Report from Dagstuhl Seminar 13021

Symbolic Methods in Testing

Edited by

Thierry Jéron[1], Margus Veanes[2], and Burkhart Wolff[3]

1 **INRIA – Rennes – Bretagne Atlantique, FR,** `Thierry.Jeron@inria.fr`
2 **Microsoft Research – Redmond, US,** `margus@microsoft.com`
3 **Université Paris Sud, FR,** `wolff@lri.fr`

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 13021 "Symbolic Methods in Testing". The aim of the seminar was to bring together leading researchers of this field; the seminary ended up with 38 participants from 10 countries: France, The Netherlands, The Unites States, Germany, Switzerland, United Kingdom, Brazil, Norway, Estonia and Italy. Through a series of presentations, discussions, and working group meetings, the seminar attempted to get a coherent picture of the field, which transcends the borders of applications and disciplines, of existing approaches and problems in formal testing. The seminar brought together, on the one hand, researchers from the different camps and various tools. The main outcome of the seminar is the exchange of information between different groups and the discussion of new trends (parallelization, cloud-computing).

Seminar 06.–11. January, 2013 – www.dagstuhl.de/13021
1998 ACM Subject Classification D.2.5 Testing and Debugging (Symbolic execution, Testing tools (e.g., data generators, coverage testing), Tracing), D.2.4 Software/Program Verification (Formal Methods, Assertion Checkers, Class invariants, Programming by Contract, Validation), B.2.3 Reliability, Testing, and Fault-Tolerance (Test generation)
Keywords and phrases Automated Deduction, White-box testing, Black-box Testing Fuzz-Testing Unit-Testing Theorem prover-based Testing
Digital Object Identifier 10.4230/DagRep.3.1.1

1 Executive Summary

Thierry Jéron
Margus Veanes
Burkhart Wolff

Recent breakthroughs in deductive techniques such as satisfiability modulo theories (SMT), abstract interpretation, model-checking, and interactive theorem proving, have paved the way for new and practically effective techniques in the area of software testing and analysis. It is common to these techniques that statespaces, model-elements, program-fragments or automata are represented symbolically making systems amenable to analysis that have formerly been out of reach. Several research communities apply similar techniques to attack the classical problem of state space explosion by using symbolic representation and symbolic execution: parametrized unit testing, fuzz testing, model-based testing, theoremprover based test case generation techniques, and real-time system testing. Moreover, several areas where symbolic methods are used in testing, are often considered more closely related to verification and end up in conferences specialized on those topics rather than at testing conferences. There is little synergy between the different communities although many of them use similar underlying symbolic techniques.

In the following areas, symbolic analysis techniques have recently had significant impact, both industrially as well as in academia. The following areas capture some topics of interest for the proposed seminar, assuming focus on the use of symbolic techniques in each area: Unit Testing, Symbolic Automata Theory in Testing, Model Based Testing, Fuzz Testing, Security Testing, Real-time System Testing, Theorem-Prover-based Test-Case Generation, Hybrid System Testing, and Mutation Testing.

2 Table of Contents

3 Overview of Talks

3.1 Symbolic methods for efficient mutation testing

Sébastien Bardin (CEA – Gif sur Yvette, FR)

Automatic white-box test data generation techniques have recently made huge progress, especially through the Dynamic Symbolic Execution (DSE) paradigm. However, these methods are limited to rather basic coverage criteria, typically instruction or decision coverage. On the other hand, mutation is a powerful testing criterion, but it is poorly supported by current automatic testing tools. We present in this talk some ongoing work aiming both at efficiently automating mutation testing and at leveraging DSE to sophisticated coverage criteria. We show that (a subset of) weak mutations can be reduced to predicate reachability, allowing to reuse all the standard machinery developed for software verification in the framework of weak mutations. Especially, we focus on the following issues: automatic test generation – ATG – in order to achieve high mutation score (using DSE and smart instrumentation), automatic mutant-equivalence checking (using static analysis and theorem proving) and efficient computation of the mutation score (again through instrumentation). Especially, for ATG, while a direct instrumentation yields an exponential blowup of the search space, we present a tight instrumentation with only a linear growth of the search space.

3.2 Towards symbolic and timed testing with JTorX

Axel Belinfante (University of Twente, NL)

We describe our model-based testing tool JTorX and the current plans for extensions to improve the support of symbolic and timed testing. JTorX [1, 3] is a tool for online (on-the-fly) test derivation and execution, based on the ioco-theory and its extensions uioco (underspecification), tioco (time) and sioco (data). Tests are derived from models given in Graphml, Aldebaran (.aut), GraphViz, mCRL2, STS-as-XML (.sax) or as a network of timed automata. Alternatively, JTorX interfaces to all models accessible via the LTSmin or CADP tool sets. JTorX accesses models on-the-fly (on demand), which allows it to deal with infinite models, as long as they are finitely branching. It can be used in 2 modes: manual (interactive), or automatic (random). In either mode, an optional test purpose (model to guide test derivation) can be specified. It has built-in adapter support to connect to a model used as system-under-test, and to connect to implementations that interact using labels of the model, over standard input/output, or via TCP. The latter adapter also comes in a time-aware variant. JTorX can be used (in automatic mode) from the command line, or via a GUI. During a test run, the GUI offers visualization of the testing progress in the model, and in a message sequence chart. In addition, the GUI contains a built-in interactive simulator, and a checker (by Lars Frantzen) that checks whether two non-symbolic models are (u)ioco-related. JTorX is used for teaching and for case studies; recent ones include testing of a software bus at Neopost [5], of an X-ray detector at PANalytical, and of railway interlocking software. For most of above-mentioned formalisms, the model is accessed through a uniform model-access interface, via separate, formalism-specific, exploration components (for a few automaton-based formalisms, support is built-in in JTorX). The model-access interface gives access, not to an LTS, QTS or STS, but to a PTS: a parameterized transition system. A PTS has two kinds of transitions: those labeled with a parameterized label, and those labeled with an instantiation. A parameterized label is like an LTS label, but with parameters (like an open term) and a constraint over the parameters (and possibly, over parameters that have been introduced in an earlier transition on the path from the initial state, and that have not yet been instantiated). An instantiation is a list of parameter-value bindings. The interface allows JTorX to request(1) the (id of the) initial state, and (2) the outgoing transitions of a given a state id, where for each transition a parameterized label and the destination state id are given. Moreover, it allows JTorX to propose an instantiation, to feed back to the model, the concrete parameter values that were used during an interaction with the system under test. When the instantiation succeeds, i.e. when the PTS contains a corresponding transition, the (id of the) destination is given to JTorX. Semantic manipulation of parameterized labels is not done in JTorX, but in the formalism-specific model explorers. When JTorX needs concrete values for the parameters in a parameterized label that it wants to use as stimulus, it obtains those from a formalism-specific instantiator (which can either be part of the formalism-specific model explorer, or a separate tool component). JTorX uses this interface to access models via Lars Frantzen's STSimulator [4], and Henrik Bohnenkamp's timed automata explorer [2]. (An extension to STSimulator has been proposed and implemented in a JTorX variant: lazy-on-the-fly, by David Farago). For timed testing, each label contains a parameter (or, when instantiated, a value) that represents the time interval (or timestamp) at which the corresponding interaction with the SUT has to take (or has taken) place. The time-aware adapter variant takes care of applying stimuli in time, and of time-stamping observations.

References

1 Axel Belinfante. *JTorX: A Tool for On-Line Model-Driven Test Derivation and Execution.* In: Proceedings of TACAS 2010. LNCS, vol. 6015, pp. 266-270. Springer (2010)

2 Henrik Bohnenkamp, Axel Belinfante. *Timed testing with TorX.* In: FM 2005, Newcastle, UK. LNCS, vol. 3582, pp. 173-188. Springer (2005)

3 *JTorX Web page*: http://fmt.ewi.utwente.nl/tools/jtorx/

4 *STSimulator project page*: https://stsimulator.dev.java.net/

5 Sijtema, M. and Stoelinga, M.I.A. and Belinfante, A.F.E. and Marinelli, L. *Experiences with Formal Engineering: Model-Based Specification, Implementation and Testing of a Software Bus at Neopost.* In: Proceedings of FMICS 2011. LNCS, vol. 6959, pp. 117-133. Springer (2011)

3.3 An update on Z3

Nikolaj Bjørner (Microsoft Research – Redmond, US)

The overview talk provides a high-level summary of current directions for the SMT solver Z3 from Microsoft Research and newer applications. These include efficient engines for solving non-linear polynomial arithmetic, checking satisfiability of Horn clauses for symbolic model checking of software. I will also summarize the main (newer) API features of Z3 and illustrate some of the ways they can be used by applications. Z3 was recently made available as shared source on z3.codeplex.com.

