

TESTING
BASADO EN
PROPIEDADES

QUIÉN SOY

Roberto Serrano

Desarrollo backend en Scala

Trabajo en remoto para 47 degrees

bilki @  

¿POR QUÉ ESCRIBIMOS TESTS?

¿POR QUÉ ESCRIBIMOS TESTS?

Verificación 

¿POR QUÉ ESCRIBIMOS TESTS?

Verificación 

Documentación 

TIPOS DE TESTS

- Unitarios
- Integración
- E2E
- Smoke
- Aceptación
- Golden
- UI
- Performance
- Chaos
- ...

¿QUÉ ES UN TEST?



ESTRUCTURA DE UN TEST

(Casi) todos los componentes de la definición

```
class FactorialSuite extends FunSuite {
  test("Factorial zero base case") {
    val input      = 0                      // Entrada
    val expected   = 1                      // Resultado esperado
    val result     = factorial(input)        // Salida
    assertEquals(result, expected)          // Aserción
  }
}
```

ENTRADA

Consideremos la entrada a una función

ENTRADA

Consideremos la entrada a una función

Tipo	Habitantes
Boolean	→ True False
Int	→ [-2147483648, 2147483647]
String	→ ???

ENTRADA

ENTRADA



¿Tests necesarios para una cobertura completa?

ENTRADA



¿Tests necesarios para una cobertura completa?



Tantos como habitantes en el tipo de entrada

ASERCIÓN

Ejecutamos el código a verificar
¿Se cumple el predicado?

ASERCIÓN

Ejecutamos el código a verificar
¿Se cumple el predicado?

```
def sum(x: Int, y: Int): Int = x + y

// >-----VV-<
val zero  = sum(0, 0) == 0
// zero: Boolean = true
val two   = sum(1, 1) == 2
// two: Boolean = true
```

TESTS BASADOS EN EJEMPLOS

Muy directos 

Entrada y salida manuales 

TESTS BASADOS EN EJEMPLOS

Muy directos 

Entrada y salida manuales 

```
def factorial(n: Int): Int =  
  if (n <= 1) 1 else n * factorial(n - 1)  
  
val f0 = factorial(0) == 1  
// f0: Boolean = true  
val f1 = factorial(1) == 1  
// f1: Boolean = true  
val f4 = factorial(4) == 24  
// f4: Boolean = true
```

PROBLEMA CON LOS EJEMPLOS



¿Cuántos tests son suficientes?

¿QUÉ DICE TDD DE ESTO?

Write production code only to make a failing unit test pass.

¿QUÉ DICE TDD DE ESTO?

Write production code only to make a failing unit test pass.

☰ Pero, ¿cuáles?

¿QUÉ DICE TDD DE ESTO?

Write production code only to make a failing unit test pass.

☰ Pero, ¿cuáles?

