Property-Based Testing mit Java

JavaLand, Brühl
14. März 2018

@johanneslink

johanneslink.net

Softwaretherapeut

"In Deutschland ist die Bezeichnung Therapeut allein oder ergänzt mit bestimmten Begriffen gesetzlich nicht geschützt und daher kein Hinweis auf ein erfolgreich abgeschlossenes Studium oder auch nur fachliche Kompetenz." Quelle: Wikipedia

Examples vs Properties

Ein *Beispiel* zeigt, dass unser Code
bei ganz konkreten Eingaben
ein ganz konkretes Ergebnis liefert.

```
@Example
void reverseList() {
   List<Integer> aList = Arrays.asList(1, 2, 3);
   Collections.reverse(aList);
   assertThat(aList).containsExactly(3, 2, 1);
}
```

Funktioniert *reverse()* nur für die getesteten Beispiele?

Wie repräsentativ sind unsere Tests?

Examples vs Properties

Eine *Property* zeigt, dass unser Code
für eine Klasse von Eingaben (Vorbedingung)
bestimmte allgemeine Eigenschaften (Invariante)
erfüllt.

```
@Property
void reverseList() {
    // Vorbedingung?
    // Invariante?
}
```

Haskell: Quick Check

```
prop_reversed :: [Int] -> Bool
prop_reversed xs =
    reverse (reverse xs) == xs
```

Java: Jqwik

```
@Property
boolean reverseTwiceIsOriginal(@ForAll List<Integer> original) {
    List<Integer> copy = new ArrayList<>(original);
    Collections.reverse(copy);
    Collections.reverse(copy);
    return copy.equals(original);
}
```

Demo 1

- Reverse
- Length of String
- Absolute value of Integer
- Sum of two integers
- Einbindung in Gradle und IntelliJ

Was ist jqwik?

http://jqwik.net

- Eine Test-Engine für die JUnit5–Plattform
- Ein Generator für Testfälle mit
  - zufälligen und typischen Eingabewerten
  - und Kombinationen von Eingabewerten

Was ist jqwik nicht?

- Es ist kein vollständig randomisiertes Testwerkzeug, das man ohne Nachdenken auf sein Programm loslässt.
- Es ist kein Silver Bullet für alle Testprobleme
- Properties werden nicht bewiesen, sondern widerlegt (aka falsifiziert)

```
static <E> List<E> brokenReverse(List<E> aList) {
    if (aList.size() < 4) {
        aList = new ArrayList<>(aList);
        reverse(aList);
    }
    return aList;
}

@Property(shrinking = ShrinkingMode.OFF)
boolean reverseShouldSwapFirstAndLast(@ForAll List<Integer> aList) {
    Assume.that(!aList.isEmpty());
    List<Integer> reversed = brokenReverse(aList);
    return aList.get(0) == reversed.get(aList.size() - 1);
}
```

```
org.opentest4j.AssertionFailedError:
Property [reverseShouldSwapFirstAndLast] falsified with sample
   [[0, -2147483648, 2147483647, -7997, 7997, -3223, -6474, 1915, -7151,
   3102, 4362, 714, 3053, 1919, -445, 7498, -2424, 3016, -5127, -7401, -7946,
   -3801, -305]]
```

```
static <E> List<E> brokenReverse(List<E> aList) {
    if (aList.size() < 4) {
        aList = new ArrayList<>(aList);
        reverse(aList);
    }
    return aList;
}

@Property
boolean reverseShouldSwapFirstAndLast(@ForAll List<Integer> aList) {
    Assume.that(!aList.isEmpty());
    List<Integer> reversed = brokenReverse(aList);
    return aList.get(0) == reversed.get(aList.size() - 1);
}
```

```
org.opentest4j.AssertionFailedError:
Property [reverseShouldSwapFirstAndLast] falsified with sample
   [[0, 0, 0, -1]]
```

