

# *Model-based Testing of Networks and Network Components*

*(the “HOL-TestGen-XT” project)*

Burkhart Wolff

Université Paris-Sud, LRI, CNRS

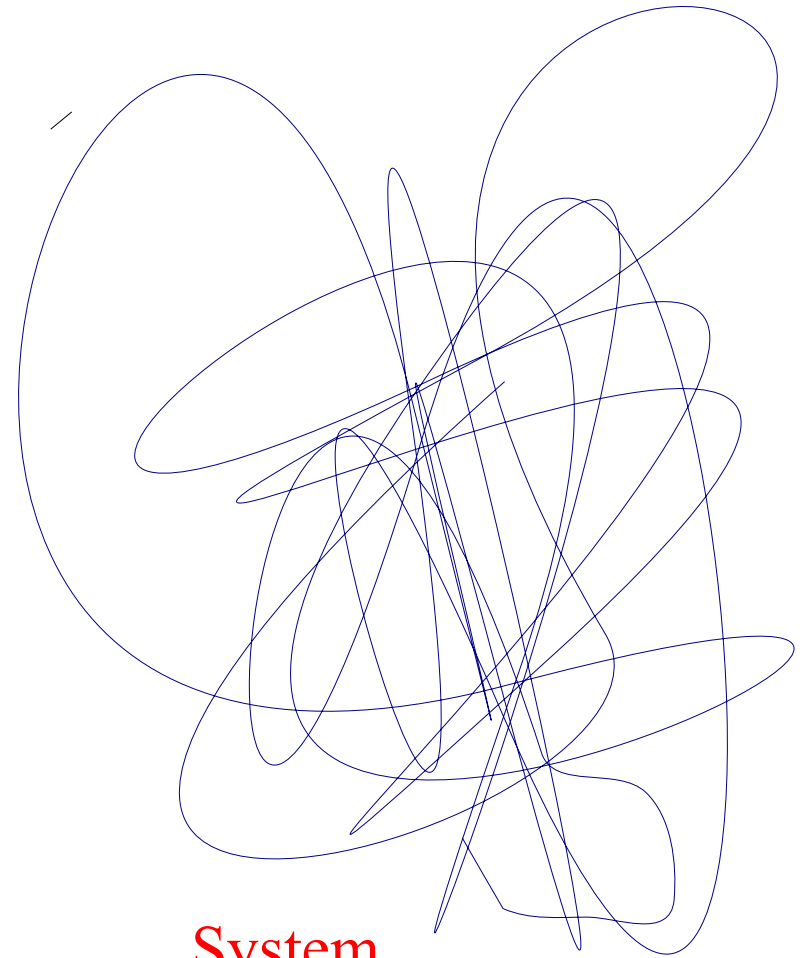
# Intro: Definition and Summary

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.**

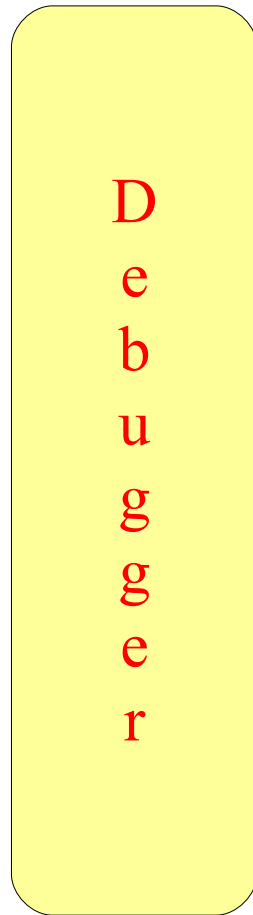
# Models of Systems for Tests

.

