Property-based Testing with

ScalaCheck

Motivation - the benefits of testing

- Testing is important
  - Confident and timely changes
  - Verification of functionality - reduces bugs
  - Leads to more modular, general code
  - Forces rigorous definition of software contracts
  - Provides documentation of last resort

Motivation - the pain of writing Unit Tests

- Writing good tests is HARD
  - Cannot rely on humans to check all (or any) edge cases all the time
  - Test suites are time consuming to write, a burden to maintain
  - Test suites can be very information-sparse, not always clear what is being tested

Solution

“Don’t write unit tests…

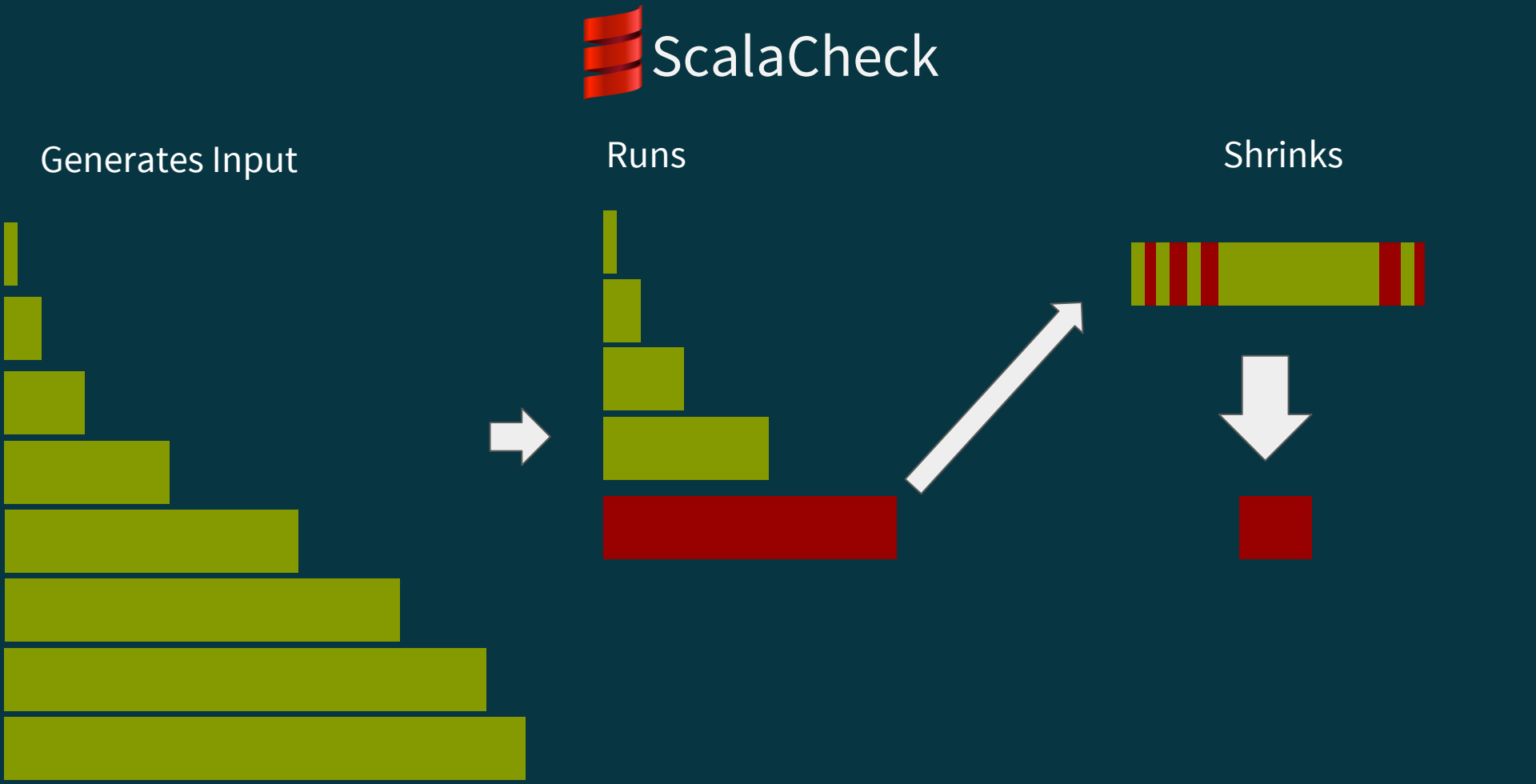
generate them!”

