Combining Formal Testing and Proving

Burkhart Wolff

Université Paris-Sud, LRI, CNRS

Instead of a Motivation:

- "Dijkstra's Verdict" :

, Program testing can be used to show the presence of bugs, but never to show their absence!

Instead of a Motivation:

- “Dijkstra's Verdict” :

    , Program testing can be used to show the presence of bugs, but never to show their absence!

- Well, Dijkstra was highly biased in the scientific debate (and contributed a lot to the approach); so can he be trusted ?

Instead of a Motivation:

- "Dijkstra's Verdict" :

  , Program testing can be used to show the presence of bugs, but never to show their absence!

- Wouldn't we question a statement by a boss of the nuclear industry that "coal-fired powerplants constitute a substantial risk for the environment" ???

Instead of a Motivation:

- “Dijkstra's Verdict” :

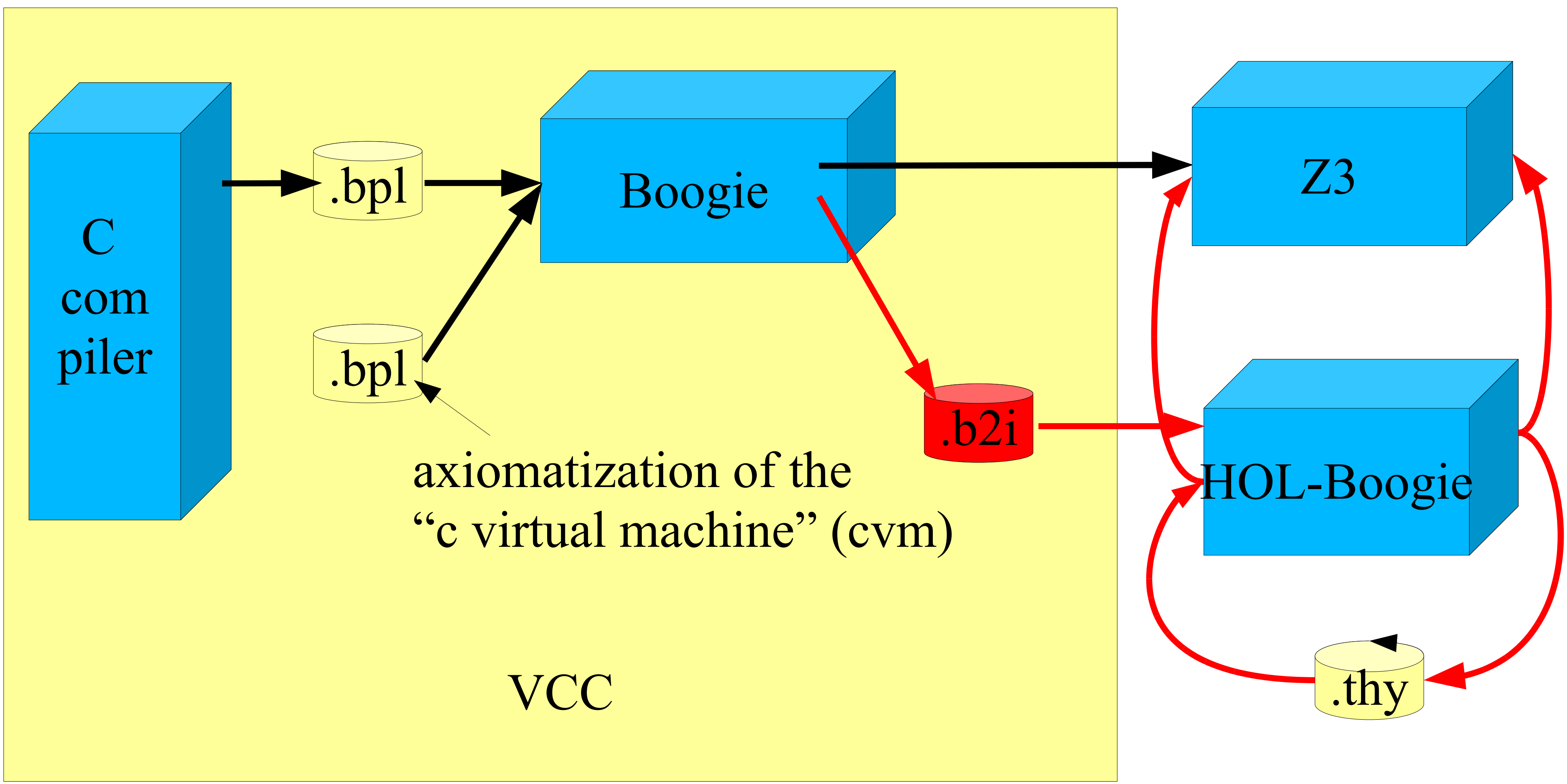
  , Program testing can be used to show the presence of bugs, but never to show their absence!

