HOL-TestGen
An Interactive Test-case Generation Framework

Achim D. Brucker[1] and Burkhart Wolff[2]

[1] SAP Research, Vincenz-Priessnitz-Str. 1, 76131 Karlsruhe, Germany
achim.brucker@sap.com
[2] Université Paris-Sud, Parc Club Orsay Université, 91893 Orsay Cedex, France
wolff@lri.fr

Abstract We present HOL-TestGen, an extensible test environment for specification-based testing build upon the proof assistant Isabelle. HOL-TestGen leverages the semi-automated generation of test theorems (a form of partitioning the test input space), and their refinement to concrete test-data, as well as the automatic generation of a test driver for the execution and test result verification.
HOL-TestGen can also be understood as a unifying technical and conceptual framework for presenting and investigating the variety of unit test and sequence test techniques in a logically consistent way.
Keywords: symbolic test-case generations, black box testing, white box testing, theorem proving, interactive testing

1 Introduction

HOL-TestGen (`http://www.brucker.ch/projects/hol-testgen/`) is an *interactive*, i. e., semi-automated, test tool for specification based tests built upon Isabelle/HOL. HOL-TestGen allows one to write test specifications in higher-order logic (HOL), (semi-) automatically partition the input space, resulting in abstract test-cases, automatically select concrete test-data, automatically generate test scripts for testing arbitrary implementations.

2 The HOL-TestGen Architecture and Workflow

In this section, we briefly review the main concepts and outline the standard workflow (see Figure 1) of HOL-TestGen [1–3]. The latter is divided into five phases: first, the *test theory* containing the basic datatypes and key predicates of the problem-domain has to be written. Since the test theory can be written in classical higher-order logic (HOL), i. e., a functional programming language extended by logical quantifiers, our approach is extremely flexible. Second, the test-engineer has to write the *test specification*, i. e., the concrete property the system under test is tested for. Third comes the generation of *test-cases* along with a *test theorem*, forth the generation of *test-data* (TD), and fifth the *test*

M. Chechik and M. Wirsing (Eds.): FASE 2009, LNCS 5503, pp. 417–420, 2009.

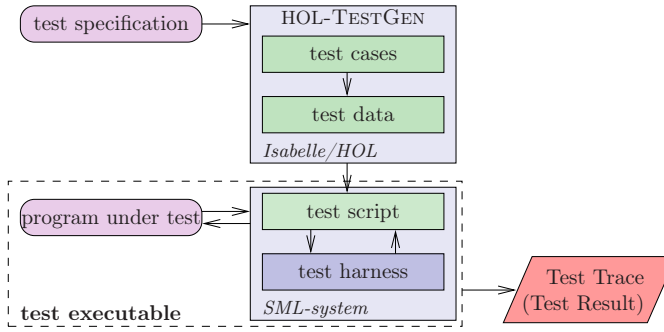

Figure 1. Overview of the Standard Workflow of HOL-TestGen

execution (*result verification*) phase involving runs of the "real code" of the program under test. Once a test theory is completed, an integrated documentation (i. e., a formal test plan) with all definitions and results can be generated.

The properties of the *program under test* are specified in HOL in the *test specification* (TS). A test-specification, typically, will have the form pre $x \rightarrow$ post x $(PUT\ x)$, where pre and post represent pre and post conditions of the program under test PUT, which is just a variable in the test-specification. Instead of just a partition of the input spaces, our system will decompose the test specification in the test-case generation phase into a semantically equivalent *test theorem* which has the form:

$$[\![\mathsf{TD}_1; \ldots; \mathsf{TD}_n; \mathsf{THYP}\ H_1; \ldots; \mathsf{THYP}\ H_m]\!] \Longrightarrow \mathsf{TS}$$

where THYP is a constant used to mark the *test hypotheses* that are underlying this test. At present, HOL-TestGen uses only uniformity and regularity Hypothesis; for example, a uniformity hypothesis means informally "if the program conforms to one instance of a case to TS, it conforms to all instances of this case to TS." Thus, a test theorem has the following meaning: *If the program under test passes the tests for all* TD_i *successfully, and if it satisfies all test hypothesis, it conforms to the test specification.* In this sense, a test theorem bridges the gap between test and verification. The TD_i are just formulae so far, containing variables and arbitrary predicates of the test theory as well as the free variable referring to the system under test. In the data-selection phase, which is implemented by a constraint-resolution based on Isabelle's proof procedures, ground instances for these variables were constructed. Both the test-case generation and the test-data-selection phase can be improved by adding lemmas (derived within Isabelle), or all sorts of logical massage can be realized by Isabelle on the test specification, the test theorem, the test-data, etc. This is how advanced users can improve the power of the deduction process dramatically.

