

Theorem-prover based Testing with HOL-TestGen

Burkhart Wolff¹

¹Université Paris-Sud, LRI, Orsay, France
wolff@lri.fr

M2R: Test des Systemes Informatiques
Orsay, 14 Jan 2010

Outline

- 1 Advanced Test Scenarios
- 2 Case Studies
- 3 Conclusion

Outline

- 1 Advanced Test Scenarios
- 2 Case Studies
- 3 Conclusion

Tuning the Workflow by Interactive Proof

Observations:

- Test-theorem generations is fairly **easy** ...
- Test-data generation is fairly **hard** ...
(it does not really matter if you use random solving or just plain enumeration !!!)
- Both are **scalable** processes ...
(via parameters like depth, iterations, ...)
- There are **bad** and **less bad** forms of test-theorems !!!
- **Recall:** Test-theorem and test-data generation are normal form computations:
⇒ More Rules, better results ...

What makes a Test-case “Bad”

- redundancy.
- many unsatisfiable constraints.
- many constraints with unclear logical status.
- constraints that are **difficult** to solve.
(like arithmetics).

Case Studies: Red-black Trees

Motivation

Test a non-trivial and widely-used data structure.

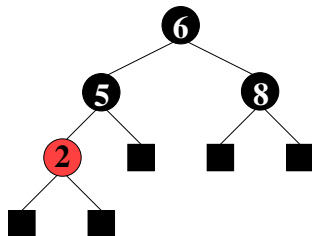
- part of the SML standard library
- widely used internally in the sml/NJ compiler, e. g., for providing efficient implementation for Sets, Bags, . . . ;
- very hard to generate (balanced) instances randomly

Modeling Red-black Trees I

Red-Black Trees:

Red Invariant: each red node has a black parent.

Black Invariant: each path from the root to an empty node (leaf) has the same number of black nodes.



datatype

color = R | B

tree = E | T color (α tree) ($\beta::\text{ord item}$) (α tree)

Modeling Red-black Trees II

- Red-Black Trees: Test Theory

consts

redinv :: tree \Rightarrow bool

blackinv :: tree \Rightarrow bool

recdef blackinv measure (λ t. (size t))

blackinv E = True

blackinv (T color a y b) =

((blackinv a) \wedge (blackinv b)

\wedge ((max B (height a)) = (max B (height b))))

recdev redinv measure ...

Red-black Trees: Test Specification

- Red-Black Trees: Test Specification

test_spec:

```
"isord t ∧ redinv t ∧ blackinv t
  ∧ isin (y::int) t
  →
  (blackinv(prog(y,t)))"
```

where prog is the program under test (e. g., delete).

- Using the standard-workflows results, among others:

```
RSF → blackinv (prog (100, T B E 7 E))
blackinv (prog (-91, T B (T R E -91 E) 5 E))
```

Red-black Trees: A first Summary

Observation:

Guessing (i. e., random-solving) valid red-black trees is difficult.

- On the one hand:
 - random-solving is nearly impossible for solutions which are “difficult” to find
 - only a small fraction of trees with depth k are balanced
- On the other hand:
 - we can quite easily construct valid red-black trees interactively.

Red-black Trees: A first Summary

Observation:

Guessing (i. e., random-solving) valid red-black trees is difficult.

- On the one hand:
 - random-solving is nearly impossible for solutions which are “difficult” to find
 - only a small fraction of trees with depth k are balanced
- On the other hand:
 - we can quite easily construct valid red-black trees interactively.
- **Question:**
Can we improve the test-data generation by using our knowledge about red-black trees?

Red-black Trees: Hierarchical Testing I

Idea:

Characterize valid instances of red-black tree in more detail and use this knowledge to guide the test data generation.

- First attempt:
enumerate the height of some trees without black nodes

lemma maxB_0_1:

"max_B_height (E:: int tree) = 0"

lemma maxB_0_5:

"max_B_height (T R (T R E 2 E) (5::int) (T R E 7 E)) = 0"

- But this is tedious ...