- So: can proof-based verifications guarantee the

  “abscence of bugs” ?

An Architecture of a Program Verifier (VCC)

- HOL-Boogie [Böhme, Wolff]



The Reality:

- In reality, proof-based verifications make a lot of assumptions (besides being costly in brain-power!)
  - operational semantics should be faithfully executed
  - complex memory-machine model consistent (VCC: 800 axioms)
  - correctness of the vc generation (for concurrent C with "ownership", "locks", ... !):
  - correctness of the vc generator and prover
  - abscence of an environment that manipulates the underlying state.

Instead of a Motivation:

- "Dijkstra's Verdict" :

  , Program testing can be used to show the presence of bugs, but never to show their absence!

- Then: is program verification by proof at least always better than testing ?

The Reality:

- Well, euh, strictly speaking not.
  - in general, both techniques use mutually independent assumptions, so …
  - … nothing well-founded can be said in general !!! It all depends on the concrete assumptions and the concrete setting !
  - there are actually cases in the literature where bugs in "verified systems" (meaning: systems verified by proof) were revealed by tests !

Instead of a Motivation:

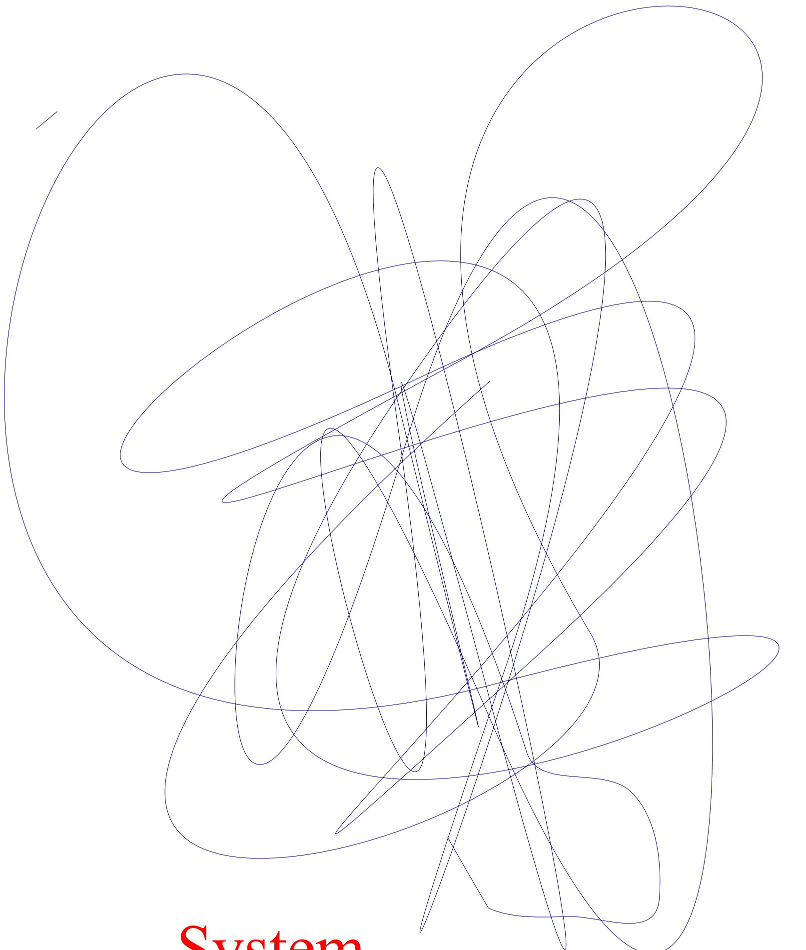
- "Dijkstra's Verdict" :

  , Program testing can be used to show the presence of bugs, but never to show their absence!

