Grzegorz Balcerek

# Property-based testing with ScalaCheck

```
scala>
```

```
scala> import org.scalacheck._, Prop._
```

```
scala> import org.scalacheck._, Prop._
import org.scalacheck._
import Prop._

scala>
```

```
scala> import org.scalacheck._, Prop._
import org.scalacheck._
import Prop._

scala> def times2property = forAll( (x:Int) => x+x == 2*x )
```

```
scala> import org.scalacheck._, Prop._
import org.scalacheck._
import Prop._

scala> def times2property = forAll( (x:Int) => x+x == 2*x )
times2property: org.scalacheck.Prop

scala>
```

```
scala> import org.scalacheck._, Prop._
import org.scalacheck._
import Prop._

scala> def times2property = forAll( (x:Int) => x+x == 2*x )
times2property: org.scalacheck.Prop

scala> times2property.check
```

```
scala> import org.scalacheck._, Prop._
import org.scalacheck._
import Prop._

scala> def times2property = forAll( (x:Int) => x+x == 2*x )
times2property: org.scalacheck.Prop

scala> times2property.check
+ OK, passed 100 tests.
```

```
scala>
```

```
scala> def times2property2 = forAll{ (x:Int) =>
     |    println(x)
     |    x+x == 2*x
     | }
```

```
scala> def times2property2 = forAll{ (x:Int) =>
     |    println(x)
     |    x+x == 2*x
     | }
times2property2: org.scalacheck.Prop

scala>
```

```
scala> def times2property2 = forAll{ (x:Int) =>
     |    println(x)
     |    x+x == 2*x
     | }
times2property2: org.scalacheck.Prop

scala> times2property2.check
```

```
scala> def times2property2 = forAll{ (x:Int) =>
     |    println(x)
     |    x+x == 2*x
     | }
times2property2: org.scalacheck.Prop

scala> times2property2.check
-2147483648
-2131273578
-1027452779
1
546825679
0
(...)
-2147483648
2015403282
+ OK, passed 100 tests.
```

```
scala>
```

```
scala> 12.abs
```

```
scala> 12.abs
res2: Int = 12

scala>
```

```
scala> 12.abs
res2: Int = 12

scala> -12.abs
```

```
scala> 12.abs
res2: Int = 12

scala> -12.abs
res3: Int = 12

scala>
```

```
scala> 12.abs
res2: Int = 12

scala> -12.abs
res3: Int = 12

scala> def absNotNegative = forAll( (x:Int) => x.abs >= 0 )
```

```
scala> 12.abs
res2: Int = 12

scala> -12.abs
res3: Int = 12

scala> def absNotNegative = forAll( (x:Int) => x.abs >= 0 )
absNotNegative: org.scalacheck.Prop

scala>
```

```
scala> 12.abs
res2: Int = 12

scala> -12.abs
res3: Int = 12

scala> def absNotNegative = forAll( (x:Int) => x.abs >= 0 )
absNotNegative: org.scalacheck.Prop

scala> absNotNegative.check
```

```
scala> 12.abs
res2: Int = 12

scala> -12.abs
res3: Int = 12

scala> def absNotNegative = forAll( (x:Int) => x.abs >= 0 )
absNotNegative: org.scalacheck.Prop

scala> absNotNegative.check
! Falsified after 2 passed tests.
> ARG_0: -2147483648

scala>
```

```
scala> 12.abs
res2: Int = 12

scala> -12.abs
res3: Int = 12

scala> def absNotNegative = forAll( (x:Int) => x.abs >= 0 )
absNotNegative: org.scalacheck.Prop

scala> absNotNegative.check
! Falsified after 2 passed tests.
> ARG_0: -2147483648

scala> Integer.MIN_VALUE
```

```
scala> 12.abs
res2: Int = 12

scala> -12.abs
res3: Int = 12

scala> def absNotNegative = forAll( (x:Int) => x.abs >= 0 )
absNotNegative: org.scalacheck.Prop

scala> absNotNegative.check
! Falsified after 2 passed tests.
> ARG_0: -2147483648

scala> Integer.MIN_VALUE
res5: Int = -2147483648

scala>
```

```
scala> 12.abs
res2: Int = 12

scala> -12.abs
res3: Int = 12

scala> def absNotNegative = forAll( (x:Int) => x.abs >= 0 )
absNotNegative: org.scalacheck.Prop

scala> absNotNegative.check
! Falsified after 2 passed tests.
> ARG_0: -2147483648

scala> Integer.MIN_VALUE
res5: Int = -2147483648

scala> Integer.MIN_VALUE.abs
```

```
scala> 12.abs
res2: Int = 12

scala> -12.abs
res3: Int = 12

scala> def absNotNegative = forAll( (x:Int) => x.abs >= 0 )
absNotNegative: org.scalacheck.Prop

scala> absNotNegative.check
! Falsified after 2 passed tests.
> ARG_0: -2147483648

scala> Integer.MIN_VALUE
res5: Int = -2147483648

scala> Integer.MIN_VALUE.abs
res6: Int = -2147483648
```

```
scala>
```

```
scala> def absExceptMinNotNegative = forAll( (x:Int) =>
     |   x != Integer.MIN_VALUE ==> x.abs >= 0 )
```

```
scala> def absExceptMinNotNegative = forAll( (x:Int) =>
     |   x != Integer.MIN_VALUE ==> x.abs >= 0 )
absExceptMinNotNegative: org.scalacheck.Prop

scala>
```

```
scala> def absExceptMinNotNegative = forAll( (x:Int) =>
     |   x != Integer.MIN_VALUE ==> x.abs >= 0 )
absExceptMinNotNegative: org.scalacheck.Prop

scala> absExceptMinNotNegative.check
```

```
scala> def absExceptMinNotNegative = forAll( (x:Int) =>
     |   x != Integer.MIN_VALUE ==> x.abs >= 0 )
absExceptMinNotNegative: org.scalacheck.Prop

scala> absExceptMinNotNegative.check
+ OK, passed 100 tests.
```

```
scala>
```

```
scala> forAll( (x:Int) => x/x == 1 ).check
```

```
scala> forAll( (x:Int) => x/x == 1 ).check
! Exception raised on property evaluation.
> ARG_0: 0
> Exception: java.lang.ArithmeticException: / by zero
(...)

scala>
```

```
scala> forAll( (x:Int) => x/x == 1 ).check
! Exception raised on property evaluation.
> ARG_0: 0
> Exception: java.lang.ArithmeticException: / by zero
(...)

scala> forAll( (x:Int) => x != 0 ==> (x/x == 1) ).check
```

```
scala> forAll( (x:Int) => x/x == 1 ).check
! Exception raised on property evaluation.
> ARG_0: 0
> Exception: java.lang.ArithmeticException: / by zero
(...)

scala> forAll( (x:Int) => x != 0 ==> (x/x == 1) ).check
+ OK, passed 100 tests.
```

```
scala>
```

```
scala> forAll( (a:Set[Int],b:Set[Int]) =>
     |   (a++b).size >= a.size ).check
```

```
scala> forAll( (a:Set[Int],b:Set[Int]) =>
     |   (a++b).size >= a.size ).check
+ OK, passed 100 tests.

scala>
```

```
scala> forAll( (a:Set[Int],b:Set[Int]) =>
     |   (a++b).size >= a.size ).check
+ OK, passed 100 tests.

scala> forAll( (a:List[Char],b:List[Char],c:List[Char]) =>
     |   (a++b)++c == a++(b++c) ).check
```

```
scala> forAll( (a:Set[Int],b:Set[Int]) =>
     |   (a++b).size >= a.size ).check
+ OK, passed 100 tests.

scala> forAll( (a:List[Char],b:List[Char],c:List[Char]) =>
     |   (a++b)++c == a++(b++c) ).check
+ OK, passed 100 tests.
```

```
def maxOption(xs: List[Int]): Option[Int] = {
  def max(acc: Option[Int], y: Int): Option[Int] = acc match {
    case None => Some(y)
    case Some(x) => Some(if (x > y) x else y)
  }
  xs.foldLeft[Option[Int]](None)(max)
}
```

```
def maxOption(xs: List[Int]): Option[Int] = {
  def max(acc: Option[Int], y: Int): Option[Int] = acc match {
    case None => Some(y)
    case Some(x) => Some(if (x > y) x else y)
  }
  xs.foldLeft[Option[Int]](None)(max)
}

scala>
```

```
def maxOption(xs: List[Int]): Option[Int] = {
  def max(acc: Option[Int], y: Int): Option[Int] = acc match {
    case None => Some(y)
    case Some(x) => Some(if (x > y) x else y)
  }
  xs.foldLeft[Option[Int]](None)(max)
}

scala> def maxOptionProperty =
     |   forAll( (xs: List[Int]) => maxOption(xs) match {
     |     case None => xs == Nil
     |     case Some(max) =>
     |       !xs.isEmpty && xs.forall( (x:Int) => max >= x )
     |   })
```

```
def maxOption(xs: List[Int]): Option[Int] = {
  def max(acc: Option[Int], y: Int): Option[Int] = acc match {
    case None => Some(y)
    case Some(x) => Some(if (x > y) x else y)
  }
  xs.foldLeft[Option[Int]](None)(max)
}

scala> def maxOptionProperty =
     |   forAll( (xs: List[Int]) => maxOption(xs) match {
     |     case None => xs == Nil
     |     case Some(max) =>
     |       !xs.isEmpty && xs.forall( (x:Int) => max >= x )
     |   })
maxOptionProperty: org.scalacheck.Prop

scala>
```

```
def maxOption(xs: List[Int]): Option[Int] = {
  def max(acc: Option[Int], y: Int): Option[Int] = acc match {
    case None => Some(y)
    case Some(x) => Some(if (x > y) x else y)
  }
  xs.foldLeft[Option[Int]](None)(max)
}

scala> def maxOptionProperty =
     |   forAll( (xs: List[Int]) => maxOption(xs) match {
     |     case None => xs == Nil
     |     case Some(max) =>
     |       !xs.isEmpty && xs.forall( (x:Int) => max >= x )
     |   })
maxOptionProperty: org.scalacheck.Prop

scala> maxOptionProperty.check
```

```
def maxOption(xs: List[Int]): Option[Int] = {
  def max(acc: Option[Int], y: Int): Option[Int] = acc match {
    case None => Some(y)
    case Some(x) => Some(if (x > y) x else y)
  }
  xs.foldLeft[Option[Int]](None)(max)
}

scala> def maxOptionProperty =
     |   forAll( (xs: List[Int]) => maxOption(xs) match {
     |     case None => xs == Nil
     |     case Some(max) =>
     |       !xs.isEmpty && xs.forall( (x:Int) => max >= x )
     |   })
maxOptionProperty: org.scalacheck.Prop

scala> maxOptionProperty.check
+ OK, passed 100 tests.
```

