Property Based Testing

An introduction to lightweight formal methods.

Benedikt Maderbacher

About Me

Formal methods researcher and PhD student at TU Graz.

I am working on synthesis and am co-teaching a Masters course on Verification and Testing.

Functional programming enthusiast.

Software Correctness

What does it mean for software to be correct?

What does it mean for software to be correct?

- It does what we want it to do.

What does it mean for software to be correct?

- It does what we want it to do.
- It does not crash.
- It matches examples we have in mind.
- It satisfies some properties.
- It conforms to an existing reference.

How to check software is correct?

How to check software is correct?

- Run it and compare outputs with our intuition.
- Write automatic tests.
- Explore exhaustively.
- Prove using mathematics.

Where to get test inputs?

Where to get test inputs?

- What ever came to your mind first.
- Common use cases.
- Corner cases.
- Random data.

Hypothesis Tutorial

Hypothesis

- Python library for property based testing.
- Generates random test inputs.
- Inspired by QuickCheck.

Run Length Encoding

```
def encode(input_string):
    count = 1
    prev = ""
    lst = []
    for character in input_string:
        if character != prev:
            if prev:
                entry = (prev, count)
                lst.append(entry)
            count = 1
            prev = character
        else:
            count += 1
    entry = (character, count)
    lst.append(entry)
    return lst
```

Run Length Encoding

```
def decode(lst):
    q = ""
    for character, count in lst:
        q += character * count
    return q
```

Unit Testing

```
def test_encode():
  assert encode('1111566') == [(1,4),(5,1),(6,2)]

def test_decode():
  assert decode([(1,4),(5,1),(6,2)]) == '1111566'
```

A property test

```python
from hypothesis import given
from hypothesis.strategies import text


@given(text())
def test_decode_inverts_encode(s):
    assert decode(encode(s)) == s
```

A property test

```
from hypothesis import given
from hypothesis.strategies import text


@given(text())
def test_decode_inverts_encode(s):
    assert decode(encode(s)) == s


Falsifying example: test_decode_inverts_encode(s='')

UnboundLocalError: local variable 'character'
referenced before assignment
```

Bug fix

```
def encode(input_string):
    if not input_string:
        return []
    ...
```

Adding examples

```python
@given(text())
@example("")
def test_decode_inverts_encode(s):
    assert decode(encode(s)) == s
```

Excluding ”forbidden” cases

```
@given(text())
def test_decode_inverts_encode(s):
    assume(s != "")
    assert decode(encode(s)) == s
```

Another error

```
def encode(input_string):
    if not input_string: return []
    count = 1
    prev = ""
    lst = []
    for character in input_string:
        if character != prev:
            if prev:
                entry = (prev, count)
                lst.append(entry)
            # count = 1  # Missing reset operation
            prev = character
        else:
            count += 1
    entry = (character, count)
    lst.append(entry)
    return lst
```

Random Tests and Shrinking

```
@given(text())
@example("")
def test_decode_inverts_encode(s):
    assert decode(encode(s)) == s
```

Random Tests and Shrinking

```
@given(text())
@example("")
def test_decode_inverts_encode(s):
    assert decode(encode(s)) == s
```

```
Falsifying example: test_decode_inverts_encode(s='001')
```

Note

The tester found the smallest failing input.

More Property Based Testing

Strategies ...

- ... are used in a @given decorator.
- ... generate random inputs of a certain type.
- ... provide a way to shrink/minimize test cases.
- ... can be combined to build more complex strategies.

Example Strategies

```
booleans()

integers(0,100)

text(printable)

emails()

dictonaries(keys=integers(), values=text())

lists(integers() | text() | booleans())
```

Composite Strategies

```
@composite
def list_and_index(draw, elements=integers()):
    xs = draw(lists(elements, min_size=1))
    i = draw(integers(min_value=0,
                      max_value=len(xs) - 1))
    return (xs, i)
```

How To Find Properties

Generalized unit tests

Replace dummy data with randomly generated data.

Fuzzing

Does it crash?

Round trip properties

Serialize/deserialize, insert/extract, etc. must be compatible.

Models

Compare with a reference implementation.

Stateful Testing

Most programs are not simple functions, they have state.

Stateful Testing

Most programs are not simple functions, they have state.

- Generate random sequences of interactions.
- Check for crashes, invariants and other properties.
- Possible interactions are described as a state machine.
- Can be applied to complex systems.

A heap class

```
def heapnew():
    ...

def heapempty(heap):
    ...

def heappush(heap, value):
    ...

def heappop(heap):
    ...
```

Testing a heap class

```
class HeapMachine(RuleBasedStateMachine):
    def __init__(self):
        super(HeapMachine, self).__init__()
        self.heap = []

    @rule(value=integers())
    def push(self, value):
        heappush(self.heap, value)

    @rule()
    @precondition(lambda self: self.heap)
    def pop(self):
        correct = min(self.heap)
        result = heappop(self.heap)
        assert correct == result
```

A discovered bug:

```
E       AssertionError: assert 0 == 1

binheap.py:90: AssertionError
----- Captured stdout call -----
Step #1: push(value=1)
Step #2: push(value=0)
Step #3: push(value=0)
Step #4: pop()
Step #5: pop()
```

Bug

The heap is not rebalanced after pop!

Extend your tests by sparkling in some randomness.

- Hypothesis is a python library for property based testing.
- Similar libraries are available for many languages (QuickCheck, ScalaCheck, FsCheck)
- Try it on your own code: start with fuzzing or by generalizing your unit tests.

References i

D. R. MacIver.
Rule based stateful testing.
https://hypothesis.works/articles/rule-based-stateful-testing/, 2016.

D. R. MacIver.
Hypothesis quickstart guide.
https://hypothesis.readthedocs.io/en/latest/quickstart.html, 2021.

S. Wlaschin.
Choosing properties for property-based testing.
https://fsharpforfunandprofit.com/posts/property-based-testing-2/, 2014.