The test theory containing test specifications, configurations of the test-data and test script generation, possibly extended by proofs for rules that support the overall process, is written in an extension of the Isar language [6]. It can be

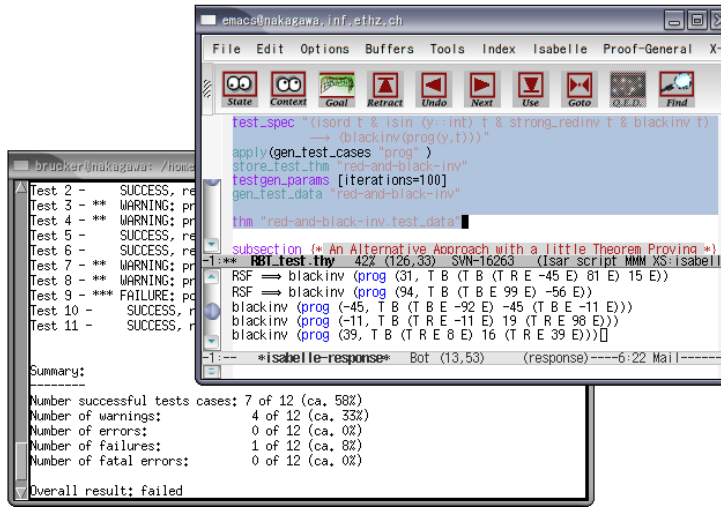

Figure 2. A HOL-TestGen Session Using Proof General

processed in batch mode, but also using the Proof General interface interactively, see Figure 2. This interface allows for interactively stepping through a test theory (in the upper sub-window) and the sub-window below shows the corresponding system state. A system state may be a proof state in a test theorem development, or the result of inspections of generated test-data or a list of test hypothesis.

After test-data generation, HOL-TestGen can produce a *test script* driving the test using the provided *test harness*. The test script together with the test harness stimulate the code for the program under test built into the *test executable*. Executing the *test executable* runs the test and results in a *test trace* showing possible errors in the implementation (see lower window in Figure 2).

3 Case Studies

HOL-TestGen was used successfully in several case studies, among them:

Unit testing of red-black trees: In this case study [2], we generated test-cases for recursive data-structures. In particular, we generated test-cases for red-black trees testing the red-black properties (i. e., both the insertion and deletion operation preserve these properties). We also generated test-data and test scripts for this scenario and used them for testing the red-black tree implementation of the SML/NJ library. Our work revealed a major bug in this implementation which has not been detected during the last 12 years.

Unit testing of packet filters: In this case study [4], we modeled stateless packet filters (firewalls) and their security policy in HOL. Based on this specification, we generated test-cases for testing that a real firewall implements a specific security policy. Furthermore, we exploited the framework

aspect of HOL-TestGen and developed a domain-specific test case generator: HOL-TestGen/fw. HOL-TestGen/fw provides both domain specific test-case and test-data generation heuristics and domain-specific extensions of the theorem prover, e. g., supporting the simplification of firewall policies.

Sequence testing of application level firewalls: In this case study [3], we applied HOL-TestGen to different sequence-testing scenarios. In particular, we modeled stateful communication protocols (e. g., ftp and voice-over-ip) and used these models as basis for the test-case generation. Overall, this provides a method for testing the compliance of an application level, stateful firewall to a give security policy.

In all these applications, we made the experience that combining theorem proving techniques and testing techniques can improve the overall quality of the generated test-cases and test-data.

4 Conclusion

We provide a test environment for specification-based (also called model-based) unit and sequence testing. Moreover, our test environment bridges the gap between formal verification and testing techniques, i. e., testing, in a logically consistent way. The system has been used in several substantial case studies [2–4] and for test-theoretical work [5].

References

[1] Achim D. Brucker and Burkhart Wolff. HOL-TestGen 1.0.0 user guide. Technical Report 482, ETH Zurich, April 2005.

[2] Achim D. Brucker and Burkhart Wolff. Symbolic test case generation for primitive recursive functions. In Jens Grabowski and Brian Nielsen, editors, *Formal Approaches to Testing of Software*, number 3395 in Lecture Notes in Computer Science, pages 16–32. Springer-Verlag, 2004. doi: 10.1007/b106767.

[3] Achim D. Brucker and Burkhart Wolff. Test-sequence generation with HOL-TestGen – with an application to firewall testing. In Bertrand Meyer and Yuri Gurevich, editors, *TAP 2007: Tests And Proofs*, number 4454 in Lecture Notes in Computer Science, pages 149–168. Springer-Verlag, 2007. doi: 10.1007/978-3-540-73770-4_9.

[4] 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 Lecture Notes in Computer Science, pages 103–118. Springer-Verlag, 2008. doi: 10.1007/978-3-540-68524-1_9.

[5] Achim D. Brucker, Lukas Brügger, and Burkhart Wolff. Verifying test-hypotheses: An experiment in test and proof. *Electronic Notes in Theoretical Computer Science*, 220(1):15–27, 2008. ISSN 1571-0661. doi: 10.1016/j.entcs.2008.11.003. Proceedings of the Fourth Workshop on Model Based Testing (MBT 2008).

[6] Markus M. Wenzel. *Isabelle/Isar – a versatile environment for human-readable formal proof documents*. PhD thesis, TU München, München, February 2002.