3.4 Counterexamples for Isabelle: Ground and Beyond

Jasmin Christian Blanchette (TU München, DE)

Joint work of Berghofer, Stefan; Bulwahn, Lukas; Nipkow, Tobias;
Main reference J.C. Blanchette, L. Bulwahn, T. Nipkow, "Automatic proof and disproof in Isabelle/HOL," in FroCoS 2011, pp. 12–27, LNAI 6989, Springer, 2011.
URL http://dx.doi.org/10.1007/978-3-642-24364-6_2

Users of the Isabelle/HOL proof assistant can rely on two counterexample generators to test their conjectures: Nitpick and Quickcheck. Nitpick grounds problems to SAT, but some of its optimizations are symbolic. Quickcheck combines three ground strategies and one symbolic strategy under one roof: random testing, bounded exhaustive testing, mode-based predicate compilation, and narrowing. For future work, we would like to expand the supported HOL fragment of both tools through more symbolic methods.

3.5 Model-based Conformance Testing of Security Properties

Achim D. Brucker (SAP Research – Karlsruhe, DE)

Joint work of Brucker, Achim D.; Brügger, Lukas; Wolff, Burkhart
Main reference A.D. Brucker, L. Brügger, P. Kearney, B. Wolff, "Verified Firewall Policy Transformations for Test-Case Generation," in Third Int'l Conf. on Software Testing, Verification, and Validation (ICST), pp. 345–354, IEEE CS, 2010.
URL http://www.brucker.ch/projects/hol-testgen

Modern systems need to comply with large and complex security policies (e. g., based on company rules, privacy laws, or regulations such as HIPAA or SOX) that need to be enforced at runtime. These policies are often expressed in declarative languages such as XACML (in case of high-level access control policies) or iptables (in case of firewall policies) and enforced by highly-efficient (and, thus, difficult to implement) enforcement engines. The correct configuration of such systems is highly error-prone, mainly due to their complexity. However, for the overall security both the correct implementation of the enforcement engines as well as their correct configuration is crucial. We are addressing these issues by presenting an approach for the model-based conformance testing of security policies. It is using HOL-TestGen [5, 4], a mode-based testing tool based on an interactive theorem proving environment. In more detail, we present a model-based testing approach encompassing the complete testing process using modular specifications of security policies (e.g., access control policies [2] or firewall policies [1, 3]). The generated test cases can be used for both testing the correctness of the security infrastructure as well as the conformance of its configuration to a high-level security policy. A particular emphasis is put on a partial solution to the scalability problem inherent in model-based policy testing.

References

1 Achim D. Brucker, Lukas Brügger, Paul Kearney, and Burkhart Wolff. *Verified firewall policy transformations for test-case generation.* In Third International Conference on Software Testing, Verification, and Validation (ICST), pages 345–354. IEEE Computer Society, 2010.

2 Achim D. Brucker, Lukas Brügger, Paul Kearney, and Burkhart Wolff. *An approach to modular and testable security models of real-world health-care applications.* In ACM SACMAT, pages 133–142. ACM Press, 2011.

3 Achim D. Brucker, Lukas Brügger, and Burkhart Wolff. *Model-based firewall conformance testing.* In Kenji Suzuki and Teruo Higashino, editors, Testcom/FATES 2008, number 5047 in LNCS, pages 103–118. Springer, 2008.

4 Achim D. Brucker and Burkhart Wolff. *HOL-TestGen: An interactive test-case generation framework.* In Marsha Chechik and Martin Wirsing, editors, Fundamental Approaches to Software Engineering (FASE), number 5503 in LNCS, pages 417–420. Springer, 2009.

5 Achim D. Brucker and Burkhart Wolff. *On theorem prover-based testing.* Formal Aspects of Computing, 2012.

3.6 Symbolic Execution for Evolving Software

Cristian Cadar (Imperial College London, GB)

Joint work of Cadar, Cristian; Marinescu, Paul; Collingbourne, Peter; Kelly, Paul

One of the distinguishing characteristics of software systems is that they evolve: new patches are committed to software repositories and new versions are released to users on a continuous basis. Unfortunately, many of these changes bring unexpected bugs that break the stability of the system or affect its security. In this talk, I describe two techniques based on symbolic execution for testing and verifying evolving software: a technique for automatic and comprehensive testing of code patches, which combines symbolic execution with several novel heuristics based on static and dynamic program analysis; and a technique that combines symbolic execution and crosschecking to test and verify the correctness of optimizations such as SIMD and GPGPU optimizations.

3.7 Collaborative Verification and Testing with Explicit Assumptions

Maria Christakis (ETH Zürich, CH)

Joint work of Christakis, Maria; Müller, Peter; Wüstholz, Valentin;
Main reference M. Christakis, P. Müller, V. Wüstholz, "Collaborative Verification and Testing with Explicit Assumptions," in FM 2012: Formal Methods, pp. 132–146, LNCS, 7436, Springer, 2012.
URL http://dx.doi.org/10.1007/978-3-642-32759-9_13

Most mainstream static program checkers make a number of compromises to increase automation, improve performance, and reduce both the number of false alarms and the annotation overhead for the programmer. These compromises include not checking certain program properties, and making implicit, unsound assumptions. As a result, static checkers that make such compromises cannot provide definite guarantees about program correctness, thus rendering unclear which properties remain to be tested. To address this problem, we have proposed a tool architecture that: 1) makes the compromises of static checkers explicit and their verification results precise with a simple language extension that facilitates collaborative verification, i.e., the integration of multiple, complementary static checkers, and 2) reinforces static checking with automated, specification-based testing to make up for any soundness limitations of the upstream checkers. In this context, we have also discussed an approach for testing object invariants.

3.8 A Certified Constraint Solver over Finite Domains

Catherine Dubois (ENSIIE – Evry, FR)

Joint work of Carlier, Matthieu;Dubois, Catherine; Gotlieb, Arnaud;
Main reference M. Carlier, C. Dubois, A. Gotlieb, "A Certified Constraint Solver over Finite Domains," in Formal Methods – 18th International Symposium (FM 2012) , Vol. 7436 of LNCS: 116–131, 2012, Paris, France.
URL http://dx.doi.org/10.1007/978-3-642-32759-9_12

Constraint solvers are often used within verification or testing tools. These tools are complex, implement clever heuristics and thus may contain bugs. When these tools are used for critical software, a skeptical regard on the implementation of constraint solvers especially when the result is that a constraint problem has no solution, i.e., unsatisfiability. We review some state-of-the art solutions allowing more confidence. We present a Coq formalisation of a constraint filtering algorithm and a simple labeling procedure, focing on arc-consistency and bound-consistency. The proof of their soundness and completeness has been completed using Coq. As a result, a formally certified constraint solver written in OCaml (the first one as far as we know) has been automatically extracted from the Coq development. The solver, yet not as efficient as specialized existing (unsafe) implementations, can be used to formally certify that a constraint system is unsatisfiable.

3.9 Diagnosis Modulo Theories for Hybrid Systems

Juhan Ernits (Tallinn University of Technology, EE)

Joint work of Ernits, Juhan; Dearden, Richard

Diagnosis of hybrid systems involves tracking the state of the system on the basis of observations and control input to distinguish nominal behaviour from faulty. If faulty behaviour is encountered the approach proceeds to identify the causes of faults. Consistency-based diagnosis (CBD) is one of possible approaches designed to achieve such goals. In CBD the nominal and faulty behaviour of the system are modelled in terms of constraints present in each state and transitions between those states. To date the models used are mostly discrete, so system variables must be discretised before diagnosis can be performed. We introduce a new approach that uses a satisfiability modulo theories (SMT) solver in the implementation of the conflict directed A* algorithm – the core of the consistency-based diagnosis procedure. It is thus possible to use constraints from the theories supported by the SMT solver in the model, which is novel in diagnosis and provides a new application for

SMT solvers. The application has become practical only recently due to the integration of efficient non linear arithmetic theory decision procedures into SMT solvers.

