Catching the Bugs You're Missing

Catching SOME OF THE Bugs You're Missing

Before
We Start

```
function(x) {
  return x + 1
}
```

```
(x) => {
  return (x + 1)
}
```

```
(x) => (
  x + 1
)
```

Kinds of Testing

Example-Based Testing

+

```
assert.equal(1 + 2, 3)
```

```
1 + 2 === 3
```

```
1 + 2 === 3
```

```
1 + 2 === 3
```

```
add(1,2) === 3
add(1,2) === 3
add(2,1) === 3
add(1,2) === 3
add(2,1) === 3
add(1,0) === 1
add(1,2) === 3
add(2,1) === 3
add(1,0) === 1
add(0,1) === 1
add(1,2) === 3
add(2,1) === 3
add(1,0) === 1
add(0,1) === 1
add(-1,0) === -1
add(1,2) === 3
add(2,1) === 3
add(1,0) === 1
add(0,1) === 1
add(-1,0) === -1
add(0,-1) === -1
```

Property-Based Testing

Given two inputs:
number, number
(any two numbers)

```
  add(x,y) === add(y,x)
```

```
jsv.property(
  "has swappable args",
  number, number,
  (x,y) => (
    add(x,y) === add(y,x)
  )
)
```

Given one input:
number (any number)

```
add(x,0) === x
```

```
jsv.property(
  "has a do-nothing value",
  number,
  (x) => (
    add(x,0) === x
  )
)
```

Given one input:
number (any number)

```
add(x,x) === x * 2
```

```
jsv.property(
  "matches multiplication",
  number,
  (x) => (
    add(x,x) === x * 2
  )
)
```

```
add(x,y) == add(y,x)

add(x,0) == x

add(x,x) == x * 2
```

```
var jsv = require('jsverify')
  , number = jsv.number()
jsv.property(
  "matches multiplication",
  number,
  (x) => (
    add(x,x) == x * 2
  )
)
```

```
var jsv = require('jsverify')
  , number = jsv.number()
jsv.property(
  "matches multiplication",
  number,
  (x) => (
    add(x,x) == x * 2
  )
)
```

One More Example

```
jsv.property(
  "concatenation",
  jsonVal, //1,"a",[1],{}…
  (x) => (_.eq(
    [1,2].concat(x),
    [1,2,x]
  ))
)
```

```
jsv.property(
  "concatenation",
  jsonVal, //1,"a",[1],{}…
  (x) => (_.eq(
    [1,2].concat(x),
    [1,2,x]
  ))
)
```

