Lecture: Property-Based Testing

```
import org.scalatest.FunSuite
import org.scalatest.prop.GeneratorDrivenPropertyChecks
import org.scalacheck._

class ListTestSuite extends FunSuite with GeneratorDrivenPropertyChecks {

  def split(sep: Char, astring: String): List[String] = {
    def splitRec(chars: List[Char]): List[String] = {
      val prefix = chars.takeWhile(ch => ch != sep).mkString
      chars.dropWhile(ch => ch != sep) match {
        case Nil => Nil
        case _ :: suffix => prefix :: splitRec(suffix)
      }
    }
    splitRec(astring.toList)
  }

  def join(sep: Char, strings: List[String]): String = strings match {
    case Nil => ""
    case List(x) => x
    case x1 :: x2 :: xs => x1 + sep + join(sep, x2 :: xs)
  }

  test("reverse-reverse") {
    forAll { (alist: List[Int]) =>
      assert(alist.reverse.reverse == alist)
    }
  }

  test("reverse concat") {
    forAll { (alist1: List[Int], alist2: List[Int]) =>
      assert((alist1 ++ alist2).reverse == alist2.reverse ++ alist1.reverse)
    }
  }

  test("concat distributes over map") {
    def incr(x: Int) = x + 1

    forAll { (lst1: List[Int], lst2: List[Int]) =>
      assert((lst1 ++ lst2).map(incr) == lst1.map(incr) ++ lst2.map(incr))
    }
  }

  def qsort(lst: List[Int]): List[Int] = {
    lst match {
      case Nil => Nil
      case pivot :: rest => {
        val lhs = rest.filter(_ < pivot)
        val rhs = rest.filter(_ >= pivot)
        qsort(lhs) ++ List(pivot) ++ qsort(rhs)
      }
    }
  }

  def isSorted(lst: List[Int]): Boolean = lst match {
    case Nil => true
    case List(x) => true
    case x :: y:: rest => x <= y && isSorted(y :: rest)
  }

  test("qsort checking") {
    forAll { (lst: List[Int]) =>
      val sortedList = qsort(lst)
      assert(isSorted(sortedList))
      assert(sortedList.length == lst.length)
    }
  }

}
```

Figure 22.1: Checking properties of simple list processing functions.

```
import org.scalatest.FunSuite
import org.scalatest.prop.GeneratorDrivenPropertyChecks
import org.scalacheck._

class FibTestSuite extends FunSuite with GeneratorDrivenPropertyChecks {

  def fib(n: Int): Int = {
    if (n == 0) 1
    else if (n == 1) 1
    else fib(n - 1) + fib(n - 2)
  }

  def optfib(n2: Int, n1: Int, n: Int, i: Int): Int = {
    info(s"optfib($n2, $n1, $n, $i)")
    if (i == 0) n else optfib(n1 ,n, n + n1, i - 1)
  }

  def altFib(n: Int): Int = {
    if (n <= 1) 1
    else optfib(1, 1, 2, n - 2)
  }

  test("altFib(n) = fib(n)") {
    forAll(Gen.choose(0, 20)) { n =>
      assert(altFib(n) == fib(n))
    }
  }

}
```

Figure 22.2: Checking that an optimized implementation of the fibonacci function is equivalent to the naive implementation.

```
import org.scalatest._
import org.scalatest.prop._
import org.scalacheck._
import JoinList._

class JoinListSuite extends FunSuite with GeneratorDrivenPropertyChecks {

  def isEven(n: Int): Boolean = n % 2 == 0

  test("filter for joinlists") {

    forAll { (lst: List[Int]) =>
      assert(toList(filter(isEven, fromList(lst))) == lst.filter(isEven))
    }
  }

  test("flatten for joinlists") {
    forAll { (lst: List[List[Int]]) =>
      val jl: JoinList[JoinList[Int]] = fromList(lst.map(fromList))
      assert (toList(flatten(jl)) == lst.flatten)
    }
  }

}
```

Figure 22.3: Checking properties of join lists.