Show me the code </>

MÁS ALLÁ DE TDD

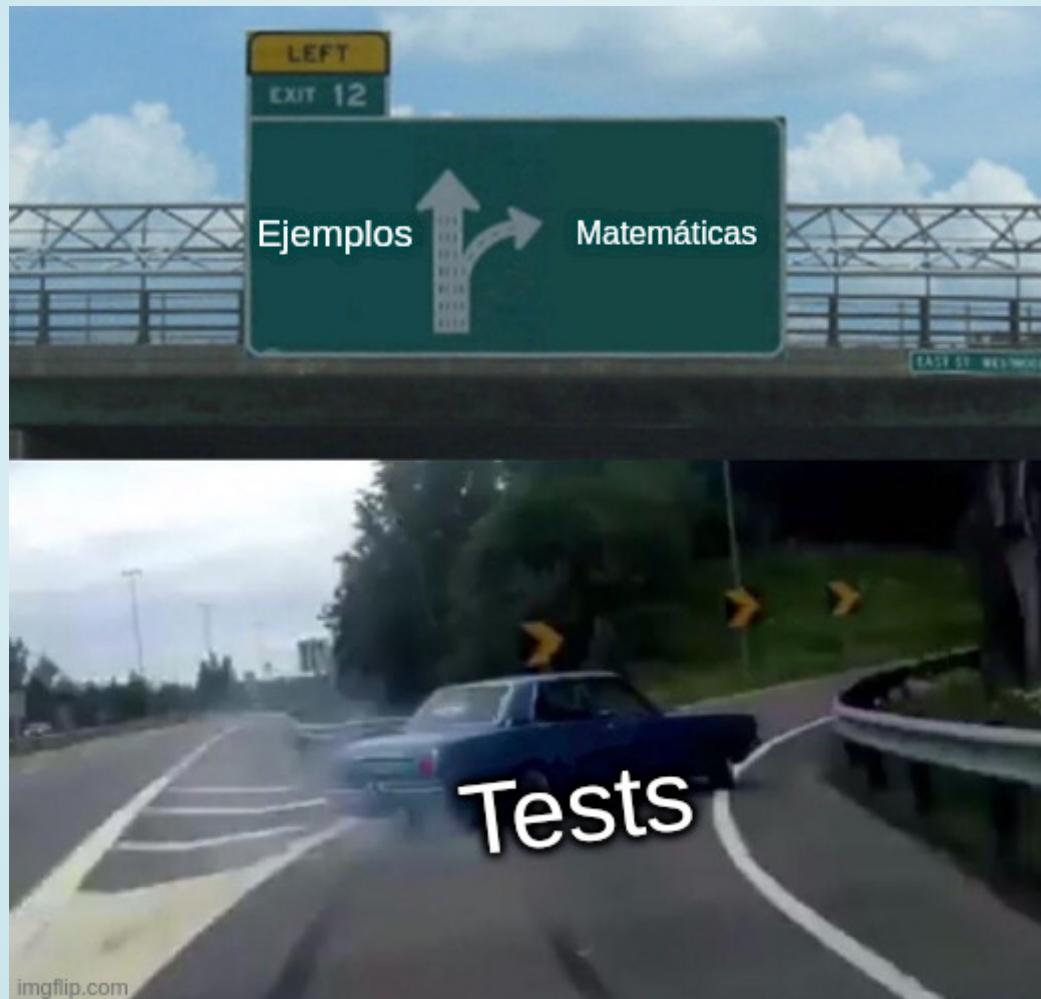
- Cobertura insuficiente
- Implementación dirigida por ejemplos
- Incertidumbre

WAIT A MINUTE...



¿Y si hubiera alguna otra forma de **probar**?

¿MATEMÁTICAS?



PROPIEDAD MATEMÁTICA

Una propiedad p para todos los elementos de un conjunto X normalmente se define como $p: X \rightarrow \{\text{true}, \text{false}\}$

PROPIEDAD MATEMÁTICA

*Una propiedad p para todos los elementos de un conjunto X normalmente se define como
 $p: X \rightarrow \{\text{true}, \text{false}\}$*

*O bien como el subconjunto de X para cuyos elementos la propiedad evalúa a verdadero
 $\{x \mid p(x) = \text{true}\}$*

PROPIEDAD MATEMÁTICA

Commutatividad

$$x * y = y * x$$
$$\forall x, y \in S$$

PROPIEDADES EN COMPUTACIÓN

¿Cómo evitar los ejemplos?

PROPIEDADES EN COMPUTACIÓN

¿Cómo evitar los ejemplos?