3.10 Theorem-Prover Based Test Generation for Circus

Abderrahmane Feliachi (Université Paris Sud, FR)

Joint work of Feliachi, Abderrahmane; Gaudel, Marie-Claude; Wolff, Burkhart

HOL-TestGen [2] is a theorem-prover based environment for specification and test generation. Starting from a data-oriented (HOL) specification of a system under test, HOL-TestGen automatically derives test cases and test data for this system. Built on top of the Isabelle/HOL theorem prover, it allow for combining test generation tactics with symbolic computations and proof methods in a sound formal way. Since real complex systems combines complex data and behavioral aspects. We introduce, in the basis of Isabelle/HOL, a formal environment [1] for specifying and verifying complex systems. Specifications are written in Circus [3], a combination of Z and CSP, with a well defined and unified semantics. The semantics embedding of the Circus language in HOL is the basis of our environment. It makes it possible to reason on Circus specifications using HOL standard rules and proof methods. This environment is combined with HOL-TestGen, to provide a test generation environment covering complex data and behavioral aspects. Different symbolic representations and computations are used to define and generate tests from Circus specifications. This includes the embedding of symbolic variables, constraints over them and the resolution of these constraints for test data generation. A concrete application of this testing environment on a real system is presented in this talk.

References

1 Abderrahmane Feliachi, Marie-Claude Gaudel, and Burkhart Wolff. *Isabelle/Circus: A process specification and verification environment.* In Rajeev Joshi, Peter Müller, and Andreas Podelski, editors, Verified Software: Theories, Tools, Experiments, volume 7152 of Lecture Notes in Computer Science, pages 243–260. Springer Berlin / Heidelberg, 2012.
2 Achim Brucker and Burkhart Wolff. *On theorem prover-based testing.* Formal Aspects of Computing, pages 1–39, 2012. 10.1007/s00165-012-0222-y.
3 Jim Woodcock and Ana Cavalcanti. *The semantics of Circus.* In Proceedings of the 2nd International Conference of B and Z Users on Formal Specification and Development in Z and B, (ZB '02), pages 184–203, London, UK, UK, 2002. Springer-Verlag.

3.11 Off-Line Test Case Generation for Timed Symbolic Model-Based Conformance Testing

Christophe Gaston (CEA – Gif sur Yvette, FR)

Joint work of Bannour, Boutheina; Escobedo, Jose Pablo; Gaston, Christophe; Le Gall, Pascale
Main reference B. Bannour, J. P. Escobedo, C. Gaston, P. Le Gall, "Off-Line Test Case Generation for Timed Symbolic Model-Based Conformance Testing," in Proc. of the 24th Int'l Conf. on Testing Software and Systems, ICTSS 2012, pp. 119–135, LNCS, Volume 7641, Springer.

Model-based conformance testing of reactive systems consists in taking benefit from the model for mechanizing both test data generation and verdicts computation. On-line test case generation allows one to apply on- the-fly analysis techniques to generate the next inputs to be sent and to decide whether or not observed outputs meet intended behaviors. On the other hand, in off-line approaches, test suites are pre-computed from the model and stored under a format that can be later performed on test benches. In this seminar, we presented an off-line approach in two phases: (1) for the test generation part, a test suite is a predefined timed sequence of input data; (2) For the verdict production part, a post treatment is performed based on an analysis of the timed sequence of output data produced by the system under test with respect to the model. Our models are Timed Input Output Symbolic Transition Systems. Therefore, our off-line algorithm involves symbolic execution and constraint solving techniques.

References

1 Boutheina Bannour, Jose Pablo Escobedo, Christophe Gaston and Pascale Le Gall. *Off-Line Test Case Generation for Timed Symbolic Model-Based Conformance Testing*. In Proceedings of the 24th IFIP WG 6.1 International Conference on Testing Software and Systems, ICTSS 2012. Springer, LNCS, Volume 7641, pages 119–135. Aalborg, Denmark, November 19–21, 2012.

3.12 Distributed and Asynchronous Model Based Testing

Robert M. Hierons (Brunel University, GB)

Some systems interact with their environment at several physically distributed interfaces, called ports, and when testing such a system it is normal to place a local tester at each port. If the local testers cannot interact with one another during testing and there is no global clock, then each local tester observes only the sequence of inputs and outputs at its interfaces (a local trace). This can make it impossible to reconstruct the global trace that occurred. Similarly, we might not directly observe the input and output of the system under test (SUT) if there is an asynchronous communications channel between the tester and the SUT: the observation of output produced by the SUT is delayed, as is the SUT receiving input from the tester. Both situations lead to some loss of information regarding the sequence of events that the SUT performed and so they change the nature of testing and require us to define new implementation relations [6, 1]. They also affect certain standard testing problems. For example, in distributed testing it is undecidable whether there is a test case that is guaranteed to take a model to a particular state or to distinguish two given states and this result holds even if we are testing from a deterministic finite state machine [3]. It is also undecidable whether there is a test case that is capable of distinguishing two states [2] and the Oracle problem is NP-hard [5]. There are currently fewer results for asynchronous testing but some problems, such as checking conformance, are known to be undecidable [1]. However, some test generation problems are decidable when there are FIFO channels and we are testing from a finite state model that is not output-divergent (there are no states from which one can take an infinite sequence of transitions without receiving an input) [4].

References

1 R. M. Hierons. *Implementation relations for testing through asynchronous channels.* The Computer Journal, to appear.

2 R. M. Hierons. *Verifying and comparing finite state machines for systems that have distributed interfaces.* IEEE Transactions on Computers, to appear.

3 Robert M. Hierons. *Reaching and distinguishing states of distributed systems.* SIAM Journal on Computing, 39(8):3480–3500, 2010.

4 Robert M. Hierons. *The complexity of asynchronous model based testing.* Theoretical Computer Science, 451:70–82, 2012.

5 Robert M. Hierons. *Oracles for distributed testing.* IEEE Transactions on Software Engineering, 38(3):629–641, 2012.

6 Robert M. Hierons, Mercedes G. Merayo, and Manuel Núñez. *Implementation relations and test generation for systems with distributed interfaces.* Distributed Computing, 25(1):35–62, 2012.

3.13 Model based conformance testing with ioco/tioco and Symbolic techniques

Thierry Jéron (INRIA Bretagne Atlantique – Rennes, FR)

Joint work of Jéron, Thierry; Jard, Claude; Jeannet, Bertrand; Rusu, Vlad; Zinovieva, Elena; Bertrand, Nathalie; Stainer, Amélie; Krichen, Moez; Chédor, Sébastien; Morvan, Christophe

This talk reviews some of the works of our group related to automatic test generation for reactive systems in the ioco/tioco testing framework. Starting with the finite state model case, we review the main ingredients of the ioco testing theory. We also generalize ioco to the io-refinement/abstraction pre-order relation. As the io-abstraction relation preserves the soundness of test cases, we show that it is a key for test generation when undecidability questions arise, due to infinity. We recall the principles of off-line test generation using test purposes, in the finite state case. The generation process is based on suspension, determinization and co-reachability analysis. We then extend test generation to infinite state systems, trying to mimic the finite state case. In particular, we focus on symbolic and approximation methods used to overcome problems due to infinite state as well as undecidability problems. First for models with data (IOSTS), limited to the deterministic case (determinization is an issue for this model), we show how abstract interpretation can be used for approximate co-reachability analysis. The consequences for test generation are examined: preservation of soundness and exhaustiveness, but loss of control to target. We then consider the case of timed automata (TAIO) in the general case (with internal actions and non-determinism, invariants for urgency). The principles of approximate determinization using games and symbolic represenations by regions or zones, which produces an io-abstraction of a TAIO, is sketched. On the other side, co-reachability is exact, thanks to the abstract symbolic representation by zones and regions. Overall the test generation process allows to preserve the soundness of test cases, but may lead to a loss of exhaustiveness, stricness and precision only when determinization is not exact. recursion or time (in the tioco extension for real-time systems). Finally, the last (unpresented) part deals with recursive models in the form of recursive tiles systems (RTS), a sort of graph grammars. Determinization of RTSs being an issue, off-line test generation is restricted to a determinizable class, weighted RTSs, while on-line test generation can be applied on the full model. On the other hand, co-reachability is exact in the general case and can be decided on the RTS. Overall, the properties of soundness, exhaustiveness, strictness and precision of test cases are preserved in both cases.

References

1 C. Jard, T. Jéron. *TGV: theory, principles and algorithms, A tool for the automatic synthesis of conformance test cases for non-deterministic reactive systems.* Software Tools for Technology Transfer (STTT), 6, October 2004.

