Monoids

Piyush Mishra
Software Consultant
Knoldus Software LLP

Topics Covered

What is Monoid

Formal definition of a Monoid

Define a String Monoid

Define a Int Monoid

Define a List Monoid

Fold lists with Monoids

What is Monoid

Monoids are certain common patterns followed by the algebras of various data types of a programming language .

Integers with addition. We know that (a+b)+c == a+(b+c) and 0+n == n+0 == n

same with multiplication: (a*b)*c == a*(b*c) and 1*n == n*1 == n

Strings with concatenation. a+(b+c)==(a+b)+c; ""+s==s and s+"" == s, etc.

Lists with concatenation, like List(1,2)+List(3,4) == List(1,2,3,4)

Sets with their union, like Set(1,2,3)+Set(2,4) == Set(1,2,3,4).

This + binary operation is the common pattern.

Formal definition of a monoid

Given a type A, a binary operation Op:(A,A) => A, and an instance Zero: A, with the properties that will be specified below, the triple (A, Op, Zero) is called a monoid. Here are the properties:

Neutral or identity element : Zero \`Op\` a == a \`Op\` Zero == a
Associativity: (a \`Op\` b) \`Op\` c == a \`Op\` (b \`Op\` c)

We can express this with a Scala trait:

```
trait Monoid[A] {
def op(a1: A, a2: A): A
def zero: A
}
```

Define a String monoid

```
trait BaseMonoid[A] {
  def op(a: A, b: A): A
  def zero: A
}

class StringMonoid extends BaseMonoid[String] {
  def op(a: String, b: String) = (a.trim + " " + b.trim).trim
  def zero = ""
}
```

Here we a operation (op) in which we adding two string with space delimiter.

Define a Int monoid

```
trait BaseMonoid[A] {
  def op(a: A, b: A): A
  def zero: A
}

class IntegerMonoid extends BaseMonoid[Int] {
  def op(a: Int, b: Int) = a + b
  def zero = 0
}
```

Here we a operation (op) in which we sum two two integer

Define a List monoid

```
trait BaseMonoid[A] {
  def op(a: A, b: A): A
  def zero: A
}

class ListMonoid[A] extends BaseMonoid[List[A]] {
  def op(a: List[A], b: List[A]): List[A] = a ++ b
  def zero = Nil
}
```

Here we a operation (op) in which we adding two two Lists

Folding a list with a monoid

```
def foldRight(z: A)(f: (A, A) => A): A
def foldLeft(z: A)(f: (A, A) => A): A

val words = List("hello", "world", "Whats Up")

scala> val s = words.foldRight(stringMonoid.zero)
(stringMonoid.op)
s: String = "hello world Whats Up"
scala> val t = words.foldLeft(stringMonoid.zero)
(stringMonoid.op)
t: String = "hello world Whats Up"
```

Function which folds a list with a monoid

```
def concatenate[A](as: List[A], m: Monoid[A]): A

def concatenate[A](as: List[A], m: BaseMonoid[A]): A =
as.foldRight(m.zero)(m.op)
```

Operation using different monoids instance

```
def foldMap[A,B](as: List[A], m: Monoid[B])(f: A => B): B

def foldMap[A, B](as: List[A], m: Monoid[B])(f: A => B): B =
  as.foldLeft(m.zero)((b, a) => m.op(b, f(a)))
```

Manipulate a json using monoid

```
 val json = """{
"first": "John",
"last": "Doe",
"credit_card": 5105105105105100,
"ssn": "123-45-6789",
"salary": 70000,
"registered": true }"""

 class JsonMonoid extends Monoid[String] {
   val CREDIT_CARD_REGEX = "\\b\\d{13,16}\\b"
   val SSN_REGEX = "\\b[0-9]{3}-[0-9]{2}-[0-9]{4}\\b"
   val BLOCK_TEXT = "*********"

   def identity = ""

def op(a1: String, a2: String) = a1 + a2.replaceAll(CREDIT_CARD_REGEX,
BLOCK_TEXT).replaceAll(SSN_REGEX, BLOCK_TEXT) + ","
  }

  val monoid = new JsonMonoid
  val result = json.split(',').foldLeft(monoid.identity)(monoid.op)
```

Thanks