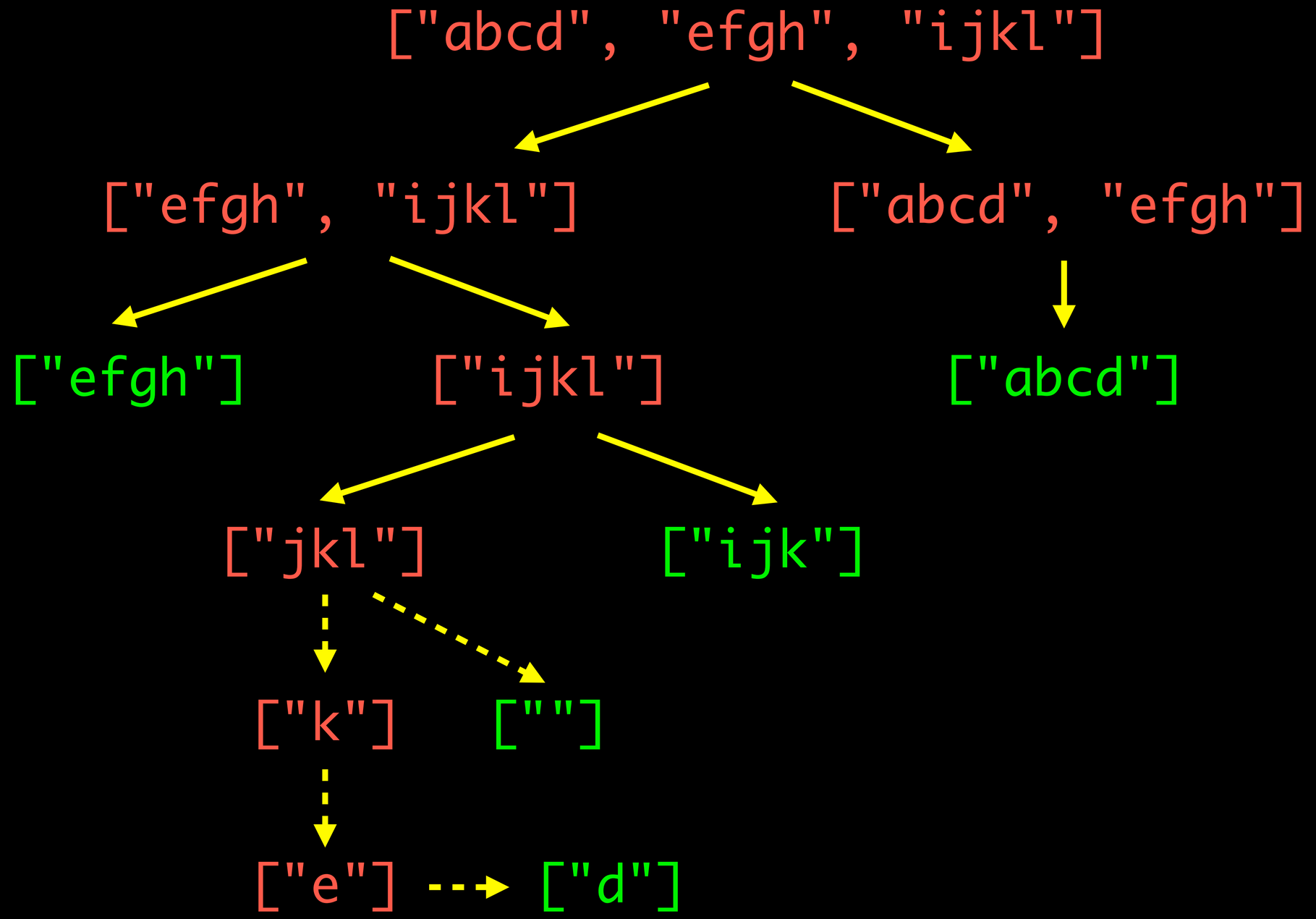
```java
static <E> List<E> brokenReverse(List<E> aList) {
    if (aList.size() < 4) {
        aList = new ArrayList<>(aList);
        reverse(aList);
    }
    return aList;
}

@Property
boolean reverseShouldSwapFirstAndLast(@ForAll List<Integer> aList) {
    Assume.that(!aList.isEmpty());
    List<Integer> reversed = brokenReverse(aList);
    return aList.get(0) == reversed.get(aList.size() - 1);
}
```

```
org.opentest4j.AssertionFailedError:
Property [reverseShouldSwapFirstAndLast] falsified with sample
   [[0, 0, 0, -1]]
```