2 B. Jeannet, T. Jéron, V. Rusu, E. Zinovieva. Symbolic *Test Selection based on Approximate Analysis.* In 11th Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'05), LNCS, Volume 3440, pages 349–364, Edinburgh (Scottland), April 2005.

3 N. Bertrand, T. Jéron, A. Stainer, M. Krichen. *Off-line Test Selection with Test Purposes for Non-Deterministic Timed Automata.* In 17th International Conference on Tools and Algorithms for the Construction And Analysis of Systems (TACAS'11), LNCS, Volume 6605, pages 96–111, Saarbrücken, Germany, April 2011.

4 S. Chédor, T. Jéron, C. Morvan. *Test generation from recursive tiles systems.* In 6th International Conference on Tests & Proofs (TAP'12), LNCS, Volume 7305, pages 99–114, Prague, May 2012.

3.14 Using State Infection Conditions to Detect Equivalent Mutants and Speed up Mutation Analysis

René Just (University of Washington – Seattle, US)

Joint work of Just, René; Ernst, Michael D.; Fraser, Gordon
Main reference R. Just, M.D. Ernst, G. Fraser, "Using State Infection Conditions to Detect Equivalent Mutants and Speed up Mutation Analysis," arXiv:1303.2784v1 [cs.SE].
URL http://arxiv.org/abs/1303.2784

Mutation analysis evaluates test suites and testing techniques by measuring how well they detect seeded defects (mutants). Even though well established in research, mutation analysis is rarely used in practice due to scalability problems — there are multiple mutations per code statement leading to a large number of mutants, and hence executions of the test suite. In addition, the use of mutation to improve test suites is futile for mutants that are equivalent, which means that there exists no test case that distinguishes them from the original program. This paper introduces two optimizations based on state infection conditions, i.e., conditions that determine for a test execution whether the same execution on a mutant would lead to a different state. First, redundant test execution can be avoided by monitoring state infection conditions, leading to an overall performance improvement. Second, state infection conditions can aid in identifying equivalent mutants, thus guiding efforts to improve test suites.

References

1 R. Just, G. M. Kapfhammer, and F. Schweiggert. *Using non-redundant mutation operators and test suite prioritization to achieve efficient and scalable mutation analysis*. In Proc. of the 23rd ISSRE. IEEE Computer Society, 2012.

2 R. Just, F. Schweiggert, and G. M. Kapfhammer. *MAJOR: An efficient and extensible tool for mutation analysis in a Java compiler*. In Proc. of the 26th ASE, IEEE Computer Society, 2011.

3.15 A Symbolic Approach to Model-based Online Testing

Marko Kaeaeramees (Tallinn University of Technology, EE)

Joint work of Kääramees, Marko;
Main reference M. Kääramees, “A Symbolic Approach to Model-based Online Testing,” PhD thesis, 2012. Tallinn University of Technology, Department of Computer Science, Estonia.
URL http://digi.lib.ttu.ee/i/?806

Non-deterministic control structures and data components provide a powerful means of abstraction for high level modelling of complex systems, at the expense of making automated test generation more challenging. Online model- based testing where test inputs are computed from the model and outputs fed back to the tester at the time of testing provides an approach where testing non-deterministic systems is possible. One of the restrictions to more widespread use of online model-based testing is the relatively high omputational overhead at runtime. We introduce an approach that addresses the computational overhead issue of online testing by pre-computation of test strategies based on the model and a formally specified test purpose. The proposed method allows the model of the IUT to be formalised as an Extended Finite State Machine over different first- order background theories. Both reachability and coverage oriented test purposes can be expressed using constraints attributed to edges of the model, called traps. We show how a testing strategy can be represented symbolically by a set of constraints and generated from the model and test purpose offline using symbolic backwards reachability analysis. The strategy can be used in online testing for efficient test input generation that guides the IUT towards fulfilment of the test purpose. The method is supported by the latest achievements of Satisfiability Module Theories (SMT) solver technology.

References

1 Marko Kääramees. *A Symbolic Approach to Model-based Online Testing*. PhD thesis, 2012. Tallinn University of Technology, Department of Computer Science, Estonia.

2 Danel Ahman and Marko Kääramees. *Constraint-based heuristic on-line test generation from non-deterministic I/O EFSMs*. In Alexander K. Petrenko and Holger Schlingloff, editors, MBT, volume 80 of EPTCS, pages 115–129, 2012.

3 Marko Kääramees, Jüri Vain, and Kullo Raiend. *Synthesis of on-line planning tester for non-deterministic EFSM models*. In Leonardo Bottaci and Gordon Fraser, editors, Testing – Practice and Research Techniques, volume 6303 of LNCS, pages 147- -154. Springer Berlin / Heidelberg, 2010.

4 Jüri Vain, Marko Kääramees, and Maili Markvardt. *Dependability and Computer Engineering : Concepts for Software-Intensive Systems*, chapter Online testing of nondeterministic systems with reactive planning tester, pages 113–150. IGI Global, Hershey, PA, 2011.

3.16 Critical Systems Development Methodology using Formal Techniques

Dominique Méry (LORIA – Nancy, FR)

License Creative Commons BY 3.0 Unported license
© Dominique Méry
Joint work of Méry, Dominique; Singh, Neeraj, Kumar;
Main reference D. Méry, N.K. Singh, "Critical systems development methodology using formal techniques," in Proc. of the 3rd Symp. on Information and Communication Technology (SoICT'12), pp. 3–12, ACM, 2012.
URL http://hal.inria.fr/hal-00747305/
URL http://dx.doi.org/10.1145/2350716.2350720

Formal methods have emerged as an alternative approach to ensuring the quality and correctness of the high confidence critical systems, overcoming limitations of the traditional validation techniques such as simulation and testing. We present a methodology for developing critical systems from requirement analysis to automatic code generation with standard safety assessment approach. This methodology combines the refinement approach with various tools including verification tool, model checker tool, real-time animator and finally, produces the source code into many languages using automatic code generation tools. This approach is intended to contribute to further the use of formal techniques for developing critical systems with high integrity and to verify complex properties, which help to discover potential problems. Assessment of the proposed methodology is given through developing a standard case study: the cardiac pacemaker. Further work remain to start for improving the development cycle by integrating testing phase for validating elements of the medical domain on the resulting system, namely the pacemaker.

References

1 D. Méry, N. K. Singh. *Functional Behavior of a Cardiac Pacing System*. International Journal of Discrete Event Control Systems (IJDECS), 2010.
2 D. Méry, N. K. Singh. *A generic framework: from modeling to code* In Innovations in Systems and Software Engineering (ISSE), pp. 1–9, 2011.
3 D. Méry, N. K. Singh. *Real-Time Animation for Formal Specification*. In Complex Systems Design & Management 2010, M. Aiguier, F. Bretaudeau, D. Krob (ed.), Springer, pp. 49–60. Paris, France, 2010.
4 D. Méry, N. K. Singh. *Trustable Formal Specification for Software Certification*. In 4th International Symposium On Leveraging Applications of Formal Methods – ISOLA 2010, T. Margaria, B. Ste (ed.), LNCS, 6416, Springer, pp. 312–326. Heraklion, Crete, Greece, 2010.
5 D. Méry, N. K. Sinhg. *Critical systems development methodology using formal techniques*.In 3rd International Symposium on Information and Communication Technology – SoICT 2012, ACM, pp. 3–12. Ha Long, Viet Nam, 2012.
6 D. Méry, N. K. Singh. *Formalization of Heart Models Based on the Conduction of Electrical Impulses and Cellular Automata* In Foundations of Health Informatics Engineering and Systems, Z. Liu, A. Wassyng (ed.), LNCS 7151, Springer Berlin Heidelberg, pp. 140–159. 2012.

3.17 Testing Real-time Systems under Uncertainty

Brian Nielsen (Aalborg University, DK)

Joint work of Nielsen, Brian; David, Alexandre; Larsen, Kim Guldstrand; Li, Shuhao; Mikucionis, Marius
Main reference A. David, K.G. Larsen, S. Li, M. Mikucionis, B. Nielsen , "Testing Real-Time Systems under Uncertainty," in Proc. of 9th Int'l Symp. on Formal Methods for Components and Objects (FMCO'10), pp. 352–371, LNCS, Vol. 6957, Springer, 2012.
URL http://dx.doi.org/10.1007/978-3-642-25271-6_19

Model-based testing is a promising technique for improving the quality of testing by automatically generating an efficient set of provably valid test cases from a system model. Testing embedded real-time systems is challenging because it must deal with timing, concurrency, processing and computation of complex mixed discrete and continuous signals, and limited observation and control. Whilst several techniques and tools have been proposed, few deals systematically with models capturing the indeterminacy resulting from concurrency, timing and limited observability and controllability. This paper proposes a number of model-based test generation principles and techniques that aim at efficient testing of timed systems under uncertainty.