💥

```
1) concatenation

0 passing (13ms)
1 failing

1) concatenation:
  Error: Failed after 3 tests
         and 5 shrinks.
  rngState: 009e47bcf23a8651d0;
  Counterexample: [];
```

```
1) concatenation

0 passing (13ms)
1 failing

1) concatenation:
  Error: Failed after 3 tests
         and 5 shrinks.
  rngState: 009e47bcf23a8651d0;
  Counterexample: [];
```

```
[ {}, {}, { '': 25, 'Ìªñ': 'þÿz' } ]
[ {}, { '': 25, 'Ìªñ': 'þÿz' } ]
[ { '': 25, 'Ìªñ': 'þÿz' } ]
[ { 'Ìªñ': 'þÿz' } ]
[ {} ]
[ ]

1) concatenation:
   Counterexample: [];
```

```
[ {}, {}, { '': 25, 'Ìᵅñ': 'þÿz' } ]
[ {}, { '': 25, 'Ìᵅñ': 'þÿz' } ]
[ { '': 25, 'Ìᵅñ': 'þÿz' } ]
[ { 'Ìᵅñ': 'þÿz' } ]
[ {} ]
[ ]

1) concatenation:
  Counterexample: [];
```

```
[ {}, {}, { '': 25, 'Ìᵃñ': 'þÿz' } ]
[ {}, { '': 25, 'Ìᵃñ': 'þÿz' } ]
[ { '': 25, 'Ìᵃñ': 'þÿz' } ]
[ { 'Ìᵃñ': 'þÿz' } ]
[ {} ]
[ ]

1) concatenation:
  Counterexample: [];
```

```
[ {}, {}, { '': 25, 'Ìªñ': 'þÿz' } ]
[ {}, { '': 25, 'Ìªñ': 'þÿz' } ]
[ { '': 25, 'Ìªñ': 'þÿz' } ]
[ { 'Ìªñ': 'þÿz' } ]
[ {} ]
[ ]

1) concatenation:
  Counterexample: [];
```

```
[ {}, {}, { '': 25, 'Ìᵃñ': 'þÿz' } ]
[ {}, { '': 25, 'Ìᵃñ': 'þÿz' } ]
[ { '': 25, 'Ìᵃñ': 'þÿz' } ]
[ { 'Ìᵃñ': 'þÿz' } ]
[ {} ]
[ ]

1) concatenation:
  Counterexample: [];
```

```
[ {}, {}, { '': 25, 'Ìªñ': 'þÿz' } ]
[ {}, { '': 25, 'Ìªñ': 'þÿz' } ]
[ { '': 25, 'Ìªñ': 'þÿz' } ]
[ { 'Ìªñ': 'þÿz' } ]
[ {} ]
[ ]

1) concatenation:
  Counterexample: [];
```

```
[ {}, {}, { '': 25, 'Ìªñ': 'þÿz' } ]
[ {}, { '': 25, 'Ìªñ': 'þÿz' } ]
[ { '': 25, 'Ìªñ': 'þÿz' } ]
[ { 'Ìªñ': 'þÿz' } ]
[ {} ]
[ ]

1) concatenation:
  Counterexample: [];
```

```
jsv.property(
  "concatenation",
  jsonVal, //1,"a",[1],{}…
  (x) => (_.eq(
    [1,2].concat(x),
    [1,2,x]
  ))
)
```

```
jsv.property(
  "concatenation",
  jsonVal, //1,"a",[1],{}…
  (x) => (_.eq(
    [1,2].concat(x),
    [1,2,x]
  ))
)
```

```
jsv.property(
  "concatenation",
  jsonVal, //1,"a",[1],{}…
  (x) => (_.eq(
    [1,2].concat(3),
    [1,2,3]
  ))
)
```

```
jsv.property(
  "concatenation",
  jsonVal, //1,"a",[1],{}…
  (x) => (_.eq(
    [1,2].concat([3]),
    [1,2,[3]]
  ))
)
```

```
jsv.property(
  "concatenation",
  jsonVal, //1,"a",[1],{}…
  (x) => (_.eq(
    [1,2].concat([3]),
    [1,2,3]
  ))
)
```

```
jsv.property(
  "concatenation",
  jsonVal, //1,"a",[1],{}…
  (x) => (_.eq(
    [1,2].concat([]),
    [1,2,[]]
  ))
)
```

```
jsv.property(
  "concatenation",
  jsonVal, //1,"a",[1],{}…
  (x) => (_.eq(
    [1,2].concat([]),
    [1,2]
  ))
)
```

Property-Based Testing in Three Steps

1) Stating a rule, eg.

```
add(x,0) === x
```

2) Generating data fitting a specific shape, eg.

```
number, (x) => (
  // ...
)
```

3) Testing the rule holds with that generated data, eg.

```
number, (x)  => (
  add(x, 0) === x
)
```

3) Testing the rule holds with that generated data, eg.

```
number, (1)  => (
  add(1, 0) === 1
)
```

3) Testing the rule holds with that generated data, eg.

```
number, (92) => (
  add(92,0) === 92
)
```

3) Testing the rule holds with that generated data, eg.

```
number, (-5) => (
  add(-5,0) === -5
)
```

Why?

Finding Edge-Cases

Honest TDD

No fudging the code to pass an anaemic test.

Why Not?

Coming up with Properties can be hard.