Entrada manual > aleatoria

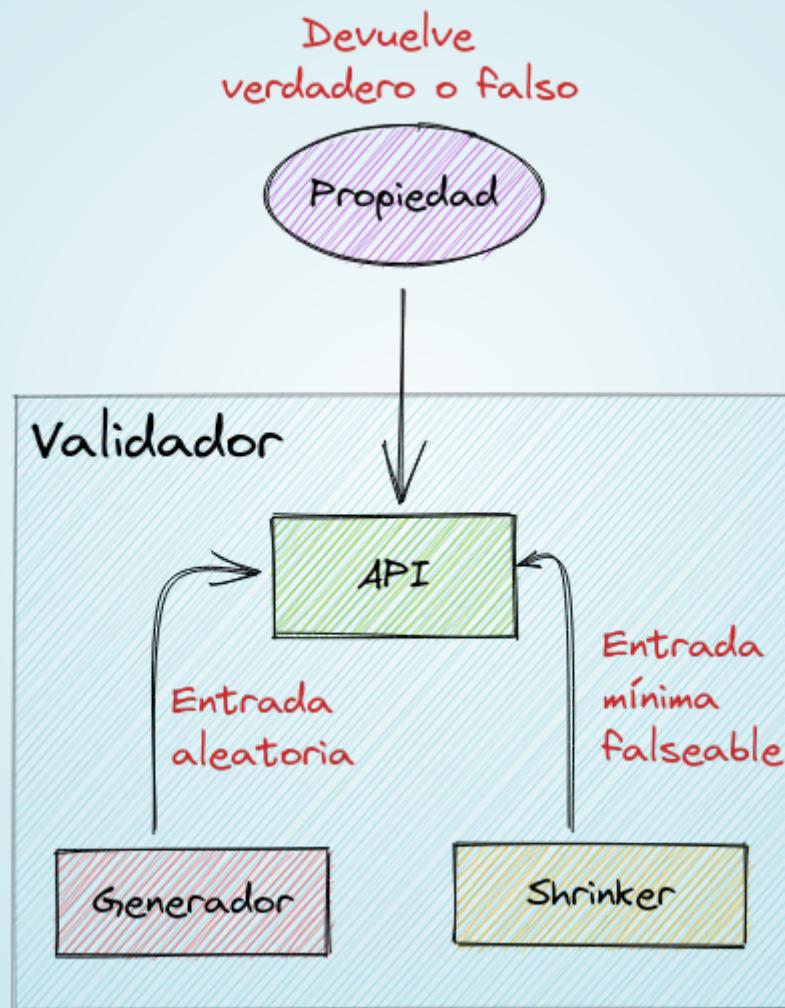
PROPIEDADES EN COMPUTACIÓN

¿Cómo evitar los ejemplos?

Entrada manual > **aleatoria**

Salida manual > **propiedades**

VALIDADOR DE PROPIEDADES



PRIMER VALIDADOR

QuickCheck - ICFP 2000

QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs

Koen Claessen
Chalmers University of Technology
koen@cs.chalmers.se

John Hughes
Chalmers University of Technology
rjmh@cs.chalmers.se

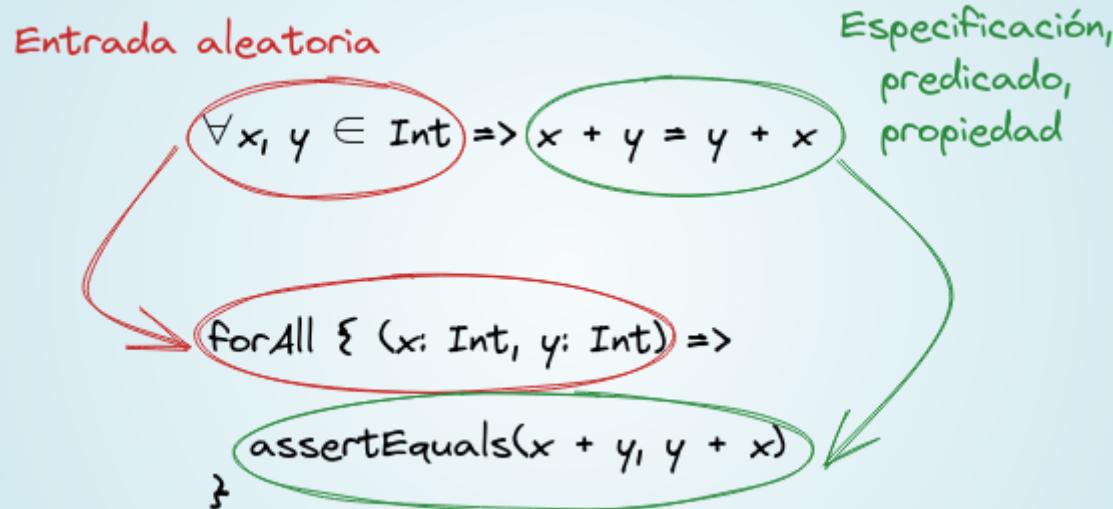
ABSTRACT

QuickCheck is a tool which aids the Haskell programmer in formulating and testing properties of programs. Properties are described as Haskell functions, and can be automatically tested on random input, but it is also possible to define custom test data generators. We present a number of case studies, in which the tool was successfully used, and also point out some pitfalls to avoid. Random testing is especially suitable for functional programs because properties can be stated at a fine grain. When a function is built from separately tested components, then random testing suffices to obtain good coverage of the definition under test.

