I DREAM OF GEN'NING

SCALACHECK IS BLACK MAGICK

Kelsey Gilmore-Innis (@kelseyinnis)
+ Stew O'Connor (@StewOConnor)

PROPERTY-BASED TESTING

```
test("pop") {
    val stack = new Stack[Int]
    stack.push(1)
    stack.push(2)
    val result = stack.pop()
    assert(result === 2)
    assert(stack.size === 1)
}
```

```
test("pop") {
    val stack = new Stack[Int]
    stack.push(1)
    stack.push(2)
    stack.push(3)
    val result = stack.pop()
    assert(result === 3)
    assert(stack.size === 2)
}
```

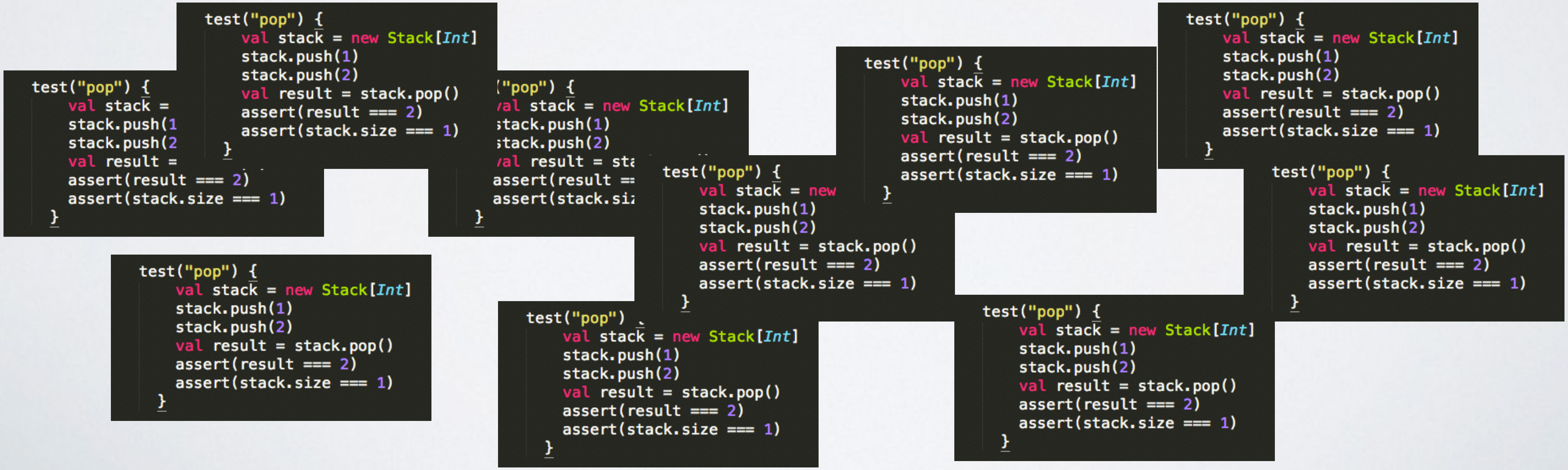
```
test("pop_again") {
    val stack = new Stack[Int]
    stack.push(1)
    stack.push(2)
    stack.push(3)
    val result = stack.pop()
    assert(result === 3)
    assert(stack.size === 2)
}
```

```
test("pop") {
  val stack = new Stack[Int]
  test("pop_again") {
    test("pop_twice") {
      val stack = new Stack[Int]
      stack.push(1)
      stack.push(2)
      stack.push(3)
    }   val result1 = stack.pop()
      val result2 = stack.pop()
      assert(result === 2)
      assert(stack.size === 1)
    }
```

```
test("pop") {
    val stack = new Stack[Int]
    stack.push(1)
    stack.push(2)
    val result = stack.pop()
    assert(result === 2)
    assert(stack.size === 1)
  }
```

```
test("pop") {
    val stack = new Stack[Int]
    stack.push(1)
    stack.push(2)
    val result = stack.pop()
    assert(result === 2)
    assert(stack.size === 1)
  }
```

```scala
test("pop") {
    val stack = new Stack[Int]
    stack.push(1)
    stack.push(2)
    val result = stack.pop()
    assert(result === 2)
    assert(stack.size === 1)
}
```