-John Hughes, co-creator of QuickCheck (Haskell)

```
"A Stack" should "pop values in last-in-first-out-order" in {
  val stack = new Stack[Int]
  stack.push(1)
  stack.push(2)
  assert(stack.pop() == 2)
  assert(stack.pop() == 1)
}
```

- Mixing of the test-case creation code and behavior validation

- Property-based testing is a more functional / declarative style
  - declare “truths” about our code, given some input
- Decouple test case generation from validation, so we can create many tests


ScalaCheck
Generates Input
Runs
Shrinks

```scala
import org.scalacheck.Prop.forAll
import org.scalacheck.Properties

object StringSpec extends Properties("Strings") {
  property("string.toUpperCase.toLowerCase should be string.toLowerCase") = forAll { a: String =>
    a.toUpperCase.toLowerCase == a.toLowerCase
  }
}
```

```
! Strings.string.toUpperCase.toLowerCase should be string.toLowerCase: Falsified after 65
passed tests.
> ARG_0: "̏Ḥ"
> ARG_0_ORIGINAL: "ʒ⑱箻헉鼭酤꿮귝楕膶꿪蓄숕皅羠쬠□伃□捡寣恒ᄒ蘸骙씯麼̏Ḥ櫌涀≮
뻬좖껼蚷滁쭞⚹漆縏揞뮉쟉刉□Ă軏̂욨幕㊉뢺隚"
```

Generators

```
scala> import org.scalacheck.Arbitrary.arbitrary
import org.scalacheck.Arbitrary.arbitrary

scala> val intGen: Gen[Int] = arbitrary[Int]
intGen: org.scalacheck.Gen[Int] = org.scalacheck.Gen$$anon$3@2a1736c5

scala> 1 to 10 flatMap (i => intGen.sample)
res1: IndexedSeq[Int] = Vector(-1, 2147483647, 1, -1488307207, 2147483647, 2147483647, 0, 37310880, -2147483648, -2147483648)
```

- Out of the box ScalaCheck comes with Gen's for most basic types:
  - Int, String, Double, Byte, Boolean, Char, etc..
- Also some not as basic types
  - Throwable, Either, Function1, Tuple2, etc..
- Also Collections
  - List, Vector, Option, Map, Set, Array

Primitive Gen's (Double, Byte, Char, Boolean…)

```
scala> 1 to 10 flatMap (i => arbitrary[Double].sample)
res60: scala.collection.immutable.IndexedSeq[Double] = Vector(7.795539215121126E307, -8.988465674311579E307,
-8.647586339452379E307, -1.0, 8.988465674311579E307, -1.0, -4.632951014592837E307, 0.0, -1.0, 0.0)

scala> 1 to 10 flatMap (i => arbitrary[Byte].sample)
res61: scala.collection.immutable.IndexedSeq[Byte] = Vector(127, -108, 1, 1, 96, 1, 114, 83, -121, -1)

scala> 1 to 10 flatMap (i => arbitrary[Char].sample)
res62: scala.collection.immutable.IndexedSeq[Char] = Vector(ഐ, ⊵, , Ꮆ, 뒭, 끖, 栦, 釗, 礑, ϭ)

scala> 1 to 10 flatMap (i => arbitrary[Boolean].sample)
res63:IndexedSeq[Boolean] = Vector(true, false, true, true, true, true, true, true, true, true)
```

Collections (tuples, Lists, Maps, etc..)

```
scala> arbitrary[(Int, Byte, Double)].sample.get

res72: (Int, Byte, Double) = (-1,-128,1.0)

scala> arbitrary[Map[Int,Byte]].sample.get

res73: Map[Int,Byte] = Map(0 -> -128, -2147483648 -> 43, 109118159 -> -1 …)
```