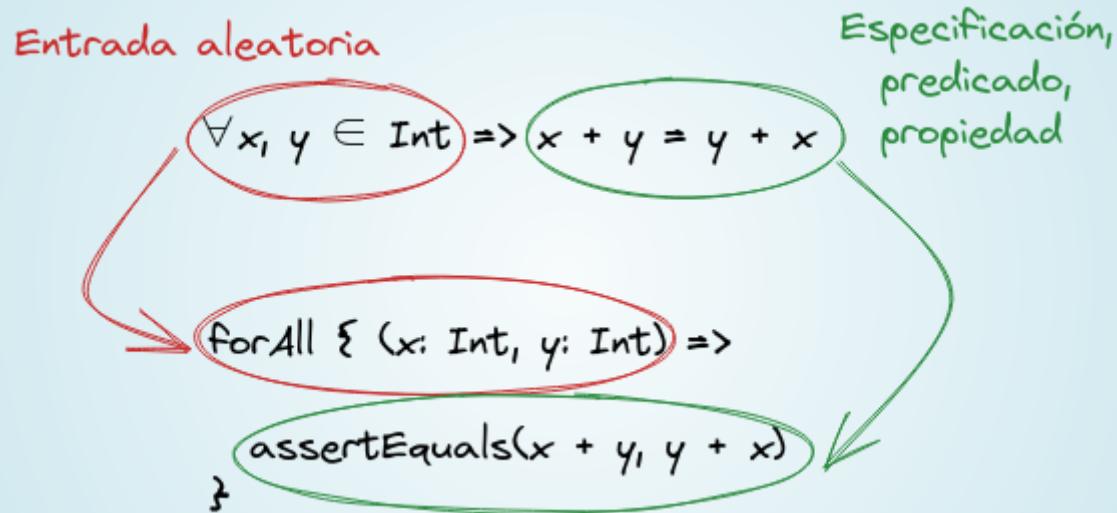
monad are hard to test), and so testing can be done at a fine grain.

A testing tool must be able to determine whether a test is passed or failed; the human tester must supply an automatically checkable criterion of doing so. We have chosen to use formal specifications for this purpose. We have designed a simple domain-specific language of *testable specifications* which the tester uses to define expected properties of the functions under test. QuickCheck then checks that the properties hold in a large number of cases. The specification language is embedded in Haskell using the class system. Properties are normally written in the same module as the functions they test, where they serve also as checkable doc-

ESTRUCTURA DE UNA PROPIEDAD



ESTRUCTURA DE UNA PROPIEDAD



Show me the code </>

GENERADORES POR DEFECTO

```
val number = Gen.posNum[Int].sample.get
// number: Int = 72

val string = Gen.stringOfN(10, Gen.alphaChar).sample.get
// string: String = "ptUbWfpjgP"

val boolean = arbitrary[Boolean].sample.get
// boolean: Boolean = false

val numbers = Gen.listOfN(5, Gen.posNum[Int]).sample.get
// numbers: List[Int] = List(20, 83, 2, 43, 49)
```

GENERADORES PERSONALIZADOS

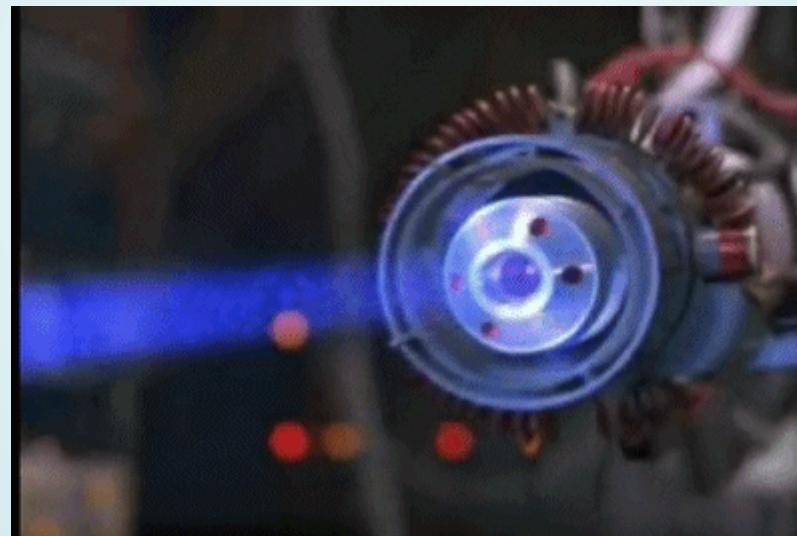
```
case class Person(name: String, age: Int)

val personGen = for {
    name <- Gen.stringOfN(10, Gen.alphaChar)
    age  <- Gen.chooseNum(1, 125)
} yield Person(name, age)
// personGen: Gen[Person] = org.scalacheck.Gen$$anon$5@2319bbcf

val person = personGen.sample.get
// person: Person = Person(name = "pFMUPZHgDr", age = 9)
```

