

INTRODUCTION TO PROPERTY TESTING IN JAVASCRIPT

property testing in plain-old-JS



What is a Property Test?

- **Example-based tests:** a particular input produces a particular output
- **Generative property tests:** for all acceptable inputs, the output has some property
- Works really well with a good type system
- Similar in denotation to contracts
- Examples:
 - *Self-inverse:* `reverse(reverse(list)) == list`
 - *Idempotency:* `sort(sort(list)) == sort(list)`
 - *Identity:* `0 + number == number`

Prior Art

- Haskell's QuickCheck
 - Originated
- `clojure.spec`
 - Popularised
- PureScript's StrongCheck
 - exhaustive and statistical tests
- PureScript's Jack
 - shrinking with provenance

Built-in Generators

```
import { types as t } from 'gentest';
```

- t.int
- t.int.nonNegative
- t.int.nonZero
- t.int.positive
- t.char
- t.string
- t.bool

Sampling a Generator

```
import { sample, types as t } from 'gentest';
```

```
sample(t.int)
```

- [1, 0, -2, -2, -3, 3, 4, 2, -1, -1]

```
sample(t.bool, 2)
```

- [true, false]

```
sample(t.char)
```

- ['A', 'y', 'n', 'J', ' ', 'q', 'D', 'M', 'P', '6']

```
import { encode, decode } from '../';
import { sample, types as t } from 'gentest';
const SAMPLE_SIZE = 100;

describe('integers should always encode to the same value', () => {
  gentest.sample( t.int , SAMPLE_SIZE).forEach(a => {
    it(`should deterministically encode ${a}`, () => {
      expect( encode(a) ).to.eql( encode(a) );
    });
  });
});
```

```
import { encode, decode } from '../';
import { sample, types as t } from 'gentest';
const SAMPLE_SIZE = 100;

describe('integers should always encode to the same value', () => {
  gentest.sample(t.int, SAMPLE_SIZE).forEach(a => {
    it(`should deterministically encode ${a}`, () => {
      expect(encode(a)).toEqual(encode(a));
    });
  });
});
```

```
import { encode, decode } from '../';
import { sample, types as t } from 'gentest';
const SAMPLE_SIZE = 100;

describe('integers should round-trip', () => {
  gentest.sample(t.int, SAMPLE_SIZE).forEach(a => {
    it(`should round-trip ${a}`, () => {
      expect(decode(encode(a))).toEqual(a);
    });
  });
});
```


Functions Which Produce Generators

```
sample(t.elements(['a', 'b', 'c']))
```

- ['b', 'c', 'b', 'b', 'b', 'a', 'a', 'b', 'b', 'a']

```
sample(t.elements([true, false]), 4)
```

- [true, false, false, false]

```
sample(t.elements([1]))
```

- [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

```
sample(t.constantly(1))
```

- [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

Generator Combinators

```
sample(t.arrayOf(t.int))
```

- ```
[[],
 [-1],
 [1, 1],
 [0],
 [1, 1],
 [0],
 [2, -2, -1],
 [],
 [-3, 2],
 [-5, -5, 0]]
```

# Generator Combinators

`sample(t.arrayOf(t.bool))`

- `[ [ true ],  
 [],  
 [ true ],  
 [ false, false ],  
 [ false, true ],  
 [ false, false ],  
 [ false, false ],  
 [ true, false, true ],  
 [ false, false, false ],  
 [ false ] ]`