# Test Parameters

```
scala>
```

```
scala> import Test.Parameters
```

```
scala> import Test.Parameters
import Test.Parameters

scala>
```

```
scala> import Test.Parameters
import Test.Parameters

scala> def myParameters = new Parameters.Default {
     |   override val minSuccessfulTests = 300
     | }
```

```
scala> import Test.Parameters
import Test.Parameters

scala> def myParameters = new Parameters.Default {
     |   override val minSuccessfulTests = 300
     | }
myParameters: org.scalacheck.Test.Parameters.Default

scala>
```

```
scala> import Test.Parameters
import Test.Parameters

scala> def myParameters = new Parameters.Default {
     |   override val minSuccessfulTests = 300
     | }
myParameters: org.scalacheck.Test.Parameters.Default

scala> property1.check(myParameters)
```

```
scala> import Test.Parameters
import Test.Parameters

scala> def myParameters = new Parameters.Default {
     |   override val minSuccessfulTests = 300
     | }
myParameters: org.scalacheck.Test.Parameters.Default

scala> property1.check(myParameters)
+ OK, passed 300 tests.
```

```
scala>
```

```
scala> def squareIsSmaller = Prop.forAll( (x:Double) => x*x < x )
```

```
scala> def squareIsSmaller = Prop.forAll( (x:Double) => x*x < x )
squareIsSmaller: org.scalacheck.Prop

scala>
```

```
scala> def squareIsSmaller = Prop.forAll( (x:Double) => x*x < x )
squareIsSmaller: org.scalacheck.Prop

scala> squareIsSmaller.check
```

```
scala> def squareIsSmaller = Prop.forAll( (x:Double) => x*x < x )
squareIsSmaller: org.scalacheck.Prop

scala> squareIsSmaller.check
! Falsified after 0 passed tests.
> ARG_0: -4.43205713363996O3E307

scala>
```

```
scala> def squareIsSmaller = Prop.forAll( (x:Double) => x*x < x )
squareIsSmaller: org.scalacheck.Prop

scala> squareIsSmaller.check
! Falsified after 0 passed tests.
> ARG_0: -4.43205713363996O3E307

scala> squareIsSmaller.check
```

```
scala> def squareIsSmaller = Prop.forAll( (x:Double) => x*x < x )
squareIsSmaller: org.scalacheck.Prop

scala> squareIsSmaller.check
! Falsified after 0 passed tests.
> ARG_0: -4.4320571336399603E307

scala> squareIsSmaller.check
! Falsified after 0 passed tests.
> ARG_0: 2.90859936081905 8E307

scala>
```

```
scala> def squareIsSmaller = Prop.forAll( (x:Double) => x*x < x )
squareIsSmaller: org.scalacheck.Prop

scala> squareIsSmaller.check
! Falsified after 0 passed tests.
> ARG_0: -4.4320571336399603E307

scala> squareIsSmaller.check
! Falsified after 0 passed tests.
> ARG_0: 2.90859936081905 8E307

scala> squareIsSmaller.check
```

```
scala> def squareIsSmaller = Prop.forAll( (x:Double) => x*x < x )
squareIsSmaller: org.scalacheck.Prop

scala> squareIsSmaller.check
! Falsified after 0 passed tests.
> ARG_0: -4.4320571336399603E307

scala> squareIsSmaller.check
! Falsified after 0 passed tests.
> ARG_0: 2.90859936081905 8E307

scala> squareIsSmaller.check
! Falsified after 0 passed tests.
> ARG_0: 0.0
```

```
scala>
```

```
scala> def myParameters = new Parameters.Default{
     |   override val rng = new Random(19)
     | }
```

```
scala> def myParameters = new Parameters.Default{
     |   override val rng = new Random(19)
     | }
myParameters: org.scalacheck.Test.Parameters.Default

scala>
```

```
scala> def myParameters = new Parameters.Default{
     |   override val rng = new Random(19)
     | }
myParameters: org.scalacheck.Test.Parameters.Default

scala> squareIsSmaller.check(myParameters)
```

```
scala> def myParameters = new Parameters.Default{
     |   override val rng = new Random(19)
     | }
myParameters: org.scalacheck.Test.Parameters.Default

scala> squareIsSmaller.check(myParameters)
! Falsified after 0 passed tests.
> ARG_0: 8.988465674311579E307

scala>
```

```
scala> def myParameters = new Parameters.Default{
     |   override val rng = new Random(19)
     | }
myParameters: org.scalacheck.Test.Parameters.Default

scala> squareIsSmaller.check(myParameters)
! Falsified after 0 passed tests.
> ARG_0: 8.988465674311579E307

scala> squareIsSmaller.check(myParameters)
```

```
scala> def myParameters = new Parameters.Default{
     |   override val rng = new Random(19)
     | }
myParameters: org.scalacheck.Test.Parameters.Default

scala> squareIsSmaller.check(myParameters)
! Falsified after 0 passed tests.
> ARG_0: 8.988465674311579E307

scala> squareIsSmaller.check(myParameters)
! Falsified after 0 passed tests.
> ARG_0: 8.988465674311579E307

scala>
```

```
scala> def myParameters = new Parameters.Default{
     |   override val rng = new Random(19)
     | }
myParameters: org.scalacheck.Test.Parameters.Default

scala> squareIsSmaller.check(myParameters)
! Falsified after 0 passed tests.
> ARG_0: 8.988465674311579E307

scala> squareIsSmaller.check(myParameters)
! Falsified after 0 passed tests.
> ARG_0: 8.988465674311579E307

scala> squareIsSmaller.check(myParameters)
```

```
scala> def myParameters = new Parameters.Default{
     |   override val rng = new Random(19)
     | }
myParameters: org.scalacheck.Test.Parameters.Default

scala> squareIsSmaller.check(myParameters)
! Falsified after 0 passed tests.
> ARG_0: 8.988465674311579E307

scala> squareIsSmaller.check(myParameters)
! Falsified after 0 passed tests.
> ARG_0: 8.988465674311579E307

scala> squareIsSmaller.check(myParameters)
! Falsified after 0 passed tests.
> ARG_0: 8.988465674311579E307
```

# Grouping Properties

```
import org.scalacheck._, Prop._
object SetProperties extends Properties("Set Properties") {
  property("subset1") =
    forAll( (s1:Set[Byte], s2:Set[Byte]) => s1.subsetOf(s1++s2) )
  property("subset2") =
    forAll( (s1:Set[Byte], s2:Set[Byte]) => s2.subsetOf(s1++s2) )
  property("max1") =
    forAll( (s:Set[Byte]) => s.nonEmpty ==> s.contains(s.max) )
}
```

```
import org.scalacheck._, Prop._
object SetProperties extends Properties("Set Properties") {
  property("subset1") =
    forAll( (s1:Set[Byte], s2:Set[Byte]) => s1.subsetOf(s1++s2) )
  property("subset2") =
    forAll( (s1:Set[Byte], s2:Set[Byte]) => s2.subsetOf(s1++s2) )
  property("max1") =
    forAll( (s:Set[Byte]) => s.nonEmpty ==> s.contains(s.max) )
}
object ListProperties extends Properties("List Properties") {
  property("reverse1") =
    forAll( (a:List[Byte]) => a == a.reverse.reverse )
  property("reverse2") =
    forAll( (a:List[Byte]) => a.headOption == a.reverse.lastOption )
}
```

```
import org.scalacheck._, Prop._
object SetProperties extends Properties("Set Properties") {
  property("subset1") =
    forAll( (s1:Set[Byte], s2:Set[Byte]) => s1.subsetOf(s1++s2) )
  property("subset2") =
    forAll( (s1:Set[Byte], s2:Set[Byte]) => s2.subsetOf(s1++s2) )
  property("max1") =
    forAll( (s:Set[Byte]) => s.nonEmpty ==> s.contains(s.max) )
}
object ListProperties extends Properties("List Properties") {
  property("reverse1") =
    forAll( (a:List[Byte]) => a == a.reverse.reverse )
  property("reverse2") =
    forAll( (a:List[Byte]) => a.headOption == a.reverse.lastOption )
}
object CollectionsProperties extends
    Properties("Collections Properties") {
  include(SetProperties)
  include(ListProperties)
}
```

```
scala>
```

```
scala> SetProperties.check
```

```
scala> SetProperties.check
+ Set Properties.subset1: OK, passed 100 tests.
+ Set Properties.subset2: OK, passed 100 tests.
+ Set Properties.max1: OK, passed 100 tests.

scala>
```

```
scala> SetProperties.check
+ Set Properties.subset1: OK, passed 100 tests.
+ Set Properties.subset2: OK, passed 100 tests.
+ Set Properties.max1: OK, passed 100 tests.

scala> ListProperties.check
```

```
scala> SetProperties.check
+ Set Properties.subset1: OK, passed 100 tests.
+ Set Properties.subset2: OK, passed 100 tests.
+ Set Properties.max1: OK, passed 100 tests.

scala> ListProperties.check
+ List Properties.reverse1: OK, passed 100 tests.
+ List Properties.reverse2: OK, passed 100 tests.

scala>
```

```
scala> SetProperties.check
+ Set Properties.subset1: OK, passed 100 tests.
+ Set Properties.subset2: OK, passed 100 tests.
+ Set Properties.max1: OK, passed 100 tests.

scala> ListProperties.check
+ List Properties.reverse1: OK, passed 100 tests.
+ List Properties.reverse2: OK, passed 100 tests.

scala> CollectionsProperties.check
```

```
scala> SetProperties.check
+ Set Properties.subset1: OK, passed 100 tests.
+ Set Properties.subset2: OK, passed 100 tests.
+ Set Properties.max1: OK, passed 100 tests.

scala> ListProperties.check
+ List Properties.reverse1: OK, passed 100 tests.
+ List Properties.reverse2: OK, passed 100 tests.

scala> CollectionsProperties.check
+ Collections Properties.Set Properties.subset1: OK, passed 100 tests.
+ Collections Properties.Set Properties.subset2: OK, passed 100 tests.
+ Collections Properties.Set Properties.max1: OK, passed 100 tests.
+ Collections Properties.List Properties.reverse1: OK, passed 100 tests.
+ Collections Properties.List Properties.reverse2: OK, passed 100 tests.

scala>
```

$

```
$ scala CollectionsProperties
```

```
$ scala CollectionsProperties
+ Collections Properties.Set Properties.subset1: OK, passed 100 tests.
+ Collections Properties.Set Properties.subset2: OK, passed 100 tests.
+ Collections Properties.Set Properties.max1: OK, passed 100 tests.
+ Collections Properties.List Properties.reverse1: OK, passed 100 tests.
+ Collections Properties.List Properties.reverse2: OK, passed 100 tests.

$
```

```
$ scala CollectionsProperties
+ Collections Properties.Set Properties.subset1: OK, passed 100 tests.
+ Collections Properties.Set Properties.subset2: OK, passed 100 tests.
+ Collections Properties.Set Properties.max1: OK, passed 100 tests.
+ Collections Properties.List Properties.reverse1: OK, passed 100 tests.
+ Collections Properties.List Properties.reverse2: OK, passed 100 tests.

$ scala SetProperties -minSuccessfulTests 200
```

```
$ scala CollectionsProperties
+ Collections Properties.Set Properties.subset1: OK, passed 100 tests.
+ Collections Properties.Set Properties.subset2: OK, passed 100 tests.
+ Collections Properties.Set Properties.max1: OK, passed 100 tests.
+ Collections Properties.List Properties.reverse1: OK, passed 100 tests.
+ Collections Properties.List Properties.reverse2: OK, passed 100 tests.

$ scala SetProperties -minSuccessfulTests 200
+ Set Properties.subset1: OK, passed 200 tests.
+ Set Properties.subset2: OK, passed 200 tests.
+ Set Properties.max1: OK, passed 200 tests.

$
```