3.18 Identifying suspicious values in programs with floating-point numbers

Michel Rueher (Université de Nice, FR)

Joint work of Rueher, Michel; Ponsini, Olivier; Michel, Claude
Main reference O. Ponsini, C. Michel, M. Rueher, "Refining abstract interpretation based value analysis with constraint programming techniques," in Proc. of CP 2012, LNCS, Vol. 7514, pp. 593–607, Springer, 2012.
URL http://dx.doi.org/10.1007/978-3-642-33558-7_43

Programs with floating-point computations control complex and critical physical systems in various domains such as transportation, nuclear energy, or medicine. Floating-point computations are an additional source of errors and famous computer bugs are due to errors in floating-point computations. Value analysis is often used to check the absence of run-time errors, such as invalid integer or floating-point operations, as well as simple user assertions. Value analysis can also help with estimating the accuracy of floating-point computations with respect to the same sequence of operations in an idealized semantics of real numbers. Existing automatic tools are mainly based on abstract interpretation techniques. For instance, a state-of-the-art static analyser like FLUCTUAT computes an over-approximation of the domains of the variables for a C program considered with semantics on real numbers. It also computes an over- approximation of the error due to floating-point operations at each program point. However, these over-approximations may be very coarse even for usual programming constructs and expressions. As a consequence, numerous false alarms may be generated. We introduce here a hybrid technique -called RAiCP- for value analysis of floating- point programs that combines abstract interpretation and constraint programming techniques. We show that constraint solvers over floating-point and real numbers can significantly refine the over-approximations computed by abstract interpretation. Experiments show that RAiCP is substantially more precise than FLUCTUAT, especially on C programs that are difficult to handle with abstract interpretation techniques. This is mainly due to the refutation capabilities of filtering algorithms over the real numbers and the floating-point numbers used in RAiCP. Reducing these two over-approximations is a critical issue for program testing since it shrinks the set of suspicious values that have to be tested. RAiCP could also eliminate 13 false alarms generated by FLUCTUAT on a set of 57 standard benchmarks proposed by D'Silva et al to evaluate CDFL, a program analysis tool that embeds an abstract domain in the conflict driven clause- learning algorithm of a SAT solver. Moreover, RAiCP is on average at least 5 times faster than CDFL on this set of benchmarks.

References

1 Olivier Ponsini, Claude Michel and Michel Rueher. *Refining abstract interpretation based value analysis with constraint programming techniques.* Proc of CP 2012 , Springer Verlag, LNCS 7514, pp. 593-607, 2012

3.19 Pex4Fun: Serious Gaming powered by Symbolic Execution

Nikolai Tillman (Microsoft Research – Redmond, US)

Pex4Fun (http://www.pex4fun.com/) is a web-based serious gaming environment for learning and teaching programming. With Pex4Fun, a student edits code in any browser, supported by auto-completion. Pex4Fun then executes the code and analyzes it in the cloud. The analysis is based on the automated test generator Pex which uses symbolic execution and the SMT solver Z3 to generate high coverage test suites and find counterexamples to assertions. In particular, Pex4Fun finds interesting and unexpected input values that help students understand what their code is actually doing. The real fun starts with Coding Duels where students write code to implement a teacher's specification. Pex4Fun finds discrepancies in behavior between the student's code and the specification. The student wins when Pex4Fun cannot find any behavioral differences. This tutorial will discuss the technologies that lie beneath the Pex4Fun platform, show how to use Pex4Fun in teaching and learning, explore existing course materials and illustrate how to create new puzzles.

3.20 Symbolic Automata

Margus Veanes (Microsoft Research – Redmond, US)

The symbolic automata toolkit lifts classical automata analysis to work modulo rich alphabet theories. It uses the power of state-of-the-art constraint solvers for automata analysis that is both expressive and efficient, even for automata over large finite alphabets. The toolkit supports analysis of finite symbolic automata and transducers over strings. It also handles transducers with registers. Constraint solving is used when composing and minimizing automata, and a much deeper and powerful integration is also obtained by internalizing automata as theories.

3.21 Using Interpolation for Test-Case Generation for Security Protocols

Luca Vigano (Università di Verona, IT)

Joint work of Dalle Vedove, Giacomo; Rocchetto, Marco; Vigano, Luca; Volpe, Marco

Interpolation has been successfully applied in formal methods for model checking and test-case generation for sequential programs. Security protocols, however, exhibit such idiosyncrasies that make them unsuitable to the direct application of such methods. In this paper, we address this problem and present an interpolation-based method for test-case generation for security protocols. Our method starts from a formal protocol specification and combines Craig interpolation, symbolic execution and the standard Dolev-Yao intruder model to search for goals (i.e., test cases representing possible attacks on the protocol). Interpolants are generated as a response to search failure in order to prune possible useless traces and speed up the exploration. We illustrate our method by means of two concrete examples.

3.22 Paths to property violation: a structural approach for analyzing counter-examples

Hélène Waeselynck (LAAS – Toulouse, FR)

Joint work of Bochot, Thomas; Virelizier, Pierre; Waeselynck, Hélène; Wiels, Virginie;
Main reference T. Bochot, P. Virelizier, H. Waeselynck, V. Wiels, "Paths to Property Violation: A Structural Approach for Analyzing Counter-Examples," in Proc. of IEEE 12th Int'l Symp. onHigh-Assurance Systems Engineering (HASE'10), pp. 74–83, IEEE, 2010.
URL http://dx.doi.org/10.1109/HASE.2010.15

At Airbus, flight control software is developed using SCADE formal models, from which 90% of the code can be generated. Having a formal design leaves open the possibility of introducing model checking techniques. But, from our analysis of cases extracted from real software, a key issue concerns the exploitation of counterexamples showing property violation. To address this issue, we propose an automated structural analysis that identifies paths of the model that are activated by a counterexample over time. This analysis allows us to extract minimal information to explain the observed violation. It is also used to guide the model checker toward the search for different counterexamples, exhibiting new path activation patterns. The approach is closely related to path-based analysis techniques developed for testing purposes.

3.23 An Introduction to Model-based Testing with Isabelle/HOL-TestGen

Burkhart Wolff (Université Paris Sud, FR)

Main reference A.D. Brucker, B. Wolff, "On Theorem Prover-based Testing," Accepted the 07-08-2011. In Formal Aspects of Computing.
URL http://dx.doi.org/10.1007/s00165-012-0222-y

Techniques for the automated generation of test-cases – be it from specifications in form of pre- and postconditions, from transition systems or from annotated programs – suffer from state-space explosion similarly to model-checking techniques. One possible anwer to the challenge is to use symbolic representations of models, their normal forms, symbolic states, the resulting test-cases (partitions of test-data) and constraint-solving techniques for test-data selection. HOL-TestGen is a model-based test-generation environment based on the Theorem-proving based approach. In this talk, we will give an introduction into architecture, accommodation scenarios for a wide range of testing problems ranging from unit, sequence, reactive testing scenarios, and present an overview over major case-studies done with the system. This introduction was accompanied by a tutorial available from the material website.

