Model-based Testing: Techniques and Industrial Applications

Lukas Brügger               Burkhart Wolff

Information Security Group, ETH Zürich
LRI, Université Paris-Sud
Abstract

Model-based testing has seen a wider range of industrial applications recently – enabling to systematically generate test-cases instead of speculate them or analyse post-hoc system traces or memory dumps. Test-case generation techniques vitally depend on symbolic computation and constraint-solving techniques. Their limits therefore represent limits for model-based testing as a whole. The HOL-TestGen system is designed as plug-in into the state-of-the-art theorem proving environment Isabelle/HOL. Thus, powerful modeling languages as well as powerful automated and interactive proof methods for constraint resolution are available. The talk is going to be a guided tour through theory, pragmatics, and recent industrial applications.
Intro: Definition and Summary

WHAT IS MODEL-BASED TESTING?

Model-Based Testing is the automatic generation of efficient test procedures/vectors using models of system requirements and specified functionality.
Models of Systems for Tests

System as awkward it might be ...
Models of Systems for Tests

Test-Oracle correct function or behaviour

System as awkward it might be ...

Observe

a posteriori run-time testing
Models of Systems for Tests

Model describing function or behaviour

System as awkward it might be ...

a priori run-time testing

a posteriori run-time testing
Modeling ... 

- ... aims at “blueprints” that can be analysed BEFORE the system is actually built.

- ... does not guarantee the absence of any error (only the conformance between a model and the “system”).

- ... can (and must) be integrated into the software development cycle ...
Modeling ...

- ... can be done post-hoc; significant projects “reverse engineer” the model of a legacy system.

- ... can help system integration processes by assuring that third-party components are in fact usable in a larger system.

The model gets the role of a “contract” in this scenario.
Vision

• Model-development should be integrated into the classical software development process; thus into:

  • Requirements documents; Design documents ...

  • Test-Cases should be used early for Animation and “Reverse Engineering” ...

  • ... in some cases, a combination with verification techniques might be useful ...
HOL-TestGen: A Solution

- HOL-TestGen is a Model-based TestCase Generation System

- Unlike e.g. Spec-Explorer (by Microsoft, available as VisualStudio Plugin), it emphasizes *(well, we are academic ;-) )*:
  - logical cleanliness and an expressiveness. Modeling Language HOL instead of, say, an OO-language with quantifiers
  - symbolic computations having their roots in Theorem Proving instead of plain enumeration and model-checking
Agenda

- TestGen and its Method by Example
- Overview on Symbolic Test Case Generation
- Own Case Studies
- Industrial Applications
- Conclusion
HOL-TestGen by Example

• Step I in the TestGen - method:

  • write Test Document containing HOL Definitions

    text{* We include the TestGen system and start with a little example *}

    Triangle = Testing +

    text{* The result type is defined by: *}

    datatype triangle = equilateral | scalene | isosceles | error

    constdefs triangle :: "[nat,nat,nat] => bool"
    "triangle x y z == (0<x ∧ 0<y ∧ 0<z ∧
                        (z<x+y) ∧ (x<y+z) ∧ (y<x+z))"
HOL-TestGen by Example

• Step II in the TestGen - method:
  • containing a Test Specification TS in HOL ... (ctd'd):

    . . .
    testspec TS:
    “prog(x, y, z) =
    if triangle x y z
    then if x = y
      then if y = z then equilateral
      else isosceles
    else if y = z then isosceles
    else if x = z then isosceles
    else scalene
    else error”
    . . .

• where prog is the program under test
HOL-TestGen by Example

• Step III in the TestGen - method:

• fire generate cases tactic and get proof-state:

        apply(gen_test_cases 3 1 simp: add_commute)
HOL-TestGen by Example

- Step III in the TestGen - method:

- fire generate cases tactic and get **proof-state**:

```
. . .
⟦0 < z; z < z + z⟧ ⇒
progra(z, z, z) = equilateral

⟦x ≤ z; 0 < x; 0 < z; z < x + z; x < z + z⟧ ⇒
progra(x, z, z) = isosceles

⟦y ≤ z; z ≤ y;¬z < z + y⟧ ⇒
progra(z, y, z) = error
```
A Step Back: Test-Theorem

• Step III in the TestGen - method:

  • consisting of 26 test cases $C_1$ to $C_{26}$
    (having the form of Horn clauses, where the
    premises are called constraints)

  • where the proof state corresponds to an equivalent
    test theorem of the form:

    $$C_1 \implies \ldots (C_{26} \implies \text{TS}) \quad (\text{written: } \llbracket C_1 ; \ldots ; C_{26} \rrbracket \implies \text{TS})$$
HOL-TestGen by Example

• Step V in the TestGen – method:

• fire generate cases tactic and get proof-state and produce test statements (i.e. premises of the form):

  \[
  \ldots
  \text{gen_test_data} \text{ “Triangle”}
  \]
HOL-TestGen by Example

- Step V in the TestGen – method:

  - fire generate cases tactic and get proof-state and produce test statements (i.e. premises of the form):

    \[
    \begin{align*}
    \text{prog}(3, 3, 3) & = \text{equilateral} \\
    \text{prog}(4, 6, 0) & = \text{error}
    \end{align*}
    \]
HOL-TestGen by Example

- Step VI in the TestGen – method:
  - Convert test-data automatically into a test driver.

  ```
  gen_test_script "Triangle"
  ```

In our case, this is an SML program that fires the test-harness, which can be linked to any .o file containing the program under test... (so, the SUT must not be SML, rather C, Java, ...)
Symbolic Computations Involved

- Basis for **TestGen package** (comprising Test Case and Test Data Generation tactics)
  - Isabelle/HOL library: 10000 derived rules . . .
  - about 500 are organized in larger data-structures used by Isabelle's proof procedures . . .

- How are tactics organized?
  - Rewriting Normal Form Computation (RNF)
  - Tableaux Normal Form Computation (HCNF)
  - Testing Normal Form Computation (TNF)
  - Testing Normal Form Minimization (MTNF)
  - Generating and Using Test Hypothesis
Own Case Study: Red Black Trees

Red-Black-Trees: Test Specification

testspec :  
(redinv t ∧ blackinv t) 
→ 
(redinv (delete x t) ∧ 
blackinv (delete x t))

where delete is the program under test.
Own Case Study: Red Black Trees

- Statistics:

  348 test cases were generated, within 2 min.

- one Error in the SML library was found, that makes crucial violation against redblack-invariants; makes lookup linear

- ... error not found within 12 years ... 

- ... reproduced meanwhile by random test tool
Own Case Study: Firewalls

- Statistics:
  
  10000 test cases were generated, within 8 h.

- ... realistic scenarios of analysis require quite advanced techniques for case-splitting and deduction

- ... uses real theorem proving
Industrial Applications

- Windows 98-Server Protocol: the story so far

- 2000: EU and US administration ruled Microsoft is a Monopoly in the Server Market (applying older Antitrust rules in the Telecommunication market)

- 2002: EU required the “ specification” of the server protocols in order to allow third-party vendors access to the market

- Polished internal documents of Microsoft were considered “insufficient” by the EU referees ...
Industrial Applications

- 2003: Microsoft legally contested this ruling, considering protocols as protected being IPR

- 2005: Microsoft lost the legal battle, was fined by 700 mio €, and forced to produce a document which:
  - also provides a formal specification
  - provides evidence that the model is actually compliant to the implemented system.

Since then, a team of 200 people started to reverse engineer the Protocol (developed in 1995), essentially using a tool-family on the basis of Spec-Explorer
Industrial Applications
Industrial Applications

· Windows Server 98 Protocol:

Wolfgang Grieskamp[2008]:

Using Model-Based Testing for Quality Assurance of Protocol Documentation

http://research.microsoft.com/users/wrwg/MBTETAPS.pdf
Conclusion

- Nowadays, model-based Testing is viable Technology

  - ... for systematic Testing
    (unit, sequence, reactive sequence, protocol testing)

  - ... for reverse-engineering Systems
    and integrating components of third-parties

  - ... to comply with future, legally required documentation standards
Conclusion

• HOL-TestGen

• Specs were written in HOL

• proof-state explosion controllable by abstraction

• although logically puristic, systematic test of a “real” library code or network components security policies has been shown feasible ...

• besides: HOL-TestGen is a verified tool inside a (well-known) theorem prover