The Importance of Being Shrunk

- "Schrumpfen" einer falsifizierten Property: Finde das einfachste Eingabe-Beispiel, das immer noch fehlschlägt.
- Manchmal gibt es das einfachste Beispiel nicht, oder die Suche danach würde sehr lange dauern.
- Benutze Heuristiken um Werte zu schrumpfen, z.B.
  - Versuche Zahlen-Werte näher bei Null
  - Verkleinere Listen, Mengen, Arrays


["abcd", "efgh", "ijkl"]
["efgh", "ijkl"]
["abcd", "efgh"]
["efgh"]
["ijkl"]
["abcd"]
["jkl"]
["ijk"]
["k"]
[""]
["e"]
["d"]

Type-based vs Integrated Shrinking

- Type-Based Shrinking: Nur der Typ von Werten dient als Constraint für die Schrumpfversuche
  - Problem: Schrumpfen kann zu Ergebnissen führen, die eigentlich bei der Generierung ausgeschlossen wurden
- Integrated Shrinking: Alle Schritte und Bedingungen der Generierung werden beim Schrumpfen berücksichtig
- jqwik implementiert integriertes Schrumpfen

```java
@Property
boolean shrinkingCanBeComplicated(
    @ForAll("first") String first,
    @ForAll("second") String second
) {
    String aString = first + second;
    return aString.length() > 5 || aString.length() < 4;
}

@Provide
Arbitrary<String> first() {
    return Arbitraries.strings('a', 'z', 1, 10)
        .filter(string -> string.endsWith("h"));
}

@Provide
Arbitrary<String> second() {
    return Arbitraries.strings('0', '9', 0, 10)
        .filter(string -> string.length() >= 1);
}
```

```java
public interface RandomGenerator<T> {
    Shrinkable<T> next(Random random);
}
```

```java
public interface Shrinkable<T> {

    Set<ShrinkResult<Shrinkable<T>>> shrinkNext(Predicate<T> falsifier);

    T value();

    int distance();
}
```

Patterns of PBT

- Obvious Property
- Fuzzying
- Inverse functions
- Idempotent functions
- Commutativity
- Black-box testing
- Induction
- Test oracle
- Invariant properties
- Stateful Testing

Demo: Fizz Buzz

- Zähle aufwärts von 1 bis 100
- "Normale" Zahlen werden normal gezählt
- Vielfache von 3 werden "Fizz" gezählt
- Vielfache von 5 werden "Buzz" gezählt
- Vielfache von 3 und 5 werden "FizzBuzz" gezählt

Probleme beim PBT

- Manchmal sind konkrete Beispiele hilfreicher beim Code-Verstehen
- Interaktionen mit der Außenwelt machen PBT langsam
- Randomisierte Tests können nicht-deterministisch sein

Die Zukunft von jqwik

- Mehr Default-Providers
  - z.B. für java.time.*
- Vollständige statt zufällige Generierung
- Unterstützung für *Stateful Testing*
- Groovy-DSL

Alternative PBT-Tools für Java

- JUnit-Quickcheck: Enge Integration mit JUnit 4
- QuickTheories: Arbeitet mit beliebigen Test-Libraries zusammen
- Vavr: Die funktionale Java-Bibliothek hat auch ein eigenes PBT-Modul

jqwik auf Github:

http://github.com/jlink/jqwik

Mitstreiter gesucht!

Code:

http://github.com/jlink/property-based-testing