- Can we always avoid testing ?

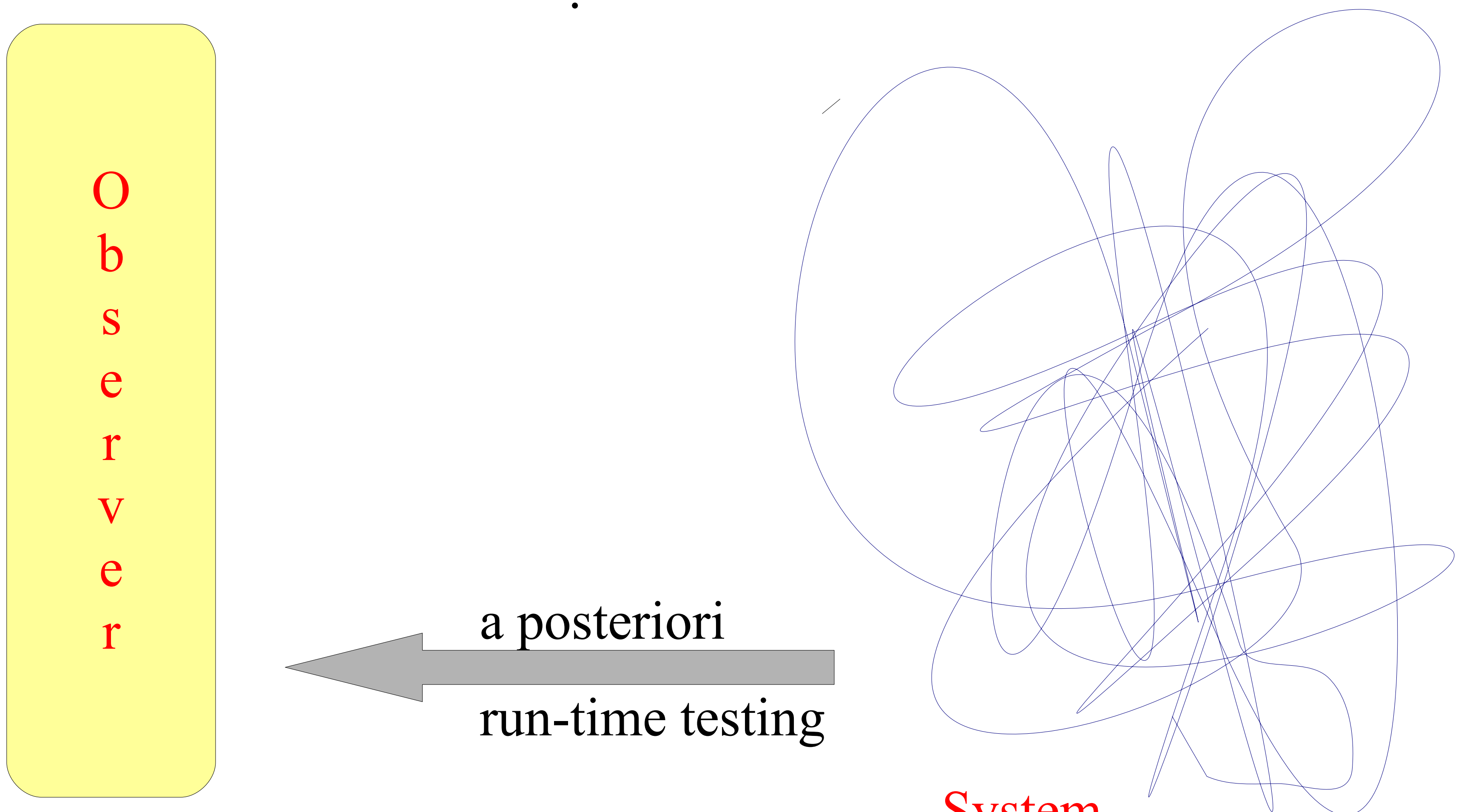
Models of Systems for Tests

.