BACKGROUND

GENERATES • RUNS • SHRINKS

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

object ShakespeareSpec extends Properties("Bubble") {

  def spell(words: List[String], mysticNumber: Int) =
    mysticNumber.toString +
      words.mkString + "toil" + "trouble"

  property("bubble") = forAll { (a: List[String], b:Int) =>
    spell(a, b).length ==
      Math.ceil(Math.log10(b + 1)) + a.map(_.length).sum + 11
  }
}
```

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

object ShakespeareSpec extends Properties("Bubble") {

  def spell(words: List[String], mysticNumber: Int) =
    mysticNumber.toString +
      words.mkString + "toil" + "trouble"

  property("bubble") = forAll { (a: List[String], b:Int) =>
    spell(a, b).length ==
      Math.ceil(Math.log10(b + 1)) + a.map(_.length).sum + 11
  }
}
```

```
[info] ! Bubble.bubble: Falsified after 2 passed tests.
[info] > ARG_0: List("")
[info] > ARG_0_ORIGINAL: List("커 돍 ")
[info] > ARG_1: 0
[info] > ARG_1_ORIGINAL: -677524853
[error] Failed: Total 1, Failed 1, Errors 0, Passed 0
```

GENERATORS

OUT OF THE BOTTLE

- Boolean, Byte, Char, String, Int, Double, Long
- Throwable, Date
- Option, Either, Tuple2-Tuple9, Function1-Function5
- Array, List, ArrayList, Stream, Set, Map

EDGE CASE BIAS

HELPERS

- alphaChar, alphaNumChar, alphaLowerChar, alphaUpperChar

- alphaStr, numStr, identifier

```
forAll(listOf(alphaStr), posNum[Int]) { (a, b) =>
  spell(a, b).length ==
    Math.ceil(Math.log10(b + 1)) +
      a.map(_.length).sum + 11
}
```

FANCY GENS

.choose, .oneOf, .someOf

```
forAll(Gen.choose(1, 10)) { magicNumber =>
  magicNumber.invoke == motherHecate
}

forAll(Gen.oneOf(eyeOfNewt, toeOfFrog, wingOfBat)) {
 (ingredient) =>
    ingredient.addToCauldron == hellBroth
}

forAll(Gen.someOf(tongueOfDog, addersFork, blindwormsSting,
                  lizardsLeg, howletsWing)) {
  (ingredients) =>
    boil(ingredients.map(_.addToCauldron)) ==
        charmOfPowerfulTrouble
}
```

```
val slimy =
  Gen.oneOf(eyeOfNewt, toeOfFrog, lizardsLeg)
val crawly =
  Gen.oneOf(addersFork, blindwormsSting, lizardsLeg)
val creepy =
  Gen.oneOf(wingOfBat, tongueOfDog, howletsWing)

forAll(Gen.someOf(slimy, crawly, creepy)) {
  (ingredients) =>
      boil(ingredients.map(_.addToCauldron)) ==
          charmOfPowerfulTrouble
}

forAll(Gen.listOf(slimy)) {(ingredients) =>
  touch(ingredients) == skinCrawl
}
```

COMPOSING GENS

map • flatMap • filter/suchThat

```scala
val spellGen = arbitrary[String].map(_ + ", so mote it be")
```

```scala
val spellGen = arbitrary[String].map(_ + ", so mote it be")
```

```scala
val titleGen = arbitrary[String].flatMap{ name =>
  Gen.oneOf("East", "West", "North", "South").map { dir =>
    name + ", Wicked Witch of the " + dir
  }
}
```

```scala
val spellGen = arbitrary[String].map(_ + ", so mote it be")
```

```scala
val titleGen = arbitrary[String].flatMap{ name =>
  Gen.oneOf("East", "West", "North", "South").map { dir =>
    name + ", Wicked Witch of the " + dir
  }
}
```

```scala
case class Spell(ingredients: List[(Int, Ingredient)],
                 boilingTime: Int)

val spellRecipeGen = for {
  ingredientList <- Gen.someOf(slimy, crawly, creepy)
  amounts <- Gen.listOfN(ingredientList.length,
                         Gen.choose(1, 10))
  bubblesFor <- Gen.posNum[Int].suchThat(_ < 720)
} yield {
  Spell(amounts.zip(ingredientList), bubblesFor)
}
```