ENCOGIENDO

¿Qué entrada mínima produce un fallo?



SHRINKER

Genera valores cada vez más pequeños

SHRINKER

Genera valores cada vez más pequeños

```
def adultsFrom(persons: List[Person]): List[Person] =  
  persons.filter(_.age > 16) // debería ser > 17
```

SHRINKER

Genera valores cada vez más pequeños

```
def adultsFrom(persons: List[Person]): List[Person] =  
  persons.filter(_.age > 16) // debería ser > 17
```

Mínima entrada posible errónea

```
val adults = adultsFrom(List(Person("Pepe", 17))) // vacío?  
// adults: List[Person] = List(Person(name = "Pepe", age = 17))
```

SEMILLAS

Si la entrada es aleatoria...

SEMILLAS

Si la entrada es aleatoria...

¿Cómo repetimos el test en el futuro?

SEMILLAS

Si la entrada es aleatoria...

¿Cómo repetimos el test en el futuro?

You can reproduce this failure by adding the following override to your suite:

```
override val scalaCheckInitialSeed =  
  "89puZ9LMPc0bM1_qSIt9tRxapPu-hWG5R4XcwVwH7BH="
```

ENCONTRAR PROPIEDADES

Es... complicado

Pero nos fuerza a reflexionar

ENCONTRAR PROPIEDADES

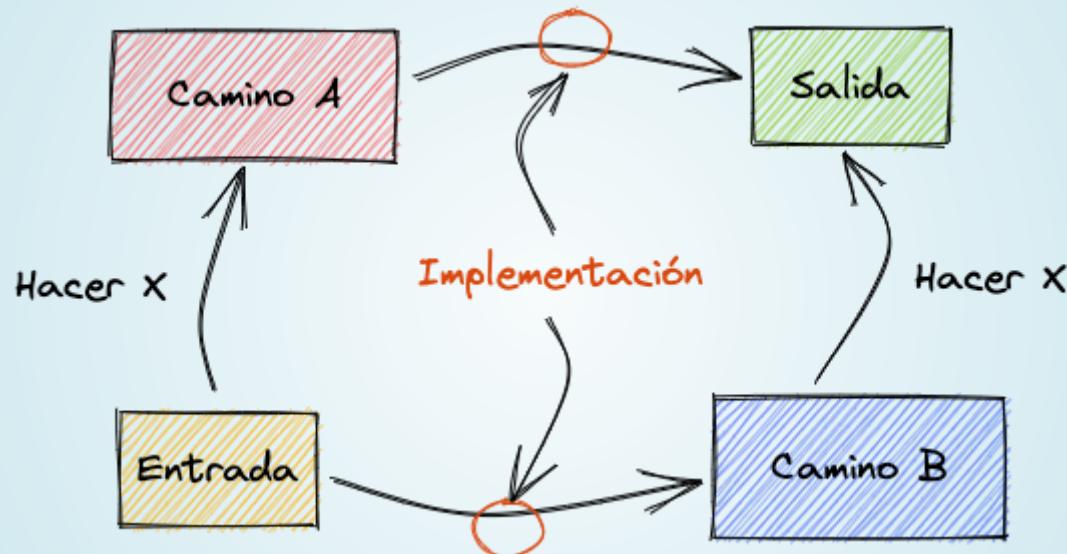
Es... complicado

Pero nos fuerza a reflexionar

¿Cuál es en realidad la especificación?

