Qualité Logicielle : Tests par Propriétés (PBT)

Sébastien Mosser
UQAM, 21.06.2018

Où sommes-nous ?

POO & Réflexivité

Java

Tests Unitaires

JUnit 5

Méthodologies

Propriétés

Charge

Mutations

Intégration

Acceptation

Polytest

Socle Technologique | **Concepts et Utilisation d'outils** | **Création d'outils**

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

> As a first example, we take the standard function `reverse` which reverses a list. This satisfies a number of useful laws, such as
>
> ```
> reverse [x]            =  [x]
> reverse (xs++ys)       =  reverse ys++reverse xs
> reverse (reverse xs)   =  xs
> ```

https://www.eecs.northwestern.edu/~robby/courses/395-495-2009-fall/quick.pdf

Comment tester la fonction "reverse" ?

Tests Unitaires

Définition des cas de tests

Contextes

Assertions

"Avec un bon marteau, tous les problèmes ressemblent à des clous"


```
reverse [x]            =  [x]
reverse (xs++ys)       =  reverse ys++reverse xs
reverse (reverse xs)   =  xs
```

```
@Test public void reverseTheSingleList() {
    assertEquals(
            list(42),
            reverse(list(42)));
}

@Test public void distributeReverseOverConcatenation() {
    List<Integer> xs = list(1, 1, 2);
    List<Integer> ys = list(3, 5, 8);
    List<Integer> reversed = reverse(ys);
    reversed.addAll(reverse(xs));
    assertEquals(
            reverse(list(1, 1, 2, 3, 5, 8)),
            reversed);
}

@Test public void doubleReverse() {
    assertEquals(
            list(1, 1, 2, 3, 5, 8),
            reverse(reverse(list(1, 1, 2, 3, 5, 8))));
}
```

Assertions & Oracles

Java

Un test unitaire démontre un comportement attendu pour une situation donnée

$$3 \times (2 + 5) = (3 \times 2) + (3 \times 5)$$

```java
@Test public void multiplicationIsDistributiveOverAddition() {
    assertEquals((3*(2+5)) , (3*2) + (3*5));
}
```

JUnit 5

Principes :

Exprimer des **propriétés** **symboliques**

Explorer automatiquement l'espace des valeurs

```java
public void multiplicationIsDistributiveOverAddition() {
    do {
        Integer a = anInt();
        Integer b = anInt();
        Integer c = anInt();

        assertEquals(a * (b + c), (a * b) + (a * c));

    } while(notEnoughConfidence());
}
```


Outillage : QuickCheck

Java

Explorer

Symboles

```java
@RunWith(JUnitQuickcheck.class)
public class IntegerProperties {

    @Property
    public void multiplicationIsDistributiveOverAddition(
            Integer a, Integer b, Integer c) {

        assertEquals(a * (b + c), (a * b) + (a * c));
    }
}
```

Propriété

Exploration de l'espace

```java
@Property
public void additionLeadsToBiggerIntegers(Integer a, Integer b) {
    assertTrue(a+b >= a);
    assertTrue(a+b >= b);
}
```

Integer ≠ $\mathbb{N}^+$

Dépassement de capacité !

Configuration par annotation

```java
@Property
public void additionLeadsToBiggerIntegers(
  @InRange(minInt = 0, maxInt = Integer.MAX_VALUE/2) Integer a,
  @InRange(minInt = 0, maxInt = Integer.MAX_VALUE/2) Integer b) {

    assertTrue(a+b >= a);
    assertTrue(a+b >= b);

}
```

Heuristiques d'exploration

BICEP 💪

CORRECT ✅

Définition de Générateurs

```java
public interface Heap {

    boolean isEmpty();

    void insert(Integer elem);
    void merge(Heap that);

    Integer head();
    Integer next();
    List<Integer> flush();

    Heap copy();

}
```

?

Générateur du Tas Vide

Classe

```java
public class EmptyHeapGenerator extends Generator<Heap> {

    public EmptyHeapGenerator() { super(Heap.class); }

    @Override
    public Heap generate(SourceOfRandomness sourceOfRandomness,
            GenerationStatus generationStatus) {

        return MyHeap.empty();

    }

}
```

Fonction de génération

Générateur du Tas Non Vide

```
public class NonEmptyHeapGenerator extends Generator<Heap> {

    public NonEmptyHeapGenerator() { super(Heap.class); }

    private int minElements = 1;
    private int maxElements = 100;

    @Override
    public Heap generate(
      SourceOfRandomness sor, GenerationStatus status) {

        int howManyElements = sor.nextInt(minElements,maxElements);
        Heap theHeap = MyHeap.empty();
        for(int i = 0; i < howManyElements; i++) {
            theHeap.insert(sor.nextInt());
        }
        return theHeap;

    }
}
```

Arbitrairement Complexe

Bilan

- **Exprimer des Propriétés**
  - et non plus des assertions sur des cas connus
- **Explorer l'espace des valeurs**
  - Aléatoire + Heuristiques
  - Chercher un contre-exemple
  - Dans un espace borné

Composition de Générateurs

```
public class HeapGenerator extends Generator<Heap> {

    public HeapGenerator() { super(Heap.class); }

    private boolean isEmpty;

    @Override
    public Heap generate(SourceOfRandomness sor,
      GenerationStatus generationStatus) {
        if(isEmpty) {
            return gen().make(EmptyHeapGenerator.class)
                    .generate(sor, generationStatus);
        } else {
            return gen().make(NonEmptyHeapGenerator.class)
                    .generate(sor, generationStatus);
        }
    }

    public void configure(Empty empty) { this.isEmpty = true; }

}
```