.suchThat

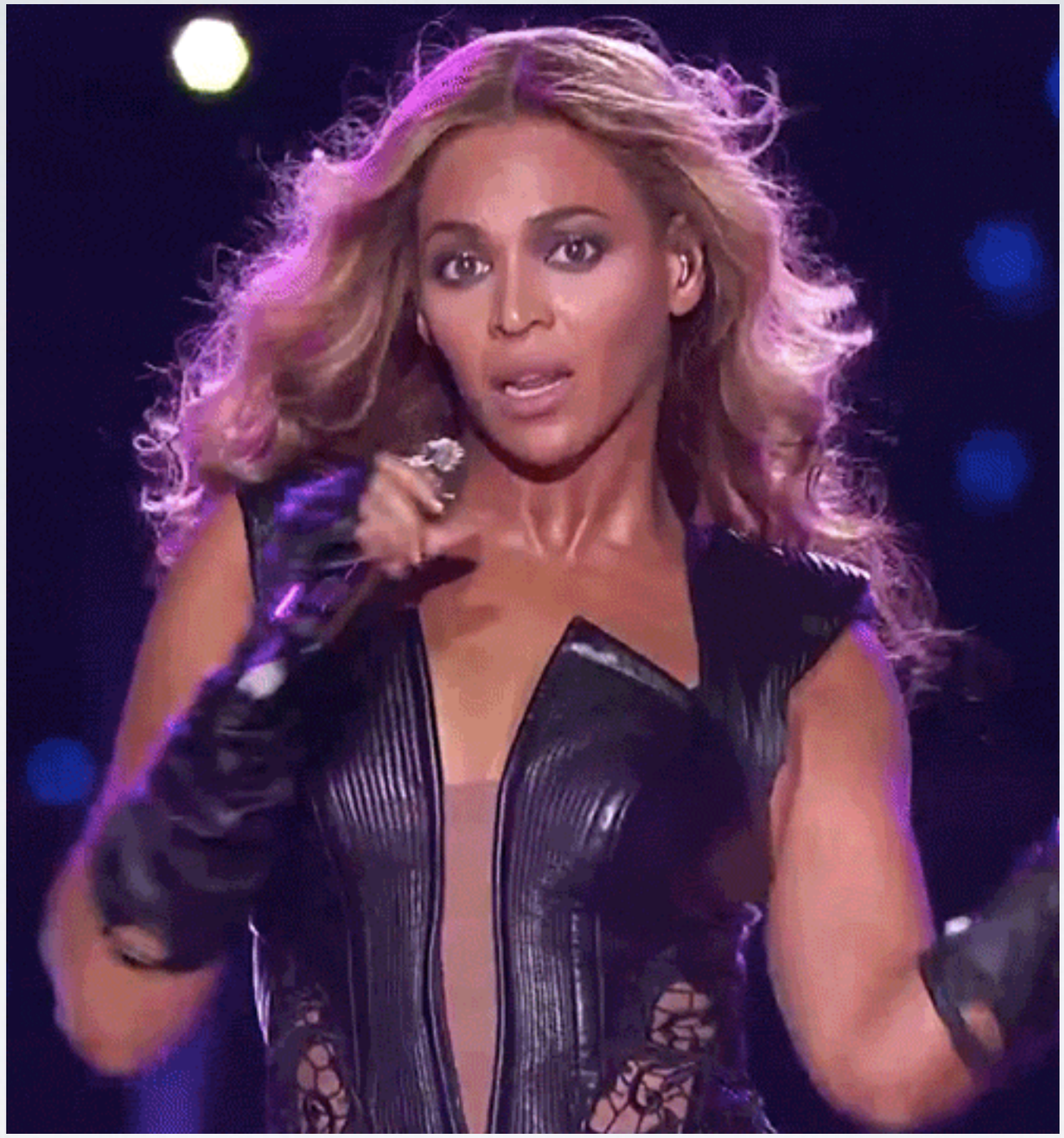
```scala
case class Person(name: String, age: Int)

val personGen = for {
  name <- Gen.alphaStr
  age <- Gen.choose(1, 100)
} yield {
  Person(name, age)
}
```

```scala
case class Person(name: String, age: Int)

val personGen = for {
  name <- Gen.alphaStr
  age <- Gen.choose(1, 100)
} yield {
  Person(name, age)
}
```

```scala
implicit def arbitraryJsonType: Arbitrary[JSONType] =
  Arbitrary { sized(depth => jsonType(depth)) }

def jsonType(depth: Int): Gen[JSONType] = oneOf(jsonArray(depth),
                                                jsonObject(depth))

def jsonObject(depth: Int): Gen[JSONObject] = for {
  n    <- choose(1, 4)
  ks   <- keys(n)
  vals <- values(n, depth)
} yield JSONObject(Map((ks zip vals):_*))

def values(n: Int, depth: Int) = listOfN(n, value(depth))

def value(depth: Int) =
  if (depth == 0)
    terminalType
  else
    oneOf(jsonType(depth - 1), terminalType)

def terminalType = oneOf(1, 2, "m", "n", "o")
```