Kinds of Properties You Can Write

Reversible

```
n + 1 - 1 === n
```

Reversible

```
n + 1 - 1 === n
```

```
x.split(" ").join(" ") === x
```

Reversible

```
n + 1 - 1 === n
```

```
x.split(" ").join(" ") === x
```

```
_.eq(
 zip.decompress(zip.compress(x)), x
)
```

Reversible

```
n + 1 - 1 === n
```

```
x.split(" ").join(" ") === x
```

```
_.eq(
 zip.decompress(zip.compress(x)), x
)
```

```
_.eq(
 xFromJson(xToJson(x)), x
)
```

Repeatable

```
_.eq(
  sort(sort(list)),
  sort(list)
)
```

Invariants

```
sort(list).length
  === list.length
```

Invariants

```
sort(list).length
  === list.length

_.all(
  sort(list),
  (x) => (
    _.contains(list, x)
  )
)
```

Prove a Small Part

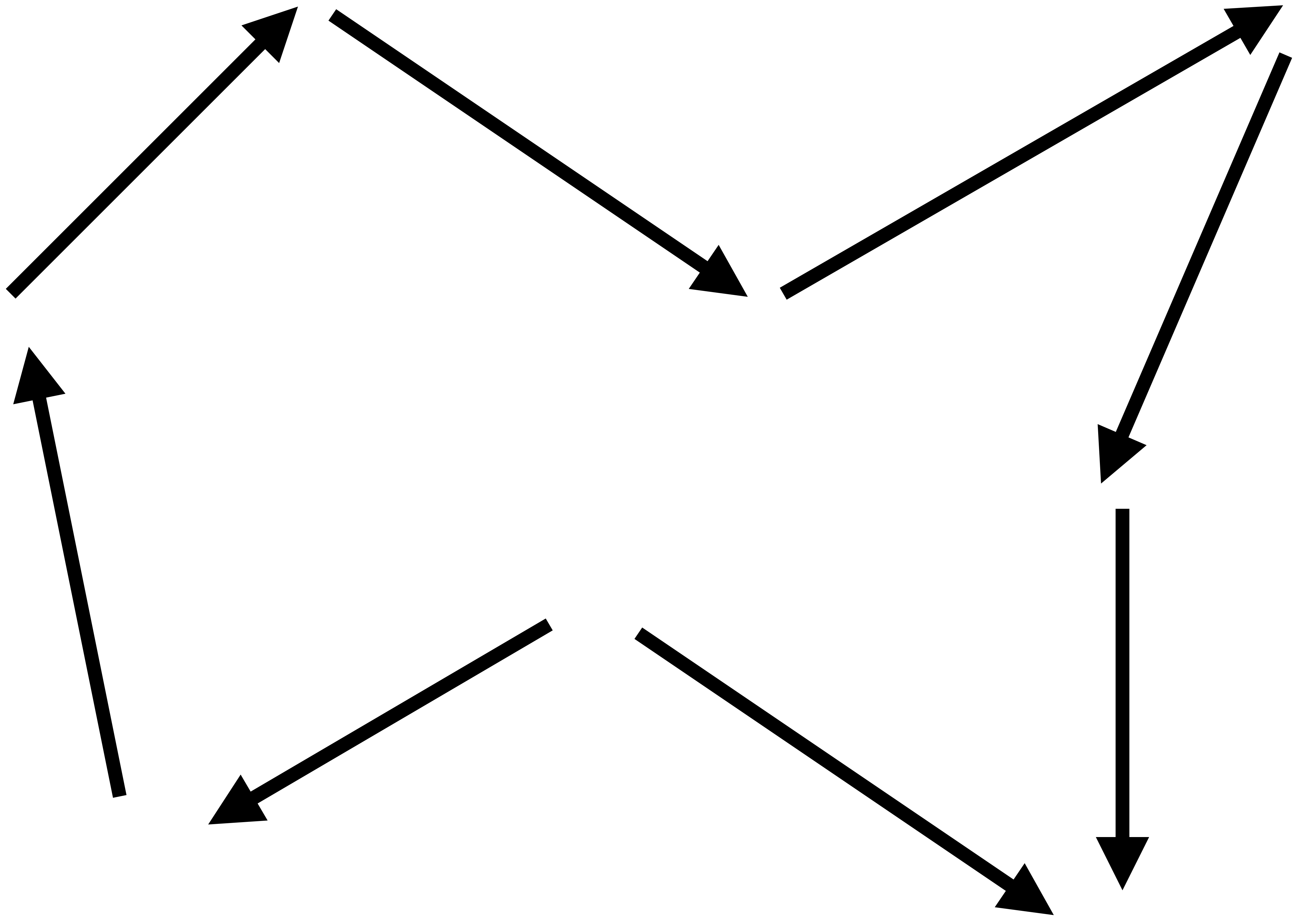
```
let sorted = sort(list)
_.all(
  toPairs(sorted),
  function(pair) {
    return (pair[0] <=
pair[1])
  }
)

// toPairs([1,2,3])
// => [[1,2], [2,3]]
```