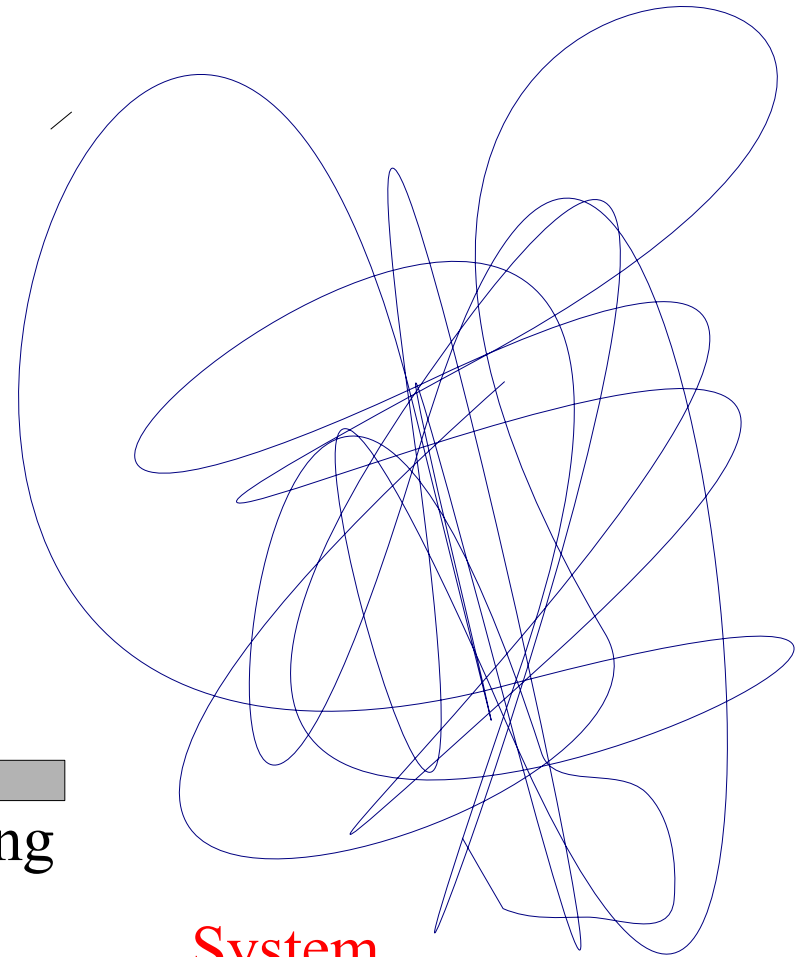
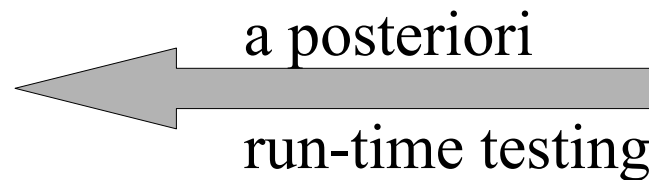


