COMP3141

Software System Design and Implementation

Property Based Testing Practice

Curtis Millar
CSE, UNSW (and Data61)

17 June 2020

Exercise 1

1. **Simple Picture**: add the chimney and smoke
2. **Moving Objects**: implement `movePictureObject`
3. **Generating a Picture**: generate pictures of circles using `simpleCirclePic`

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 Prime 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 a small number of 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.

Semigroup and Monoid Properties

Last week we proved by hand that a list forms a semigroup with ++ as its associative operator and a monoid with [] as its identity element.
We can show the same properties much faster (although less completely) with property based testing.

QuickCheck Properties

```
-- Semigroup laws
prop_listAssociative xs yz zs = ((xs ++ ys) ++ zs) == (xs ++ (ys ++ zs))

-- Monoid laws
prop_listLeftIdentity xs = xs == [] ++ xs
prop_listRightIdentity xs = xs == xs ++ []
```

Reverse Involution

Last week we also proved by hand that the `reverse` function is an *involution*.
This took over twenty minutes.
Let's see how long it takes QuickCheck.

QuickCheck Property

```
prop_reverseInvolution xs = reverse (reverse xs) == xs
```

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.

```
canMakeRansom :: RansomNote -> Magazine -> Bool
```

1. Write a specification
2. Create an efficient implementation
3. Test the implementation

In Haskell.

Graphics

Write some specifications for the following functions, use them to create properties, and then test an implementation.

1. Horizontal flip
2. Vertical flip
3. Rotate 180 degrees

Example (Demo Task)

Implement the above for a single `Path`. (You might want to try and implement these for other `PictureObject` constructors or for an entire `Image` as self-practice.) In Haskell.

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.
- 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. Last week's quiz is due on Friday. Make sure you submit your answers.
2. The second programming exercise is due by the start if my next lecture (in 7 days).
3. This week's quiz is also up, it's due next Friday (in 9 days).

Consultations

Tomorrow, 9am to 11am on Blackboard Collaborate

Link on course website.

Be ready to share your screen with REPL (`ghci` or `stack repl`) and editor set up.