Unlocking the Power of Property-Based Testing

Merlin Pahič

EIDU

Example-based testing (conventional unit testing)

Unit tests are commonly written this way.

Programmer defines expected outputs/behaviour for specific inputs.

Challenging to come up with examples covering every edge case.

```
def test_reversing_string():
    assert reverse("abc") == "cba"
def test_reversing_empty_string():
    assert reverse("") == ""
```

EIDU

What is Property-Based Testing?

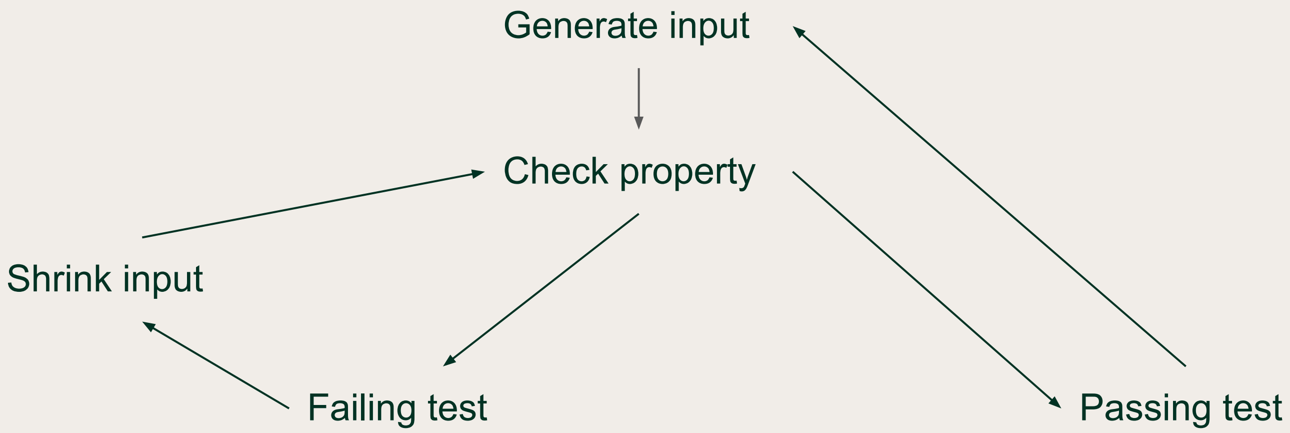
Programmers specify properties: general truths or invariants that should hold for all inputs, or under certain conditions.

Inputs are automatically generated – tests run hundreds or thousands of times.

```
@given(strategies.text())
def test_reversing_all_the_strings(s):
    assert reverse(reverse(s)) == s
```

Running a property-based test

When a test fails, the input used may be simplified (“shrunk”) to ease debugging.



EIDU

Input generation

Inputs are generated randomly but not uniformly to cover edge cases.

Specify required inputs as “generators” or arbitraries”,
e.g. string, real number, list of integers

Generated values include typical edge cases:

- Empty string, non-ASCII characters
- 0, 1, -1, ± Infinity, NaN
- Lists of various lengths, including duplicate items.

Generators can be combined to produce more complex data.

Each test run uses a specific random seed to allow reproducing any failures.

Properties

We can consider properties as invariants.

- What holds true across (a class of) inputs?
- What does not change after applying the function under test?

But anything can be a property.

```
for all (x, y, …)
[satisfying precondition(x, y, …)]
predicate(x, y, …) evaluates to true
```

E.g. given two strings (or lists) a and b, `len(a + b) == len(a) + len(b)`

Finding properties

Distributivity: `len(a + b) == len(a) + len(b)` (many other transformations)

Commutativity: `a + b == b + a`

- `image.flipX().flipY() == image.flipY().flipX()`
- `data.filter(…).sort() == data.sort().filter(…)`

Associativity: `(a + b) + c == a + (b + c)`

Invertible functions

- reversing a string, flipping an image
- serialization, encryption, lossless compression, undo/redo

Idempotency: `sort(x) == sort(sort(x))`

Libraries are available for most languages

Language	**Property-based testing library**	**License**
JavaScript/TypeScript	fast-check	MIT
Python	Hypothesis	MPL 2.0
Rust	Proptest	Apache 2.0, MIT
Go	Rapid	MPL 2.0
PHP	Eris	MIT
C#/.NET	FsCheck	BSD-3-Clause
Java/JUnit5	jqwik	EPL 2.0
Scala	ScalaCheck	BSD-3-Clause
Kotlin	Kotest-property	Apache 2.0
C	theft	ISC
C++	RapidCheck	BSD-2-Clause
Swift	SwiftCheck	MIT
Haskell	QuickCheck	BSD-3-Clause
Elm	elm-explorations/test	BSD-3-Clause

EIDU

Thank you!

Thanks for listening.

Now’s the time for any questions you may have.

EIDU