# Test Data Generation

```
scala>
```

```
scala> forAll( (x:Char) =>
     |   (x >= 'A' && x <= 'Z') ==>
     |   (x+32).toChar.isLower ).check
```

```
scala> forAll( (x:Char) =>
     |   (x >= 'A' && x <= 'Z') ==>
     |   (x+32).toChar.isLower ).check
! Gave up after only 0 passed tests. 101 tests were discarded.

scala>
```

```
scala> forAll( (x:Char) =>
     |   (x >= 'A' && x <= 'Z') ==>
     |   (x+32).toChar.isLower ).check
! Gave up after only 0 passed tests. 101 tests were discarded.

scala> forAll( (x:Byte) =>
     |   (x >= 'A' && x <= 'Z') ==>
     |   (x+32).toChar.isLower ).check
```

```
scala> forAll( (x:Char) =>
     |   (x >= 'A' && x <= 'Z') ==>
     |   (x+32).toChar.isLower ).check
! Gave up after only 0 passed tests. 101 tests were discarded.

scala> forAll( (x:Byte) =>
     |   (x >= 'A' && x <= 'Z') ==>
     |   (x+32).toChar.isLower ).check
! Gave up after only 7 passed tests. 94 tests were discarded.
```

```
scala>
```

```
scala> import Gen._
```

```
scala> import Gen._
import Gen._

scala>
```

```
scala> import Gen._
import Gen._

scala> forAll(choose('A','Z'))( (x:Char) =>
     |   (x+32).toChar.isLower ).check
```

```
scala> import Gen._
import Gen._

scala> forAll(choose('A','Z'))( (x:Char) =>
     |   (x+32).toChar.isLower ).check
+ OK, passed 100 tests.

scala>
```

```
scala> import Gen._
import Gen._

scala> forAll(choose('A','Z'))( (x:Char) =>
     |   (x+32).toChar.isLower ).check
+ OK, passed 100 tests.

scala> choose('A','Z').sample
```

```
scala> import Gen._
import Gen._

scala> forAll(choose('A','Z'))( (x:Char) =>
     |   (x+32).toChar.isLower ).check
+ OK, passed 100 tests.

scala> choose('A','Z').sample
res26: Option[Char] = Some(W)

scala>
```

```
scala> import Gen._
import Gen._

scala> forAll(choose('A','Z'))( (x:Char) =>
     |   (x+32).toChar.isLower ).check
+ OK, passed 100 tests.

scala> choose('A','Z').sample
res26: Option[Char] = Some(W)

scala> choose('A','Z').sample
```

```
scala> import Gen._
import Gen._

scala> forAll(choose('A','Z'))( (x:Char) =>
     |   (x+32).toChar.isLower ).check
+ OK, passed 100 tests.

scala> choose('A','Z').sample
res26: Option[Char] = Some(W)

scala> choose('A','Z').sample
res27: Option[Char] = Some(C)

scala>
```

```
scala> import Gen._
import Gen._

scala> forAll(choose('A','Z'))( (x:Char) =>
     |   (x+32).toChar.isLower ).check
+ OK, passed 100 tests.

scala> choose('A','Z').sample
res26: Option[Char] = Some(W)

scala> choose('A','Z').sample
res27: Option[Char] = Some(C)

scala> choose('A','Z').sample
```

```
scala> import Gen._
import Gen._

scala> forAll(choose('A','Z'))( (x:Char) =>
     |   (x+32).toChar.isLower ).check
+ OK, passed 100 tests.

scala> choose('A','Z').sample
res26: Option[Char] = Some(W)

scala> choose('A','Z').sample
res27: Option[Char] = Some(C)

scala> choose('A','Z').sample
res28: Option[Char] = Some(M)

scala>
```

```
scala> import Gen._
import Gen._

scala> forAll(choose('A','Z'))( (x:Char) =>
     |   (x+32).toChar.isLower ).check
+ OK, passed 100 tests.

scala> choose('A','Z').sample
res26: Option[Char] = Some(W)

scala> choose('A','Z').sample
res27: Option[Char] = Some(C)

scala> choose('A','Z').sample
res28: Option[Char] = Some(M)

scala> choose('A','Z').sample
```

```
scala> import Gen._
import Gen._

scala> forAll(choose('A','Z'))( (x:Char) =>
     |   (x+32).toChar.isLower ).check
+ OK, passed 100 tests.

scala> choose('A','Z').sample
res26: Option[Char] = Some(W)

scala> choose('A','Z').sample
res27: Option[Char] = Some(C)

scala> choose('A','Z').sample
res28: Option[Char] = Some(M)

scala> choose('A','Z').sample
res29: Option[Char] = Some(E)
```

```
scala>
```

```
scala> import Arbitrary.arbitrary
```

```
scala> import Arbitrary.arbitrary
import Arbitrary.arbitrary

scala>
```

```
scala> import Arbitrary.arbitrary
import Arbitrary.arbitrary

scala> forAll(arbitrary[Byte])( (x:Byte) => x*x >= 0 ).check
```

```
scala> import Arbitrary.arbitrary
import Arbitrary.arbitrary

scala> forAll(arbitrary[Byte])( (x:Byte) => x*x >= 0 ).check
+ OK, passed 100 tests.

scala>
```

```
scala> import Arbitrary.arbitrary
import Arbitrary.arbitrary

scala> forAll(arbitrary[Byte])( (x:Byte) => x*x >= 0 ).check
+ OK, passed 100 tests.

scala> arbitrary[Byte].sample
```

```
scala> import Arbitrary.arbitrary
import Arbitrary.arbitrary

scala> forAll(arbitrary[Byte])( (x:Byte) => x*x >= 0 ).check
+ OK, passed 100 tests.

scala> arbitrary[Byte].sample
res31: Option[Byte] = Some(-128)

scala>
```

```
scala> import Arbitrary.arbitrary
import Arbitrary.arbitrary

scala> forAll(arbitrary[Byte])( (x:Byte) => x*x >= 0 ).check
+ OK, passed 100 tests.

scala> arbitrary[Byte].sample
res31: Option[Byte] = Some(-128)

scala> arbitrary[Byte].sample
```

```
scala> import Arbitrary.arbitrary
import Arbitrary.arbitrary

scala> forAll(arbitrary[Byte])( (x:Byte) => x*x >= 0 ).check
+ OK, passed 100 tests.

scala> arbitrary[Byte].sample
res31: Option[Byte] = Some(-128)

scala> arbitrary[Byte].sample
res32: Option[Byte] = Some(-55)

scala>
```

```
scala> import Arbitrary.arbitrary
import Arbitrary.arbitrary

scala> forAll(arbitrary[Byte])( (x:Byte) => x*x >= 0 ).check
+ OK, passed 100 tests.

scala> arbitrary[Byte].sample
res31: Option[Byte] = Some(-128)

scala> arbitrary[Byte].sample
res32: Option[Byte] = Some(-55)

scala> arbitrary[Int].sample
```

```
scala> import Arbitrary.arbitrary
import Arbitrary.arbitrary

scala> forAll(arbitrary[Byte])( (x:Byte) => x*x >= 0 ).check
+ OK, passed 100 tests.

scala> arbitrary[Byte].sample
res31: Option[Byte] = Some(-128)

scala> arbitrary[Byte].sample
res32: Option[Byte] = Some(-55)

scala> arbitrary[Int].sample
res33: Option[Int] = Some(-1419545063)

scala>
```

```
scala> import Arbitrary.arbitrary
import Arbitrary.arbitrary

scala> forAll(arbitrary[Byte])( (x:Byte) => x*x >= 0 ).check
+ OK, passed 100 tests.

scala> arbitrary[Byte].sample
res31: Option[Byte] = Some(-128)

scala> arbitrary[Byte].sample
res32: Option[Byte] = Some(-55)

scala> arbitrary[Int].sample
res33: Option[Int] = Some(-1419545063)

scala> arbitrary[Int].sample
```

```
scala> import Arbitrary.arbitrary
import Arbitrary.arbitrary

scala> forAll(arbitrary[Byte])( (x:Byte) => x*x >= 0 ).check
+ OK, passed 100 tests.

scala> arbitrary[Byte].sample
res31: Option[Byte] = Some(-128)

scala> arbitrary[Byte].sample
res32: Option[Byte] = Some(-55)

scala> arbitrary[Int].sample
res33: Option[Int] = Some(-1419545063)

scala> arbitrary[Int].sample
res34: Option[Int] = Some(464683479)
```

```
[noun] book - livre (m)
[verb] go - aller
[noun] car - voiture (f)
[verb] call - appeler
[noun] computer - ordinateur (m)
```

```
[noun] book - livre (m)
[verb] go - aller
[noun] car - voiture (f)
[verb] call - appeler
[noun] computer - ordinateur (m)
```

```
<?xml version='1.0' encoding='utf-8'?>
<dict>
<entry kind="noun">
  <en><word>book</word></en>
  <fr><word>livre</word><gender>m</gender></fr></entry>
<entry kind="verb">
  <en><word>go</word></en>
  <fr><word>aller</word></fr></entry>
<entry kind="noun">
  <en><word>car</word></en>
  <fr><word>voiture</word><gender>f</gender></fr></entry>
<entry kind="verb">
  <en><word>call</word></en>
  <fr><word>appeler</word></fr></entry>
<entry kind="noun">
  <en><word>computer</word></en>
  <fr><word>ordinateur</word><gender>m</gender></fr></entry>
</dict>
```

```
scala>
```

```
scala> val dict1 = "[noun] book - livre (m)"
```

```
scala> val dict1 = "[noun] book - livre (m)"
dict1: String = [noun] book - livre (m)

scala>
```

```
scala> val dict1 = "[noun] book - livre (m)"
dict1: String = [noun] book - livre (m)

scala> val dict1xml = Dict.toXml(dict1)
```

```
scala> val dict1 = "[noun] book - livre (m)"
dict1: String = [noun] book - livre (m)

scala> val dict1xml = Dict.toXml(dict1)
dict1xml: String =
<?xml version='1.0' encoding='utf-8'?>
<dict>
<entry kind="noun">
  <en><word>book</word></en>
  <fr><word>livre</word><gender>m</gender></fr></entry>
</dict>

scala>
```

```
scala> val dict1 = "[noun] book - livre (m)"
dict1: String = [noun] book - livre (m)

scala> val dict1xml = Dict.toXml(dict1)
dict1xml: String =
<?xml version='1.0' encoding='utf-8'?>
<dict>
<entry kind="noun">
  <en><word>book</word></en>
  <fr><word>livre</word><gender>m</gender></fr></entry>
</dict>

scala> Dict.toText(dict1xml)
```

```
scala> val dict1 = "[noun] book - livre (m)"
dict1: String = [noun] book - livre (m)

scala> val dict1xml = Dict.toXml(dict1)
dict1xml: String =
<?xml version='1.0' encoding='utf-8'?>
<dict>
<entry kind="noun">
  <en><word>book</word></en>
  <fr><word>livre</word><gender>m</gender></fr></entry>
</dict>

scala> Dict.toText(dict1xml)
res35: String = [noun] book - livre (m)
```

```
val dict2 = """[noun] book - livre (m)
[verb] go - aller
[noun] car - voiture (f)
[verb] call - appeler
[noun] computer - ordinateur (m)"""
```

```
val dict2 = """[noun] book - livre (m)
[verb] go - aller
[noun] car - voiture (f)
[verb] call - appeler
[noun] computer - ordinateur (m)"""

scala>
```

```
val dict2 = """[noun] book - livre (m)
[verb] go - aller
[noun] car - voiture (f)
[verb] call - appeler
[noun] computer - ordinateur (m)"""

scala> Dict.toText(Dict.toXml(dict2)) == dict2
```

```
val dict2 = """[noun] book - livre (m)
[verb] go - aller
[noun] car - voiture (f)
[verb] call - appeler
[noun] computer - ordinateur (m)"""

scala> Dict.toText(Dict.toXml(dict2)) == dict2
res36: Boolean = true
```

```
scala>
```

```
scala> def wordGen1 = listOf(choose('a','z'))
```

```
scala> def wordGen1 = listOf(choose('a','z'))
wordGen1: org.scalacheck.Gen[List[Char]]

scala>
```

```
scala> def wordGen1 = listOf(choose('a','z'))
wordGen1: org.scalacheck.Gen[List[Char]]

scala> wordGen1.sample
```

```
scala> def wordGen1 = listOf(choose('a','z'))
wordGen1: org.scalacheck.Gen[List[Char]]

scala> wordGen1.sample
res37: Option[List[Char]] = Some(List(s))

scala>
```

```
scala> def wordGen1 = listOf(choose('a','z'))
wordGen1: org.scalacheck.Gen[List[Char]]

scala> wordGen1.sample
res37: Option[List[Char]] = Some(List(s))

scala> wordGen1.sample
```

```