Swap the Ordering

```
_.eq(
  sort(list).map(x => x + 1),
  sort(list.map(x => x + 1))
)
```

Hard to Solve, Easy to Check



Consult an Oracle

```
_.eq(
  sort(list),
  ultraCoolSort(list)
)
```

Consult an Oracle

```
_.eq(
  sort(list),
  ultraCoolSort(list)
)
```

```
newCode(input)
     ===
oldCode(input)
```

A Regular Test with a Hole

```
property(…, user, (u) => (
  createTestUser(u)
   .then((u) => (
     page.login(u)
   ))
   .then((r) => {
     assertLocation(r, '/account')
     return page.logout()
   }).then((r) => {
     assertStatus(r, 302)
     assertLocation(r, '/')
   })
))
```

A Regular Test with a Hole

```
property(…, user, (u) => (
  createTestUser(u)
   .then((u) => (
     page.login(u)
   ))
   .then((r) => {
     assertLocation(r, '/account')
     return page.logout()
   }).then((r) => {
     assertStatus(r, 302)
     assertLocation(r, '/')
   })
))
```

Mathsy

```
add(x,0) === x
/* Operation w/ Identity */
```

Mathsy

```
add(x,0) === x
/* Operation w/ Identity */

add(x,y) === add(y,z)
/* Commutative */
```

Mathsy

```
add(x,0) === x
/* Operation w/ Identity */

add(x,y) === add(y,z)
/* Commutative */

add(add(x,y), z)
===
add(x, add(y,z))
/* Associative */
```

Generating Data

Data Generation

Existing Generators

```
jsv
  ….number
  ….string
  ….boolean
  ….json
  ….array(…)
  ….nearray(…) // non-empty
  ….dict(…) // object
  …
```

Utilities

```
jsv
 ….oneof([number, string, …])
 ….constant(undefined)
 ….constant(6) // or whatever
 ….recursive(…)
    // ^-- used to make .json
 …
```

BYO

```
var whatever = jsc.bless({
  generator: function () {
    switch (jsc.random(0, 2)) {
      case 0: return "foo";
      case 1: return "bar";
      case 2: return "quux";
    }
  }
});
```

Examples
Doing
Real Things

Oracle Check

```
prop_ifBool v = compareHelpers
  [("val", Handlebars.Bool v)]
  "{{~#if val~}}
     True
   {{~else~}}
     False
   {{~/if~}}"
  (if v then "True" else "False")
```

Reversible Checks

```
prop_roundTripDayOfWeek :: DayOfWeek -> Property
prop_roundTripDayOfWeek d =
  (dayOfWeekFromInt . dayOfWeekToInt) d === is d

prop_roundTripNextMonth :: Date -> Bool
prop_roundTripNextMonth m =
  (prevMonth . nextMonth) m == m &&
  (nextMonth . prevMonth) m == m

prop_roundTripNextDay :: Date -> Bool
prop_roundTripNextDay d =
  (nextDay . prevDay) d == d &&
  (prevDay . nextDay) d == d
```

Invariant

```
prop_withinRange =
  forAll boundedInt $ \seed ->
    forAll boundedInt $ \lo ->
      forAll boundedInt $ \hi ->
        let
          gen = Random (nextInt lo hi)
          num = fst (runRandom (Seed seed) gen)
        in
          property $ num >= low && num <= high
```