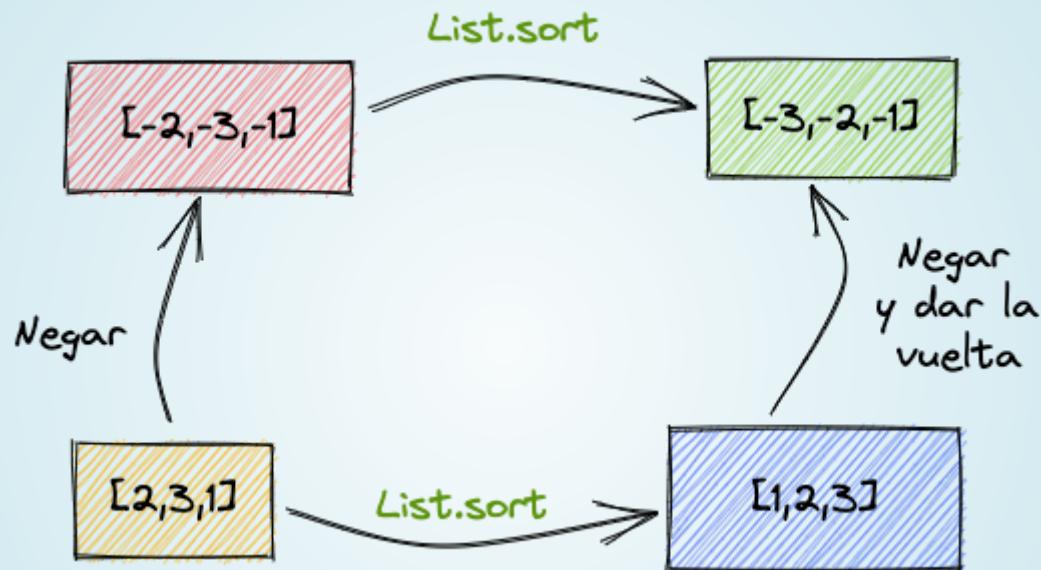


VARIOS CAMINOS (I)



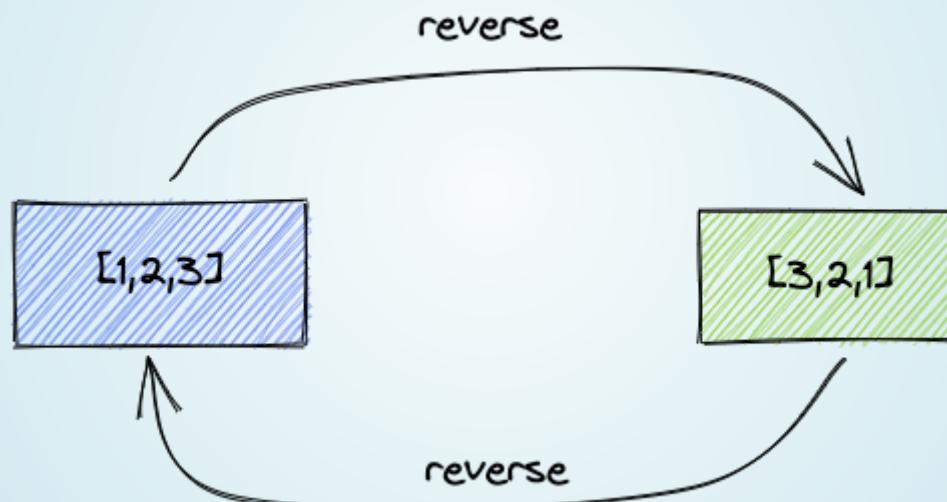
X y luego F = F y luego X

VARIOS CAMINOS (I)

