CS 2800: Logic & Computation

Property-based Testing

Hemann

Who enjoys writing tests?

Tedious

Byzantine

Repetitive

Mindless

Never-ending

Tedious

Byzantine

Repetitive

Mindless

Never-ending

What kinds of tasks are computers great at?

- Tedious
- Byzantine
- Repetitive
- Mindless
- Never-ending

Yahoo - A Guide to WWW

[What's New? | What's Cool? | What's Popular? | Stats | A Random Link]

Top | Up | Search | Mail | Add | Help

- Art(466) NEW
- Business(6426) NEW
- Computers(2609) NEW
- Economy(743) NEW
- Education(1487) NEW
- Entertainment(6199) NEW
- Environment and Nature(193) NEW
- Events(53) NEW
- Government(1031) NEW
- Health(367) NEW
- Humanities(163) NEW
- Law(163) NEW
- News(185)
- Politics(148) NEW
- Reference(474) NEW
- Regional Information(2606) NEW
- Science(2634) NEW
- Social Science(93) NEW
- Society and Culture(648) NEW

23836 entries in *Yahoo* [Yahoo / Up / Search / Mail / Add / Help]

---

yahoo@akebono.stanford.edu

Testing by hand??

What is a property?

Property = a statement which either holds or not, i.e., the statement is either true or false

Example: *"the two functions below always return the same result when the input is a natural number"*

```
(definec even-natp (x :nat) :bool
  (natp (/ x 2)))

(definec even-intp (x :int) :bool
  (integerp (/ x 2)))
```

What is a property?

Property = a statement which either holds or not, i.e., the statement is either true or false

Example: *"the two functions below always return the same result when the input is a natural number"*

```
(definec even-natp (x :nat) :bool
  (natp (/ x 2)))

(definec even-intp (x :int) :bool
  (integerp (/ x 2)))
```

```
(implies (natp n)
         (equal (even-natp n)
                (even-intp n)))
```

Testing v proving

Testing:

- Generate many examples and test on each example whether property holds
  - In the preceding, an **example** is **one specific n**
- The test passes if the tool cannot find an example violating the property: a **counter-example**
- More powerful than a just a single test of one example

Theorem proving:

- **Prove** that the property holds for **every n**
- More powerful than mere testing (**why?**)
- Theorem proving techniques fundamentally different from testing

PBT Possible outcomes

a. Generates a counter-example (property is false, does not hold)

b. All examples pass (property might or might not hold)

c. Proof of correctness (sometimes PBT’ers can manage a proof)

PBT Possible outcomes

a. Generates a counter-example (property is false, does not hold)

b. All examples pass **(property might or might not hold!)**

c. Proof of correctness (sometimes PBT'ers can manage a proof)

PBT: The Wrong Way

```
;; Testing Reverse
;; (Listof Int) -> Property
reverseCorrect(xs) =
   return reverse(xs) == ???
```

```
;; Testing Reverse
;; (Listof Int) -> Property
reverseCorrect(xs) =
   return reverse(xs) == correctBehaviorForReverse(xs)
```

```
;; Testing Reverse
;; (Listof Int) -> Property
reverseCorrect(xs) =
   return reverse(xs) == correctBehaviorForReverse(xs)
```

```
;; Testing Reverse
;; (Listof Int) -> Property
reverseCorrect(xs) =
   return reverse(reverse(xs)) == xs
```

1. **Generate** tests from the properties

2. **Shrink** failing tests to minimal failure

QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs

Koen Claessen
Chalmers University of Technology
koen@cs.chalmers.se

John Hughes
Chalmers University of Technology
rjmh@cs.chalmers.se

ABSTRACT

QuickCheck is a tool which aids the Haskell programmer in formulating and testing properties of programs. Properties are described as Haskell functions, and can be automatically tested on random input, but it is also possible to define custom test data generators. We present a number of case studies, in which the tool was successfully used, and

monad are hard to test), and so testing can be done at a fine grain.

A testing tool must be able to determine whether a test is passed or failed; the human tester must supply an automatically checkable criterion of doing so. We have chosen to use formal specifications for this purpose. We have designed a simple domain-specific language of *testable specifi-*

Last Published: July 31, 2020

junit-quickcheck: Property-based testing, JUnit-style

junit-quickcheck is a library that supports writing and running property-based tests in JUnit inspired by QuickCheck for Haskell.

junit-quickcheck is source/target-compatible with JDK 8.

Basic example

```
import com.pholser.junit.quickcheck.Property;
import com.pholser.junit.quickcheck.runner.JUnitQuickcheck;
import org.junit.runner.RunWith;

import static org.junit.Assert.*;

@RunWith(JUnitQuickcheck.class)
public class StringProperties {
    @Property public void concatenationLength(String s1, String s2) {
        assertEquals(s1.length() + s2.length(), (s1 + s2).length());
    }
}
```

Re-implementations of QuickCheck exist for several languages:

- C[2][3][4]
- C++[5][6][7]
- Chicken[8]
- Clojure[9][10][11]
- Common Lisp[12]
- Coq[13]
- D[14]
- Elm[15]
- Elixir[16]
- Erlang[17]
- F#, and C#, Visual Basic .NET (VB.NET)[18]
- Factor[19]
- Go[20]
- Io[21]
- Java[22][23][24][25][26][27][28]
- JavaScript[29][30][31]
- Julia[32]
- Logtalk[33]
- Lua[34]
- Node.js[35]
- Objective-C[36]
- OCaml[37]
- Perl[38]
- Prolog[39][40]
- PHP[41]
- Pony[42]
- Python[43]
- R[44]
- Racket[45]
- Ruby[46]
- Rust[47][48]
- Scala[49][50][51]
- Scheme[52]
- Smalltalk[53]
- Standard ML[54]
- Swift[55]
- TypeScript[56]
- Whiley[57]

Wikipedia (2020) *Quickcheck*

Links, References, and More

- youtube.com/watch?v=zvRAyq5wj38 (Hughes, Writing Properties)
- Claessen, & Hughes, “Quickcheck: a lightweight tool …” ICFP 2000
- “Test & Code” Podcast (incl. Test Automation for SwEngineering)
- odp.org (DMOZ Archive)