How to test proper{t,l}y

Lars Hupel
Scala Matsuri
2019-06-29

INNOQ

Basic idea

- specify a property with variables
- let the framework instantiate

パラメータ化された属性を指定して例を自動生成する

Example

```
// * is associative
Prop.forAll { (x: Int, y: Int, z: Int) =>
  (x * y) * z == x * (y * z)
}
```

掛け算の結合法則のテスト

Example

```
// * is associative
Prop.forAll { (x: Int, y: Int, z: Int) =>
  (x * y) * z == x * (y * z)
}

// + OK, passed 100 tests.
```

100 例のテストをパスしたので、OK

Example

```
// * is associative
Prop.forAll { (x: Float, y: Float, z: Float) =>
  (x * y) * z == x * (y * z)
}
```

Float 値の場合

Example

```
// * is associative
Prop.forAll { (x: Float, y: Float, z: Float) =>
  (x * y) * z == x * (y * z)
}

// ! Falsified after 2 passed tests.
// > ARG_0: 1.2072811E-5
// > ARG_1: 1.026218E-4
// > ARG_2: -5.731085E-20
```

2 例のテストをパスした後、偽が証明された

Common features

- automatic and custom generators
- filtering of inputs
- shrinking of counterexamples
- classification of inputs
- input and property labelling

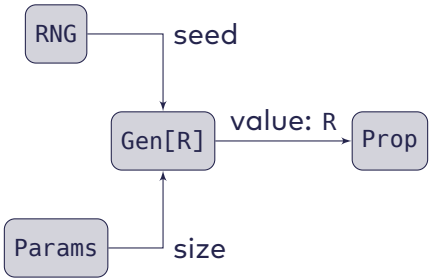
Framework ecosystem

- Haskell
  - QuickCheck
  - SmallCheck
  - Hedgehog
- Scala
  - ScalaCheck
  - scalaprops
- Java
  - Vavr
- Rust
  - QuickCheck
  - proptest
- Erlang
  - QuickCheck
  - triq
- Python
  - pytest-quickcheck
  - Hypothesis
- Clojure
  - simple-check
  - test.check
- your favourite language ...

様々な言語で実装されたフレームワーク

Deep Dive

How do we come up with values?



どのように値を生成しているのでしょうか？

Generating functions

```
def getItemByName(string: String): Future[Int] = ???
```

- assume: this function makes an expensive database access
- bad for testing!
- can we generate a random implementation?

データベースへのアクセスを避けたいので、適当な実装を生成したい

How can we generate functions?

Precompute a table: impractical 😢

事前に計算したテーブルは実用的ではない

How can we generate functions?

Precompute a table: impractical 😢

Make up values on the spot: side-effecting 🤯

その場で生成するのは、副作用が。。。

How can we generate functions?

Precompute a table: impractical 😢

Make up values on the spot: side-effecting 🤯

Constant functions: not good enough 🙃

不変の関数では不十分。。。

✨ Cogen ✨

- function outputs must only depend on its inputs
- when generating a function, inputs are not known yet!

関数の出力は、入力で決まる。関数の生成時、まだ入力は分からない。

✨ Cogen ✨

- function outputs must only depend on its inputs
- when generating a function, inputs are not known yet!
- hack: use inputs to **perturb** random generator

入力を乱数ジェネレーターのシードとしてではなく、

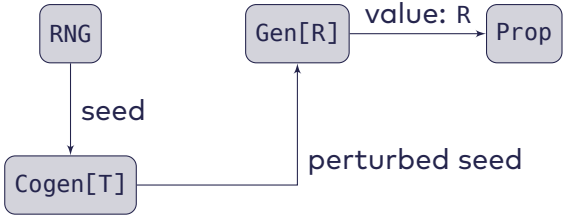
✨ Cogen ✨

- function outputs must only depend on its inputs
- when generating a function, inputs are not known yet!
- hack: use inputs to **perturb** random generator

乱数ジェネレーターを摂動させるために使う

✨ Cogen ✨

- function outputs must only depend on its inputs
- when generating a function, inputs are not known yet!
- hack: use inputs to **perturb** random generator



乱数ジェネレーターを摂動させるために使う

✨ Cogen ✨

- function outputs must only depend on its inputs
- when generating a function, inputs are not known yet!
- hack: use inputs to **perturb** random generator

```
trait Cogen[T] {
  def perturb(seed: Seed, t: T): Seed
}
```

乱数ジェネレーターを摂動させるために使う

Demo

Randomly generate all the things?

Pros

- ⊕ likely to find input you didn't think about
- ⊕ can easily cover wide variety of input

Cons

- ⊖ not reproducible
- ⊖ often high discard ratio
- ⊖ existential properties are difficult

長所: 想定したのことの無い入力が得られる
短所: 再現性に欠ける

Enumerating things

```
trait Enumerable[T] {
  def enumerate(size: Int): Stream[T]
}
```

列挙可能なもの

SmallCheck

Haskell implementation

- supports quantifiers: $\exists$, $\exists_1$, $\forall$
- supports implication
- supports quantifiers nested in implications

Haskell: 量化子、包含をサポート

Demo

What if my properties are polymorphic?

```
Prop.forAll { xs: List[A] =>
  xs.reverse.reverse == xs
}
```

ポリモーフィックなプロパティの場合

What if my properties are polymorphic?

```
Prop.forAll { xs: List[A] =>
  xs.reverse.reverse == xs
}
```

- how should we instantiate A?