Red-black Trees: Hierarchical Testing I

Idea:

Characterize valid instances of red-black tree in more detail and use this knowledge to guide the test data generation.

- First attempt:
enumerate the height of some trees without black nodes

lemma maxB_0_1:

"max_B_height (E:: int tree) = 0"

lemma maxB_0_5:

"max_B_height (T R (T R E 2 E) (5::int) (T R E 7 E)) = 0"

- But this is tedious ... and error-prone

How to Improve Test-Theorems

- New simplification rule establishing **unsatisfiability**.
- New rules establishing equational constraints for variables.

$$(\max_B_height (T x t1 \text{ val } t2) = 0) \implies (x = R)$$

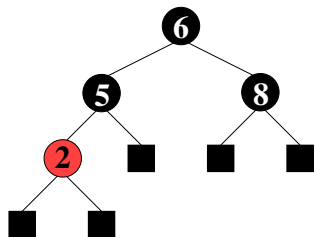
$$(\max_B_height x = 0) = \\ (x = E \vee \exists a y b. x = T R a y b \wedge \\ \max(\max_B_height a) \\ (\max_B_height b) = 0)$$

- Many rules are domain specific —
few hope that automation pays really off.

Improvement Slots

- logical massage of test-theorem.
- in-situ improvements:
add new rules into the context before `gen_test_cases`.
- post-hoc logical massage of test-theorem.
- in-situ improvements:
add new rules into the context before `gen_test_data`.

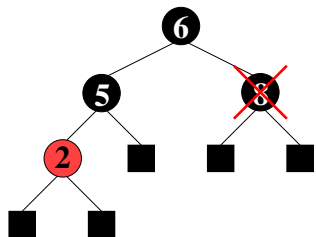
Red-black Trees: sml/NJ Implementation



(a) pre-state

Figure: Test Data for Deleting a Node in a Red-Black Tree

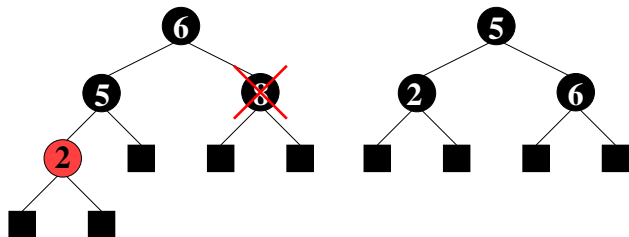
Red-black Trees: sml/NJ Implementation



(b) pre-state: delete "8"

Figure: Test Data for Deleting a Node in a Red-Black Tree

Red-black Trees: sml/NJ Implementation

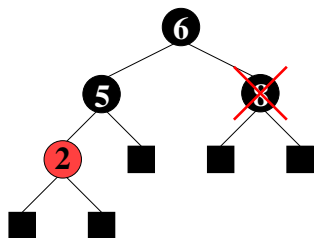


(b) pre-state: delete "8"

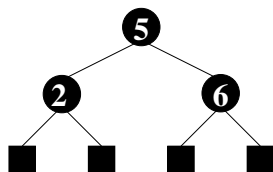
(c) correct result

Figure: Test Data for Deleting a Node in a Red-Black Tree

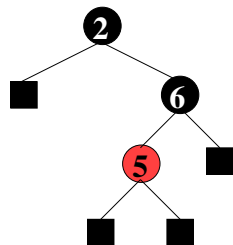
Red-black Trees: sml/NJ Implementation



(b) pre-state: delete "8"



(c) correct result



(d) result of sml/NJ

Figure: Test Data for Deleting a Node in a Red-Black Tree

Red-black Trees: Summary

- Statistics: 348 test cases were generated (within 2 minutes)
- One error found: crucial violation against red/black-invariants
- Red-black-trees degenerate to linked list (insert/search, etc. only in linear time)
- Not found within 12 years
- Reproduced meanwhile by random test tool

