QuickCheck

Koen Lindström Claessen

QuickCheck

- A testing library for Haskell in 2000
  - Koen Claessen
  - John Hughes
- Now the “standard” way of testing Haskell programs
- Also: Erlang, C, C++, Java, OCaml, Python, Isabelle, Coq, ...

Quviq AB

- A testing company founded in the early 2000s

- Commercial version of Erlang QuickCheck
  - State machines
  - Property libraries for industrial applications
  - ...

QuickCheck Success Stories

XMonad, darcs

PULSE - Concurrent Software

Compiler testing

model checkers

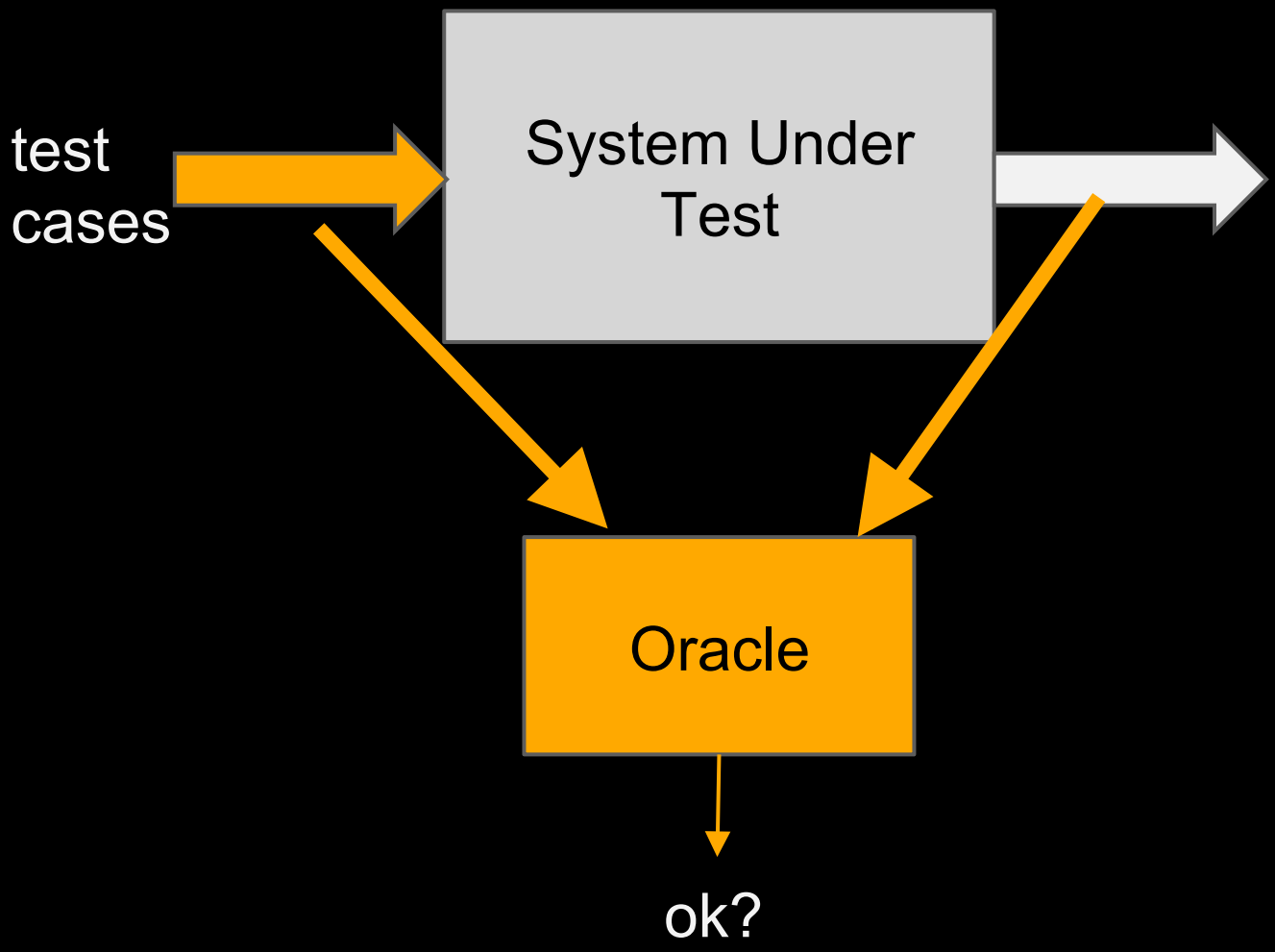
QuickCheck

- Properties
  - one aspect of functionality of the code
- Random test data
  - Each time you get a new test case
  - Library for crafting generators
- Shrinking
  - Understanding the failing case
  - (Avoid getting to the same failing case every time!)