Higher Order/Utility Gens

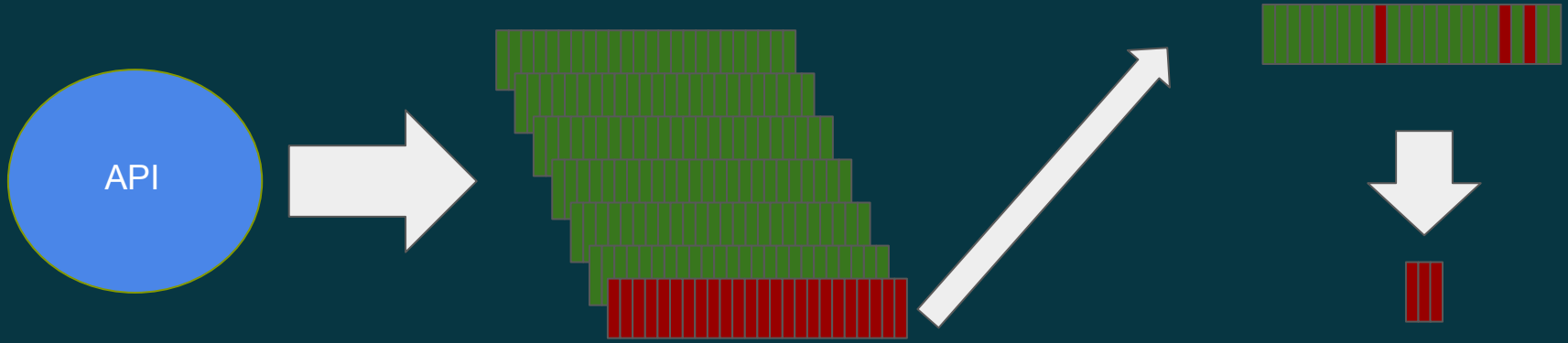
frequency, oneOf, someOf, mapOf, listOf, const, choose

```
val monthOfYearGen = Gen.frequency( 31 -> 1, 28.25 -> 2, 31 -> 3, …

val fruitGen = Gen.oneOf("Banana", "Apple", "Pear", ”Grape”)

case class Email(from: String, to: Set[String], cc: Set[String], message: String, attachments: List[File])

val employeeGen = Gen.oneOf("Josh", "Jeremy", "Kevin", ”Maia”)

// I know you can send emails to yourself, just for an example...
val emailGen: Gen[Email] =
  for {
    from <- employeeGen
    to <- someOf(employeeGen.filter(_ != from))
    cc <- someOf(employeeGen.filter(e => e != from && !to.contains(e)))
    message <- arbitrary[String]
    attachments <- Gen.listOf(Gen.oneOf(textFileGen, mp3Gen, jpgGen))
  } yield Email(from, to, cc, message, attachments)
```

