Let's start with a disclaimer
Property based testing in Elixir

Tonći Galić / @Tuxified
@tuxified@mastodon.social
A little intro

- Tonći Galić / @Tuxified
- Live in 🇳🇱 (near Amsterdam)
- Family 🐶🐱🐱🐱🐱🐱
- Like (computer) languages 😊
- Like doing sports (urances...)

Sometimes feel 🧑‍🦰
Makes you think harder, not more

All kinds of cases, often edge cases

Combination of features (hence tests) possible

Once an issue is found, search for minimal input

Generate

Shrinking

Complexity

Rewires

TL;DL:

🧠 Complexity

Generate

Shrinking

Complexity

Rewires

Makes you think harder, not more
Table of contents

01 Why
   This talk

02 Intro
   Into property based testing (PBT)

03 Examples
   To get a feeling what it’s useful for

04 Conclusion
   When should/shouldn’t you resort to PBT
Unit testing

A.k.a example testing
Why talk about Unit testing?

Good
Testing is good as it gives us confidence, prevents disasters and helps drive design

Boring
Coming up with good examples for our tests is boring and tedious

Hard
How many tests should we write? How will our test suite grow as we add features
Write tests for the plus operator (Kernel.+/2)
Write tests for the plus operator (Kernel.+/2)

```ruby
1   test "addition works fine" do
2     assert 2 + 2 == 4
3     refute 2 + 2 == 5
4     assert 2 + 2.0 == 4
5   end
```
Testing can be boring
<table>
<thead>
<tr>
<th>N features</th>
<th>O(n) tests</th>
<th>No problem</th>
</tr>
</thead>
</table>

How many tests?
How many tests?

<table>
<thead>
<tr>
<th></th>
<th>$O(n)$ tests</th>
<th>No problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>N features</td>
<td>$O(n)$ tests</td>
<td>No problem</td>
</tr>
<tr>
<td>Pairs of N features</td>
<td>$O(n^2)$</td>
<td>This is a step up, but doable</td>
</tr>
</tbody>
</table>
## How many tests?

<table>
<thead>
<tr>
<th></th>
<th>(O(n)) tests</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>N features</td>
<td>No problem</td>
<td></td>
</tr>
<tr>
<td>Pairs of N features</td>
<td>(O(n^2))</td>
<td>This is a step up, but doable</td>
</tr>
<tr>
<td>Triples of N features</td>
<td>(O(n^3))</td>
<td>Starting to get out of hand</td>
</tr>
</tbody>
</table>
### How many tests?

<table>
<thead>
<tr>
<th></th>
<th>O(n) tests</th>
<th>No problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>N features</td>
<td>O(n)</td>
<td>No problem</td>
</tr>
<tr>
<td>Pairs of N features</td>
<td>O(n^2)</td>
<td>This is a step up, but doable</td>
</tr>
<tr>
<td>Triples of N features</td>
<td>O(n^3)</td>
<td>Starting to get out of hand</td>
</tr>
<tr>
<td>M x N features</td>
<td>💥</td>
<td>🔥</td>
</tr>
</tbody>
</table>
Testing can be hard
How to fix?
Property based testing
“Instead of writing examples, we define properties and let the computer come up with cases”

— Folks at Quviq
“Instead of writing examples, we define properties and let the computer come up with cases”
... and some more.

– Folks at Quviq
Write tests for string reversal

test "String.reverse reverses a string" do
  assert String.reverse("FOSDEM") == "MEDSOF"
  refute String.reverse("FOSDEM") == "FOSDEM"
  assert String.reverse("") == ""
end

Does this spark confidence?
Write properties for string reversal
Write properties for string reversal

```ruby
property "reversing a string twice returns original" do
  check all shirt ← string(:ascii) do
    assert String.reverse(String.reverse(shirt)) == shirt
  end
end
```
What are other properties of Any.reverse/1?

- first item becomes last item (for non-empty list)
- last item becomes first item (for non-empty list)
- palindromes stay the same after reversal
- amount of items stay consistent
- ....
Write properties for string reversal

```ruby
property "reversing a string twice returns original" do
  check all shirt ← string(:utf8) do
    assert String.reverse(String.reverse(shirt)) == shirt
  end
end
```
1) property reversing a string twice returns original (PbtExamplesTest)
   test/pbt_examples_test.exs:12
   Failed with generated values (after 8 successful runs):

   * Clause:   shirt <- string(:utf8)
       Generated: "??"

   Assertion with == failed
   code:   assert String.reverse(String.reverse(shirt)) == shirt
   left:   "??"
   right:  "??"
   stacktrace:
      test/pbt_examples_test.exs:14: anonymous fn/2 in PbtExamplesTest."property reversing twice returns original"/1
 (stream_data 0.6.0) lib/stream_data.exs:2367: StreamData.shrink_failure/6
 (stream_data 0.6.0) lib/stream_data.exs:2327: StreamData.check_all/7
    test/pbt_examples_test.exs:13: (test)

...
Edge case found !!
Case shrinked
Examples*
*Use cases
Volvo's AUTOSAR
3000 pages

specification

1,000,000

Lines of vendor code tested

200

Issues found
Klarna’s heisenbug
6 weeks

No result

1 GB

Files

< 3 day

To find issue with QuickCheck
What are other good occasions to use PBT

- symmetrical functions (serialize <-> deserialization)
- mathematical proof
- compare systems
- testing concurrency
Conclusion
Conclusion

**Generate**
All kinds of cases, often edge cases

**Shrinking**
Once an issue is found, search for minimal input

**Complexity**
Combination of features (hence tests) possible

Makes you think harder, not more
Thanks!

Does anyone have any questions?
Tonći Galić / @Tuxified
@tuxified@mastodon.social

CREDITS: This presentation template was created by Slidesgo, including icons by Flaticon, and infographics & images by Freepik

Please keep this slide for attribution
Additional resources

BEAM: PropEr, QuickCheck, StreamData, Triq, etc etc

Haskell: QuickCheck (by Quviq)

Python: https://hypothesis.works/

Book: https://propertesting.com/

Talk: John Hughes - Keynote: How to specify it!
https://www.youtube.com/watch?v=G0NUOst-53U