# Quiz: Property Based Testing

Find and fix the bug in the second property test below.

```
1  object BrokenSpec extends org.scalacheck.Properties("broken"):
2
3    // Our usual sequence Option-List. No surprises. Don't read if you remember
4    def sequence[A] (aos: List[Option[A]]): Option[List[A]] =
5      aos.foldRight[Option[List[A]]] (Some (Nil)) {
6        (oa, z) => z.flatMap { l => oa map (_::l) } }
7
8    property("Returns Some if the list has no failures") =
9      // Override the default generator to create lists of Some of A
10     given def arbList[A] (using arb: Arbitrary[List[A]]) =
11       val genL = arb.arbitrary.map { l => l.map { Some(_) } }
12       Arbitrary[List[Option[A]]] (genL)
13     forAll { (l: List[Option[Int]]) => sequence(l).isDefined }
14
15   property("Returns None if the list has one failure") =
16     forAll { (l: List[Option[Int]]) => sequence(l).isEmpty }
```

# Quiz: Property Based Testing

The bug is fixed in the second property test below.

```
1 object OptionSpec extends org.scalacheck.Properties("option"):
2
3   def sequence[A] (aos: List[Option[A]]) : Option[List[A]] =
4     aos.foldRight[Option[List[A]]] (Some(Nil)) {
5       (oa,z) => z.flatMap (l => oa.map (a => a::l)) }
6
7   property("Returns Some if the list has no failures") =
8     // Override the default generator to create lists of Some of A
9     given def arbList[A] (using arb: Arbitrary[List[A]]) =
10      val genL = arb.arbitrary.map { l => l.map { Some(_) } }
11      Arbitrary[List[Option[A]]](genL)
12    forAll { (l: List[Option[Int]]) => sequence(l).isDefined }
13
14  property("Returns None if the list has one failure") =
15    given def arbFailingList[A] (using arb: Arbitrary[List[Option[A]]]) =
16      val genL = arb.arbitrary.filter { l => l.exists { _.isEmpty } }
17      Arbitrary(genL)
18    forAll { (l: List[Option[Int]]) => sequence(l).isEmpty }
```