http://tiny.cc/json

FREQUENCY

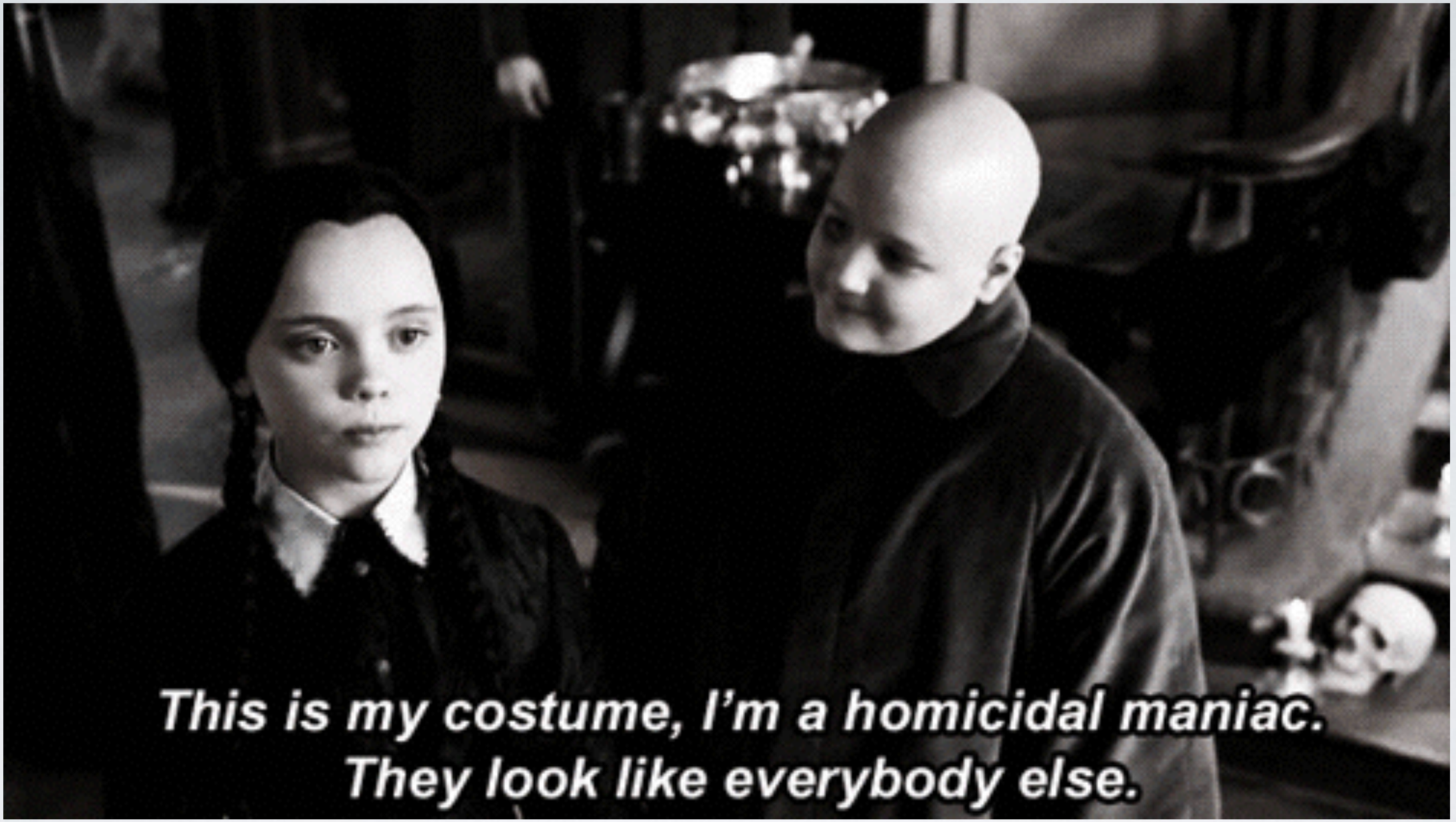


P1 BUG: BEYONCÉ NOT FOUND

PERFORMANCE TESTING

RUNNING SCALACHECK TESTS

PARAMS



INTEGRATION

ROUND-TRIP PROPERTIES

```
forAll { p: Prince =>
    princessKiss(turnIntoFrog(p)) == p
}
```

NEGATIVE TESTING

http://tiny.cc/poolboy

SCALACHECK + SELENIUM

http://tiny.cc/selenium

COMMANDS

TESTING SERVER NETWORKS

http://tiny.cc/servers

SIMULATION TESTING

http://tiny.cc/simulant

SHRINKING

```scala
implicit lazy val shrinkInt: Shrink[Int] = Shrink { n =>

  log.shriek("I'LL GET YOU MY PRETTIES!")

  def halfs(n: Int): Stream[Int] =
    if(n == 0) empty else cons(n, halfs(n/2))

  if(n == 0) empty else {
    val ns = halfs(n/2).map(n - _)
    cons(0, interleave(ns, ns.map(-1 * _)))
  }
}
```

MAKING IT LOOK EASY

```
/Users/stew/devel/test:
total used in directory 8 available 53010364
drwxr-xr-x    4 stew  staff   136 Aug  1 01:58 .
-rw-r--r--    1 stew  staff   130 Aug  1 01:43 build.sbt
drwxr-xr-x  134 stew  staff  4556 Jul 30 14:17 ..
drwxr-xr-x    3 stew  staff   102 Jul  1 00:09 src
```

```
-:%%-  test<devel>          All (4,49)     [(Dired by date MRev yas Projectile)]
```


More witches!

BYE!