scala> def wordGen1 = listOf(choose('a','z'))
wordGen1: org.scalacheck.Gen[List[Char]]

scala> wordGen1.sample
res37: Option[List[Char]] = Some(List(s))

scala> wordGen1.sample
res38: Option[List[Char]] = Some(List(r, u, b, p, n, c, h, m, x
, c, u, i, i, x, y, b, x, f, d, e, z, j, t, u, k, y, l, w, z, m
, m, d, k, o, x, p, g, b, e, w, i, v, o, x, t, o, r))

scala>
```

```
scala> def wordGen1 = listOf(choose('a','z'))
wordGen1: org.scalacheck.Gen[List[Char]]

scala> wordGen1.sample
res37: Option[List[Char]] = Some(List(s))

scala> wordGen1.sample
res38: Option[List[Char]] = Some(List(r, u, b, p, n, c, h, m, x
, c, u, i, i, x, y, b, x, f, d, e, z, j, t, u, k, y, l, w, z, m
, m, d, k, o, x, p, g, b, e, w, i, v, o, x, t, o, r))

scala> wordGen1.sample
```

```
scala> def wordGen1 = listOf(choose('a','z'))
wordGen1: org.scalacheck.Gen[List[Char]]

scala> wordGen1.sample
res37: Option[List[Char]] = Some(List(s))

scala> wordGen1.sample
res38: Option[List[Char]] = Some(List(r, u, b, p, n, c, h, m, x
, c, u, i, i, x, y, b, x, f, d, e, z, j, t, u, k, y, l, w, z, m
, m, d, k, o, x, p, g, b, e, w, i, v, o, x, t, o, r))

scala> wordGen1.sample
res39: Option[List[Char]] = Some(List(d, b, k, v, j, o, j, b, d
, a, i, a, j, t, g, d, j, e, b, l, s, e, v, c, q, j, j, v, p, i
, q, q, z, r, p, z, y, y, j, r, p, m, g, p, j, a, p, p, x, s, t
, k, n, a, q, n, f, f, w, z, p, p, q, o, w, r, r, i, w, a, r, a
, h, q, w, j, c, k, g, y, c, m, u, l))

scala>
```

```
scala> def wordGen1 = listOf(choose('a','z'))
wordGen1: org.scalacheck.Gen[List[Char]]

scala> wordGen1.sample
res37: Option[List[Char]] = Some(List(s))

scala> wordGen1.sample
res38: Option[List[Char]] = Some(List(r, u, b, p, n, c, h, m, x
, c, u, i, i, x, y, b, x, f, d, e, z, j, t, u, k, y, l, w, z, m
, m, d, k, o, x, p, g, b, e, w, i, v, o, x, t, o, r))

scala> wordGen1.sample
res39: Option[List[Char]] = Some(List(d, b, k, v, j, o, j, b, d
, a, i, a, j, t, g, d, j, e, b, l, s, e, v, c, q, j, j, v, p, i
, q, q, z, r, p, z, y, y, j, r, p, m, g, p, j, a, p, p, x, s, t
, k, n, a, q, n, f, f, w, z, p, p, q, o, w, r, r, i, w, a, r, a
, h, q, w, j, c, k, g, y, c, m, u, l))

scala> wordGen1.sample
```

```
scala> def wordGen1 = listOf(choose('a','z'))
wordGen1: org.scalacheck.Gen[List[Char]]

scala> wordGen1.sample
res37: Option[List[Char]] = Some(List(s))

scala> wordGen1.sample
res38: Option[List[Char]] = Some(List(r, u, b, p, n, c, h, m, x
, c, u, i, i, x, y, b, x, f, d, e, z, j, t, u, k, y, l, w, z, m
, m, d, k, o, x, p, g, b, e, w, i, v, o, x, t, o, r))

scala> wordGen1.sample
res39: Option[List[Char]] = Some(List(d, b, k, v, j, o, j, b, d
, a, i, a, j, t, g, d, j, e, b, l, s, e, v, c, q, j, j, v, p, i
, q, q, z, r, p, z, y, y, j, r, p, m, g, p, j, a, p, p, x, s, t
, k, n, a, q, n, f, f, w, z, p, p, q, o, w, r, r, i, w, a, r, a
, h, q, w, j, c, k, g, y, c, m, u, l))

scala> wordGen1.sample
res40: Option[List[Char]] = Some(List(c, r, i, m, y, j, f, i, l
, e, k, g, n, m, j, b, u))
```

```
scala>
```

```
scala> def wordGen2 = resize(7,listOf(choose('a','z')))
```

```
scala> def wordGen2 = resize(7,listOf(choose('a','z')))
wordGen2: org.scalacheck.Gen[List[Char]]

scala>
```

```
scala> def wordGen2 = resize(7,listOf(choose('a','z')))
wordGen2: org.scalacheck.Gen[List[Char]]

scala> wordGen2.sample
```

```
scala> def wordGen2 = resize(7,listOf(choose('a','z')))
wordGen2: org.scalacheck.Gen[List[Char]]

scala> wordGen2.sample
res41: Option[List[Char]] = Some(List(p, o, p, f, g, e))

scala>
```

```
scala> def wordGen2 = resize(7,listOf(choose('a','z')))
wordGen2: org.scalacheck.Gen[List[Char]]

scala> wordGen2.sample
res41: Option[List[Char]] = Some(List(p, o, p, f, g, e))

scala> wordGen2.sample
```

```
scala> def wordGen2 = resize(7,listOf(choose('a','z')))
wordGen2: org.scalacheck.Gen[List[Char]]

scala> wordGen2.sample
res41: Option[List[Char]] = Some(List(p, o, p, f, g, e))

scala> wordGen2.sample
res42: Option[List[Char]] = Some(List())

scala>
```

```
scala> def wordGen2 = resize(7,listOf(choose('a','z')))
wordGen2: org.scalacheck.Gen[List[Char]]

scala> wordGen2.sample
res41: Option[List[Char]] = Some(List(p, o, p, f, g, e))

scala> wordGen2.sample
res42: Option[List[Char]] = Some(List())

scala> wordGen2.sample
```

```
scala> def wordGen2 = resize(7,listOf(choose('a','z')))
wordGen2: org.scalacheck.Gen[List[Char]]

scala> wordGen2.sample
res41: Option[List[Char]] = Some(List(p, o, p, f, g, e))

scala> wordGen2.sample
res42: Option[List[Char]] = Some(List())

scala> wordGen2.sample
res43: Option[List[Char]] = Some(List(i, u, y, f, n, n, e))

scala>
```

```
scala> def wordGen2 = resize(7,listOf(choose('a','z')))
wordGen2: org.scalacheck.Gen[List[Char]]

scala> wordGen2.sample
res41: Option[List[Char]] = Some(List(p, o, p, f, g, e))

scala> wordGen2.sample
res42: Option[List[Char]] = Some(List())

scala> wordGen2.sample
res43: Option[List[Char]] = Some(List(i, u, y, f, n, n, e))

scala> wordGen2.sample
```

```
scala> def wordGen2 = resize(7,listOf(choose('a','z')))
wordGen2: org.scalacheck.Gen[List[Char]]

scala> wordGen2.sample
res41: Option[List[Char]] = Some(List(p, o, p, f, g, e))

scala> wordGen2.sample
res42: Option[List[Char]] = Some(List())

scala> wordGen2.sample
res43: Option[List[Char]] = Some(List(i, u, y, f, n, n, e))

scala> wordGen2.sample
res44: Option[List[Char]] = Some(List(d, h, z))
```

```
scala>
```

```
scala> def wordGen3 = resize(7,listOf1(choose('a','z')))
```

```
scala> def wordGen3 = resize(7,listOf1(choose('a','z')))
wordGen3: org.scalacheck.Gen[List[Char]]

scala>
```

```
scala> def wordGen3 = resize(7,listOf1(choose('a','z')))
wordGen3: org.scalacheck.Gen[List[Char]]

scala> wordGen3.sample
```

```
scala> def wordGen3 = resize(7,listOf1(choose('a','z')))
wordGen3: org.scalacheck.Gen[List[Char]]

scala> wordGen3.sample
res45: Option[List[Char]] = Some(List(g, z, v, y))

scala>
```

```
scala> def wordGen3 = resize(7,listOf1(choose('a','z')))
wordGen3: org.scalacheck.Gen[List[Char]]

scala> wordGen3.sample
res45: Option[List[Char]] = Some(List(g, z, v, y))

scala> wordGen3.sample
```

```
scala> def wordGen3 = resize(7,listOf1(choose('a','z')))
wordGen3: org.scalacheck.Gen[List[Char]]

scala> wordGen3.sample
res45: Option[List[Char]] = Some(List(g, z, v, y))

scala> wordGen3.sample
res46: Option[List[Char]] = Some(List(a, d, z, e, c, p, k))

scala>
```

```
scala> def wordGen3 = resize(7,listOf1(choose('a','z')))
wordGen3: org.scalacheck.Gen[List[Char]]

scala> wordGen3.sample
res45: Option[List[Char]] = Some(List(g, z, v, y))

scala> wordGen3.sample
res46: Option[List[Char]] = Some(List(a, d, z, e, c, p, k))

scala> wordGen3.sample
```

```
scala> def wordGen3 = resize(7,listOf1(choose('a','z')))
wordGen3: org.scalacheck.Gen[List[Char]]

scala> wordGen3.sample
res45: Option[List[Char]] = Some(List(g, z, v, y))

scala> wordGen3.sample
res46: Option[List[Char]] = Some(List(a, d, z, e, c, p, k))

scala> wordGen3.sample
res47: Option[List[Char]] = Some(List(i))

scala>
```

```
scala> def wordGen3 = resize(7,listOf1(choose('a','z')))
wordGen3: org.scalacheck.Gen[List[Char]]

scala> wordGen3.sample
res45: Option[List[Char]] = Some(List(g, z, v, y))

scala> wordGen3.sample
res46: Option[List[Char]] = Some(List(a, d, z, e, c, p, k))

scala> wordGen3.sample
res47: Option[List[Char]] = Some(List(i))

scala> wordGen3.sample
```

```
scala> def wordGen3 = resize(7,listOf1(choose('a','z')))
wordGen3: org.scalacheck.Gen[List[Char]]

scala> wordGen3.sample
res45: Option[List[Char]] = Some(List(g, z, v, y))

scala> wordGen3.sample
res46: Option[List[Char]] = Some(List(a, d, z, e, c, p, k))

scala> wordGen3.sample
res47: Option[List[Char]] = Some(List(i))

scala> wordGen3.sample
res48: Option[List[Char]] = Some(List(j, z))
```

```
scala>
```

```
scala> def wordGen = resize(7,listOf1(alphaLowerChar)).
     |   map(_.mkString)
```

```
scala> def wordGen = resize(7,listOf1(alphaLowerChar)).
     |   map(_.mkString)
wordGen: org.scalacheck.Gen[String]

scala>
```

```
scala> def wordGen = resize(7,listOf1(alphaLowerChar)).
     |   map(_.mkString)
wordGen: org.scalacheck.Gen[String]

scala> wordGen.sample
```

```
scala> def wordGen = resize(7,listOf1(alphaLowerChar)).
     |   map(_.mkString)
wordGen: org.scalacheck.Gen[String]

scala> wordGen.sample
res49: Option[String] = Some(ca)

scala>
```

```
scala> def wordGen = resize(7,listOf1(alphaLowerChar)).
     |   map(_.mkString)
wordGen: org.scalacheck.Gen[String]

scala> wordGen.sample
res49: Option[String] = Some(ca)

scala> wordGen.sample
```

```
scala> def wordGen = resize(7,listOf1(alphaLowerChar)).
     |   map(_.mkString)
wordGen: org.scalacheck.Gen[String]

scala> wordGen.sample
res49: Option[String] = Some(ca)

scala> wordGen.sample
res50: Option[String] = Some(d)

scala>
```

```
scala> def wordGen = resize(7,listOf1(alphaLowerChar)).
     |   map(_.mkString)
wordGen: org.scalacheck.Gen[String]

scala> wordGen.sample
res49: Option[String] = Some(ca)

scala> wordGen.sample
res50: Option[String] = Some(d)

scala> wordGen.sample
```

```
scala> def wordGen = resize(7,listOf1(alphaLowerChar)).
     |   map(_.mkString)
wordGen: org.scalacheck.Gen[String]

scala> wordGen.sample
res49: Option[String] = Some(ca)

scala> wordGen.sample
res50: Option[String] = Some(d)

scala> wordGen.sample
res51: Option[String] = Some(yjfpknf)

scala>
```

```
scala> def wordGen = resize(7,listOf1(alphaLowerChar)).
     |   map(_.mkString)
wordGen: org.scalacheck.Gen[String]

scala> wordGen.sample
res49: Option[String] = Some(ca)

scala> wordGen.sample
res50: Option[String] = Some(d)

scala> wordGen.sample
res51: Option[String] = Some(yjfpknf)

scala> wordGen.sample
```

```
scala> def wordGen = resize(7,listOf1(alphaLowerChar)).
     |   map(_.mkString)
wordGen: org.scalacheck.Gen[String]

scala> wordGen.sample
res49: Option[String] = Some(ca)

scala> wordGen.sample
res50: Option[String] = Some(d)

scala> wordGen.sample
res51: Option[String] = Some(yjfpknf)

scala> wordGen.sample
res52: Option[String] = Some(u)
```

```
scala>
```

```
scala> def verbGen = wordGen.map2(wordGen)(
     |   "[verb] " + _ + " - " + _)
