Got To Test Them All

Steve Love

ACCU April 2017

@IAmSteveLove
steve@arventech.com
https://perfectcobalt.blogspot.co.uk/

Tests

What?

Tests

Why?

Procedural testing

```
[ Test ]
public void Procedural()
{
    for( int n = 1; n <= 100; ++n )
    {
        var result = Fizz.Buzzed( n );
        if( n % 3 == 0 )
            Assert.That( result, Does.StartWith( "Fizz" ) );
        if( n % 5 == 0 )
            Assert.That( result, Does.EndWith( "Buzz" ) );
        if( n % 15 == 0 )
            Assert.That( result, Is.EqualTo( "FizzBuzz" ) );
    }
}
```

Necessity and sufficiency

```
if( ! ( n % 3 == 0 || n % 5 == 0 ) )
     Assert.That( result, Is.EqualTo( n.ToString() ) );
```

Test replicates the algorithm

Selective testing

```
[ TestCase( 3 ) ]
[ TestCase( 6 ) ]
// ...
[ TestCase( 96 ) ]
[ TestCase( 99 ) ]
public void All_values_are_Fizz( int i )
{
    Assert.That( Fizz.Buzzed( i ), Is.EqualTo( "Fizz" ) );
}
```

Not really scalable

```
[ TestCase( 5 ) ]
[ TestCase( 10 ) ]
// ...
public void All_values_are_Buzz( int i )
{
    // ...

[ TestCase( 15 ) ]
[ TestCase( 30 ) ]
// ...
public void All_values_are_FizzBuzz( int i )
{
    // ...

[ TestCase( 1 ) ]
[ TestCase( 2 ) ]
// ...
public void All_values_are_numerical_representation( int i )
{
    //...
```

Looked at sideways

```
[ Test ]
public void Is_a_Fizz_when_divisible_by_3()
{
    var values = Enumerable.Range( 1, 100 )
                           .Where( i => i % 3 == 0 );

    Assert.That( values.Select( Fizz.Buzzed ),
                 Is.All.EqualTo( "Fizz" ) );
}
```

```
1) Failed : Is_a_Fizz_when_divisible_by_3
  Expected: all items equal to "Fizz"
  But was:   < "Fizz", "Fizz", "Fizz", "Fizz", "FizzBuzz", "Fizz", "Fizz", "Fizz", "Fizz", "FizzBuzz"... >
```

Converging...

Back to replicating the algorithm

```
[ Test ]
public void Is_a_Fizz_when_divisible_by_3_and_not_5()
{
    var values = Enumerable.Range( 1, 100 )
        .Where( i => i % 3 == 0 && i % 5 != 0 );

    Assert.That( values.Select( Fizz.Buzzed ),
        Is.All.EqualTo( "Fizz" ) );
}
```

...but it’s closer to what we want

Almost there

Implication

```
.Where( i => i % 3 == 0 && i % 5 != 0 );
```

Property

```
Is.All.EqualTo( "Fizz" ) );
```

What we really really want

- Randomly generated items
  - …possibly filtered
- A report of the *specific* value that disproves the assertion
  - …or better yet, the simplest example that disproves it

Another testing framework?

Haskell's QuickCheck

- F# FsCheck
- Python Hypothesis
- Scala ScalaCheck
- C++ CppQuickCheck

...and many many others

Properties

- **Multiples of both 3 and 5**
  exhibit property FizzBuzz

```
let ``All multiples of 3 and 5 are FizzBuzzes`` x =
    match x with
    | MultipleOf3 & MultipleOf5 ->  x |> Fizz.Buzzer = "FizzBuzz"
    | _ ->                          true
```

...and implications

- **Fizz** *implies that* the input value is a multiple of 3

```
let ``All Fizzes are multiples of 3`` x =
    match x |> Fizz.Buzzer with
    | "Fizz" ->     x |> IsMultipleOf3
    | _ ->          true
```

Generators

- Custom types

...and filters

- Positive integers
- Even numbers
- Beware: insufficient examples

Shrinkers

Simplification of the input to find the smallest example.

Arbitrary

- Generator
- Shrinker

For *arbitrary* values

A more interesting example

Least Common Multiple of 2 digits

- If either input is 0, the LCM is 0
- Both digits divide the LCM exactly

F# LCM property tests

```
let ``LCM of 0 and 0 is 0`` () =
    lcm 0 0 = 0

let ``LCM of 0 and n is 0`` n =
    lcm 0 n = 0

let ``LCM of n and 0 is 0`` n =
    lcm n 0 = 0

let ``LCM of x and y is an exact multiple of x`` x y =
    ( lcm x y ) % x = 0
```

A naive implementation

```
let lcm x y = x * y
```

And our first failure

```
let ``LCM of x and y is an exact multiple of x`` x y =
    ( lcm x y ) % x = 0
```

```
GcdLcm.LcmTest.LCM of x and y is an exact multiple of x [FAIL]
       FsCheck.Xunit.PropertyFailedException :
       Falsifiable, after 1 test (0 shrinks) (StdGen (1453944661,296289872)):
       Original:
       (0, 0)
       ---- System.DivideByZeroException : Attempted to divide by zero.
```

Built in filters

With the corresponding extra test while we're here

```
let ``LCM of x and y is an exact multiple of x`` (NonZeroInt x) y =
    ( lcm x y ) % x = 0
let ``LCM of x and y is an exact multiple of y`` x (NonZeroInt y) =
    ( lcm x y ) % y = 0
```

Now the tests pass.

Except, the implementation *can't* be right, can it?