Outline

1 Advanced Test Scenarios

2 Case Studies

3 Conclusion

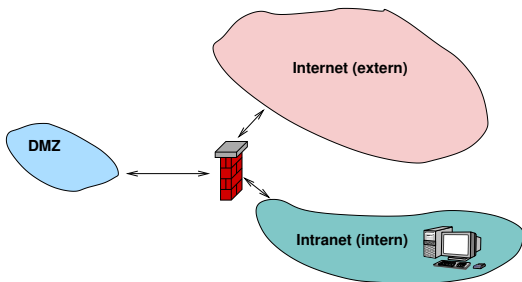
Specification-based Firewall Testing

Objective: test if a firewall configuration implements a given firewall policy

Procedure: as usual:

- 1 model firewalls (e.g., networks and protocols) and their policies in HOL
- 2 use HOL-TestGen for test-case generation

A Typical Firewall Policy



→	Intranet	DMZ	Internet
Intranet	-	smtp, imap	all protocols except smtp
<DMZ	∅	-	smtp
Internet	∅	http,smtp	-

A Bluffers Guide to Firewalls

- A Firewall is a
 - state-less or
 - state-fullpacket filter.
- The filtering (i.e., either accept or deny a packet) is based on the
 - source
 - destination
 - protocol
 - possibly: internal protocol state

The State-less Firewall Model I

First, we model a packet:

types (α, β) packet = "id \times protocol \times α src \times α dest \times β content"

where

id: a unique packet identifier, e. g., of type Integer

protocol: the protocol, modeled using an enumeration type (e.g., ftp, http, smtp)

α src (α dest): source (destination) address, e.g., using IPv4:

types

ipv4_ip = "(int \times int \times int \times int)"

ipv4 = "(ipv4_ip \times int)"

β content: content of a packet

The State-less Firewall Model II

- A **firewall** (packet filter) either accepts or denies a packet:

datatype

α out = accept α | deny

- A **policy** is a map from packet to packet out:

types

(α, β) Policy = " (α, β) packet \rightarrow ((α, β) packet) out"

where $\alpha \rightarrow \beta$ is a *partial function* defined as type synonym $\alpha \rightarrow \beta$ option

The State-less Firewall Model II

Now, we define a number of combinators on Policies like:

- $\text{empty} \equiv \lambda x. \text{None}$
- $\text{update}: m(x \setminus \langle \text{maplet} \rangle a) \equiv \lambda y. \text{if } x=y \text{ then } a \text{ else } m \ y$
- override :

$$m \ ++ \ n \equiv \lambda x. \text{case } m \ x \ \mathbf{of}$$

$$\quad \text{None} \Rightarrow n \ x$$

$$\quad | \ \text{Some } x \Rightarrow \text{Some } x$$

- and another 50 firewall-specific policies like ...

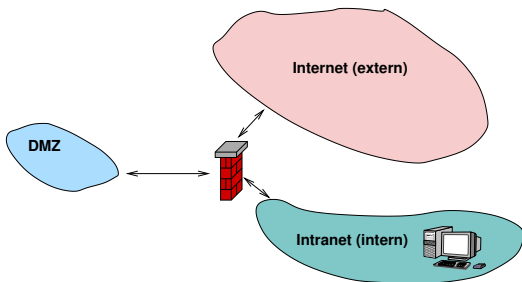
The State-less Firewall Model II

Policy-Specific Combinators:

constdefs

```
allow_prot_from_to :: "protocol  $\Rightarrow \alpha :: \text{net set set} \Rightarrow \alpha :: \text{net set set} \Rightarrow (\alpha, \beta)$  Rule
"allow_prot_from_to prot src_net dest_net  $\equiv$  allow_all |'
  {pa. src pa  $\sqsubseteq$  src_net  $\wedge$  dest pa  $\sqsubseteq$  dest_net  $\wedge$  protocol pa = prot}"
```