```

```
scala> def verbGen = wordGen.map2(wordGen)(
     |   "[verb] " + _ + " - " + _)
verbGen: org.scalacheck.Gen[String]

scala>
```

```
scala> def verbGen = wordGen.map2(wordGen)(
     |   "[verb] " + _ + " - " + _)
verbGen: org.scalacheck.Gen[String]

scala> verbGen.sample
```

```
scala> def verbGen = wordGen.map2(wordGen)(
     |   "[verb] " + _ + " - " + _)
verbGen: org.scalacheck.Gen[String]

scala> verbGen.sample
res53: Option[String] = Some([verb] zxm - mesmnq)

scala>
```

```
scala> def verbGen = wordGen.map2(wordGen)(
     |   "[verb] " + _ + " - " + _)
verbGen: org.scalacheck.Gen[String]

scala> verbGen.sample
res53: Option[String] = Some([verb] zxm - mesmnq)

scala> verbGen.sample
```

```
scala> def verbGen = wordGen.map2(wordGen)(
     |   "[verb] " + _ + " - " + _)
verbGen: org.scalacheck.Gen[String]

scala> verbGen.sample
res53: Option[String] = Some([verb] zxm - mesmnq)

scala> verbGen.sample
res54: Option[String] = Some([verb] kua - pgdxvr)

scala>
```

```
scala> def verbGen = wordGen.map2(wordGen)(
     |   "[verb] " + _ + " - " + _)
verbGen: org.scalacheck.Gen[String]

scala> verbGen.sample
res53: Option[String] = Some([verb] zxm - mesmnq)

scala> verbGen.sample
res54: Option[String] = Some([verb] kua - pgdxvr)

scala> verbGen.sample
```

```
scala> def verbGen = wordGen.map2(wordGen)(
     |   "[verb] " + _ + " - " + _)
verbGen: org.scalacheck.Gen[String]

scala> verbGen.sample
res53: Option[String] = Some([verb] zxm - mesmnq)

scala> verbGen.sample
res54: Option[String] = Some([verb] kua - pgdxvr)

scala> verbGen.sample
res55: Option[String] = Some([verb] r - mmdse)

scala>
```

```
scala> def verbGen = wordGen.map2(wordGen)(
     |   "[verb] " + _ + " - " + _)
verbGen: org.scalacheck.Gen[String]

scala> verbGen.sample
res53: Option[String] = Some([verb] zxm - mesmnq)

scala> verbGen.sample
res54: Option[String] = Some([verb] kua - pgdxvr)

scala> verbGen.sample
res55: Option[String] = Some([verb] r - mmdse)

scala> verbGen.sample
```

```
scala> def verbGen = wordGen.map2(wordGen)(
     |   "[verb] " + _ + " - " + _)
verbGen: org.scalacheck.Gen[String]

scala> verbGen.sample
res53: Option[String] = Some([verb] zxm - mesmnq)

scala> verbGen.sample
res54: Option[String] = Some([verb] kua - pgdxvr)

scala> verbGen.sample
res55: Option[String] = Some([verb] r - mmdse)

scala> verbGen.sample
res56: Option[String] = Some([verb] eu - gbtdpa)
```

```
scala>
```

```
scala> def verbGen1 = for {
     |   en <- wordGen
     |   fr <- wordGen
     | } yield "[verb] " + en + " - " + fr
```

```
scala> def verbGen1 = for {
     |   en <- wordGen
     |   fr <- wordGen
     | } yield "[verb] " + en + " - " + fr
verbGen1: org.scalacheck.Gen[String]

scala>
```

```
scala> def verbGen1 = for {
     |   en <- wordGen
     |   fr <- wordGen
     | } yield "[verb] " + en + " - " + fr
verbGen1: org.scalacheck.Gen[String]

scala> verbGen1.sample
```

```
scala> def verbGen1 = for {
     |   en <- wordGen
     |   fr <- wordGen
     | } yield "[verb] " + en + " - " + fr
verbGen1: org.scalacheck.Gen[String]

scala> verbGen1.sample
res57: Option[String] = Some([verb] tfl - m)

scala>
```

```
scala> def verbGen1 = for {
     |   en <- wordGen
     |   fr <- wordGen
     | } yield "[verb] " + en + " - " + fr
verbGen1: org.scalacheck.Gen[String]

scala> verbGen1.sample
res57: Option[String] = Some([verb] tfl - m)

scala> verbGen1.sample
```

```
scala> def verbGen1 = for {
     |   en <- wordGen
     |   fr <- wordGen
     | } yield "[verb] " + en + " - " + fr
verbGen1: org.scalacheck.Gen[String]

scala> verbGen1.sample
res57: Option[String] = Some([verb] tfl - m)

scala> verbGen1.sample
res58: Option[String] = Some([verb] jdxzmkp - thr)

scala>
```

```
scala> def verbGen1 = for {
     |   en <- wordGen
     |   fr <- wordGen
     | } yield "[verb] " + en + " - " + fr
verbGen1: org.scalacheck.Gen[String]

scala> verbGen1.sample
res57: Option[String] = Some([verb] tfl - m)

scala> verbGen1.sample
res58: Option[String] = Some([verb] jdxzmkp - thr)

scala> verbGen1.sample
```

```
scala> def verbGen1 = for {
     |   en <- wordGen
     |   fr <- wordGen
     | } yield "[verb] " + en + " - " + fr
verbGen1: org.scalacheck.Gen[String]

scala> verbGen1.sample
res57: Option[String] = Some([verb] tfl - m)

scala> verbGen1.sample
res58: Option[String] = Some([verb] jdxzmkp - thr)

scala> verbGen1.sample
res59: Option[String] = Some([verb] muheq - mc)

scala>
```

```
scala> def verbGen1 = for {
     |   en <- wordGen
     |   fr <- wordGen
     | } yield "[verb] " + en + " - " + fr
verbGen1: org.scalacheck.Gen[String]

scala> verbGen1.sample
res57: Option[String] = Some([verb] tfl - m)

scala> verbGen1.sample
res58: Option[String] = Some([verb] jdxzmkp - thr)

scala> verbGen1.sample
res59: Option[String] = Some([verb] muheq - mc)

scala> verbGen1.sample
```

```
scala> def verbGen1 = for {
     |   en <- wordGen
     |   fr <- wordGen
     | } yield "[verb] " + en + " - " + fr
verbGen1: org.scalacheck.Gen[String]

scala> verbGen1.sample
res57: Option[String] = Some([verb] tfl - m)

scala> verbGen1.sample
res58: Option[String] = Some([verb] jdxzmkp - thr)

scala> verbGen1.sample
res59: Option[String] = Some([verb] muheq - mc)

scala> verbGen1.sample
res60: Option[String] = Some([verb] xlkcm - ncqi)
```

```
scala>
```

```
scala> def nounMGen = wordGen.map2(wordGen)(
     |   "[noun] " + _ + " - " + _ + " (m)")
```

```
scala> def nounMGen = wordGen.map2(wordGen)(
     |   "[noun] " + _ + " - " + _ + " (m)")
nounMGen: org.scalacheck.Gen[String]

scala>
```

```
scala> def nounMGen = wordGen.map2(wordGen)(
     |   "[noun] " + _ + " - " + _ + " (m)")
nounMGen: org.scalacheck.Gen[String]

scala> def nounFGen = wordGen.map2(wordGen)(
     |   "[noun] " + _ + " - " + _ + " (f)")
```

```
scala> def nounMGen = wordGen.map2(wordGen)(
     |   "[noun] " + _ + " - " + _ + " (m)")
nounMGen: org.scalacheck.Gen[String]

scala> def nounFGen = wordGen.map2(wordGen)(
     |   "[noun] " + _ + " - " + _ + " (f)")
nounFGen: org.scalacheck.Gen[String]

scala>
```

```
scala> def nounMGen = wordGen.map2(wordGen)(
     |   "[noun] " + _ + " - " + _ + " (m)")
nounMGen: org.scalacheck.Gen[String]

scala> def nounFGen = wordGen.map2(wordGen)(
     |   "[noun] " + _ + " - " + _ + " (f)")
nounFGen: org.scalacheck.Gen[String]

scala> nounMGen.sample
```

```
scala> def nounMGen = wordGen.map2(wordGen)(
     |   "[noun] " + _ + " - " + _ + " (m)")
nounMGen: org.scalacheck.Gen[String]

scala> def nounFGen = wordGen.map2(wordGen)(
     |   "[noun] " + _ + " - " + _ + " (f)")
nounFGen: org.scalacheck.Gen[String]

scala> nounMGen.sample
res61: Option[String] = Some([noun] ledyn - w (m))

scala>
```

```
scala> def nounMGen = wordGen.map2(wordGen)(
     |   "[noun] " + _ + " - " + _ + " (m)")
nounMGen: org.scalacheck.Gen[String]

scala> def nounFGen = wordGen.map2(wordGen)(
     |   "[noun] " + _ + " - " + _ + " (f)")
nounFGen: org.scalacheck.Gen[String]

scala> nounMGen.sample
res61: Option[String] = Some([noun] ledyn - w (m))

scala> nounMGen.sample
```

```
scala> def nounMGen = wordGen.map2(wordGen)(
     |   "[noun] " + _ + " - " + _ + " (m)")
nounMGen: org.scalacheck.Gen[String]

scala> def nounFGen = wordGen.map2(wordGen)(
     |   "[noun] " + _ + " - " + _ + " (f)")
nounFGen: org.scalacheck.Gen[String]

scala> nounMGen.sample
res61: Option[String] = Some([noun] ledyn - w (m))

scala> nounMGen.sample
res62: Option[String] = Some([noun] rmt - iytgy (m))

scala>
```

```
scala> def nounMGen = wordGen.map2(wordGen)(
     |   "[noun] " + _ + " - " + _ + " (m)")
nounMGen: org.scalacheck.Gen[String]

scala> def nounFGen = wordGen.map2(wordGen)(
     |   "[noun] " + _ + " - " + _ + " (f)")
nounFGen: org.scalacheck.Gen[String]

scala> nounMGen.sample
res61: Option[String] = Some([noun] ledyn - w (m))

scala> nounMGen.sample
res62: Option[String] = Some([noun] rmt - iytgy (m))

scala> nounFGen.sample
```

```
scala> def nounMGen = wordGen.map2(wordGen)(
     |   "[noun] " + _ + " - " + _ + " (m)")
nounMGen: org.scalacheck.Gen[String]

scala> def nounFGen = wordGen.map2(wordGen)(
     |   "[noun] " + _ + " - " + _ + " (f)")
nounFGen: org.scalacheck.Gen[String]

scala> nounMGen.sample
res61: Option[String] = Some([noun] ledyn - w (m))

scala> nounMGen.sample
res62: Option[String] = Some([noun] rmt - iytgy (m))

scala> nounFGen.sample
res63: Option[String] = Some([noun] x - wqawk (f))

scala>
```

```
scala> def nounMGen = wordGen.map2(wordGen)(
     |   "[noun] " + _ + " - " + _ + " (m)")
nounMGen: org.scalacheck.Gen[String]

scala> def nounFGen = wordGen.map2(wordGen)(
     |   "[noun] " + _ + " - " + _ + " (f)")
nounFGen: org.scalacheck.Gen[String]

scala> nounMGen.sample
res61: Option[String] = Some([noun] ledyn - w (m))

scala> nounMGen.sample
res62: Option[String] = Some([noun] rmt - iytgy (m))

scala> nounFGen.sample
res63: Option[String] = Some([noun] x - wqawk (f))

scala> nounFGen.sample
```

```
scala> def nounMGen = wordGen.map2(wordGen)(
     |   "[noun] " + _ + " - " + _ + " (m)")
nounMGen: org.scalacheck.Gen[String]

scala> def nounFGen = wordGen.map2(wordGen)(
     |   "[noun] " + _ + " - " + _ + " (f)")
nounFGen: org.scalacheck.Gen[String]

scala> nounMGen.sample
res61: Option[String] = Some([noun] ledyn - w (m))

scala> nounMGen.sample
res62: Option[String] = Some([noun] rmt - iytgy (m))

scala> nounFGen.sample
res63: Option[String] = Some([noun] x - wqawk (f))

scala> nounFGen.sample
res64: Option[String] = Some([noun] sucjd - ft (f))
```

```
scala>
```

```
scala> def entryGen = oneOf(verbGen, nounMGen, nounFGen)
```

```
scala> def entryGen = oneOf(verbGen, nounMGen, nounFGen)
entryGen: org.scalacheck.Gen[String]

scala>
```

```
scala> def entryGen = oneOf(verbGen, nounMGen, nounFGen)
entryGen: org.scalacheck.Gen[String]

scala> entryGen.sample
```

```
scala> def entryGen = oneOf(verbGen, nounMGen, nounFGen)
entryGen: org.scalacheck.Gen[String]

scala> entryGen.sample
res65: Option[String] = Some([noun] lpjpyv - lsrbgs (f))

scala>
```

```
scala> def entryGen = oneOf(verbGen, nounMGen, nounFGen)