System
(hard + software +
environment)

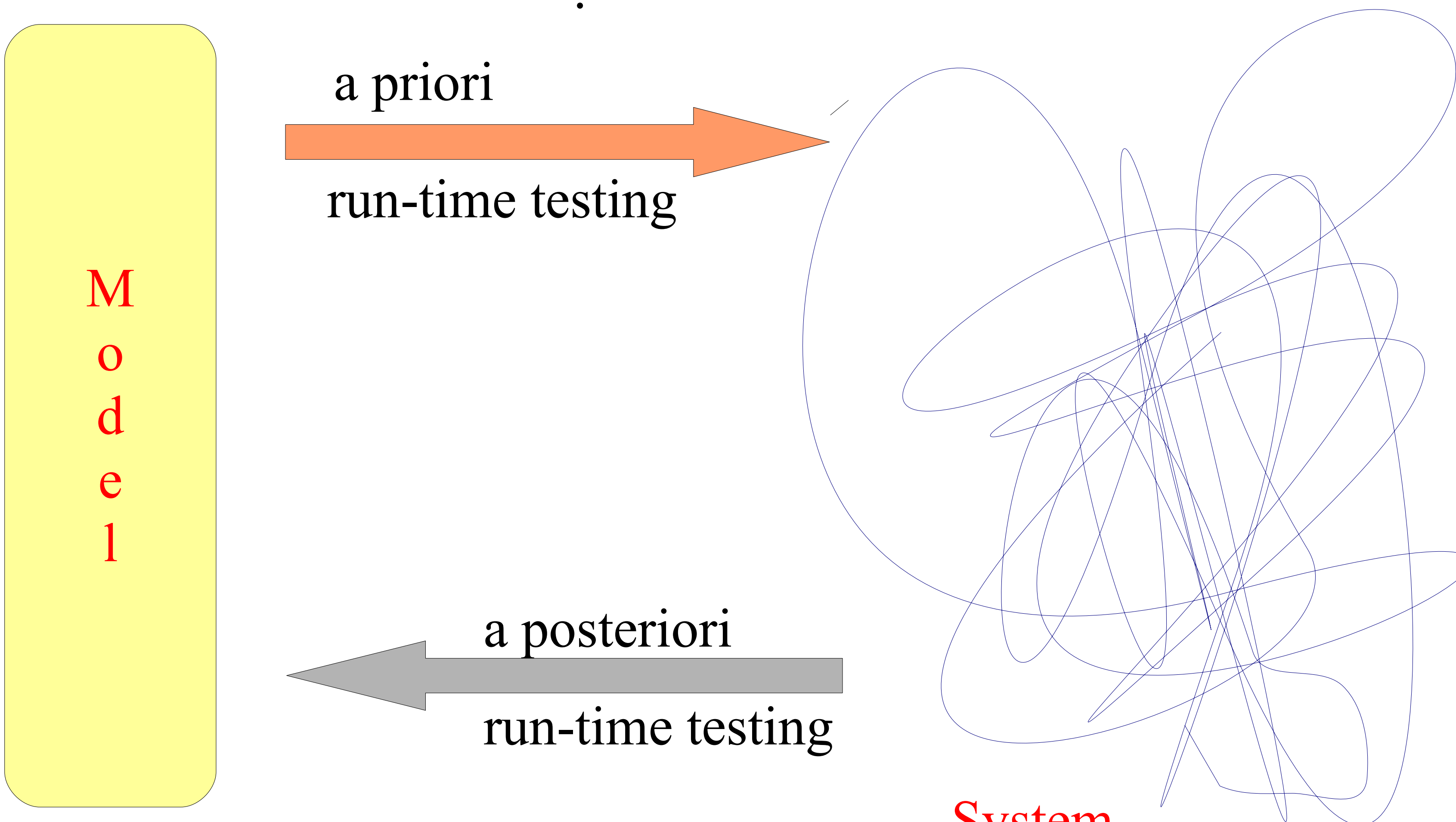
Models of Systems for Tests



System
(hard + software +
environment)