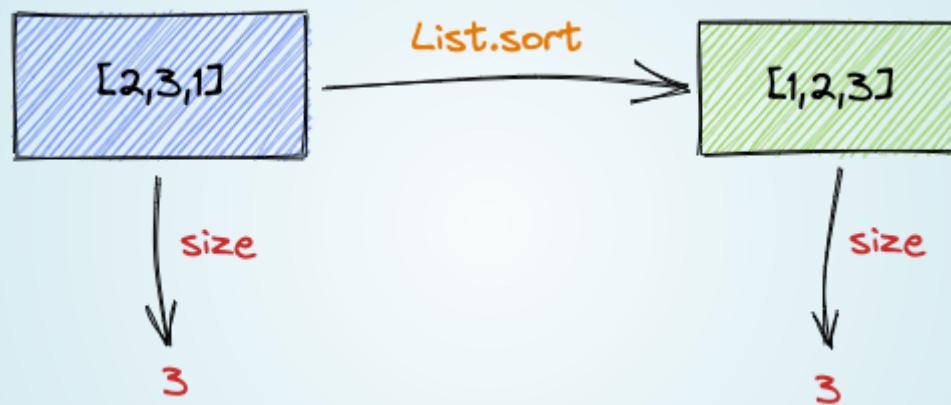


Negar, ordenar =
Ordenar, negar y reverso

IDA Y VUELTA

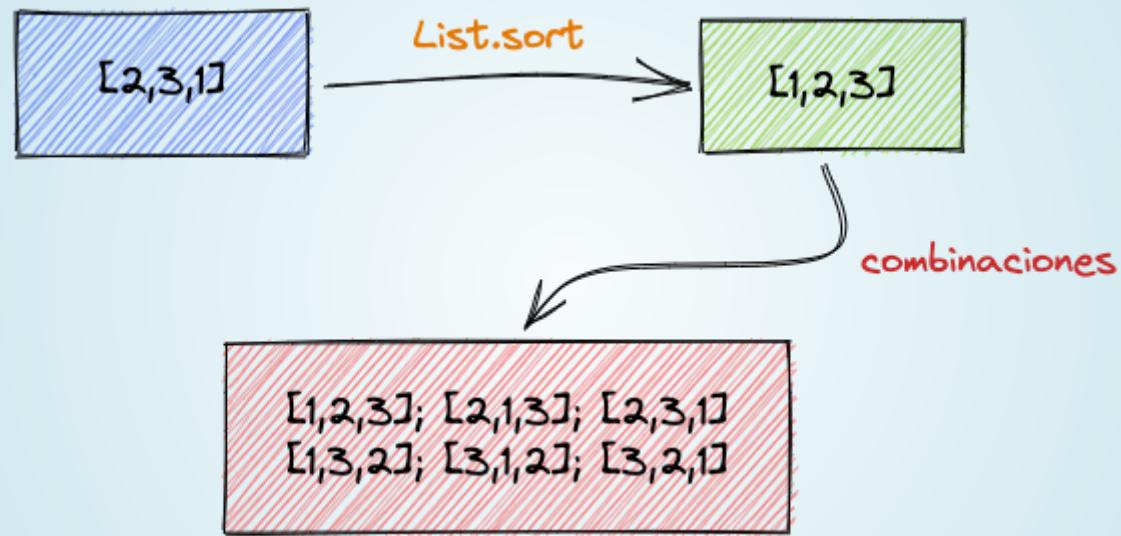


INARIANTES ()



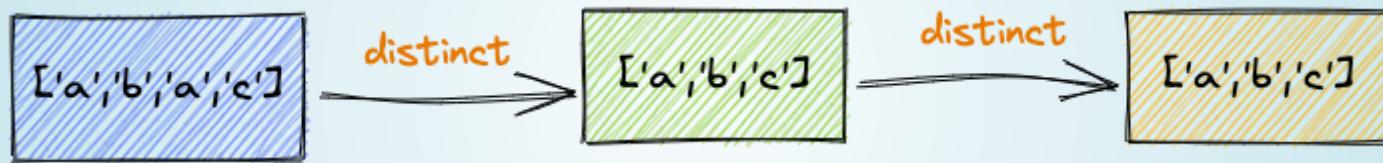
Una propiedad constante de la entrada y salida

INARIANTES (II)