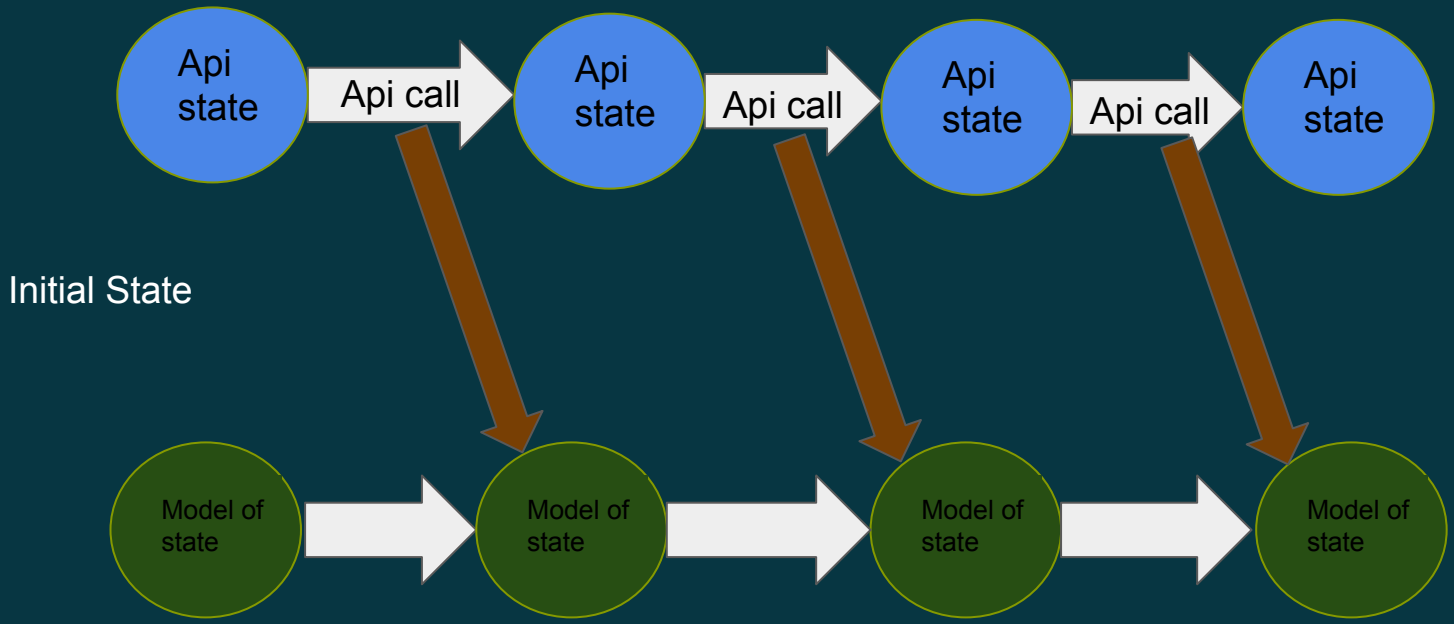
```
  Site.countAll() must_== before
  Site.create("EIS1", "0001a")
  Site.countAll() must_== (before + 1)
  Site.create("EIS2", "0001b")
  Site.create("EIS3", "0001c")
  Site.create("EIS4", "0001d")
  Site.create("EIS5", "0001e")
  Site.create("EIS6", "0001f")
  Site.countAll() must_== (before + 6)
}
"Site.findAll() should find all customers" in new AutoRollback {
  val before = Site.findAll()
  Site.findAll() must_== before
  Site.create("EIS1", "0001a")
  Site.findAll().length must_== (before.length + 1)
  Site.create("EIS2", "0001b")
  Site.create("EIS3", "0001c")
  Site.create("EIS4", "0001d")
  Site.create("EIS5", "0001e")
  Site.create("EIS6", "0001f")
  Site.findAll().length must_== (before.length + 6)
}
"Site.find() should find the right customer" in new AutoRollback {
  val siteOption = Site.create("EIS1", "0001a")
  assert(siteOption.isDefined)
  assert(siteOption.get.siteEcrmId == "EIS1")
  assert(siteOption.get.customerEcrmId == "0001a")
  val unrealId = Site.findAll.map(_.id).max + 1
  val doesNotExist = Site.find(unrealId)
  assert(doesNotExist.isEmpty)
}
"Site.findBySiteEcrmId() should find the right customer" in new AutoRollback {
  Site.create("EIS1", "0001a")
  val siteOption = Site.findBySiteEcrmId("EIS1")
  assert(siteOption.isDefined)
  assert(siteOption.get.customerEcrmId == "0001a")
  assert(siteOption.get.siteEcrmId == "EIS1")
  val doesNotExist = Site.findBySiteEcrmId("FAKE")
  assert(doesNotExist.isEmpty)
}
"Site.insertOrUpdate() should insert new customer" in new AutoRollback {
  val (siteEcrmId, customerEcrmId) = ("siteEcrmId", "customerEcrmId")
  Site.insertOrUpdate(siteEcrmId, customerEcrmId)
  assert(Site.findBySiteEcrmId(siteEcrmId) match {
    case Some(Site(_,`siteEcrmId`,`customerEcrmId`,_,_,_,_)) => true
    case _ => false
  })
}
"Site.insertOrUpdate() should update existing customer's ecrmFetchedAt time" in
```

```
property("translated organizationDto should have name = customer's default locale's display label") {
    forAll { (customer: Customer) =>
      customer.toUtilityOrganization.toDto.name shouldBe customer.display_labels(customer.default_locale_name)
    }
  }

  property("translated organizationDto should have timeZone = customer's ") {
    forAll { (customer: Customer) =>
      customer.toUtilityOrganization.toDto.timeZone shouldBe customer.tz
    }
  }

  property("translated organizationDto should have primaryLocale = customer's default_locale_name") {
    forAll { (customer: Customer) =>
      customer.toUtilityOrganization.toDto.primaryLocale shouldBe customer.default_locale_name
    }
  }

  property("translated organizationDto should have labels containing customer's default_locale_name") {
    forAll { (customer: Customer) =>
      customer.toUtilityOrganization.toDto.labels.keys should contain(customer.default_locale_name)
    }
  }

  property("translated organizationDto should have labels = customer's display_labels") {
    forAll { (customer: Customer) =>
      customer.toUtilityOrganization.toDto.labels shouldBe customer.display_labels
    }
  }
```