entryGen: org.scalacheck.Gen[String]

scala> entryGen.sample
res65: Option[String] = Some([noun] lpjpyv - lsrbgs (f))

scala> entryGen.sample
```

```
scala> def entryGen = oneOf(verbGen, nounMGen, nounFGen)
entryGen: org.scalacheck.Gen[String]

scala> entryGen.sample
res65: Option[String] = Some([noun] lpjpyv - lsrbgs (f))

scala> entryGen.sample
res66: Option[String] = Some([noun] qggcxk - cnesr (m))

scala>
```

```
scala> def entryGen = oneOf(verbGen, nounMGen, nounFGen)
entryGen: org.scalacheck.Gen[String]

scala> entryGen.sample
res65: Option[String] = Some([noun] lpjpyv - lsrbgs (f))

scala> entryGen.sample
res66: Option[String] = Some([noun] qggcxk - cnesr (m))

scala> entryGen.sample
```

```
scala> def entryGen = oneOf(verbGen, nounMGen, nounFGen)
entryGen: org.scalacheck.Gen[String]

scala> entryGen.sample
res65: Option[String] = Some([noun] lpjpyv - lsrbgs (f))

scala> entryGen.sample
res66: Option[String] = Some([noun] qggcxk - cnesr (m))

scala> entryGen.sample
res67: Option[String] = Some([verb] ij - xk)

scala>
```

```
scala> def entryGen = oneOf(verbGen, nounMGen, nounFGen)
entryGen: org.scalacheck.Gen[String]

scala> entryGen.sample
res65: Option[String] = Some([noun] lpjpyv - lsrbgs (f))

scala> entryGen.sample
res66: Option[String] = Some([noun] qggcxk - cnesr (m))

scala> entryGen.sample
res67: Option[String] = Some([verb] ij - xk)

scala> entryGen.sample
```

```
scala> def entryGen = oneOf(verbGen, nounMGen, nounFGen)
entryGen: org.scalacheck.Gen[String]

scala> entryGen.sample
res65: Option[String] = Some([noun] lpjpyv - lsrbgs (f))

scala> entryGen.sample
res66: Option[String] = Some([noun] qggcxk - cnesr (m))

scala> entryGen.sample
res67: Option[String] = Some([verb] ij - xk)

scala> entryGen.sample
res68: Option[String] = Some([noun] qkq - tdpgoj (m))
```

```
scala>
```

```
scala> def entryGen = verbGen | nounFGen | nounMGen
```

```
scala> def entryGen = verbGen | nounFGen | nounMGen
entryGen: org.scalacheck.Gen[String]

scala>
```

```
scala> def entryGen = verbGen | nounFGen | nounMGen
entryGen: org.scalacheck.Gen[String]

scala> entryGen.sample
```

```
scala> def entryGen = verbGen | nounFGen | nounMGen
entryGen: org.scalacheck.Gen[String]

scala> entryGen.sample
res69: Option[String] = Some([noun] k - pxze (f))

scala>
```

```
scala> def entryGen = verbGen | nounFGen | nounMGen
entryGen: org.scalacheck.Gen[String]

scala> entryGen.sample
res69: Option[String] = Some([noun] k - pxze (f))

scala> entryGen.sample
```

```
scala> def entryGen = verbGen | nounFGen | nounMGen
entryGen: org.scalacheck.Gen[String]

scala> entryGen.sample
res69: Option[String] = Some([noun] k - pxze (f))

scala> entryGen.sample
res70: Option[String] = Some([verb] wnzgbh - nzquan)

scala>
```

```
scala> def entryGen = verbGen | nounFGen | nounMGen
entryGen: org.scalacheck.Gen[String]

scala> entryGen.sample
res69: Option[String] = Some([noun] k - pxze (f))

scala> entryGen.sample
res70: Option[String] = Some([verb] wnzgbh - nzquan)

scala> entryGen.sample
```

```
scala> def entryGen = verbGen | nounFGen | nounMGen
entryGen: org.scalacheck.Gen[String]

scala> entryGen.sample
res69: Option[String] = Some([noun] k - pxze (f))

scala> entryGen.sample
res70: Option[String] = Some([verb] wnzgbh - nzquan)

scala> entryGen.sample
res71: Option[String] = Some([noun] jqlvke - azzd (f))

scala>
```

```
scala> def entryGen = verbGen | nounFGen | nounMGen
entryGen: org.scalacheck.Gen[String]

scala> entryGen.sample
res69: Option[String] = Some([noun] k - pxze (f))

scala> entryGen.sample
res70: Option[String] = Some([verb] wnzgbh - nzquan)

scala> entryGen.sample
res71: Option[String] = Some([noun] jqlvke - azzd (f))

scala> entryGen.sample
```

```
scala> def entryGen = verbGen | nounFGen | nounMGen
entryGen: org.scalacheck.Gen[String]

scala> entryGen.sample
res69: Option[String] = Some([noun] k - pxze (f))

scala> entryGen.sample
res70: Option[String] = Some([verb] wnzgbh - nzquan)

scala> entryGen.sample
res71: Option[String] = Some([noun] jqlvke - azzd (f))

scala> entryGen.sample
res72: Option[String] = Some([noun] s - wvmldv (f))
```

```
scala>
```

```
scala> def entryGen = frequency((2,verbGen),
     |   (1,nounFGen),(1,nounMGen))
```

```
scala> def entryGen = frequency((2,verbGen),
     |   (1,nounFGen),(1,nounMGen))
entryGen: org.scalacheck.Gen[String]

scala>
```

```
scala> def entryGen = frequency((2,verbGen),
     |   (1,nounFGen),(1,nounMGen))
entryGen: org.scalacheck.Gen[String]

scala> entryGen.sample
```

```
scala> def entryGen = frequency((2,verbGen),
     |   (1,nounFGen),(1,nounMGen))
entryGen: org.scalacheck.Gen[String]

scala> entryGen.sample
res73: Option[String] = Some([verb] smfej - ygalg)

scala>
```

```
scala> def entryGen = frequency((2,verbGen),
     |   (1,nounFGen),(1,nounMGen))
entryGen: org.scalacheck.Gen[String]

scala> entryGen.sample
res73: Option[String] = Some([verb] smfej - ygalg)

scala> entryGen.sample
```

```
scala> def entryGen = frequency((2,verbGen),
     |   (1,nounFGen),(1,nounMGen))
entryGen: org.scalacheck.Gen[String]

scala> entryGen.sample
res73: Option[String] = Some([verb] smfej - ygalg)

scala> entryGen.sample
res74: Option[String] = Some([noun] srzoi - bh (m))

scala>
```

```
scala> def entryGen = frequency((2,verbGen),
     |   (1,nounFGen),(1,nounMGen))
entryGen: org.scalacheck.Gen[String]

scala> entryGen.sample
res73: Option[String] = Some([verb] smfej - ygalg)

scala> entryGen.sample
res74: Option[String] = Some([noun] srzoi - bh (m))

scala> entryGen.sample
```

```
scala> def entryGen = frequency((2,verbGen),
     |   (1,nounFGen),(1,nounMGen))
entryGen: org.scalacheck.Gen[String]

scala> entryGen.sample
res73: Option[String] = Some([verb] smfej - ygalg)

scala> entryGen.sample
res74: Option[String] = Some([noun] srzoi - bh (m))

scala> entryGen.sample
res75: Option[String] = Some([noun] tcfgvtb - rqeprly (f))

scala>
```

```
scala> def entryGen = frequency((2,verbGen),
     |   (1,nounFGen),(1,nounMGen))
entryGen: org.scalacheck.Gen[String]

scala> entryGen.sample
res73: Option[String] = Some([verb] smfej - ygalg)

scala> entryGen.sample
res74: Option[String] = Some([noun] srzoi - bh (m))

scala> entryGen.sample
res75: Option[String] = Some([noun] tcfgvtb - rqeprly (f))

scala> entryGen.sample
```

```
scala> def entryGen = frequency((2,verbGen),
     |   (1,nounFGen),(1,nounMGen))
entryGen: org.scalacheck.Gen[String]

scala> entryGen.sample
res73: Option[String] = Some([verb] smfej - ygalg)

scala> entryGen.sample
res74: Option[String] = Some([noun] srzoi - bh (m))

scala> entryGen.sample
res75: Option[String] = Some([noun] tcfgvtb - rqeprly (f))

scala> entryGen.sample
res76: Option[String] = Some([verb] xezvu - itqncf)
```

```
scala>
```

```
scala> def dictGen = resize(10,listOf1(entryGen).map(_.mkString("\n")))
```

```
scala> def dictGen = resize(10,listOf1(entryGen).map(_.mkString("\n")))
dictGen: org.scalacheck.Gen[String]

scala>
```

```
scala> def dictGen = resize(10,listOf1(entryGen).map(_.mkString("\n")))
dictGen: org.scalacheck.Gen[String]

scala> dictGen.sample
```

```
scala> def dictGen = resize(10,listOf1(entryGen).map(_.mkString("\n")))
dictGen: org.scalacheck.Gen[String]

scala> dictGen.sample
res77: Option[String] =
Some([verb] bycq - ea
[noun] ktekq - rxh (f)
[noun] tpj - a (m)
[noun] tuqdzko - h (m)
[noun] byttafh - x (f))
```

```
import org.scalacheck.Prop._
import org.scalacheck.Gen._
def wordGen = resize(6,listOf1(alphaLowerChar)).map(_.mkString)
def verbGen = wordGen.map2(wordGen)("[verb] " + _ + " - " + _)
def nounMGen = wordGen.map2(wordGen)("[noun] " + _ + " - " + _ + " (m)")
def nounFGen = wordGen.map2(wordGen)("[noun] " + _ + " - " + _ + " (f)")
def entryGen = frequency((2,verbGen),(1,nounFGen),(1,nounMGen))
def dictGen = resize(10,listOf1(entryGen).map(_.mkString("\n")))
```

```
scala>
```

```
scala> def dictConversions = forAll(dictGen)( (dict:String) =>
     |   Dict.toText(Dict.toXml(dict)) == dict )
```

```
scala> def dictConversions = forAll(dictGen)( (dict:String) =>
     |   Dict.toText(Dict.toXml(dict)) == dict )
dictConversions: org.scalacheck.Prop

scala>
```

```
scala> def dictConversions = forAll(dictGen)( (dict:String) =>
     |   Dict.toText(Dict.toXml(dict)) == dict )
dictConversions: org.scalacheck.Prop

scala> dictConversions.check
```

```
scala> def dictConversions = forAll(dictGen)( (dict:String) =>
     |   Dict.toText(Dict.toXml(dict)) == dict )
dictConversions: org.scalacheck.Prop

scala> dictConversions.check
! Falsified after 5 passed tests.
> ARG_0: "["
```

```
scala>
```

```
scala> implicit val aString: Arbitrary[String] =
     |   Arbitrary(dictGen)
```

```
scala> implicit val aString: Arbitrary[String] =
     |   Arbitrary(dictGen)
aString: org.scalacheck.Arbitrary[String] = org.scalacheck.Arbi
trary$$anon$2@9253cc

scala>
```

```
scala> implicit val aString: Arbitrary[String] =
     |   Arbitrary(dictGen)
aString: org.scalacheck.Arbitrary[String] = org.scalacheck.Arbi
trary$$anon$2@9253cc

scala> arbitrary[String].sample
```

```
scala> implicit val aString: Arbitrary[String] =
     |   Arbitrary(dictGen)
aString: org.scalacheck.Arbitrary[String] = org.scalacheck.Arbi
trary$$anon$2@9253cc

scala> arbitrary[String].sample
res79: Option[String] =
Some([verb] qreyfpz - hwro
[noun] xallron - nxgu (f)
[noun] rcl - lxphfpc (m))
```

```
scala>
```

```
scala> def dictConversionsDefault = forAll( (dict:String) =>
     |   Dict.toText(Dict.toXml(dict)) == dict )
```

```
scala> def dictConversionsDefault = forAll( (dict:String) =>
     |   Dict.toText(Dict.toXml(dict)) == dict )
dictConversionsDefault: org.scalacheck.Prop

scala>
```

```
scala> def dictConversionsDefault = forAll( (dict:String) =>
     |   Dict.toText(Dict.toXml(dict)) == dict )
dictConversionsDefault: org.scalacheck.Prop

scala> dictConversionsDefault.check
```

```
scala> def dictConversionsDefault = forAll( (dict:String) =>
     |   Dict.toText(Dict.toXml(dict)) == dict )
dictConversionsDefault: org.scalacheck.Prop

scala> dictConversionsDefault.check
! Falsified after 9 passed tests.
> ARG_0: "["
```