**System**  
as awkward  
it might be ...

# Models of Systems for Tests

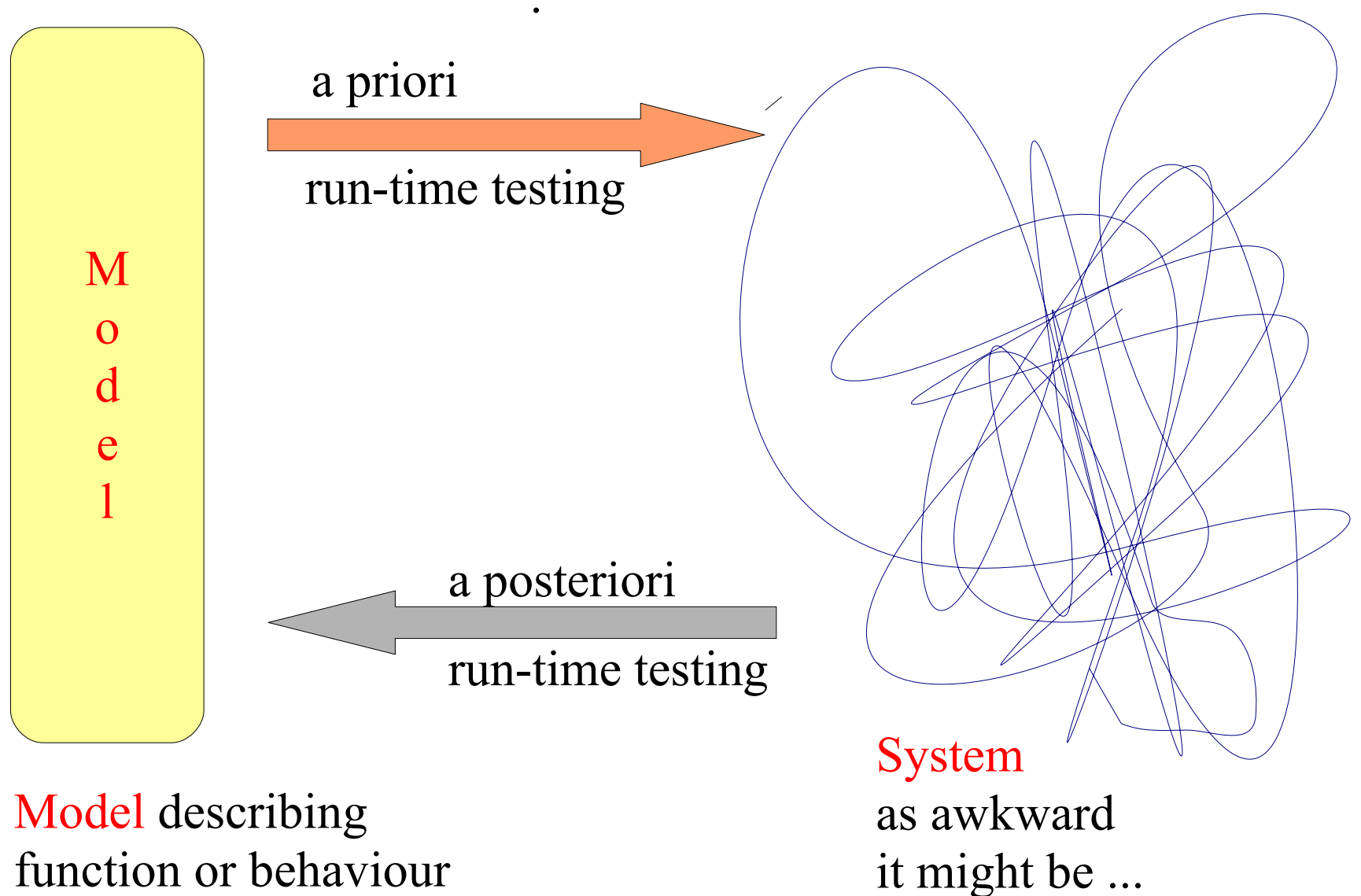


**Test-Oracle** correct  
function or behaviour



as awkward  
it might be ...

# Models of Systems for Tests



# Model-based Testing ...

- ... can be done post-hoc; significant industrial projects “reverse engineer” legacy system models
- ... attempts to find bugs in specifications **EARLY** (and can complement verification projects ...)
- ... 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.

# HOL-TestGen is ...

- ... based on HOL (Higher-order Logic):
  - “Functional Programming Language with Quantifiers”
  - plus definitional libraries on Sets, Lists, . . .
  - can be used meta-language for HoareCalculi, Z, CSP. . .
- ... implemented on top of Isabelle
  - an interactive prover implementing HOL
  - the test-engineer must decide over, abstraction level, split rules, breadth and depth of data structure exploration . . .
  - providing automated and interactive constraint-resolution techniques
  - interface: ProofGeneral
- ... by thy way, a verified test-tool

# HOL-TestGen Workflow

- Modelisation
  - writing background theory of problem domain

# HOL-TestGen Workflow

- Modelisation
  - writing background theory of problem domain
- Test-Case-Generation from Test-Specification
  - automated procedure `gen_test_case ...`
  - Test-Cases: partitions of I/O relation of the form

$$C_1(x) \implies \dots C_n(x) \implies \text{post } x \text{ (PUT } x)$$

# HOL-TestGen Workflow

- Modelisation
  - writing background theory of problem domain
- Test-Case-Generation from Test-Specification
  - automated procedure `gen_test_case ...`
  - Test-Cases: partitions of I/O relation of the form
$$C_1(x) \implies \dots C_n(x) \implies \text{post } x \text{ (PUT } x)$$
- Test-Data-Selection
  - constraint Solver `gen_test_data`
  - finds  $x$  satisfying  $C_i(x)$

# HOL-TestGen Workflow

- Modelisation
  - writing background theory of problem domain
- Test-Case-Generation from Test-Specification
  - automated procedure `gen_test_case ...`
  - Test-Cases: partitions of I/O relation of the form
$$C_1(x) \implies \dots C_n(x) \implies \text{post } x \text{ (PUT } x)$$
- Test-Data-Selection
  - constraint Solver `gen_test_data`
  - finds  $x$  satisfying  $C_i(x)$
- Test-Driver Generation
  - automatically compiled, drives external program

# HOL-TestGen Workflow

- Modelisation
  - writing **background theory** of problem domain
- Test-Case-Generation from Test-Specification
  - automated procedure `gen_test_case ...`
  - Test-Cases: **partitions of I/O relation** of the form
$$C_1(x) \implies \dots C_n(x) \implies \text{post } x \text{ (PUT } x)$$
- Test-Data-Selection
  - **constraint solver** `gen_test_data`
  - finds  $x$  satisfying  $C_i(x)$
- Test-Driver Generation
  - automatically compiled, drives external program
- Test Execution, Test-Documentation