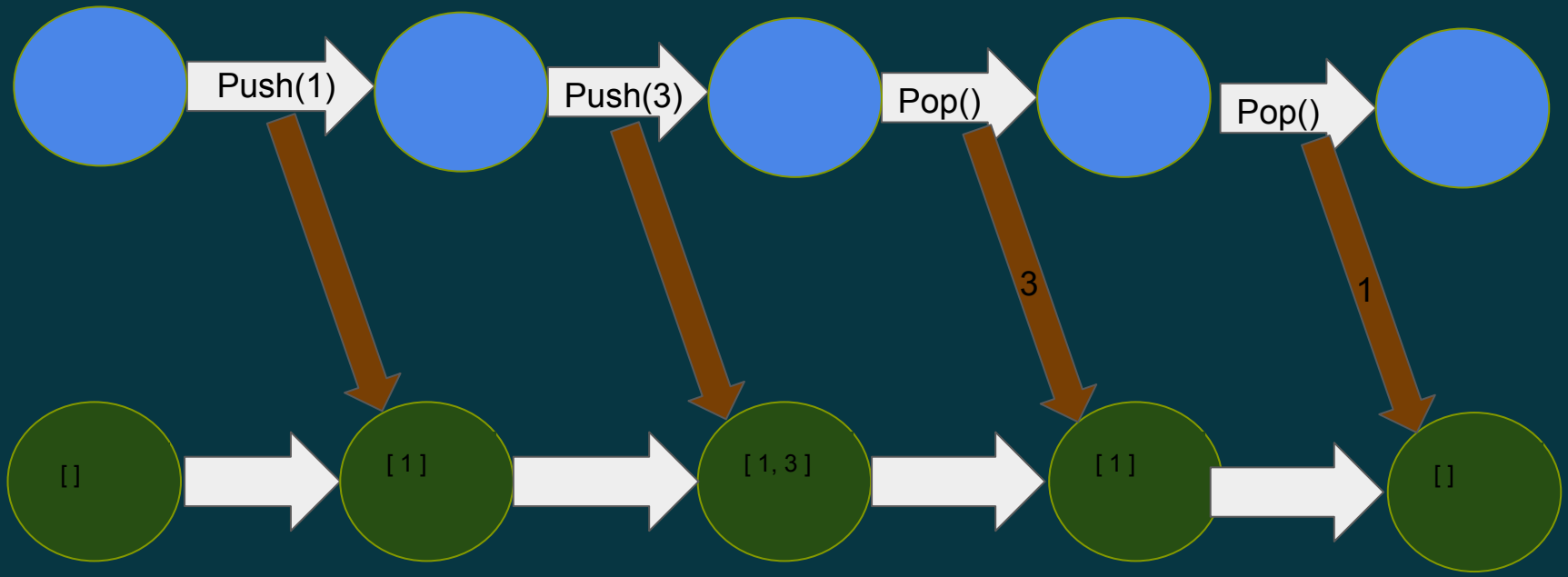
Testing State



State Machines!



