Yet a Better Error Explanation Algorithm (Extended Abstract)

Heinz Riener*
*Institute of Computer Science,
Universität Bremen, Germany
{hriener,fey}@informatik.uni-bremen.de

Görschwin Fey*†
†Institute of Space Systems,
German Aerospace Center, Germany
goerschwin.fey@dlr.de

Error explanation [GV03, GCKS06] is a formal approach to automate diagnosis of software programs with the aid of a *Satisfiability* (SAT)-based model checker. Firstly, the semantics of the program is modeled as a *Finite State Machine* (FSM) and is encoded into an instance of the SAT problem [CKL04]. Given a specification expressed in a formal logic which does not hold on the FSM, error explanation utilizes the model checker to produce a pair of similar failing and successful execution traces and highlights the differences of the execution traces as a possible explanation of the error. Thus, an *explanation* corresponds to a set of locations of the program source.

More precisely, in SAT-based model checking execution traces correspond to assignments of logic variables in the SAT instance. The logic variables are used to capture the possible valuations of the program variables. The domains of the logic variables depend on the logic in use. Similarity of execution traces can then be expressed leveraging a distance metric which counts the number of values for which the two execution traces are different. Suppose (A, B) is a pair of a failing and a successful execution trace which correspond to the sequences $a = (a_1, \ldots, a_n)$ and $b = (b_1, \ldots, b_n)$ of logic variables, respectively. The distance metric is then defined as $\sum_{i=1}^{n}(1 - \delta_{a_i b_i})$ where the *Kronecker delta function* δ_{ij} evaluates to 1 only if $i = j$ and to 0 otherwise.

Experiments on practical examples, however, revealed that using this distance metric often leads to *useless* explanations which are characterized by the fact that they do not contain the location of the real error. This may happen because error explanation produces a successful execution trace altering an input which avoids the erroneous part of the program source. As a consequence, the logic variables encoding the program variables in the erroneous part of the source code become don't cares and a SAT-based model checker can assign them any value during SAT checking. In this case, error explanation needs guidance by a human, who provides additional assumptions to explain an error accurately. Groce et al. [GCKS06] called this phenomenon the *implication-antecedent problem* when they observed this issue during property checking. For particular properties of the form $(A \rightarrow C)$, error explanation computed successful execution traces which do not satisfy the antecedent A of the implication rendering the explanation useless. In this simple case, the problem can be solved by additionally assuming that A has to hold. In general, however, more complex assumptions are needed to guide error explanation.

In this extended abstract, we outline a new distance metric for an improved error explanation in order to reduce the impact of the implication-antecedent problem. Our goal is to further reduce the

manual effort needed to localize errors. Moreover, instead of property checking we focus on equivalence checking. In equivalence checking, the specification is given as a reference implementation and thus the properties are not explicitly known.
The new distance metric considers whether program source has been executed: we introduce one additional Boolean variable v_x for each logic variable x which keeps track of whether the value of the logic variable is a don't care and add it to the SAT instance. Again, suppose $a = (a_1, \ldots, a_n)$ and $b = (b_1, \ldots, b_n)$ are two sequences of logic variables corresponding to the execution traces A and B. The new distance metric is defined as $\sum_{i=1}^{n}(1 - \hat{\delta}_{a_i b_i})$, where $\hat{\delta}_{ij}$ evaluates to 1 if $i = j$ and $v_i = 1$ and $v_j = 1$ hold and evaluates to 0 otherwise. Thus, two logic variables are only considered equal if none of the two logic variables is a don't care during SAT checking. As a drawback of the approach, keeping track of the don't care information is costly, because we have to add n additional Boolean variables for a SAT instance with n logic variables.
We have implemented the improved error explanation procedure which uses the new distance metric into the FAuST framework [RF12a]. FAuST is a software model checking tool for ANSI-C programs similar to CBMC [CKL04] which provides additional testing and debugging capabilities [RF12b]. The FAuST framework is built on top of metaSMT [HFF+11], a generic interface to multiple SAT and *SAT Modulo Theories* (SMT) solvers. We have extended metaSMT to provide a new generic meta-solver which automatically synthesizes a Boolean variable for each logic variable and conservatively computes whether the value of a variable can be guaranteed to be concrete. In particular, when the Boolean variable is 1 the logic variable is concrete. However, when the Boolean variable is 0 the logic variable may be concrete or don't care.

References

[CKL04] Clarke, Edmund, Daniel Kroening, and Flavio Lerda: *A tool for checking ANSI-C programs*. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 168–176, 2004.

[GCKS06] Groce, Alex, Sagar Chaki, Daniel Kroening, and Ofer Strichman: *Error explanation with distance metrics*. International Jounral of Software Tools for Technology Transfer, 8(3):229–247, 2006.

[GV03] Groce, Alex and Willem Visser: *What went wrong: Explaining counterexamples*. In *Interational SPIN Workshop on Model Checking of Software*, pages 121–135, 2003.

[HFF+11] Haedicke, Finn, Stefan Frehse, Görschwin Fey, Daniel Große, and Rolf Drechsler: *metaSMT: Focus on your application not on solver integration*. In *International Workshop on Design and Implemention of Formal Tools and Systems*, pages 22–29, 2011.

[RF12a] Riener, Heinz and Görschwin Fey: *FAuST: A framework for Formal verification, Automated debugging, and Software Test generation*. In *Interational SPIN Workshop on Model Checking of Software*, pages 234–240, 2012.

[RF12b] Riener, Heinz and Görschwin Fey: *Model-based diagnosis versus error explanation*. In *International Conference on Formal Methods and Models for Codesign*, pages 43–52, 2012.