Theorem-prover based Testing with HOL-TestGen (Extended Abstract)

Achim D. Brucker and Burkhart Wolff

Information Security, ETH Zurich, 8092 Zurich, Switzerland
{brucker, bwolff}@inf.ethz.ch

1 Mission

Today, essentially two software validation techniques are used: *software verification* and *software testing*. As far as symbolic verification methods and model-based testing techniques are concerned, the interest among researchers in the mutual fertilization of these fields is growing.

From the verification perspective, testing offers:

- experiences on test-adequacy criteria [6], which can be viewed as *new abstraction techniques* reducing infinite models to finite and checkable ones,
- new approaches to generate *counter-examples*, and
- new application scenarios for verification, since black-box testing can be used as a systematic experimentation method for *reverse engineering specifications* for legacy systems.

From the testing perspective, symbolic verification offers:

- ways to cope with the *state space explosions* inherent to test case generation techniques, and
- ways to log the *testing hypothesis* underlying a test explicitly.

The HOL-TestGen system [3,2,1] is designed to explore and exploit these complementary assets. Built on top of a widely-used interactive theorem prover, it provides automatic procedures for test case generation and test-data selection as well as interactive means to perform logical massages of the intermediate results by derived rules. The core of HOL-TestGen is a test case generation procedure that decomposes a *test specification* (TS), i. e., test-property over a program under test, into a semantically equivalent *test theorem* of the form:

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

where the TC_i are the *test cases* and THYP is a constant (semantically defined as identity) used to mark the explicit *test hypotheses* H_j that are underlying this test. Thus, a test theorem has the following meaning:

> If the program under test passes the tests with a witness for all TC_i successfully, and if it satisfies all test hypothesis, it satisfies TS.

In this sense, the test theorem bridges the gap between test and verification. Testing can be viewed as systematic weakening of specifications.

2 Workflow

HOL-TestGen is an *interactive* (semi-automated) test tool for specification based tests. Its theory and implementation has been described elsewhere [3,1]; here, we briefly review main concepts and outline the standard workflow. The latter is divided into four phases: writing the *test specification* TS, generation of *test cases* TC along with a *test theorem* for TS, generation of *test data* TD, i. e., constraint-free instances of TC, and the *test execution* (*result verification*) phase involving runs of the "real code" of the program under test. (See Figure 1 for the overall workflow.) Once a test theory is completed, documents can be generated that represent a formal test plan. The test plan containing test theory, test spec-

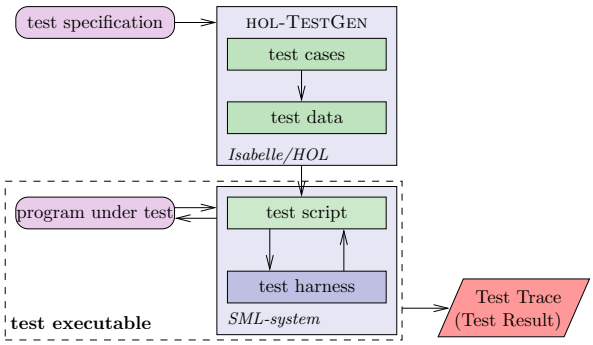

Figure 1. Overview of the Standard Workflow of HOL-TestGen

ifications, configurations of the test data and test script generation commands, possibly extended by proofs for rules that support the overall process, is written in an extension of the Isar language [5]. It can be 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 while the sub-window below shows the corresponding system state. This may be a proof state in a test theorem development, a list of generated test data or a list of test hypothesis.

The transition between the test-theorem generation and the test-data generation offers specific opportunities for HOL-TestGen. In practice, a fully automatic generated test-theorem contains too many constraints which are difficult to resolve in the test-data generation phase. Thus, logical massage by proving "little" lemmas used in automatic simplification leads to much better versions of test-theorems that lead to dramatically improved coverage.

After test data generation, HOL-TestGen produces 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 yields a *test trace* showing errors in the implementation (see lower window in Figure 2).

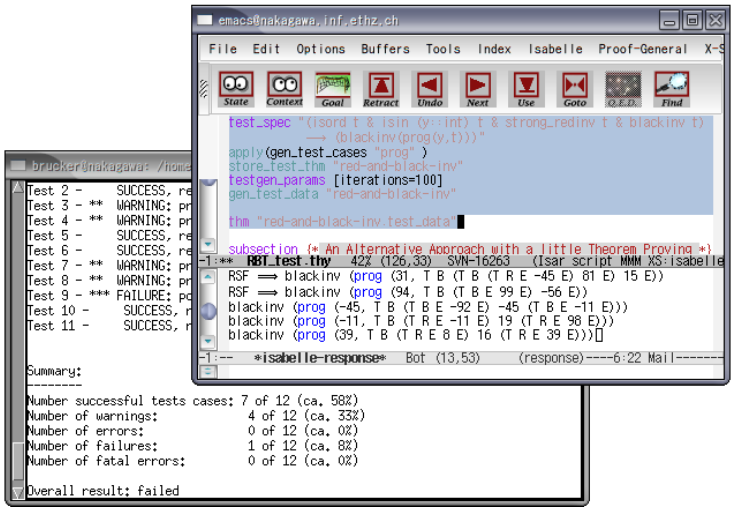

Figure 2. A HOL-TestGen Session Using Proof General

3 Case Studies

HOL-TestGen has been originally designed for unit-tests; for example, [3] discusses tests of insert and delete operations for library implementations of *red-black trees*. However, it can be shown that the procedure can also be used for sequence testing of locally non-deterministic reactive systems as well: instead of using an automaton, we build a test-specification based on its input traces. In [4], we apply this technique to a substantial case study in the field of computer security, namely the black-box test of a *firewall configuration*.

References

1. A. D. Brucker and B. Wolff. HOL-TestGen 1.0.0 user guide. Technical Report 482, ETH Zürich, 2005. HOL-TestGen is available at: `http://www.brucker.ch/projects/hol-testgen/`.
2. A. D. Brucker and B. Wolff. Interactive testing using HOL-TestGen. In W. Grieskamp and C. Weise, editors, *Formal Approaches to Testing of Software (FATES 05)*, volume 3997 of LNCS, pages 87–102. Springer-Verlag, 2005.
3. A. D. Brucker and B. Wolff. Symbolic test case generation for primitive recursive functions. In J. Grabowski and B. Nielsen, editors, *Formal Approaches to Testing of Software (FATES 04)*, volume 3395 of LNCS, pages 16–32, 2005.
4. A. D. Brucker and B. Wolff. Test-sequence generation with HOL-TestGen – with an application to firewall testing. In B. Meyer and Y. Gurevich, editors, TAP *2007: Tests And Proofs*, volume 4454 of LNCS. Springer-Verlag, 2007.
5. M. M. Wenzel. *Isabelle/Isar—a versatile environment for human-readable formal proof documents*. PhD thesis, TU München, München, 2002.
6. H. Zhu, P. A. Hall, and J. H. R. May. Software unit test coverage and adequacy. ACM *Computing Surveys*, 29(4):366–427, 1997.