But the tests still pass.

Perhaps there are other properties…?

More properties of LCM

- The LCM is the **least** common multiple

```
let ``LCM is the smallest possible multiple`` x y =
    result = lcm( x, y )
    start = max( x, y ) + 1

    assert not any( ch % x == 0 and ch % y == 0
        for ch in range( start, result ) )
```

That might take a while

GCD and LCM are related

```
let ``LCM is the smallest possible multiple`` x y =
    ( lcm x y ) * ( gcd x y ) = abs( x * y )
```

Oh, wait...

Ok we need a GCD

Time for a Hypothesis...

```
def test_gcd_of_0_and_0_is_0():
    assert gcd( 0, 0 ) == 0

@given( integers(), integers() )
def test_gcd_is_commutative( x, y ):
    assert gcd( x, y ) == gcd( y, x )

@given( integers() )
def test_gcd_is_reflexive( x ):
    assert gcd( x, x ) == x

@given( integers() )
def test_gcd_of_n_and_0_is_n( x ):
    assert gcd( 0, x ) == x
    assert gcd( x, 0 ) == x

@given( integers(), integers() )
def test_gcd_when_x_is_factor_of_y_is_x( x, y ):
    assert gcd( x, x * y ) == x
```

Straight to it

```
def gcd( x, y ):
    while y:
        x, y = y, x % y
    return x
```

And some more interesting properties

```
@given( integers(), integers(), integers() )
def test_gcd_cancels_out_common_factor( x, y, m ):
    assert gcd( x, y ) == gcd( x - y * m, y )

@given( integers(), integers(), integers() )
def test_gcd_distributes_a_common_factor( x, y, m ):
    assert abs( m ) * gcd( x, y ) == gcd( x * m, y * m )

@given( integers(), integers(), integers() )
def test_gcd_is_associative( x, y, z ):
    assert gcd( x, gcd( y, z ) ) == gcd( gcd( x, y ), z )
```

So far, so good.

We can even check with a standard library version:

```
@given( integers(), integers() )
def test_gcd_matches_builtin( x, y ):
    assert gcd( x, y ) == fractions.gcd( x, y )
```

So far, so good...so wrong

From the one true definition

> *the greatest common divisor (gcd) of two or more integers, when at least one of them is not zero, is the* ***largest positive*** *integer that is a divisor of both numbers*

— Wikipedia

Python is wrong, then?

> *If either a or b is nonzero, then the* ***absolute value*** *of gcd(a, b) is the largest integer that divides both a and b*

— Python Software Foundation

So, well, yes. Basically.

A fixed GCD

```
def gcd( x, y ):
    return x != 0 and gcd( y % x, x ) or abs( y )
```

With some judicious sprinkling of `abs` in the tests, and our tests pass once again.

So now we can test LCM

(After porting the F# to Python...)

```
@given( integers(), integers() )
    def lcm_is_the_smallest_possible_multiple( x, y )
        assert lcm( x y ) * gcd( x y ) = abs( x * y )
```

We can even say the same thing many times

```
        assert abs( x / ( lcm( x, y ) / y ) ) == gcd( x, y )
        assert abs( y / ( lcm( x, y ) / x ) ) == gcd( x, y )

        assert abs( x / gcd( x, y ) ) == abs( lcm( x, y ) / y )
        assert abs( y / gcd( x, y ) ) == abs( lcm( x, y ) / x )
```

A fixed LCM

```
def lcm( x, y ):
    return x != 0 and abs( int( x / gcd( x, y ) ) ) * y or 0
```

```
let lcm x y =
        match x with
        | 0 -> 0
        | _ -> x / ( gcd x y ) * y |> abs
```

A note on filters

```
@given( integers(), integers() )
def test_lcm_is_least_possible_over_positive_x_and_y( x, y ):
        assume( x > 0 and y > 0 )
        assert x / ( lcm( x, y ) / y ) == gcd( x, y )
        assert y / ( lcm( x, y ) / x ) == gcd( x, y )
```

vs.

```
positive_ints = integers( min_value=0 )

@given( positive_ints, positive_ints )
def test_lcm_is_least_possible_over_positive_x_and_y( x, y ):
        assume( x != 0 and y != 0 )
        ...
```

Some property patterns

Choosing properties

I go, I come back

```
assert x / ( lcm( x, y ) / y ) == gcd( x, y )
```

- List reversal
- Deserialisation

You take the high road, I'll take the low road

```
def test_no_smaller_lcm_can_be_found( x, y ):
    result = lcm( x, y )
    start = max( x, y ) + 1

    assert not any( ch % x == 0 and ch % y == 0
        for ch in range( start, result ) )
```

- Searching for an element

Status Quo

```
assert gcd( x, x ) == abs( x )
```

- Upper-cased string length
- New sorted list

Once and only once

- Idempotency
- Asking a question should not change the answer
- Finding (unintended) state

Break it down

Prove for a smaller thing

- Tree searching
- Zero sum

The Proof is in the pudding

- The test requires an implementation of the algo to check the algo worked
- Hard to prove, simple to check
- List sorting

Golden Source

- The test Oracle
- Known good (slow?) implementation

Brave new world?

- Finds the edge cases
- Explores the unhappy paths
- Identifies simple test case failures

i.e. all the things programmers hate to do

Examples vs. Samples

Property Based Testing

It’s not just fuzzing
…but don’t get hung up on it

Throwing random input data at code can be an effective way to flush out bugs

Thank you

@IAmSteveLove
steve@arventech.com
https://perfectcobalt.blogspot.co.uk/