Test-Oracle correct
function or behaviour

Models of Systems for Tests



Model describing function or behaviour

System (hard + software + environment)

Verification by Model-based Testing …

- … can be done post-hoc; significant projects "reverse engineer" the model of a legacy system

- … attempts to find bugs in specifications EARLY (and can thus complement proof-based verification …)

- … can help system integration processes in a partly unknown environment ("embedded systems")

Nothing of this can be done by proof-based verification !

Instead of a Motivation:

- "Dijkstra's Verdict" :

  , Program testing can be used to show the presence of bugs, but never to show their absence!

- Test and Proofs, are they actually adversaries?

  (Tony Hoare, POPL2012, says "meanwhile no").

Agenda

- MBT Tool HOL-TestGen (based on Isabelle/HOL) and outline its method
- Own Case Studies
- Demo
- 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 litte 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 □ ⟧ ⟹
    prog(z, z, z) = equilateral

□ ⟦x ⩽ z; 0 < x; 0 < z; z < x + z; x < z + z ⟧ ⟹
    prog(x, z, z) = isosceles

□ ⟦y ⩽ z; z  y;¬z < z + y ⟧ ⟹
    prog(z, y, z) = error
```

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

$$C_1 \Longrightarrow \ldots\ C_{26} \Longrightarrow \text{THYP } H_1 \Longrightarrow \text{THYP } H_{13} \Longrightarrow \text{TS}$$

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):

```
. . .
gen_test_data "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):

```
. . .
prog(3, 3, 3) = equilateral
prog(4, 6, 0) = error
```

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, ...)

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: Firewalls + UPF

- Access Control Policies represent a key element of security for Networks, Data-Bases, ...

- We modeled a "Unified Policy Framework" (UPF) and specialized our test-case generation approach

- ... used (internally) substantial interactive theorem proving for correctness of normalization theorem.

Own Case Study: Firewalls + UPF

- UPF (A Theory in HOL / for HOL-TestGen)
  - A Policy: A Decision Function

    datatype α decision = allow α | deny α

    types (α,β) policy = $\alpha \rightharpoonup \beta$ decision (* = $\alpha \Rightarrow \beta$ option *)

    notation α ⊕→ β = (α,β) policy

  - Operators

    definition $\emptyset \equiv \lambda$ y. None

    definition p(x $\mapsto$ t) $\equiv \lambda$ y. if y = x then A else p y

    definition A $\equiv$ {x.∃y. x = allow y} , D $\equiv$ {x.∃y. x = deny y}

    definition p(x+$\mapsto$t) $\equiv$ p(x $\mapsto$ allow t) p(x−$\mapsto$t) $\equiv$ p(x $\mapsto$ deny t)

    definition (*AllowAll*) ∀Af $\equiv \lambda$ x. allow(f x),(*DenyAll*) ∀Df $\equiv \lambda$ x. deny(f x)

    … domain / range restriction S $\lhd$ p, p $\rhd$ S, override $p_1 \oplus p_2$ ...