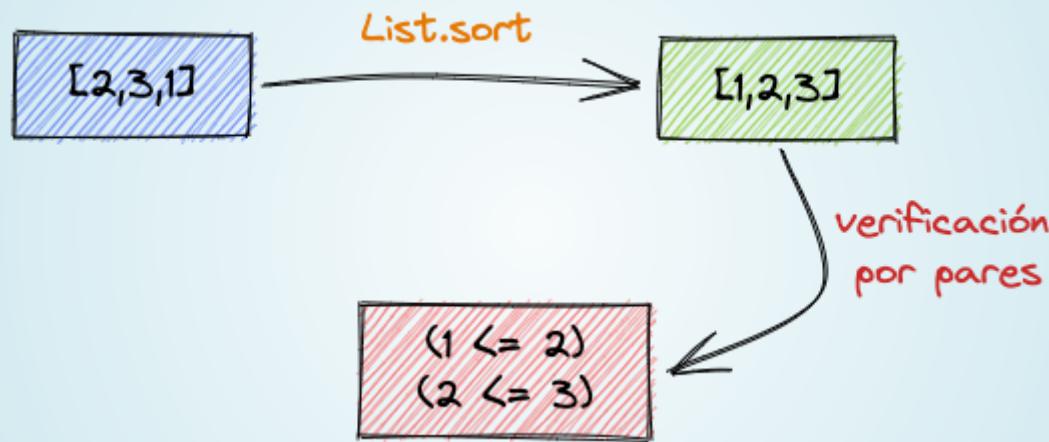
Salida limitada a ciertos valores

IDEMPOTENCIA



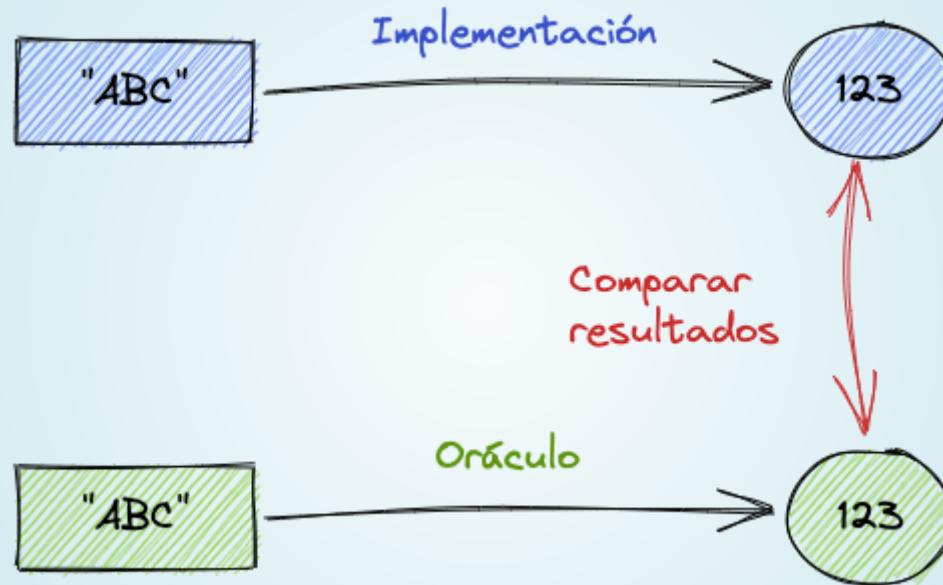
Salida estable tras repetición

REFORMULACIÓN



Problema reducido a su mínima expresión

ORÁCULO



Situación ideal ante entrada aleatoria

¿FUNCIONA CON TDD?

Sí, funciona con TDD

¿FUNCIONA CON TDD?

Sí, funciona con TDD

Quizás no de la forma que esperas

¿FUNCIONA CON TDD?

Sí, funciona con TDD

Quizás no de la forma que esperas

Show me the code </> 

PROPIEDADES VS EJEMPLOS

PROPIEDADES VS EJEMPLOS



- Más generales
- Pueden revelar casos límite
- Especificación más clara

PROPIEDADES VS EJEMPLOS



- Más generales
- Pueden revelar casos límite
- Especificación más clara



- Más difíciles
- Menos directos
- Seguimos necesitando ejemplos

PREGUNTAS



You wanna ask ME
questions ??

FUENTES

Título

User Guide

QuickCheck: A Lightweight Tool
for Random Testing of Haskell
Programs

The lazy programmer's guide to
writing thousands of tests

Property-Based Testing: Let Your
Testing Library Work for You

Autor

Scalacheck

Koen
Claessen,
John Hughes

Scott
Wlaschin

Magda
Stożek

FUENTES

Título

Property (mathematics)

Autor

Wikipedia

Refactoring the three laws of
TDD

Javier
Saldana