QuickCheck для Kotlin

Студент: Михаил Чернявский
Руководитель: Марат Ахин

QuickCheck

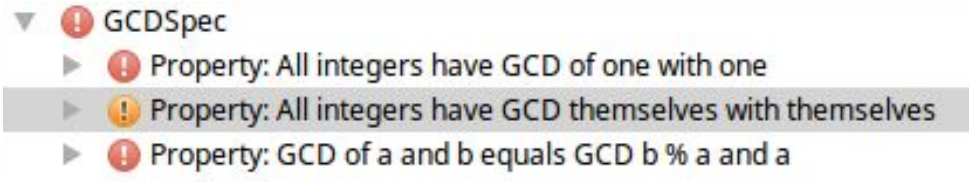
- Библиотека комбинаторов для генерации случайных тестов, первоначально написанная под Haskell
- Имеет аналоги для множества других языков
- Существует примитивная реализация для Kotlin в рамках проекта kotlintest
- Хочется иметь нормальный QuickCheck для Kotlin

Простой пример

```
fun gcd(a: Int, b: Int): Int = when {
    a == 0 -> b
    b == 0 -> a
    a > b -> gcd(b, a)
    else -> gcd(b % a, a)
}

property("All integers have GCD of one with one") {
    forAll { a: Int -> gcd(a, 1) == 1 }
}

property("All integers have GCD themselves with themselves") {
    forAll { a: Int -> gcd(a, a) == abs(a) }
}

property("GCD of a and b equals GCD b % a and a") {
    forAll { a: Int, b: Int -> gcd(a, b) == gcd(b % a, a) }
}
```

Простой пример

```
fun gcd(a: Int, b: Int): Int = when {
    a == 0 -> b
    b == 0 -> a
    a > b -> gcd(b, a)
    else -> gcd(b % a, a)
}

property("All integers have GCD of one with one") {
    forAll { a: Int -> gcd(a, 1) == 1 }
}

property("All integers have GCD themselves with themselves") {
    forAll { a: Int -> gcd(a, a) == abs(a) }
}

property("GCD of a and b equals GCD b % a and a") {
    forAll { a: Int, b: Int -> gcd(a, b) == gcd(b % a, a) }
}
```

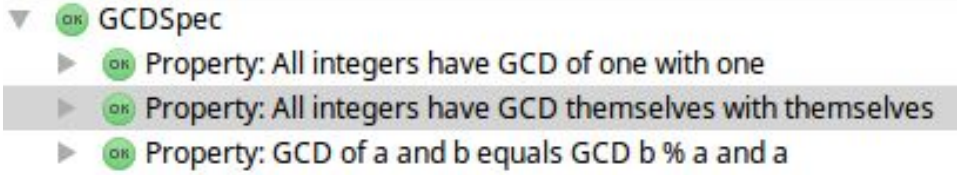


Простой пример

```
fun gcd(a: Int, b: Int): Int = when {
    a == 0 -> abs(b)
    b == 0 -> abs(a)
    a < 0 -> gcd(-a, b)
    b < 0 -> gcd(a, -b)
    a > b -> gcd(b, a)
    else -> gcd(b % a, a)
}

property("All integers have GCD of one with one") {
    forAll { a: Int -> gcd(a, 1) == 1 }
}

property("All integers have GCD themselves with themselves") {
    forAll { a: Int -> gcd(a, a) == abs(a) }
}

property("GCD of a and b equals GCD b % a and a") {
    forAll { a: Int, b: Int -> gcd(a, b) == gcd(b % a, a) }
}
```

Простой пример

```
fun gcd(a: Int, b: Int): Int = when {
    a == 0 -> abs(b)
    b == 0 -> abs(a)
    a < 0 -> gcd(-a, b)
    b < 0 -> gcd(a, -b)
    a > b -> gcd(b, a)
    else -> gcd(b % a, a)
}

property("All integers have GCD of one with one") {
    forAll { a: Int -> gcd(a, 1) == 1 }
}

property("All integers have GCD themselves with themselves") {
    forAll { a: Int -> gcd(a, a) == abs(a) }
}

property("GCD of a and b equals GCD b % a and a") {
    forAll { a: Int, b: Int -> gcd(a, b) == gcd(b % a, a) }
}
```