A をどうインスタンス化するか？

What if my properties are polymorphic?

```
Prop.forAll { xs: List[A] =>
  xs.reverse.reverse == xs
}
```

- how should we instantiate A?
- should we come up with fresh types?

新たな型を導入すべきか？

Existential types to the rescue!

```
trait Type {
  type T
  val name: String
  val gen: Gen[T]
}
```

存在型

Existential types to the rescue!

```
trait Type {
  type T
  val name: String
  val gen: Gen[T]
}
```

We know nothing about T!

T についてはなにも知らないが、

Existential types to the rescue!

```
trait Type {
  type T
  val name: String
  val gen: Gen[T]
}
```

We know nothing about T!

... but we can **generate values of it**

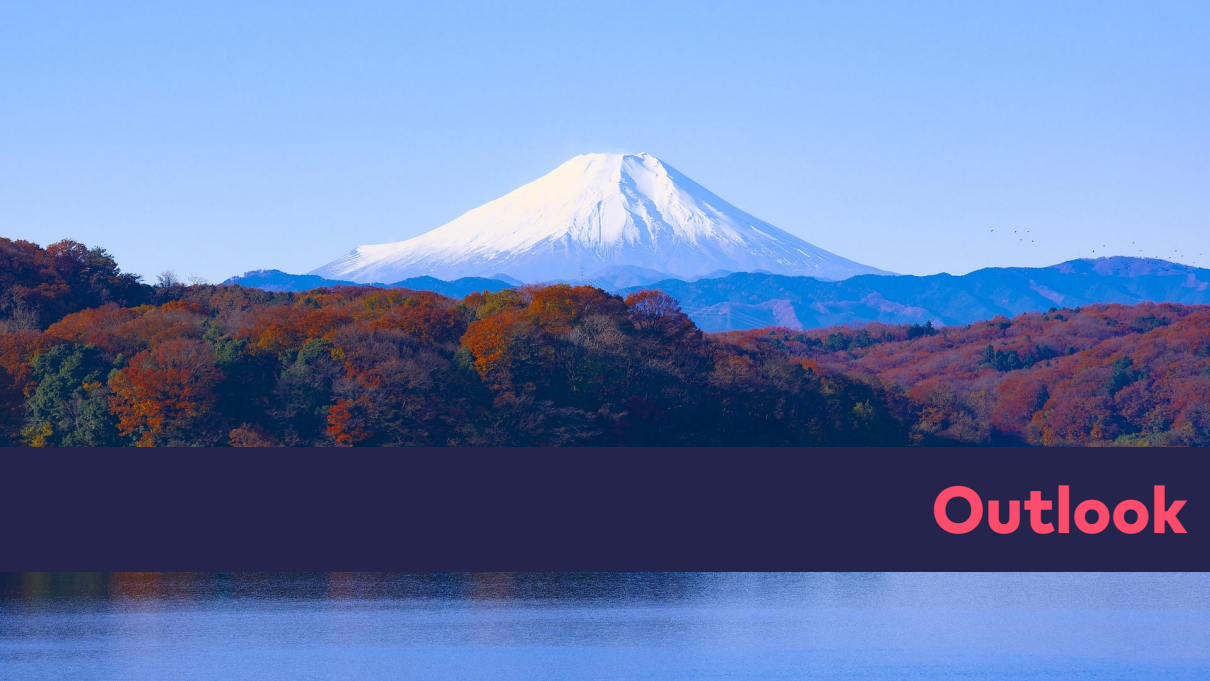
その値は生成できる。

Demo

Polymorphic testing

- perfect for testing data pipelines
- also useful for compilers
- check out Erik's talk!

ポリモーフィック・テストはデータパイプラインのテストに最適



Outlook

Outlook

There is still innovation in this space!

- law management & checking
- integrated shrinking
- automatic classification of counter-examples
- automatic generation of properties
- proofs instead of tests

検査ではなく、証明する。

Q & A

INNOQ
www.innoq.com

Lars Hupel

lars.hupel@innoq.com

@larsr_h

innoQ Deutschland GmbH

Krischerstr. 100
40789 Monheim a. Rh.
Germany
+49 2173 3366-0

Ohlauer Str. 43
10999 Berlin
Germany

Ludwigstr. 180 E
63067 Offenbach
Germany

Kreuzstr. 16
80331 München
Germany

c/o WeWork
Hermannstrasse 13
20095 Hamburg
Germany

innoQ Schweiz GmbH

Gewerbestr. 11
CH-6330 Cham
Switzerland
+41 41 743 01 11

Albulastr. 55
8048 Zürich
Switzerland

LARS HUPEL

Consultant
innoQ Deutschland GmbH

Lars enjoys programming in a variety of languages, including Scala, Haskell, and Rust. He is known as a frequent conference speaker and one of the founders of the Typelevel initiative which is dedicated to providing principled, type-driven Scala libraries.

- Chef: https://unsplash.com/photos/EHK-EH1SRzQ
- Sushi: https://pixabay.com/photos/sushi-set-nigiri-maki-fish-raw-716456/
- Scuba diver: https://pixabay.com/photos/scuba-diver-diver-diving-underwater-569333/
- Abacus: https://pixabay.com/photos/abacus-calculus-classroom-count-1866497/
- Sakura: https://pixabay.com/photos/sakura-cherry-blossum-tokyo-flower-1339106/
- Pipes: https://pixabay.com/illustrations/refinery-refining-pipes-valves-2059775/
- Mt Fuji: https://pixabay.com/photos/japan-mt-fuji-sayama-lake-reservoir-1706942/