Reusing Information in Multi-Goal Reachability Analyses [*]

Dirk Beyer[1], Andreas Holzer[2], Michael Tautschnig[3], Helmut Veith[2]

[1] University of Passau
Software Systems
Innstrasse 33
D-94032 Passau, Germany

[2] Vienna University of Technology
Institut für Informationssysteme 184/4
Favoritenstraße 9-11
A-1040 Vienna, Austria

[3] School of Electronic Engineering and Computer Science
Queen Mary University of London
Mile End Road
London E1 4NS, UK

Abstract: It is known that model checkers can generate test inputs as witnesses for reachability specifications (or, equivalently, as counterexamples for safety properties). While this use of model checkers for testing yields a theoretically sound test-generation procedure, it scales poorly for computing complex test suites for large sets of test goals, because each test goal requires an expensive run of the model checker. We represent test goals as automata and exploit relations between automata in order to reuse existing reachability information for the analysis of subsequent test goals. Exploiting the sharing of sub-automata in a series of reachability queries, we achieve considerable performance improvements over the standard approach. We show the practical use of our multi-goal reachability analysis in a predicate-abstraction-based test-input generator for the test-specification language FQL.

Overview

We consider the problem of performing many reachability queries on a program that is given as source code. Querying a model checker repeatedly for path-sensitive reachability information [BCH$^+$04b] has many interesting applications, e.g., to decompose verification tasks, but most prominently to generate test cases from counterexample paths [BCH$^+$04a, HSTV08]. If, for example, we want to achieve basic-block coverage, we will for each basic block b try to construct a path through the program that witnesses a program execution that reaches b. In our approach, we describe test-coverage criteria using the coverage-specification language FQL [HSTV08, HSTV09, HSTV10, HTSV10, Hol13], which provides a concise specification of complex coverage criteria. We translate an FQL coverage criterion into a (possibly huge) set of test goals. Each such test goal is represented as a finite automaton, called *test-goal automaton*, and specifies a reachability query. The model-checking engine then takes a test-goal automaton to restrict the state-space search to the specified paths (for which test cases are desired). Test-goal automata often have identical parts which let us reuse analysis results across several queries. We developed an

[*]This is a summary of [BHTV13]. This work was supported by the Canadian NSERC grant RGPIN 341819-07, by the Austrian National Research Network S11403-N23 (RiSE) of the Austrian Science Fund (FWF), by the Vienna Science and Technology Fund (WWTF) grant PROSEED, and by the EPSRC project EP/H017585/1.

approach that exploits the automaton structure of reachability queries to efficiently reuse reachability results when solving multiple queries. Given two test-goal automata A and A', we introduce the notion of *similarity of A and A' modulo a set X of transitions*, where X is a subset of the transitions of A'. We then identify potentially shared behavior between A and A' via a *simulation-modulo-X relation H* between the states of A and A'. If two states s and s' are related via H, then each sequence of A'-transitions starting in s' and not including a transition from X corresponds to an equivalent sequence of A-transitions starting in s. This allows us to reason about feasibility of program executions that are covered by A' based on the reachability results for A as long as we investigate transition sequences shared by both automata [BHTV13].

Because it is generally undecidable whether a test goal is satisfiable on an arbitrary given program, we use an overapproximating reachability analyses, more specifically, a CEGAR-based predicate abstraction [BKW10], to approximate the set of executions of a program until we either (i) have found a partial program execution that is described by a word in the language of the test-goal *or* (ii) we have shown that there is no such execution. The test-goal automaton guides the reachability analysis, i.e., the analysis tracks program and automaton states simultaneously and stops exploring the state space if there is no possible transition in the program state space or no possible next automaton transition. Based on the excluded transitions X, we reuse parts of the already analyzed state space (those parts which do not involve these transitions) or continue state-space exploration along the transitions in X. We implemented our approach in the test-input generator CPA/TIGER[1].

References

[BCH+04a] D. Beyer, A. J. Chlipala, T. A. Henzinger, R. Jhala, and R. Majumdar. Generating Tests from Counterexamples. In *Proc. ICSE*, pages 326–335. IEEE, 2004.

[BCH+04b] D. Beyer, A. J. Chlipala, T. A. Henzinger, R. Jhala, and R. Majumdar. The BLAST Query Language for Software Verification. In *Proc. SAS*, LNCS 3148, pages 2–18. Springer, 2004.

[BHTV13] D. Beyer, A. Holzer, M. Tautschnig, and H. Veith. Information Reuse for Multi-goal Reachability Analyses. In *Proc. ESOP*, pages 472–491, 2013.

[BKW10] D. Beyer, M. E. Keremoglu, and P. Wendler. Predicate Abstraction with Adjustable-block Encoding. In *Proc. FMCAD*, pages 189–198. FMCAD Inc, 2010.

[Hol13] A. Holzer. *Query-Based Test-Case Generation*. PhD thesis, TU Vienna, 2013.

[HSTV08] A. Holzer, C. Schallhart, M. Tautschnig, and H. Veith. FSHELL: Systematic Test Case Generation for Dynamic Analysis and Measurement. In *Proc. CAV*, pages 209–213. Springer, 2008.

[HSTV09] A. Holzer, C. Schallhart, M. Tautschnig, and H. Veith. Query-Driven Program Testing. In *Proc. VMCAI*, pages 151–166. Springer, 2009.

[HSTV10] A. Holzer, C. Schallhart, M. Tautschnig, and H. Veith. How did You Specify Your Test Suite. In *Proc. ASE*, pages 407–416. ACM, 2010.

[HTSV10] A. Holzer, M. Tautschnig, C. Schallhart, and H. Veith. An Introduction to Test Specification in FQL. In *Proc. HVC*, pages 9–22. Springer, 2010.

[1]`http://forsyte.at/software/cpatiger/`