# Test Data Shrinking

```
scala>
```

```
scala> def set111 = forAll( (x:Set[Byte]) => x == x - 111 )
```

```
scala> def set111 = forAll( (x:Set[Byte]) => x == x - 111 )
set111: org.scalacheck.Prop

scala>
```

```
scala> def set111 = forAll( (x:Set[Byte]) => x == x - 111 )
set111: org.scalacheck.Prop

scala> set111.check
```

```
scala> def set111 = forAll( (x:Set[Byte]) => x == x - 111 )
set111: org.scalacheck.Prop

scala> set111.check
! Falsified after 33 passed tests.
> ARG_0: Set(111)

scala>
```

```
scala> def set111 = forAll( (x:Set[Byte]) => x == x - 111 )
set111: org.scalacheck.Prop

scala> set111.check
! Falsified after 33 passed tests.
> ARG_0: Set(111)

scala> set111.check
```

```
scala> def set111 = forAll( (x:Set[Byte]) => x == x - 111 )
set111: org.scalacheck.Prop

scala> set111.check
! Falsified after 33 passed tests.
> ARG_0: Set(111)

scala> set111.check
! Falsified after 15 passed tests.
> ARG_0: Set(111)

scala>
```

```
scala> def set111 = forAll( (x:Set[Byte]) => x == x - 111 )
set111: org.scalacheck.Prop

scala> set111.check
! Falsified after 33 passed tests.
> ARG_0: Set(111)

scala> set111.check
! Falsified after 15 passed tests.
> ARG_0: Set(111)

scala> set111.check
```

```
scala> def set111 = forAll( (x:Set[Byte]) => x == x - 111 )
set111: org.scalacheck.Prop

scala> set111.check
! Falsified after 33 passed tests.
> ARG_0: Set(111)

scala> set111.check
! Falsified after 15 passed tests.
> ARG_0: Set(111)

scala> set111.check
! Falsified after 84 passed tests.
> ARG_0: Set(111)
```

```
scala>
```

```
scala> def set111noshrinking =
     |   forAllNoShrink(arbitrary[Set[Byte]])(
     |     (x:Set[Byte]) => x == x - 111 )
```

```
scala> def set111noshrinking =
     |   forAllNoShrink(arbitrary[Set[Byte]])(
     |     (x:Set[Byte]) => x == x - 111 )
set111noshrinking: org.scalacheck.Prop

scala>
```

```
scala> def set111noshrinking =
     |   forAllNoShrink(arbitrary[Set[Byte]])(
     |     (x:Set[Byte]) => x == x - 111 )
set111noshrinking: org.scalacheck.Prop

scala> set111noshrinking.check
```

```
scala> def set111noshrinking =
     |   forAllNoShrink(arbitrary[Set[Byte]])(
     |     (x:Set[Byte]) => x == x - 111 )
set111noshrinking: org.scalacheck.Prop

scala> set111noshrinking.check
! Falsified after 79 passed tests.
> ARG_0: Set(0, -49, -128, 93, -127, -104, 1, -82, 85, -26, -114, -36, -19,
   73, 22, 12, -10, 98, -29, -1, 127, 104, 40, 58, 4, -2, 111, 83, -88)

scala>
```

```
scala> def set111noshrinking =
     |   forAllNoShrink(arbitrary[Set[Byte]])(
     |     (x:Set[Byte]) => x == x - 111 )
set111noshrinking: org.scalacheck.Prop

scala> set111noshrinking.check
! Falsified after 79 passed tests.
> ARG_0: Set(0, -49, -128, 93, -127, -104, 1, -82, 85, -26, -114, -36, -19,
   73, 22, 12, -10, 98, -29, -1, 127, 104, 40, 58, 4, -2, 111, 83, -88)

scala> set111noshrinking.check
```

```
scala> def set111noshrinking =
     |   forAllNoShrink(arbitrary[Set[Byte]])(
     |     (x:Set[Byte]) => x == x - 111 )
set111noshrinking: org.scalacheck.Prop

scala> set111noshrinking.check
! Falsified after 79 passed tests.
> ARG_0: Set(0, -49, -128, 93, -127, -104, 1, -82, 85, -26, -114, -36, -19,
   73, 22, 12, -10, 98, -29, -1, 127, 104, 40, 58, 4, -2, 111, 83, -88)

scala> set111noshrinking.check
! Falsified after 19 passed tests.
> ARG_0: Set(0, -128, -7, 1, 27, 113, -1, 67, -13, 68, 111)

scala>
```

```
scala> def set111noshrinking =
     |   forAllNoShrink(arbitrary[Set[Byte]])(
     |     (x:Set[Byte]) => x == x - 111 )
set111noshrinking: org.scalacheck.Prop

scala> set111noshrinking.check
! Falsified after 79 passed tests.
> ARG_0: Set(0, -49, -128, 93, -127, -104, 1, -82, 85, -26, -114, -36, -19,
   73, 22, 12, -10, 98, -29, -1, 127, 104, 40, 58, 4, -2, 111, 83, -88)

scala> set111noshrinking.check
! Falsified after 19 passed tests.
> ARG_0: Set(0, -128, -7, 1, 27, 113, -1, 67, -13, 68, 111)

scala> set111noshrinking.check
```

```
scala> def set111noshrinking =
     |   forAllNoShrink(arbitrary[Set[Byte]])(
     |     (x:Set[Byte]) => x == x - 111 )
set111noshrinking: org.scalacheck.Prop

scala> set111noshrinking.check
! Falsified after 79 passed tests.
> ARG_0: Set(0, -49, -128, 93, -127, -104, 1, -82, 85, -26, -114, -36, -19,
   73, 22, 12, -10, 98, -29, -1, 127, 104, 40, 58, 4, -2, 111, 83, -88)

scala> set111noshrinking.check
! Falsified after 19 passed tests.
> ARG_0: Set(0, -128, -7, 1, 27, 113, -1, 67, -13, 68, 111)

scala> set111noshrinking.check
! Falsified after 18 passed tests.
> ARG_0: Set(0, -128, -35, 121, 1, 6, 103, 3, -57, 36, 111)
```

```
scala>
```

```
scala> def set111println = forAll{ (x:Set[Byte]) =>
     |   println(x); x == x - 111 }
```

```
scala> def set111println = forAll{ (x:Set[Byte]) =>
     |   println(x); x == x - 111 }
set111println: org.scalacheck.Prop

scala>
```

```
scala> def set111println = forAll{ (x:Set[Byte]) =>
     |   println(x); x == x - 111 }
set111println: org.scalacheck.Prop

scala> set111println.check
```

```
scala> def set111println = forAll{ (x:Set[Byte]) =>
     |   println(x); x == x - 111 }
set111println: org.scalacheck.Prop

scala> set111println.check
Set()
Set()
Set(-15, -36)
Set()
Set(1, -1)
Set(0, -87, 1, -20)
Set(-1)
Set(116, -128)
Set(5, -31, 34, 39, -1, 67, 79)
Set(1, -1, 111, 0)
Set(1, -1)
Set(111, 0)
Set(111)
Set()
! Falsified after 9 passed tests.
> ARG_0: Set(111)
```

```
scala> def dictConversionsDefault = forAll( (dict:String) =>
     |   Dict.toText(Dict.toXml(dict)) == dict )
dictConversionsDefault: org.scalacheck.Prop

scala> dictConversionsDefault.check
! Falsified after 9 passed tests.
> ARG_0: "["
```

```
scala>
```

```
scala> def dictConversionsNoShrink = forAllNoShrink(dictGen)(
     |   (dict:String) => Dict.toText(Dict.toXml(dict)) == dict )
```

```
scala> def dictConversionsNoShrink = forAllNoShrink(dictGen)(
     |   (dict:String) => Dict.toText(Dict.toXml(dict)) == dict )
dictConversionsNoShrink: org.scalacheck.Prop

scala>
```

```
scala> def dictConversionsNoShrink = forAllNoShrink(dictGen)(
     |   (dict:String) => Dict.toText(Dict.toXml(dict)) == dict )
dictConversionsNoShrink: org.scalacheck.Prop

scala> dictConversionsNoShrink.check
```

```
scala> def dictConversionsNoShrink = forAllNoShrink(dictGen)(
     |   (dict:String) => Dict.toText(Dict.toXml(dict)) == dict )
dictConversionsNoShrink: org.scalacheck.Prop

scala> dictConversionsNoShrink.check
! Falsified after 7 passed tests.
> ARG_0: "[verb] ucp - t
[noun] fw - c (f)
[noun] ae - bkmn (f)
[noun] tnkum - ilit (m)
[noun] aytyyl - dfsnnj (f)"

scala>
```

```
scala> def dictConversionsNoShrink = forAllNoShrink(dictGen)(
     |   (dict:String) => Dict.toText(Dict.toXml(dict)) == dict )
dictConversionsNoShrink: org.scalacheck.Prop

scala> dictConversionsNoShrink.check
! Falsified after 7 passed tests.
> ARG_0: "[verb] ucp - t
[noun] fw - c (f)
[noun] ae - bkmn (f)
[noun] tnkum - ilit (m)
[noun] aytyyl - dfsnnj (f)"

scala> dictConversionsNoShrink.check
```

```
scala> def dictConversionsNoShrink = forAllNoShrink(dictGen)(
     |   (dict:String) => Dict.toText(Dict.toXml(dict)) == dict )
dictConversionsNoShrink: org.scalacheck.Prop

scala> dictConversionsNoShrink.check
! Falsified after 7 passed tests.
> ARG_0: "[verb] ucp - t
[noun] fw - c (f)
[noun] ae - bkmn (f)
[noun] tnkum - ilit (m)
[noun] aytyyl - dfsnnj (f)"

scala> dictConversionsNoShrink.check
! Falsified after 4 passed tests.
> ARG_0: "[verb] nd - txfmj
[noun] nwkno - jgv (f)
[noun] c - s (f)
[noun] egoars - bolba (f)"
```

```
import org.scalacheck.Shrink

implicit val shrinkDict: Shrink[String] =
  Shrink[String] { (str: String) =>
    if (str.contains("\n")) {
      val entries = str.split("\n")
      val shrunkEntries = Shrink.shrink[Array[String]](entries)
      shrunkEntries.map(_.mkString("\n"))
    }
    else
      Stream.empty
}
```

```
scala>
```

```
scala> def dictConversions = forAll(dictGen)( (dict:String) =>
     |   Dict.toText(Dict.toXml(dict)) == dict )
```

```
scala> def dictConversions = forAll(dictGen)( (dict:String) =>
     |   Dict.toText(Dict.toXml(dict)) == dict )
dictConversions: org.scalacheck.Prop

scala>
```

```
scala> def dictConversions = forAll(dictGen)( (dict:String) =>
     |   Dict.toText(Dict.toXml(dict)) == dict )
dictConversions: org.scalacheck.Prop

scala> dictConversions.check
```

```
scala> def dictConversions = forAll(dictGen)( (dict:String) =>
     |   Dict.toText(Dict.toXml(dict)) == dict )
dictConversions: org.scalacheck.Prop

scala> dictConversions.check
! Falsified after 5 passed tests.
> ARG_0: "[noun] hjebcl - khrezc (f)"

scala>
```

```
scala> def dictConversions = forAll(dictGen)( (dict:String) =>
     |   Dict.toText(Dict.toXml(dict)) == dict )
dictConversions: org.scalacheck.Prop

scala> dictConversions.check
! Falsified after 5 passed tests.
> ARG_0: "[noun] hjebcl - khrezc (f)"

scala> dictConversions.check
```

```
scala> def dictConversions = forAll(dictGen)( (dict:String) =>
     |   Dict.toText(Dict.toXml(dict)) == dict )
dictConversions: org.scalacheck.Prop

scala> dictConversions.check
! Falsified after 5 passed tests.
> ARG_0: "[noun] hjebcl - khrezc (f)"

scala> dictConversions.check
! Falsified after 2 passed tests.
> ARG_0: "[noun] pcklo - fgrti (f)"

scala>
```

```
scala> def dictConversions = forAll(dictGen)( (dict:String) =>
     |   Dict.toText(Dict.toXml(dict)) == dict )
dictConversions: org.scalacheck.Prop

scala> dictConversions.check
! Falsified after 5 passed tests.
> ARG_0: "[noun] hjebcl - khrezc (f)"

scala> dictConversions.check
! Falsified after 2 passed tests.
> ARG_0: "[noun] pcklo - fgrti (f)"

scala> dictConversions.check
```

```
scala> def dictConversions = forAll(dictGen)( (dict:String) =>
     |   Dict.toText(Dict.toXml(dict)) == dict )
dictConversions: org.scalacheck.Prop

scala> dictConversions.check
! Falsified after 5 passed tests.
> ARG_0: "[noun] hjebcl - khrezc (f)"

scala> dictConversions.check
! Falsified after 2 passed tests.
> ARG_0: "[noun] pcklo - fgrti (f)"

scala> dictConversions.check
! Falsified after 2 passed tests.
> ARG_0: "[noun] gmcz - abhh (f)"

scala>
```

```
scala> def dictConversions = forAll(dictGen)( (dict:String) =>
     |   Dict.toText(Dict.toXml(dict)) == dict )
dictConversions: org.scalacheck.Prop

scala> dictConversions.check
! Falsified after 5 passed tests.
> ARG_0: "[noun] hjebcl - khrezc (f)"

scala> dictConversions.check
! Falsified after 2 passed tests.
> ARG_0: "[noun] pcklo - fgrti (f)"

scala> dictConversions.check
! Falsified after 2 passed tests.
> ARG_0: "[noun] gmcz - abhh (f)"

scala> dictConversions.check
```

```
scala> def dictConversions = forAll(dictGen)( (dict:String) =>
     |   Dict.toText(Dict.toXml(dict)) == dict )
dictConversions: org.scalacheck.Prop

scala> dictConversions.check
! Falsified after 5 passed tests.
> ARG_0: "[noun] hjebcl - khrezc (f)"

scala> dictConversions.check
! Falsified after 2 passed tests.
> ARG_0: "[noun] pcklo - fgrti (f)"

scala> dictConversions.check
! Falsified after 2 passed tests.
> ARG_0: "[noun] gmcz - abhh (f)"

scala> dictConversions.check
! Falsified after 4 passed tests.
> ARG_0: "[noun] kd - cv (f)"

scala>
```

```
scala> def dictConversions = forAll(dictGen)( (dict:String) =>
     |   Dict.toText(Dict.toXml(dict)) == dict )
dictConversions: org.scalacheck.Prop

scala> dictConversions.check
! Falsified after 5 passed tests.
> ARG_0: "[noun] hjebcl - khrezc (f)"

scala> dictConversions.check
! Falsified after 2 passed tests.
> ARG_0: "[noun] pcklo - fgrti (f)"

scala> dictConversions.check
! Falsified after 2 passed tests.
> ARG_0: "[noun] gmcz - abhh (f)"

scala> dictConversions.check
! Falsified after 4 passed tests.
> ARG_0: "[noun] kd - cv (f)"

scala> dictConversions.check
```

```
scala> def dictConversions = forAll(dictGen)( (dict:String) =>
     |   Dict.toText(Dict.toXml(dict)) == dict )
dictConversions: org.scalacheck.Prop

scala> dictConversions.check
! Falsified after 5 passed tests.
> ARG_0: "[noun] hjebcl - khrezc (f)"

scala> dictConversions.check
! Falsified after 2 passed tests.
> ARG_0: "[noun] pcklo - fgrti (f)"

scala> dictConversions.check
! Falsified after 2 passed tests.
> ARG_0: "[noun] gmcz - abhh (f)"

scala> dictConversions.check
! Falsified after 4 passed tests.
> ARG_0: "[noun] kd - cv (f)"

scala> dictConversions.check
! Falsified after 10 passed tests.
> ARG_0: "[noun] av - ws (f)"
```