Testing State-less Firewalls: An Example I



→	Intranet	DMZ	Internet
Intranet	-	smtp, imap	all protocols except smtp
DMZ	∅	-	smtp
Internet	∅	http, smtp	-

Testing State-less Firewalls: An Example II

src	dest	protocol	action
Internet	DMZ	http	<i>accept</i>
Internet	DMZ	smtp	<i>accept</i>
⋮	⋮	⋮	⋮
*	*	*	<i>deny</i>

constdefs Internet_DMZ :: "(ipv4, content) Rule"

"Internet_DMZ ≡

(allow_prot_from_to smtp internet dmz) ++

(allow_prot_from_to http internet dmz)"

The policy can be modelled as follows:

constdefs test_policy :: "(ipv4, content) Policy"

"test_policy ≡ deny_all ++ Internet_DMZ ++ ..."

Testing State-less Firewalls: An Example III

- Using the test specification

test_spec "FUT x = test_policy x"

- results in test cases like:

- FUT

$(6, \text{smtp}, ((192, 169, 2, 8), 25), ((6, 2, 0, 4), 2), \text{data}) =$
Some (accept

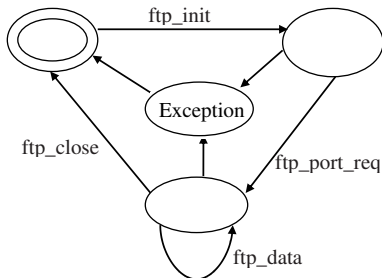
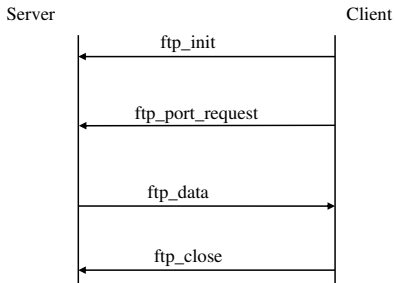
$(6, \text{smtp}, ((192, 169, 2, 8), 25), ((6, 2, 0, 4), 2), \text{data}))$

- FUT $(2, \text{smtp}, ((192, 168, 0, 6), 6), ((9, 0, 8, 0), 6), \text{data})$
= Some deny

Testing State-less Firewalls: An Example IV

- time used for policies with 13 rules : 19 hours (for example)
- see: Achim D. Brucker, Lukas Brügger, and Burkhard Wolff. Model-based firewall conformance testing. In Kenji Suzuki and Teruo Higashino, editors, Testcom/FATES 2008, LNCS 5047, pages 103-118. Springer-Verlag, Tokyo, Japan, 2008.
- improved techniques use a (proven correct) pre-normalization and a technique to decompose subnets.
time used for policies with 100 rules : 5 s (for example).
- see: Achim D. Brucker, Lukas Brügger, Paul Kearney, and Burkhard Wolff. Verified firewall policy Transformations for Test Case Generation. In Ana Cavalli and Sudipto Ghosh, editors, International Conference on Software Testing (ICST10), Lecture Notes in Computer Science. Springer-Verlag, 2010.

State-full Firewalls: An Example (ftp) I



State-full Firewalls: An Example (ftp) II

- based on our state-less model:
Idea: a firewall (and policy) has an internal state:
- the firewall state is based on the history and the current policy:

types (α, β, γ) FWState = " $\alpha \times (\beta, \gamma)$ Policy"

- where FWStateTransition maps an incoming packet to a new state

types (α, β, γ) FWStateTransition =
 " $((\beta, \gamma)$ In_Packet $\times (\alpha, \beta, \gamma)$ FWState) \rightarrow
 $((\alpha, \beta, \gamma)$ FWState)"

