COMP3141

Software System Design and Implementation

Property Based Testing Practice

Christine Rizkallah
CSE, UNSW (and Data61)
Term 2 2019

Property Based Testing

Key idea: Generate random input values, and test properties by running them.

Example (QuickCheck Property)

```
prop_reverseApp xs ys =
    reverse (xs ++ ys) == reverse ys ++ reverse xs
```

Property Based Testing

Key idea: Generate random input values, and test properties by running them.

Example (QuickCheck Property)

```
prop_reverseApp xs ys =
    reverse (xs ++ ys) == reverse ys ++ reverse xs
```

Haskell's *QuickCheck* is the first library ever invented for property-based testing. The concept has since been ported to Erlang, Scheme, Common Lisp, Perl, Python, Ruby, Java, Scala, F#, OCaml, Standard ML, C and C++.

Mersenne Number Example

Example (Demo Task)

- The n^{th} Mersenne number $M_n = 2^{n-1}$.
- M_2, M_3, M_5 and M_7 are all prime numbers.
- **Conjecture:** $\forall\, n.prime(n) \implies prime(2^{n-1})$

Mersenne Number Example

Example (Demo Task)

- The n^{th} Mersenne number $M_n = 2^{n-1}$.
- M_2, M_3, M_5 and M_7 are all prime numbers.
- **Conjecture:** $\forall\, n.prime(n) \implies prime(2^{n-1})$

Let's try using QuickCheck to answer this question.

Mersenne Number Example

Example (Demo Task)

- The n^{th} Mersenne number $M_n = 2^{n-1}$.
- M_2, M_3, M_5 and M_7 are all prime numbers.
- **Conjecture:** $\forall\, n.prime(n) \implies prime(2^{n-1})$

Let's try using QuickCheck to answer this question.

After 14 guesses and fractions of a second, QuickCheck found a counter-example to this conjecture: 11.

Mersenne Number Example

Example (Demo Task)

- The n^{th} Mersenne number $M_n = 2^{n-1}$.
- M_2, M_3, M_5 and M_7 are all prime numbers.
- **Conjecture:** $\forall\, n.prime(n) \implies prime(2^{n-1})$

Let's try using QuickCheck to answer this question.

After 14 guesses and fractions of a second, QuickCheck found a counter-example to this conjecture: 11.

It took humanity about two thousand years to do the same.

Ransom Note Example

Example (Demo Task)

Given a magazine (in `String` form), is it possible to create a ransom message (in `String` form) from characters in the magazine.

Ransom Note Example

Example (Demo Task)

Given a magazine (in `String` form), is it possible to create a ransom message (in `String` form) from characters in the magazine.

We must:

1. Count the characters in the magazine
2. Count the characters in the message
3. Check whether the magazine contains enough characters to construct the message

Ransom Note Example

Example (Demo Task)

Given a magazine (in `String` form), is it possible to create a ransom message (in `String` form) from characters in the magazine.

We must:

1. Count the characters in the magazine
2. Count the characters in the message
3. Check whether the magazine contains enough characters to construct the message

In Haskell.

Recall: Proofs

Proofs:

- Proofs must make some assumptions about the environment and the semantics of the software.

Recall: Proofs

Proofs:

- Proofs must make some assumptions about the environment and the semantics of the software.
- Proof complexity grows with implementation complexity, sometimes drastically.

Recall: Proofs

Proofs:

- Proofs must make some assumptions about the environment and the semantics of the software.
- Proof complexity grows with implementation complexity, sometimes drastically.
- If software is incorrect, a proof attempt might simply become stuck: we do not always get constructive negative feedback.

Recall: Proofs

Proofs:

- Proofs must make some assumptions about the environment and the semantics of the software.
- Proof complexity grows with implementation complexity, sometimes drastically.
- If software is incorrect, a proof attempt might simply become stuck: we do not always get constructive negative feedback.
- Proofs can be labour and time intensive ($$$), or require highly specialised knowledge ($$$).

Testing

Compared to proofs:

- Tests typically run the actual program, so requires fewer assumptions about the language semantics or operating environment.

Testing

Compared to proofs:

- Tests typically run the actual program, so requires fewer assumptions about the language semantics or operating environment.
- Test complexity does not grow with implementation complexity, so long as the specification is unchanged.

Testing

Compared to proofs:

- Tests typically run the actual program, so requires fewer assumptions about the language semantics or operating environment.
- Test complexity does not grow with implementation complexity, so long as the specification is unchanged.
- Incorrect software when tested leads to immediate, debuggable counterexamples.

Testing

Compared to proofs:

- Tests typically run the actual program, so requires fewer assumptions about the language semantics or operating environment.
- Test complexity does not grow with implementation complexity, so long as the specification is unchanged.
- Incorrect software when tested leads to immediate, debuggable counterexamples.
- Testing is typically cheaper and faster than proving.

Testing

Compared to proofs:

- Tests typically run the actual program, so requires fewer assumptions about the language semantics or operating environment.
- Test complexity does not grow with implementation complexity, so long as the specification is unchanged.
- Incorrect software when tested leads to immediate, debuggable counterexamples.
- Testing is typically cheaper and faster than proving.
- Tests care about efficiency and computability, unlike proofs.

Testing

Compared to proofs:

- Tests typically run the actual program, so requires fewer assumptions about the language semantics or operating environment.
- Test complexity does not grow with implementation complexity, so long as the specification is unchanged.
- Incorrect software when tested leads to immediate, debuggable counterexamples.
- Testing is typically cheaper and faster than proving.
- Tests care about efficiency and computability, unlike proofs.

We lose some assurance, but gain some convenience ($$$).

Verification versus Validation

"Testing shows the presence, but not the absence of bugs."
– Dijkstra (1969)

Testing is essential but is insufficient for safety-critical applications.

Homework

1. Next programming exercise is due in 6 days.
2. Last week's quiz is due this Friday. Make sure you submit your answers.
3. This week's quiz is also up, due the following Friday.