Testing Implementations of Complex Algorithms

using *contrapositive testing*, *inductive testing*, and *co-inductive testing*

Koen Lindström Claessen
Chalmers, Gothenburg, Sweden
and Utrecht University


test
cases
System Under
Test
Oracle
ok?

Oracle

- Simple
  - Simpler than the implementation
- Practically runnable
  - May need to run many tests
- Oracle should be “complete”
  - For any faulty implementation, there should exist inputs that trigger the oracle to say “no”

Shortest Path Algorithms

```
type Map
type Point
type Path


shortest : (Map, Point, Point) -> Maybe Path



( solve : Problem -> Maybe Solution )
```

Problem

- The oracle needs to know what the shortest path is

- We can be **simple**, but it is **too slow**
  - Not practical when testing
  - (Non-termination!)

- We can be **fast**, but it is **too complex**
  - We may not trust our test results

Property-based Testing

(a la QuickCheck)

Sound - If an answer is produced, it should be an actual solution

Complete - If no answer is produced, there indeed was no actual solution

Optimal - If an answer is produced, there is no actual solution that is better

Complete - If no answer is produced, there indeed was no actual solution

logically equivalent

Complete' - If there is a solution, some answer will be produced

testable

ForAll x . A(x) ==> B(x)

ForAll x in “A”. B(x)

```
ForAll mp,a,b .
    hasPath mp a b ==>
        isJust (shortest (mp, a, b))
```

```
ForAll mp,a,b in hasPathMap .
        isJust (shortest (mp, a, b))
```

Optimal - If an answer is produced, there is no actual solution that is better

logically equivalent

Optimal' - If there is a solution, then no worse answer will be produced

testable ?

Contrapositive testing

- Change your viewpoint
  - From: Stimuli / System Under Test / Oracle
  - To: Logical implication
- And take the contrapositive view to get new inspiration
- Sometimes, you have a choice! (How to make it?)

Contrapositive Testing

Shortest Distance Algorithms

```
type Map
type Point
data Distance = Inf | Fin Int


distance : (Map, Point, Point) -> Distance
```

Sound - If an answer is produced, it should be an actual solution

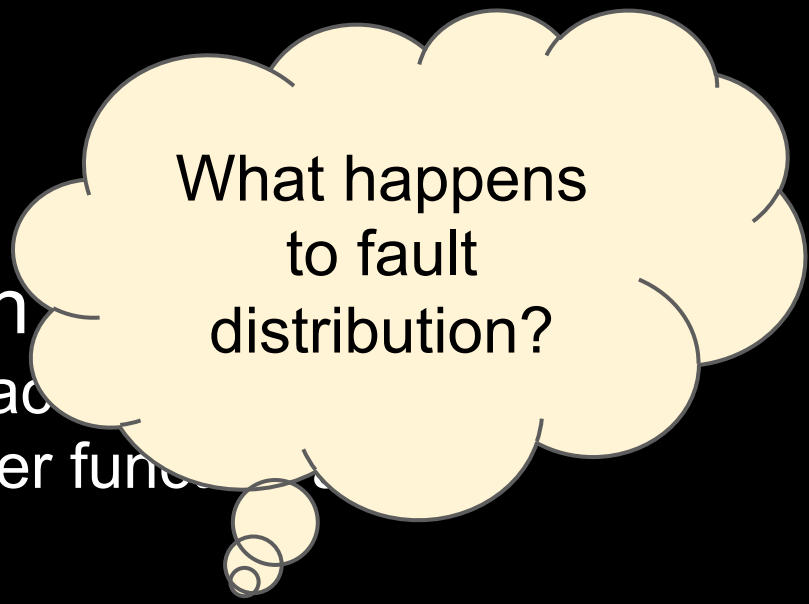
Complete - If no answer is produced, there indeed was no actual solution

Optimal - If an answer is produced, there is no actual solution that is better

```
ForAll mp,a,b .
  distance(mp,a,b) ==
    minimum [ distance(mp,a’,b) + d
            | (a’,d) <- neighbors(mp,a)
            ]
```

```
ForAll mp,a,a .
  distance(mp,a,a) == Fin 0
```

Inductive Testing



- Correctness: by induction
  - soundness: induction over ac
  - completeness: induction over func
- Induction principle
  - choose this for enabling testing
  - independent of implementation (unlike proving)
- Induction vs. recursion in implementation
  - too slow to use directly (even non-terminating)
  - Plotkin induction

Testing SAT-solvers

Testing SAT-solvers

- If model and proof are generated
  - Direct soundness
  - Direct completeness
- If only model is generated when found
  - Direct soundness
  - Contrapositive testing for completeness
- If only yes/no answer
  - Inductive testing
  - Base case: no variables
  - Step case: branch on a variable