# Generator Combinators

```
sample(t.arrayOf(t.arrayOf(t.int)))
```

- ```
[ [],  
  [],  
  [ [] ],  
  [ [ 0 ], [ -2 ] ],  
  [ [ -2, -1 ] ],  
  [ [ -1, 3, 0 ], [ -2 ] ],  
  [ [ -2, 3, -2, 1 ] ],  
  [ [], [ 0, 2, 3, -4 ], [ 3, 4, -1 ], [ 3, 3, 4, 3 ] ],  
  [ [ 3 ], [ 0 ], [ 3, -1, -3, -1, 4 ], [ 2, -4 ], [ 0 ] ],  
  [ [ -5, -2, 1 ], [ 4, -4, -2, 4 ], [ -2 ], [ ] ] ]
```

```
import { encode, decode } from '../';
import { sample, types as t } from 'gentest';
const SAMPLE_SIZE = 100;

describe('arrays should preserve length through a round-trip', () => {
  gentest.sample(t.arrayOf(t.int), SAMPLE_SIZE).forEach(a => {
    it(`should preserve length of ${JSON.stringify(a)}`, () => {
      expect(decode(encode(a)).length).toEqual(a.length);
    });
  });
});
```

Generator Combinators

```
sample(t.oneOf([t.int, t.bool]))
```

- [true, true, true, -2, -2, false, -1, 2, 3, false]

```
sample(t.oneOf([t.constantly(0), t.constantly(1)]))
```

- [0, 1, 1, 0, 0, 1, 1, 1, 0, 1]

```
sample(t.shape({x: t.int, y: t.int}), 6)
```

- [{ x: -1, y: 1 },
 { x: -1, y: -1 },
 { x: -2, y: -2 },
 { x: 1, y: -1 },
 { x: 2, y: 0 },
 { x: 2, y: 3 }]

Generator Combinators

```
sample(t.suchThat(a => a.length > 0, t.arrayOf(t.bool)), 5)
```

- [[false],
[false],
[false, false],
[true],
[false, true, true]]

```
sample(t.suchThat(a => (a & 1) == 0, t.int))
```

- [0, 0, 2, -2, -2, 2, 4, 4]

fmap

```
sample(t.fmap(a => a * 2, t.int))
```

- [0, 2, 0, 4, -2, 0, 6, 2, -10, 4]

```
sample(t.fmap(a => (a * 2) + 1, t.int))
```

- [-1, -1, -3, 5, -3, 1, 5, -5, 11, 3]

```
sample(t.fmap(a => a.toString(), t.int))
```

- ['-1', '0', '2', '0', '2', '2', '-1', '4', '4', '5']


```
import { encode, decode } from '../';
import { sample, types as t } from 'gentest';
const SAMPLE_SIZE = 100;

function isUint(typeTag) { /* implementation omitted */ }

describe('non-negative integers should be encoded as uints', () => {
  let generator = t.fmap(Math.abs, t.int);
  gentest.sample(generator, SAMPLE_SIZE).forEach(a => {
    it(`should encode ${a} as a uint`, () => {
      expect(isUint(encode(a))).to.be.true;
    });
  });
});
```

bind

```
fmap :: (a -> b)           -> Generator a -> Generator b
```

```
bind :: (a -> Generator b) -> Generator a -> Generator b
```

bind

```
function cons(head, arrayGen) {  
  return t.fmap(tail => [head].concat(tail), arrayGen);  
}
```

```
function nonEmptyArrayOf(gen) {  
  return t.bind(gen, el => cons(el, t.arrayOf(gen)));  
}
```

```
sample(nonEmptyArrayOf(t.int), 4)
```

- [[0], [0, -1], [1, -2, 0], [-1, 2, -2]]

Areas for Improvement

- Use ES2015+ features such as generators
- Shrinking
- Jack-style provenance-based shrinking
- Integration with type checker such as Flow
- Suggestions for fixing violations

<> Code

! Issues 10

🔗 Pull requests 0

📁 Projects 0

⚡ Pulse

📊 Graphs

Generative testing for JavaScript. Save time and catch more bugs by letting the computer write test cases for you. WIP

📄 45 commits

🌿 1 branch

🏷 2 releases

👤 1 contributor

📄 MIT

📖 README.md

gentest

Property-based, generative testing for JavaScript.

Don't handwrite unit tests. Save time and catch more bugs by writing properties, and let the computer generate test cases for you!

(This is a work in progress. Consider it "[Stability 1: Experimental](#)" for the time being. Feedback welcome.)

Basic example

That's it!



michaelficarra



jspedant



justgrahamthings