References

1 Achim D. Brucker, Burkhart Wolff: *On Theorem Prover-based Testing.*. In: Formal Aspects of Computing (FAOC). DOI: 10.1007/s00165-012-0222-y. Springer, 2012.
2 Achim D. Brucker, L. Brügger and Burkhart Wolff: *Model-Based Firewall Conformance Testing.* ICTSS 08, LNCS 5047, pp. 103-118, http://dx.doi.org/10.1007/978-3-540-68524-1_9, Springer, 2008.
3 Achim D. Brucker, Abderrahmane Feliachi, Yakoub Nemouchi, and Burkhart Wolff. *Test Program Generation for a Microprocessor: A Case-Study.* In TAP 2013: Tests And Proofs. To appear in LNCS, Springer-Verlag, 2013.
4 Lukas Brügger. *A Framework for Modelling and Testing of Security Policies.* ETH Dissertation No. 20513. ETH Zurich,2012.
5 Achim D. Brucker, Lukas Brügger, Paul Kearney, and Burkhart Wolff. *An Approach to Modular and Testable Security Models of Real-world Health-care Applications.* In ACM symposium on access control models and technologies (SACMAT)., pages 133-142, ACM Press, 2011.
6 Achim D. Brucker, Lukas Brügger, Paul Kearney, and Burkhart Wolff. *Verified Firewall Policy Transformations for Test-Case Generation.* In Third International Conference on Software Testing, Verification, and Validation (ICST), pages 345-354, IEEE Computer Society , 2010.
7 Matthias P. Krieger. *Test Generation and Animation Based on Object-Oriented Specifications.* University Paris-Sud XI,2011.
8 Achim D. Brucker, Matthias P. Krieger, Delphine Longuet, and Burkhart Wolff. *A Specification-based Test Case Generation Method for UML/OCL.* In MoDELS Workshops. LNCS 6627, pages 334-348, Springer-Verlag , 2010.
9 Achim D. Brucker and Burkhart Wolff. *HOL-TestGen: An Interactive Test-case Generation Framework.* In Fundamental Approaches to Software Engineering (FASE09). LNCS 5503, pages 417-420, Springer-Verlag , 2009.
10 Achim D. Brucker, Lukas Brügger, and Burkhart Wolff. *Verifying Test-Hypotheses: An Experiment in Test and Proof.* Fourth Workshop on Model Based Testing (MBT 2008). In ENTCS, 220 (1), pages 15-27, 2008.

3.24 Dijkstra's Verdict Considered Harmful

Burkhart Wolff (Université Paris Sud, FR)

Dijkstra's Verdict "Testing can only show the absence of errors, not the presence" has been very influential and even more misleading in the scientifico-political debate between model-checking, testing and deductive verification communities. Having seen too many Phds using Dijkstra's Verdict in the introduction to motivate their model-checking and proof-based verification tool in a hopelessly exaggerating way, denying the importance of testing and experimentation, I recall that no approach can guarantee the absence of bugs. In contrast, the danger of uncritical application of verification tools (based on complex memory models, implicit methodological assumptions, etc) is very real. In particular, deductive methods can in practice NOT guarantee the absence of errors in programs, as far as they are written in real programming languages and run on real machines. Strictly speaking, it can not even be safely argued that deductive methods are BETTER than testing methods, since the underlying assumptions are incomparable.

3.25 Online Verification of Value-Passing Choreographies through Property-Oriented Passive Testing

Fatiha Zaïdi (Université Paris Sud, FR)

Joint work of Huu-Nghia, Nguyen; Pascal, Poizat; Fatiha, Zaïdi
Main reference Huu Nghia Nguyen, P. Poizat, F. Zaïdi, "Online Verification of Value-Passing Choreographies through Property-Oriented Passive Testing," in Proc. of the 14th IEEE Int'l High Assurance Systems Engineering Symposium (HASE 12), pp. 106–113, IEEE, 2012.
URL http://dx.doi.org/10.1109/HASE.2012.15

Choreography supports the specification, with a global perspective, of the interactions between roles played by peers in a collaboration. Choreography conformance testing aims at verifying whether a set of distributed peers collaborates wrt. choreography. Such collaborations are usually achieved through information exchange, thus taking data into account during the testing process is necessary. We address this issue by using a non-intrusive passive testing approach based on functional properties. A property can express a critical (positive or negative) behaviour to be tested on an isolated peer (locally) or on a set of peers (globally). We support online verification of these kind of properties against local running traces of each peer in a distributed system where no global clock is needed. Our framework is fully tool supported.

3.26 Symbolic Model-Based Testing of Real-Time Systems using SYMBOLRT

Wilkerson de Lucena Andrade (Federal University of Campina Grande, BR)

The state space explosion problem is one of the challenges to be faced by test case generation techniques, particularly when data values need to be enumerated. This problem gets even worse for Real-Time Systems that also have time requirements. The usual solution consists in enumerating data values (restricted to finite domains) while treating time symbolically, thus leading to the classical state explosion problem. We propose a symbolic model for conformance testing of real-time systems software named TIOSTS that addresses both data and time symbolically [1, 3]. Moreover, a test case generation process was defined to generate tests through TIOSTS models based on a combination of symbolic execution and constraint solving for the data part and symbolic analysis for timed aspects. All the process is supported by the SYMBOLRT tool [2]. SYMBOLRT is a model-based test generation tool developed to automatically generate test cases from symbolic models in the context of real-time systems. SYMBOLRT handles both data and time requirements symbolically, avoiding the state explosion problem during the test case generation.

References

1 W. L. Andrade, P. D. L. Machado, T. Jéron, H. Marchand. *Abstracting time and data for conformance testing of real-time systems.* In 7th Workshop on Advances in Model Based Testing (A-MOST 2011) / 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), Pages 9-17, IEEE Computer Society, March 2011.
2 W. L. Andrade, D. R. Almeida, J. B. Cândido, P. D. L. Machado. *SYMBOLRT: A Tool for Symbolic Model-Based Test Case Generation for Real-Time Systems.* In 19th Tools Session of the 3rd Brazilian Conference on Software: Theory and Practice (CBSoft 2012), Pages 31-37, Natal, Brazil, September 2012.
3 W. L. Andrade, P. D. L. Machado. *Generating Test Cases for Real-Time Systems Based on Symbolic Models.* In IEEE Transactions on Software Engineering, to appear.

3.27 High-performance Analysis and Symbolic Online Test Generation

Jaco van de Pol (University of Twente, NL)

This overview talk consists of three parts: (1) A description of the LTSmin toolset for high-performance analysis, based on symbolic and multi-core model checking. The basis for the multi-core algorithms is a concurrent hashtable, with a very nice measured speedup on multi-core machines. Selecting one particular algorithm, I explain a parallel NDFS algorithm for LTL model checking. The initial variant had a small error, which was really hard to find. This work could be connected to research in symbolic testing in two directions: (a) a challenge for testing: could the error above be found by (symbolic) testing? (b) LTSmin could be used for high-performance test algorithms, e.g. determinisation, strategy synthesis, etc. (2) I shortly review my previous work in testing. First, we implemented simulated time

in TTCN-3 and applied it to model based testing of Railway Interlocking systems. Next, we proposed a symbolic online test generation algorithm, along the following lines:
- compute a data abstraction of the system (e.g. by homomorphism)
- generate an abstract test case towards some test purpose goal
- infer data parameters by constraint solving
- follow the test path for a while; re-plan when you cannot continue.

(3) Finally, I believe that the following themes are both challenging and promising as future work:
- developing robust black-box coverage criteria for symbolic testing
- optimal test case generation as strategy synthesis from a test game

As a closing remark: An important methodological issue concerning all research on testing is: What is a common yardstick to evaluate progress in testing research?

References

1 S.C.C. Blom, N. Ioustinova, J.C. van de Pol, A. Rennoch, and N. Sidorova, *Simulated Time for Testing Railway Interlockings with TTCN-3.* Proc. Formal Approaches to Software Testing (FATES'05, Edinburgh), LNCS 3997, Springer, pp. 1-15, 2006.

2 S.C.C. Blom, T. Deiß, N. Ioustinova, A. Kontio, J.C. van de Pol, A. Rennoch, and N. Sidorova, *Simulated Time for Host-Based Testing with TTCN-3.* Software Testing, Verification and Reliability, 18(1):29-49, 2008.

3 J.R. Calamé, N. Ioustinova, J.C. van de Pol, and N. Sidorova, *Data Abstraction and Constraint Solving for Conformance Testing.* Proc. Asia-Pacific Software Engineering Conference (APSEC'05, Taipei, Taiwan), IEEE, pp. 541-548, 2005.

4 Stefan Blom, Jaco van de Pol, and Michael Weber, *LTSmin: Distributed and Symbolic Reachability.* Proc. Computer Aided Verifciation (CAV'10, Edinburgh), LNCS 6174, Springer, pp. 354-359, 2010.

5 Alfons Laarman, Jaco van de Pol, and Michael Weber, *Multi-Core LTSmin: Marrying Modularity and Scalability.* Proc. Nasa Formal Methods (NFM'11, Pasadena, USA), LNCS 6617, Springer, pp. 506-511, 2011.

6 Alfons Laarman, Jaco van de Pol and Michael Weber, *Boosting Multi-Core Reachability Performance with Shared Hash Tables.* Proc. Formal Methods in Computer Aided Design (FMCAD'10, Lugano, Switzerland), IEEE, pp. 247-256, 2010.