## class Dict:

```
...
(items[1].length() == items[3].length() ? "m" : "f")
...
```

## class Dict:

```
...
(items[1].length() == items[3].length() ? "m" : "f")
...
```

## class Dict2:

```
...
"f"
...
```

## class Dict:

```
...
(items[1].length() == items[3].length() ? "m" : "f")
...
```

## class Dict2:

```
...
"f"
...
```

```
scala>
```

## class Dict:

```
...
(items[1].length() == items[3].length() ? "m" : "f")
...
```

## class Dict2:

```
...
"f"
...
```

```
scala> def dict2Conversions = forAll(dictGen)( (dict:String) =>
     |   Dict2.toText(Dict2.toXml(dict)) == dict )
```

## class Dict:

```
...
(items[1].length() == items[3].length() ? "m" : "f")
...
```

## class Dict2:

```
...
"f"
...
```

```
scala> def dict2Conversions = forAll(dictGen)( (dict:String) =>
     |   Dict2.toText(Dict2.toXml(dict)) == dict )
dict2Conversions: org.scalacheck.Prop

scala>
```

## class Dict:

```
...
(items[1].length() == items[3].length() ? "m" : "f")
...
```

## class Dict2:

```
...
"f"
...
```

```
scala> def dict2Conversions = forAll(dictGen)( (dict:String) =>
     |   Dict2.toText(Dict2.toXml(dict)) == dict )
dict2Conversions: org.scalacheck.Prop

scala> dict2Conversions.check
```

## class Dict:

```
...
(items[1].length() == items[3].length() ? "m" : "f")
...
```

## class Dict2:

```
...
"f"
...
```

```
scala> def dict2Conversions = forAll(dictGen)( (dict:String) =>
     |   Dict2.toText(Dict2.toXml(dict)) == dict )
dict2Conversions: org.scalacheck.Prop

scala> dict2Conversions.check
+ OK, passed 100 tests.
```

# Thank you!

www.grzegorzbalcerek.net

gb@grzegorzbalcerek.net