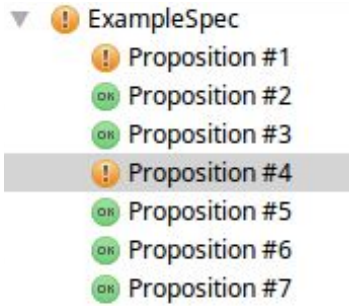


Больше примеров

```
forAll(ints(), ints(origin = 1)) { a, b -> a + b > a }

forAll(bigInts(), bigInts(origin = ONE)) { a, b -> a + b > a }

forAll { str: String -> str.reversed().reversed() == str }

forAll { xs: List<Int>, ys: List<Int> ->
    (xs + ys).reversed() == xs.reversed() + ys.reversed()
}

forAll { xs: Set<Int> ->
    xs.groupBy { it }.all { (_, group) -> group.size == 1 }
}

forAll(oneOf('a', 'e', 'i', 'o', 'i')) { ch -> ch in "aeiou" }

forAll(oneOf(ints(10, 20), ints(30, 40))) { x -> x in 10..20 || x in 30..40 }
```

Больше примеров

```
forAll(ints(), ints(origin = 1)) { a, b -> a + b > a }

forAll(bigInts(), bigInts(origin = ONE)) { a, b -> a + b > a }

forAll { str: String -> str.reversed().reversed() == str }

forAll { xs: List<Int>, ys: List<Int> ->
    (xs + ys).reversed() == xs.reversed() + ys.reversed()
}

forAll { xs: Set<Int> ->
    xs.groupBy { it }.all { (_, group) -> group.size == 1 }
}

forAll(oneOf('a', 'e', 'i', 'o', 'i')) { ch -> ch in "aeiou" }

forAll(oneOf(ints(10, 20), ints(30, 40))) { x -> x in 10..20 || x in 30..40 }
```



Генераторы для произвольных классов

```
data class BSTree(var left: BSTree?, var right: BSTree?) {
    val size: Int
        get() = 1 + (left?.size ?: 0) + (right?.size ?: 0)

    constructor() : this(null, null)
}


// Генератор руками

val leafGen = Arbitrary(object : Gen<BSTree> {
    override fun generate(): BSTree = BSTree()
})

val forkGen = Arbitrary(object : Gen<BSTree> {
    override fun generate(): BSTree = BSTree(bstGen.generate(), bstGen.generate())
})

val bstGen = oneOf(leafGen, forkGen)


forAll(bstGen) { tree: BSTree -> tree.size < 3 }
```

Автоматический вывод генераторов для произвольных классов

```
data class BSTree(var left: BSTree?, var right: BSTree?) {
    val size: Int
        get() = 1 + (left?.size ?: 0) + (right?.size ?: 0)

    constructor() : this(null, null)
}


// Или так

forAll { tree: BSTree -> tree.size < 3 }
```

```
Proposition failed. Counterexample: BSTree(left=BSTree(left=null, right=null), right=BSTree(left=null, right=null))
```

Shrinking

- Идея в том, чтобы получив контрпример для теста, попытаться его в каком-то смысле минимизировать
- Важная фича QuickCheck, отсутствующая в kotlintest

Shrinking

```
forAll { x: Int -> x < 42 }

Proposition failed. Counterexample: 42


forAll { xs: List<Int> -> xs.size < 5 }

Proposition failed. Counterexample: [0, 0, 0, 0, 0]


forAll { xs: List<Int>, ys: List<Int> ->
    (xs + ys).reversed() == xs.reversed() + ys.reversed()
}

Proposition failed. Counterexample: ([0], [1])


forAll { (first, second): Pair<Int, Int> -> first < second }

Proposition failed. Counterexample: (0, 0)
```

База контрпримеров

- Если свойство упало на каком-то случайном примере, то сохраним этот контрпример в базу
- При последующих запусках этого теста в первую очередь будем тестировать на контрпримерах из базы

Вопросы?

github.com/mchernyavsky/kotlincheck