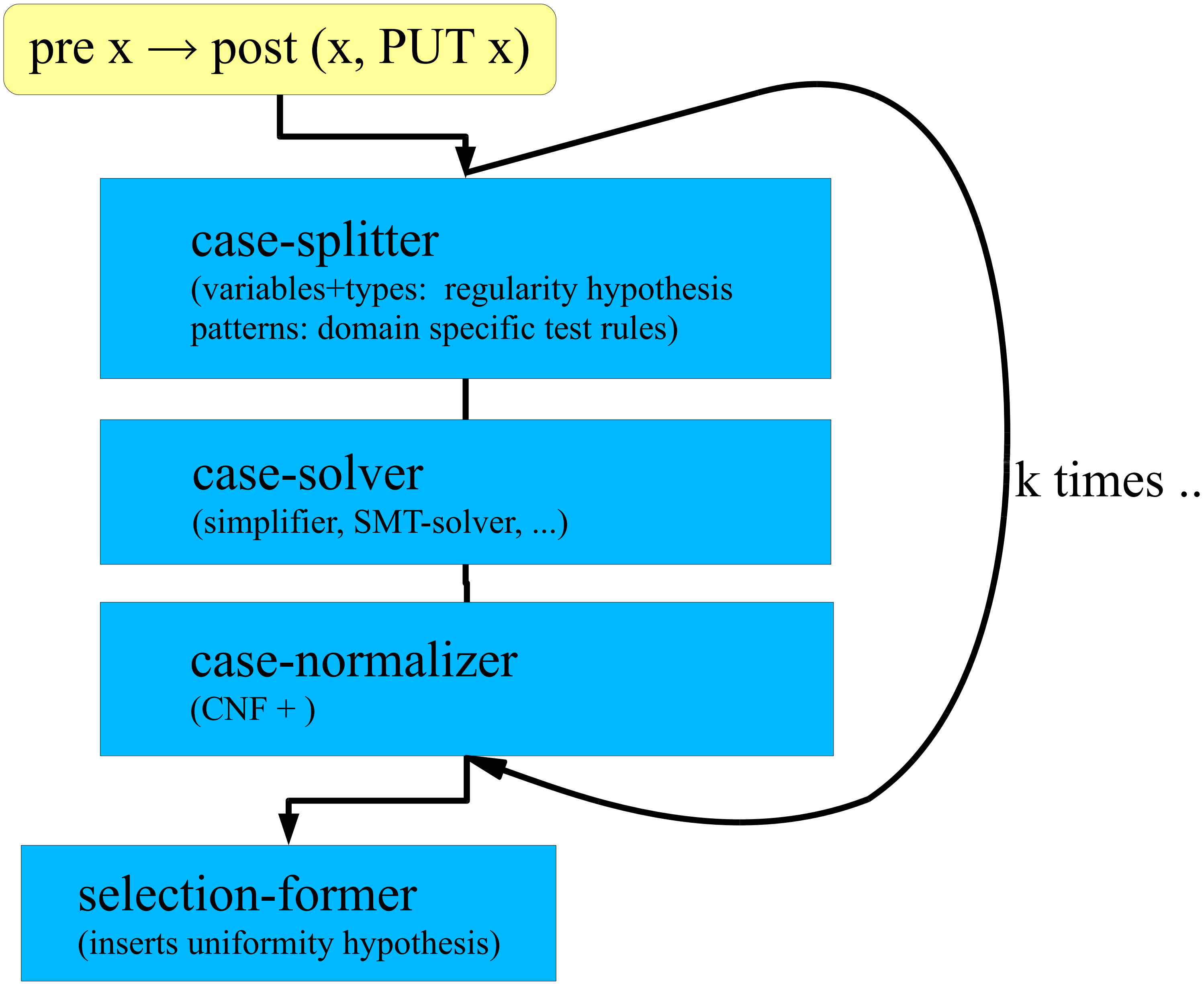
Own Case Study: Firewalls + UPF

DEMO !

Conclusion: Test & Proof

- … can never ever establish the absense of "Bugs" in a system! Never ever. Both of them.
- … can, when combined, further increase confidence in verification results by using mutually independent assumptions.
- … can, when combined, offer new ways to tackle abstraction and state space explosion. (UPF Normalization Theorem)
- … can share Tools and Tool development efforts. (Parallelization, Interfaces, Counter-Example Gen.)

TestGen: Symbolic Computations



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