Testing Sorting

Testing sorting functions

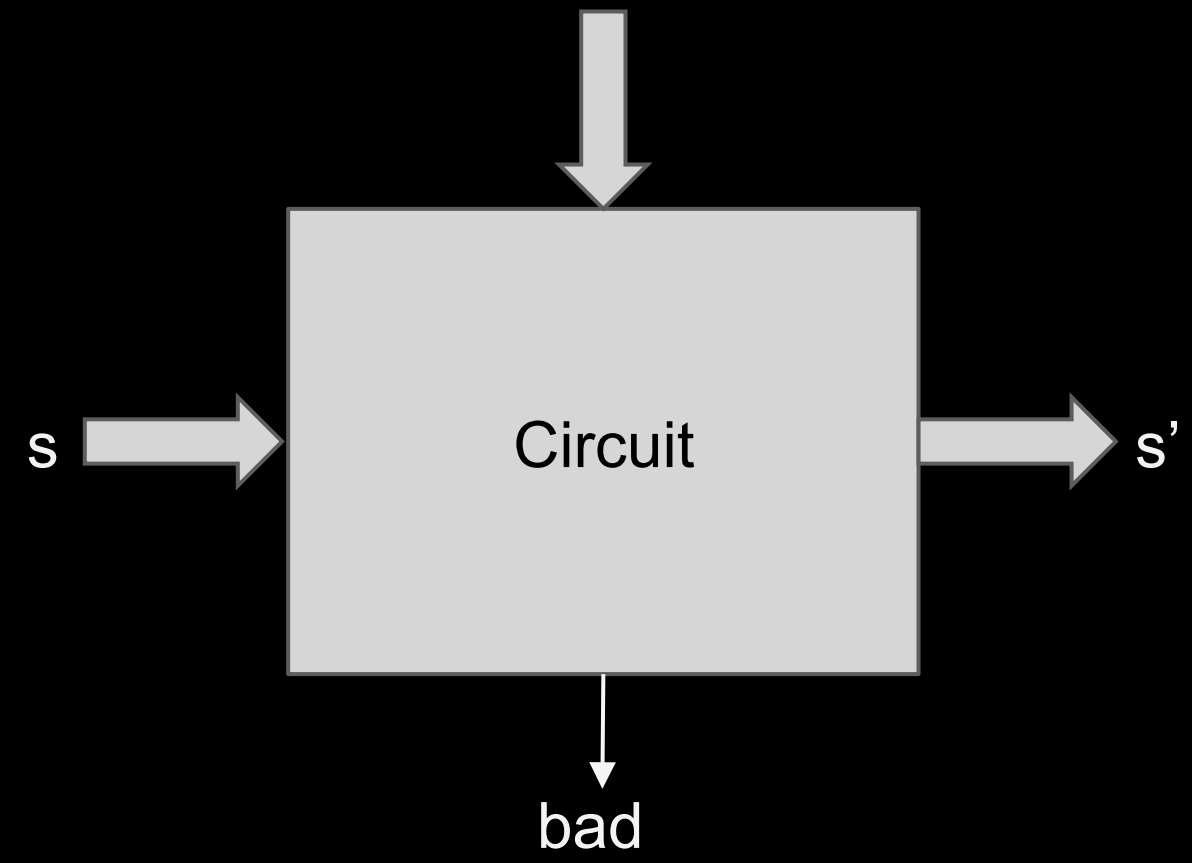
- Write down the simplest sorting function you can think of
  - *You trust this code*
- Show that the function you want to test has the same behavior
  - *How?*

Testing FFT implementations

Testing FFT

- Using exact arithmetic
  - Implementation is still fast
  - Specification is extremely slow
- Base cases
  - vectors [0,..,0,1,0,..,0]
- Step cases
  - a * fft v = fft (a*v)
  - fft v + fft w = fft (v + w)

Testing Model Checkers for Safety Properties



s0

False: The circuit is **not safe**; often produces a **trace**

check : (State, Circuit) -> Bool

True: The circuit is **safe**; (produces nothing)

step : (State, Circuit, Input) -> (Bool, State)

```
ForAll s, C .
    check(s, C) ==>
        ForAll inp .
            let (ok, s') = step(s, C, inp) in
                ok && check(s', C)
```

-
  -
  -
  -
-
  -

$$\frac{a \leq F(a)}{a \leq \mathrm{gfp}\ x\ .\ F(x)}$$

Inductive Testing

- Break away from the stimuli / system under test / oracle view
- Look at the logical meaning of the property
- Use proof techniques to “break up” into smaller properties
  - Together, they imply the original property
  - They may be easier to test
  - The system may be run several times
- What happens to the distribution of faulty test cases?

Ongoing Work

- More examples
  - Testing compilers / interpreters
  - Theorem provers for decidable logics
  - Theorem provers for semi-decidable logics
  - Unification algorithm
  - Distributed systems
  - …
- Develop “testing logic”
  - Logical equivalence
  - Testing non-equivalence
  - Cost of testing
  - Predict which testing ways are most effective