# HOL-TestGen

a

*Demo*

# Mini-Example

- Modelisation
  - `is_sorted`, `insert`, `sort`
- Test-Case-Generation from Test-Specification
  - Test-Specification: `sort x = PUT x`
  - Test-Cases:  $\dots x \leq y \implies [x,y] = \text{PUT } [x,y]$   
 $\dots x > y \implies [y,x] = \text{PUT } [x,y] \dots$
- Test-Data-Selection
  - Test Data:  $[3,9] = \text{PUT } [3,9]$   
 $[1,6] = \text{PUT } [6,1]$
- Test-Driver Generation
  - SML driver
- Test Execution, Test-Documentation

# Case-Study: Firewalls

- Modelisation
  - TCP-IP, nets and subnets, (stateful) firewalls, policies
- Test-Case-Generation from Test-Specification
  - Test-Sepcification: **policy pkt = FUT pkt**
  - Test-Cases: ... subnet x y  $\implies$  accept(http,x,d,y) =  
PUT(http,x,d,y)
- Test-Data-Selection
  - Test Data: accept(http,(132,17,24,12),  
"blob",(132,17,0,0))
- Test-Driver Generation
  - test-data fed into external driver [Diana Krueger 05)
- Test Execution, Test-Documentation
  - partially contained in our distribution

# 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
  - ... complements full-blown verification techniques

# Conclusion

- HOL-TestGen
  - Specs were written in Isabelle/HOL, Documents and Programs alike
  - 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

# A Step Back: Test-Theorem

- corresponding to a **Test Theorem**:
  - consisting of 26 **test cases**  $C_1$  to  $C_{26}$   
(having the form of Horn clauses, where the premises are called **constraints**)
  - consisting of 13 Explicit Test-Hypothesis THYP (H)
  - establishing a formal link between Test and Proof

$$C_1 \implies \dots C_{26} \implies \text{THYP } H_1 \implies \text{THYP } H_{13} \implies \text{TS}$$

# 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

The screenshot displays the Microsoft Visual Studio environment with the SlicedStateMachine exploration tool. The interface is divided into several sections:

- Machine:** A list of state machine types (StateMachine, SetupScenario, SlicedStateMachine, TestSuite) and a 'new machine' option. Buttons for 'Set as main', 'Remove', 'Validate', 'Explore (Incrementally)', 'Test', and 'Generate tests' are present.
- Config:** A configuration pane with tabs for 'Switches', 'Types', 'Actions', 'Constraints', and 'Exceptions'. It contains sections for 'Exploration', 'Solver', 'Testing', and 'Viewing', each with various settings and their inheritance status.
- State Machine Diagram:** A state transition diagram showing states S0 through S37. Transitions are labeled with events and actions, such as 'EnsureShareExists(1, DISK)', 'SetupConnectionAndSession()', 'TreeConnectRequest(0, 1, 1)', 'event TreeConnectResponse(0, 1, 0, DISK)', 'CreateRequest(1, 1, 0, Create, "test1")', 'event CreateResponse(0, 1, 0)', 'event ErrorResponse(CREATE, 0, 1)', 'CloseRequest(2, 1, 0, 0)', 'event CloseResponse(0, 1)', 'ReadRequest(2, 1, 0, 0)', 'event ReadResponse(0, 1, Seq())', 'WriteRequest(2, 1, 0, 0, Seq{1, 3, 2})', 'event WriteResponse(0, 1)', 'CloseRequest(3, 1, 0, 0)', 'event CloseResponse(0, 1)', 'WriteRequest(3, 1, 0, 0, Seq{1, 3, 2})', 'event WriteResponse(0, 1)', 'ReadRequest(3, 1, 0, 0)', 'event ReadResponse(0, 1, Seq{1, 3, 2})', 'CloseRequest(3, 1, 0, 0)', 'event CloseResponse(0, 1)', 'CloseRequest(4, 1, 0, 0)', 'event CloseResponse(0, 1)'. The diagram is a state transition graph with states represented by circles and transitions by arrows.
- Status Bar:** Shows 'Finished' with '36 states, 37 steps, 0 errors' and a timestamp '00:00:01.8220000'.

# TestGen vs. Spec-Explorer

- HOL-TestGen offers a similar approach to SE process integration (albeit on a smaller scale ...)
- 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

# Industrial Applications

- Windows Server 98 Protocol :

Wolfgang Grieskamp[2008]:

Using Model-Based Testing for Quality Assurance of Protocol Documentation

Invited Talk MBT 2008, Budapest.

<http://research.microsoft.com/users/wrwg/MBTETAPS.pdf>

# TestGen: Symbolic Computations