7 Alfons Laarman, Rom Langerak, Jaco van de Pol, Michael Weber, and Anton Wijs, *Multi-Core Nested Depth-First Search.* Proc. Automated Technology for Verification and Analysis (ATVA'11, Tapei, Taiwan), LNCS 6996, Springer, pp. 321-335, 2011.

8 Tom van Dijk, Alfons Laarman and Jaco van de Pol, *Multi-core and/or Symbolic Model Checking*, Proc. Automated Verification of Critical Systems (AVOCS'12, Bamberg, Germany), ECEASST 53, 2012.

3.28 A Conformance Testing Relation for Symbolic Timed Automata

Sabrina von Styp (RWTH Aachen, DE)

Main reference Sabrina von Styp, Henrik Bohnenkamp, and Julien Schmaltz. A Conformance Testing Relation for Symbolic Timed Automata. In Proc. FORMATS 2010. pages 243-255. Volume 6246 of LNCS. Springer-Verlag, 2010.

We introduce Symbolic Timed Automata, an amalgamation of symbolic transition systems and timed automata, which allows to express nondeterministic data- dependent control flow with inputs, outputs and real-time behaviour. In particular, input data can influence the timing behaviour. We define two semantics for STA, a concrete one as timed labelled transition systems and another one on a symbolic level. We show that the symbolic semantics is complete and correct w.r.t. the concrete one. Finally, we introduce symbolic conformance relation stioco, which is an extension of the well-known ioco conformance relation. Relation stioco is defined using FO-logic on a purely symbolic level. We show that stioco corresponds on the concrete semantic level to Krichen and Tripakis' implementation relation tioco for timed labelled transition systems.

4 Working Groups

4.1 Working Group Report: Towards a Competition in Model-based testing

Burkhart Wolff (Université Paris-Sud, FR)

The relative success of Tool- or Modeling competitions à la VSTTE or SMT- Competitions raises the question if the MBT community should develop a similar institution for its field. Competitions tend to reward otherwise neglected tool development, give indications on the technical state-of-the-art, and produce a strong feedback for choices in the foundational theories. We discuss certain scenarios and present a strategy that seems most appropriate to our field, which has the problem that *modeling* is a key issue and therefore a plethora of different modeling-languages make a simple comparison of solutions difficult. First, we identified a number of scenarios:

1. The "Archive Scenario" (Woodcock's FM Archive) ...
2. The "Grand Challenge Scenario" (The production cell scenario, MONDEX case study, etc.)
3. Open Format Competition à la VSTTE (open format, co-loc, Jury, fixed time)
4. Fixed Format Competition à la SMT and Casc (fixed format, co-location with a Conf.)

In principle, these scenarios sketch already a progression, and a strategy towards a competition. The latter should pave the path to a Modeling Competition a la VSTTE, where informal specifications of small problems were given, and hard time constraints for solutions were required by participating teams. Evaluation of solutions and tools by were performed by a Jury. The challenge for really setting this up consists essentially finding in a notable jury that sets up a call-for-paper, a prestigious prize and a reasonable conference to co-locate with ...

4.2 Working Group Report: Proof and Test

Cathérine Dubois (Université Évry, FR), Burkhart Wolff (Université Paris-Sud, FR)

Test and proofs seem to have complementary properties: while (formal) proofs are based on formal manipulation / computation, prove properties over models "once and for all", relate models by logically complete arguments, testing is traditionally seen as experimentation with the goal to finding bugs, which can relate models with concrete systems, but are inherently incomplete.

The complementary characteristic is sometimes expressed as: *Beware of bugs in the above code; I have only proved it correct, not tried it.* Given that modern deductive verification tools make a lot of assumptions over memory model, machine-model, and methodology, the risk that a formally proven program is still buggily executed on a concrete machine is very real (see the nice example on Maria Christakis Slides presented here). The *validation* of models and system assumptions is therefore a necessary complement to deductive verification.

Over the tombstones of this old debate, there is in research communities the growing consensus that there is a lot of common ground between these two and that they complement each other. Testing can help Proof by:
- MBT for model validation,
- counter example finding and checking,
- using testing techniques on executable models,
- testing for verifying assumptions for proof tools,
- ...

Proof can contribute to testing :
- use prover for test-case generation,
- proof to pruning test objectives,
- basic technology: constraint solving, SMT's,
- mature interactive formal development environments (such as Isabelle),
- ...

Quality in software development (and, by the way, high-level certifications such as Common Criteria) require a clever combination of static analysis, proving, testing, model checking, constraint solving, reviewing ...

References

1 B. Wolff. Using Theorem Provers for Testing. https://www.lri.fr/~wolff/talks/UsingTheoremProversforTesting.pdf Invited Talk at the Seminary of the Digiteo Foundation http://www.digiteo.fr/-seminaires-digiteo-, Paris, 20 march 2013.

4.3 Working Group Report: Testing and the Cloud

Wolfgang Grieskamp (Google, Seattle, US), Burkhart Wolff (Université Paris-Sud, FR)

Cloud computing is a major trend in computing, both as a technological development as well as a scientific endeavor. With respect to testing, this trend generates to directions or research:
1. *Testing the Cloud*: How to adapt testing techniques to the new hybrid, highly distributed, highly virtualized infra-structures that become part of our world-wide IT infra-structure?
2. *Testing with the Cloud*: How can the massive computing power inherent in these new infra-structures be effectively used for old and new testing techniques?

With respect to the former question, we agreed that old problems are coming back big: the problems of scalability of techniques, the problem of isolation of components under test from the environment, the security and privacy problems as well as the problems of test configuration and deployment. However, some new aspects also pop up: the monitoring makes runtime verification ("live" testing) easier and opens new ways on the virtualization level. The dynamics makes clusters of discrete systems increasingly continuous, which increases the need for symbolic methods. On the other hand, cloud computing makes the billing, concretely: the cost of testing, more explicit. In the cloud, cost a factor of availability, and production typically needs 99% availability. Here testing has an advantage over production: test departments can get away with perhaps 75% availability (which can be substantially cheaper).

Wrt. the latter question, we observe the phenomenon that more than two orders of magnitude or more of affordable computing power becomes available to testing — which can have an impact to methodology (less brain-power, more brute force ?). The other concern is, that the new infra-structures will not necessarily be *accessible* to everyone, it is a new era of company-owned computing centers, possibly a post-PC era. Further interest attracted the question, what type of testing techniques are, generally speaking, "cloudifiable". This is the case if they follow the map-reduce pattern, i.e.

- inexpensive pre-computation splitting the problem in a fixed number of independent sub-problems
- sub-problems executed on different machines, where communication via remote procedure calls is possible occurring rare in practice, and
- results can be collected easily and a potential start over to re-computation is possible.

This is the case for online model-based testing (symbolic or not), model-checking (some approaches already exist), concolic execution, and load testing.

4.4 Working Group Report: Machine Learning and Testing

Margus Veanes (Microsoft Research, Redmond, USA), Juhan Ernits (Tallinn University of Technology, Estonia)

Testing of complex systems, such as the behavior of a *quad copter* that is a real-time system controlled by hand gestures recognized through Kinect, can be very challenging. The 20 second youtube video clip[1] illustrates some aspects of what may go wrong and clearly raises questions as to how to best test such systems. There are several reasons for why testing of robotics and augmented reality applications is difficult in general and raises the difficulty bar compared to traditional approaches to testing:

- How do we define the test oracle for the system?
- How do we generate tests for the system?
- How do we measure coverage, i.e., when have we tested enough?

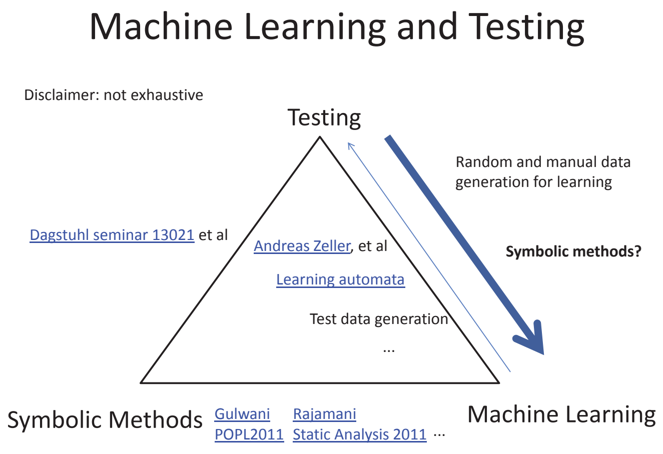
One observation we can make about the concrete scenario in the video is that, at some point, the intended gesture that was supposed to lower the copter was misinterpreted, and instead, the copter flew up. We know that the Kinect sensor uses *decision trees*[2] to detect body parts and to recognize gestures. Hypothetically, the flaw that caused the wrong behavior was in the logic of the decison tree. The decision tree has been constructed after *machine learning*

[1] https://www.youtube.com/watch?v=OQnRA6wZM-A
[2] http://dx.doi.org/10.1109/FPL.2012.6339226

of intended gestures for the given task. The question that is central to the theme of this seminar is:

- Could symbolic testing techniques be applied to decision tree classifiers in order to improve their accuracy?