State-full Firewalls: An Example (ftp) III

HOL-TestGen generates test case like:

```
FUT [(6, ftp, ((192, 168, 3, 1), 10), ((4, 7, 9, 8), 21), close),
      (6, ftp, ((4, 7, 9, 8), 21), ((192, 168, 3, 1), 3), ftp_data),
      (6, ftp, ((192, 168, 3, 1), 10), ((4, 7, 9, 8), 21), port_request 3),
      (6, ftp, ((192, 168, 3, 1), 10), ((4, 7, 9, 8), 21), init)] =
  [(6, ftp, ((192, 168, 3, 1), 10), ((4, 7, 9, 8), 21), close),
   (6, ftp, ((4, 7, 9, 8), 21), ((192, 168, 3, 1), 3), ftp_data),
   (6, ftp, ((192, 168, 3, 1), 10), ((4, 7, 9, 8), 21), port_request 3),
   (6, ftp, ((192, 168, 3, 1), 10), ((4, 7, 9, 8), 21), init)],
  new_policy)
```

time used: 7 minutes; but involves techniques like sequencing and protocol specifications not yet introduced here.

Firewall Testing: Summary

- Successful testing if a concrete configuration of a network firewall correctly implements a given policy
- Non-Trivial Test-Case Generation
- Non-Trivial State-Space (IP Adresses)
- Sequence Testing used for Stateful Firewalls
- Realistic, but amazingly concise model in HOL!

Outline

- 1 Advanced Test Scenarios
- 2 Case Studies
- 3 Conclusion

Conclusion I

- Approach based on theorem proving
 - test specifications are written in HOL
 - functional programming, higher-order, pattern matching
- Test hypothesis explicit and controllable by the user (could even be verified!)
- Proof-state explosion controllable by the user
- Although logically puristic, systematic unit-test of a “real” compiler library is feasible!
- Verified tool inside a (well-known) theorem prover

Conclusion II

- **Explicit Test Hypothesis** are controllable by the test-engineer (can be seen as proof-obligation!)
- In HOL, Sequence Testing and Unit Testing are the same!

- The Sequence Test Setting of HOL-TestGen is **effective** (see Firewall Test Case Study)
- HOL-Testgen is a **verified test-tool** (entirely based on derived rules ...)
- The **White-box Test** offers potentials to prune unfeasible paths early ... (but no large programs tried so far ...)

Conclusion II

- **Explicit Test Hypothesis** are controllable by the test-engineer (can be seen as proof-obligation!)
- In HOL, Sequence Testing and Unit Testing are the same!
TS pattern **Unit Test**:

$$\text{pre } x \longrightarrow \text{post } x(\text{prog } x)$$

- The Sequence Test Setting of HOL-TestGen is **effective** (see Firewall Test Case Study)
- HOL-Testgen is a **verified test-tool** (entirely based on derived rules ...)
- The **White-box Test** offers potentials to prune unfeasible paths early ... (but no large programs tried so far ...)

Conclusion II

- **Explicit Test Hypothesis** are controllable by the test-engineer (can be seen as proof-obligation!)
- In HOL, Sequence Testing and Unit Testing are the same!
TS pattern **Sequence Test**:

$$\text{accept trace} \implies P(\text{Mfold trace } \sigma_0 \text{ prog})$$

- The Sequence Test Setting of HOL-TestGen is **effective** (see Firewall Test Case Study)
- HOL-Testgen is a **verified test-tool** (entirely based on derived rules ...)
- The **White-box Test** offers potentials to prune unfeasible paths early ... (but no large programs tried so far ...)

Conclusion II

- **Explicit Test Hypothesis** are controllable by the test-engineer (can be seen as proof-obligation!)
- In HOL, Sequence Testing and Unit Testing are the same!
TS pattern **Reactive Sequence Test**:

$$\text{accept } trace \implies P(\text{Mfold } trace \sigma_0$$

(observer observer rebind subst prog))

- The Sequence Test Setting of HOL-TestGen is **effective** (see Firewall Test Case Study)
- HOL-Testgen is a **verified test-tool** (entirely based on derived rules ...)
- The **White-box Test** offers potentials to prune unfeasible paths early ... (but no large programs tried so far ...)

Bibliography I