Invariant

```
prop_withinRange =
  forAll boundedInt $ \seed ->
    forAll boundedInt $ \lo ->
      forAll boundedInt $ \hi ->
        let
          gen = Random (nextInt lo hi)
          num = fst (runRandom (Seed seed) gen)
        in
          property $ num >= low && num <= high
```

With
Real Bugs

Invariant

```
prop_withinRange =
  forAll boundedInt $ \seed ->
    forAll boundedInt $ \lo ->
      forAll boundedInt $ \hi ->
        let
          gen = Random (nextInt lo hi)
          num = fst (runRandom (Seed seed) gen)
        in
          property $ num >= low && num <= high
```

Invariant

```
=== prop_withinRange from BadRandom ===
*** Failed! Falsifiable (after 1 test
    and 90 shrinks) with data:

  between: [0, 642563584]
  result: -9394
```

Invariant

```
=== prop_withinRange from BadRandom ===
*** Failed! Falsifiable (after 1 test
    and 90 shrinks) with data:

  between: [0, 642563584]
  result: -9394
```

Round-Tripping

```
it "can round-trip timestamps" do
  property_of {
    (Time.current - float.abs)
  }.check { |time|
    user = create(User, login_at: time)
    expect(
      User.find(user.id).login_at
    ).to eq(time)
  }
end
```

Round-Tripping

```
1) can round-trip timestamps
 Failure/Error:
   expect(User.find(user.id).login_at)
     .to eq(time)

expected: 2015-06-13 04:39:52.835645641 +0000
     got: 2015-06-13 04:39:52.835645000 +0000
```

Round-Tripping

```
1) can round-trip timestamps
 Failure/Error:
   expect(User.find(user.id).login_at)
     .to eq(time)

expected: 2015-06-13 04:39:52.835645641 +0000
     got: 2015-06-13 04:39:52.835645000 +0000
```

Round-Tripping

```
property_of { char, integer }.check { |char,size|
  file = File.join(tmpdir, "testfile-#{size}.bin")
  zip  = File.join(tmpdir, "testfile-#{size}.zip")

  data_write = char * size  # size-length string, all char.
  filename   = char * size

  File.open(file, 'wb') { |f| f.write(data_write) }
  Zip::File.open(zip, CREATE) {|f| f.add(filename, file) }

  data_read = nil
  Zip::File.open(zip) {|f|
    data_read = f.first.get_input_stream.read
  }

  expect(data_write).to == data_read
}
```

Round-Tripping

```
Size: 65535 - Gen'd, Written,
Zipped, Unzipped. Written data
equals read data.
```

```
Size: 65536 - Gen'd, Written,
Zipped, /Users/rhoward/code/
experiments/p7zip/rubyzip/lib/
zip/inflater.rb:44:in `inflate':
invalid stored block lengths
(Zlib::DataError)
```

Round-Tripping

```
$ 7z x testfile-65536.zip
7-Zip [64] …

Processing archive: testfile-65536.zip

Errors: Headers Error
Errors: Unconfirmed start of archive
Warnings: There are data after the end of
archive

Extracting testfile-65536: Segmentation fault
```

Round-Tripping

```
$ 7z x testfile-65536.zip
7-Zip [64] …

Processing archive: testfile-65536.zip

Errors: Headers Error
Errors: Unconfirmed start of archive
Warnings: There are data after the end of
archive

Extracting testfile-65536: Segmentation fault
```

One Last Thing

ARMY
05
I HAVE NO IDEA
WHAT I'M DOING

Credits

- **fsharpforfunandprofit.com** (Property-based testing posts)
- **github.com/charleso/property-testing-preso** (Lambda Jam 2015)
- **jsverify.github.io** (JS)
- **Rantly** (Ruby)
- **QuickCheck** (Haskell)
- **Hypothesis** (Python)
- **Jack** (Haskell, PureScript, F#, hopefully JS after Railscamp...)

Catching THE Bugs You're Missing

jsverify.github.io

(or QuickCheck, Jack, Rantly, ...)

robhoward.id.au
@damncabbage