This question is very interdisciplinary. The relation between the three fields of research: *testing*, *symbolic methods*, and *machine learning* is quite interesting and is illustrated in the following figure:


Machine Learning and Testing
Disclaimer: not exhaustive
Testing
Random and manual data generation for learning
Symbolic methods?
Dagstuhl seminar 13021 et al
Andreas Zeller, et al
Learning automata
Test data generation
...
Symbolic Methods
Gulwani POPL2011
Rajamani Static Analysis 2011 ...
Machine Learning

Abtractly, a machine learning algorithm produces a classifier, e.g., in form of a decision tree, that can be characterized as a function $F : X_1 \times \cdots \times X_n \to Y$ where elements of X_i are inputs to the classifier (e.g. 2D coordicates). The output value is the value depending on the inputs, typically Y is finite and is for example the shape of the detected skeleton. There are at least two different uses of classifiers in the context of testing.

- Classifier itself is the system under test (SUT).
- Classifiers are used as test data generators for the SUT (e.g. the SUT is the copter in the above scenario).

There are several open research questions related to the above uses.

- How could a white box approach be used, e.g., can a decision tree be treated as a white box?
- How could approximate distance based methods be used for data generation?
- Can one make minimal modification to fail classifiers?[3]

There are many more open questions. For eaxmple, could mutation based testing be used to improve classifiers? One can also apply other symbolic approaches such as: predicate extraction for explaining class-boundary conditons, and precondition and postcondition checking, or invariant checking such as: "In a similar domain every *even* input is classified as class-1, is it true for this classifier?".

[3] See for example CV Dazzle: http://cvdazzle.com/.

5 Programme

Programme for Monday 7th of January:

09:00–09:30: Margus Veanes: Welcome Talk of the Organizers
09:30–10:30: Nikolaj Bjørner: An update on Z3: Features and Applications
10:30–11:00:Coffee Break
11:00–12:00:Elevator Pitch Sessions: Position Statements, Perspectives, New Ideas, Value Propositions ...
12:00–14:00:Lunch
14:00–15:30 :Logic in Testing
– L.Viganó:Using Interpolation for Test-Case Generation for Security Protocols
– B. Wolff: An Introduction to Model-based Testing with Isabelle/HOL-TestGen
– J. v.d. Pol: Formal Methods and Tools: Testing U Twente
15:30–16:00:Coffee Break 16:00–17:30:Symbolic IOCO
– T. Jéron: Model based conformance testing with iocotioco and Symbolic techniques
– C. Gaston: Symbolic execution for Off-line Test Case Generation For Timed Model-Based Testing

Programme for Tuesday 8th of January:

09:00–10:30:Symbolic Testing involving Time
– A. Belinfante: Towards symbolic and timed testing with JTorX
– S. v. Styp: A Conformance Testing Relation for Symbolic Timed Automata
– W. Andrade: Symbolic Model-Based Testing of Real-Time Systems using SYMBOLRT
10:30–11:00:Coffee Break
11:00–12:00: N. Tillmann: Pex4Fun: Serious Gaming powered by Symbolic Execution
12:00–14:00: Lunch
14:00–15:30: Combined Process & Data Testing
– F. Zaidi: Online Verification of Value-Passing Choreographies through Property-Oriented Passive Testing
– A. Feliachi: Theorem-Prover Based Test Generation for Circus
– Discussion
15:30–16:00:Coffee Break
16:00–17:30: Counter-Example Generation
– J. Blanchette: Counterexamples for Isabelle: Ground and Beyond
– H. Waeselynck: Paths to property violation: a structural approach for analyzing counter-examples
– Discussion

Programme for Wednesday 9th of January:

9:00–10:30: Alternative Approaches
– M. Christakis: Collaborative Verification and Testing with Explicit Assumptions
– J. Ernits: Diagnosis Modulo Theories
– M. Kaeaeramees: A Symbolic Approach to Model-based Online Testing
Coffee
11:00–12:00: A. Brucker, L. Brügger: Tutorial: Model-based Testing of Security-critical Systems
12:00: Dagstuhl Foto

12:00–14:00 Lunch
14:00–14:30: D. Molnar: SAGE
15:00–21:00 Excursion to Saarburg Winery

Programme for Thursday 10th of January:

09:00–10:00: Mutation Testing
– S. Bardin: Symbolic methods for efficient mutation testing
– R. Just: Using State Infection Conditions to Detect Equivalent Mutants and Speed up Mutation Analysis
10:00–10:30 2nd elevator pitch sessions
Coffee
11:00–12:00: Group session : 4 Groups
– Should there be Tool Competition or an Archive on MBT
– Proof and Test
– Machine Learning and Testing
– Testing in and of the Cloud
14:30–15:30:
– D. Mery: Critical Systems Development Methodology
– B. Nielsen: Testing real-time systems under uncertainty
Coffee
16:00–17:30
– C. Cadar: KATCH: High-Coverage Testing of Software Patches
– M. Veanes: Symbolic automata

Programme for Friday 11th of January:

09:00–10:00
– C. Dubois: A Certified Constraint Solver over Finite Domains.
– M. Rueher: Combining AI & CP to identify suspicious values
Coffee Break
10:30–12:00: Summary of groups (1h) 10mn/groups
– B. Wolff: Should there be Tool Competition or an Archive on MBT
– C. Dubois: Proof and Test
– J. Ernits: Machine Learning and Testing
– W. Grieskamp: Testing in and of the Cloud
Discussion

Participants

- Sébastien Bardin
CEA – Gif sur Yvette, FR
- Axel Belinfante
University of Twente, NL
- Nikolaj Bjorner
Microsoft Res. – Redmond, US
- Jasmin Christian Blanchette
TU München, DE
- Achim D. Brucker
SAP Research – Karlsruhe, DE
- Lukas A. Brügger
ETH Zürich, CH
- Cristian Cadar
Imperial College London, GB
- Maria Christakis
ETH Zürich, CH
- Sylvain Conchon
Université Paris Sud, FR
- Wilkerson de Lucena Andrade
Federal University of Campina Grande, BR
- Catherine Dubois
ENSIIE – Evry, FR
- Juhan Ernits
Tallinn Univ. of Technology, EE
- Abderrahmane Feliachi
Université Paris Sud, FR
- Christophe Gaston
CEA – Gif sur Yvette, FR
- Arnaud Gotlieb
Simula Reseach Laboratory – Lysaker, NO
- Wolfgang Grieskamp
Google – Sammamish, US
- Robert M. Hierons
Brunel University, GB
- Thierry Jéron
INRIA Rennes – Bretagne Atlantique, FR
- René Just
University of Washington – Seattle, US
- Marko Kääramees
Tallinn Univ. of Technology, EE
- Pascale Le Gall
Ecole Centrale – Paris, FR
- Martin Leucker
Universität Lübeck, DE
- Delphine Longuet
Université Paris Sud, FR
- Dominique Méry
LORIA – Nancy, FR
- David Molnar
Microsoft Res. – Redmond, US
- Brian Nielsen
Aalborg University, DK
- Grgur Petric Maretic
ETH Zürich, CH
- Frank Rogin
Biotronik SE&Co.KG – Berlin, DE
- Michel Rueher
Université de Nice, FR
- Nikolai Tillmann
Microsoft Res. - Redmond, US
- Jan Tretmans
Embedded Systems Institute – Eindhoven, NL
- Jaco van de Pol
University of Twente, NL
- Margus Veanes
Microsoft Res. – Redmond, US
- Luca Vigano
Università di Verona, IT
- Sabrina von Styp
RWTH Aachen, DE
- Hélène Waeselynck
LAAS – Toulouse, FR
- Burkhart Wolff
Université Paris Sud, FR
- Fatiha Zaïdi
Université Paris Sud, FR