Example: A Stack data structure



```
class Stack[T] {
  private var items = List.empty[T]
  def reset() = items = List.empty[T]
  def push(item: T) = items = item :: items
//   Bug!
  def popOption(): Option[T] = {
    val result = items.headOption
    if(items.length > 10) items = items.tail.tail
    else if(items.nonEmpty) items = items.tail
    result
  }
}
```

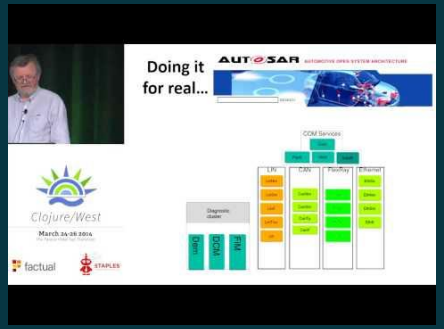
```
class StackSpecification[T](implicit arbT: Arbitrary[T]) extends Commands {
  val stack = new Stack[T]
  case class State(items: List[T])

  def initialState() = {
    stack.reset()
    State(List.empty)
  }

  case class Push(n: T) extends Command {
    def run(s: State) = stack.push(n)
    def nextState(s: State) = State(n :: s.items)
  }

  case object PopOption extends Command {
    def run(s: State): Option[T] = stack.popOption()
    def nextState(s: State) = State(if(s.items.nonEmpty) s.items.tail else s.items)

    postConditions += {
      case (s0, s1, r:Option[T]) =>
        val result = s0.items.headOption == r
        assert(result, s"popOption should be ${s0.items.headOption}, but was $r")
        result
      case _ => false
    }
  }

  def genCommand(s: State): Gen[Command] = Gen.oneOf( for(i <- arbitrary[T]) yield Push(i), Gen.const(PopOption))
}
```

```
! Exception raised on property evaluation.

> COMMANDS: Push(-689275502), Push(-1434187142), Push(1), Push(2147483647), Push(1362158311), Push(-2147483648), Push(-1623733418), Push(0), Push
(2633620), Push(0), Push(1), PopOption, PopOption

> COMMANDS_ORIGINAL: PopOption, Push(-689275502), Push(-1434187142), Push(1), Push(1), PopOption, Push(-1), PopOption, Push(2147483647), Push
(1362158311), Push(-2147483648), Push(-1623733418), Push(0), Push(2633620), PopOption, Push(2147483647), Push(0), Push(1), PopOption, PopOption,
PopOption, PopOption, PopOption, Push(1), Push(1865411318), PopOption, PopOption, PopOption, PopOption, Push(-614061289), PopOption, PopOption, Push
(2147483647), Push(-1), PopOption, Push(-2147483648), PopOption, PopOption

> Exception: java.lang.AssertionError: assertion failed: popOption() should be Some(0) but was Some(2633620)
```

Configuration

Config Parameter	Default Value
minSuccessful	100
maxDiscarded	500
minSize	0
maxSize	100
workers	1

Links



John Hughes - Testing the Hard Stuff and Staying Sane

Kelsey Gilmore-Innis - I Dream of Gen’ning: ScalaCheck Beyond the Basics

Links

.NET (C#, F#, VB)

* FsCheck http://fscheck.codeplex.com/

Python:

* Factcheck https://github.com/npryce/python-factcheck
* Hypothesis https://github.com/DRMacIver/hypothesis
* pytest-quickcheck http://pypi.python.org/pypi/pytest-quickcheck/

Ruby:
* Rantly https://github.com/hayeah/rantly

Scala:
* ScalaCheck https://github.com/rickynils/scalacheck
* Nyaya https://github.com/japgolly/nyaya

Haskell

* QuickCheck https://hackage.haskell.org/package/QuickCheck

Clojure:

* ClojureCheck https://bitbucket.org/kotarak/clojurecheck

Java:
* JavaQuickCheck http://java.net/projects/quickcheck/pages/Home

Groovy:
* Gruesome https://github.com/mcandre/gruesome

JavaScript:

* QC.js https://bitbucket.org/darrint/qc.js/