Monoids & Friends

Saulius Valatka

Recap so far

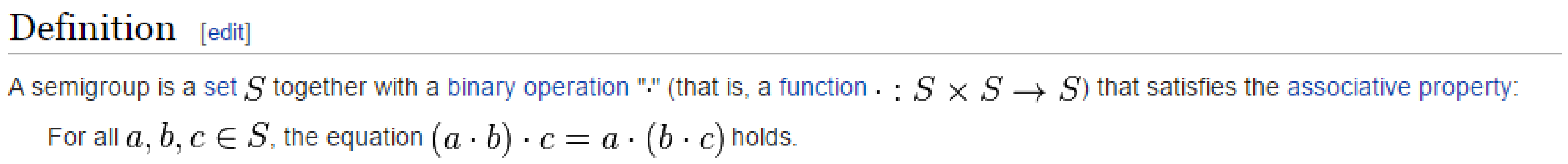
Monads are just **monoids** in the category of endofunctors

```scala
def sum[T](list: List[T]): T = ???
```

```scala
trait Adder[T] {
  def add(lhs: T, rhs: T): T
}
```

```scala
trait Adder[T] {
  def add(lhs: T, rhs: T): T
}


class IntAdder extends Adder[Int] {
  def add(lhs: Int, rhs: Int): Int = lhs + rhs
}
```

```scala
trait Adder[T] {
  def add(lhs: T, rhs: T): T
}


class IntAdder extends Adder[Int] {
  def add(lhs: Int, rhs: Int): Int = lhs + rhs
}


def sum[T](list: List[T])(adder: Adder[T]): T =
  list.reduce((a, b) => adder.add(a, b))
```



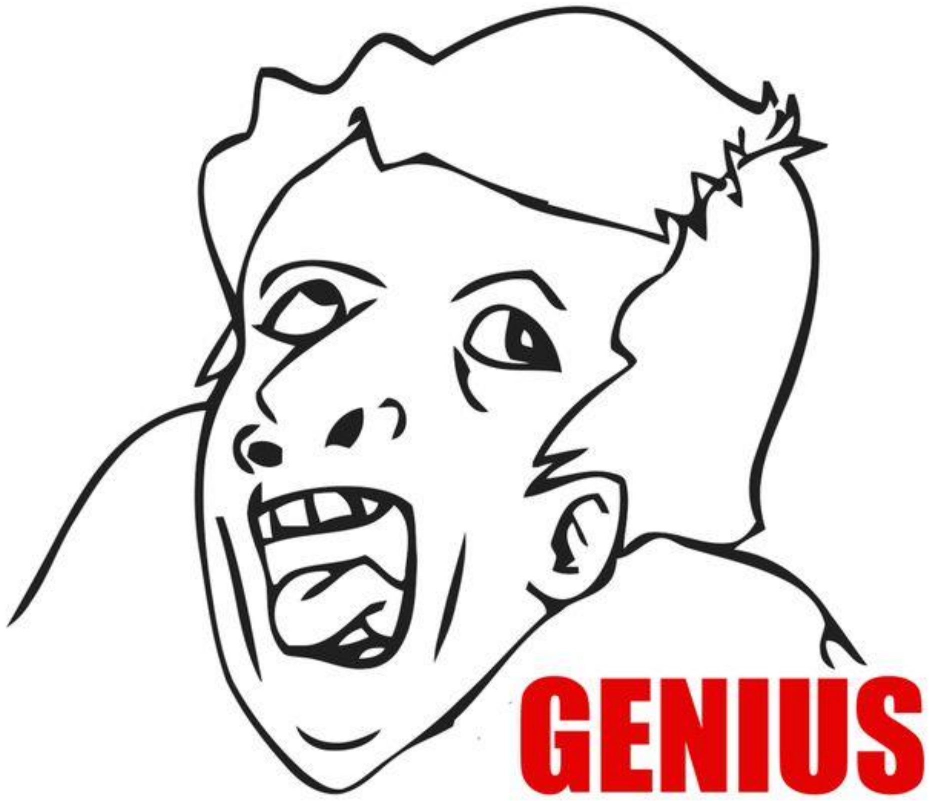
Definition [edit]

A semigroup is a set S together with a binary operation "·" (that is, a function $\cdot : S \times S \rightarrow S$) that satisfies the associative property:

For all $a, b, c \in S$, the equation $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ holds.

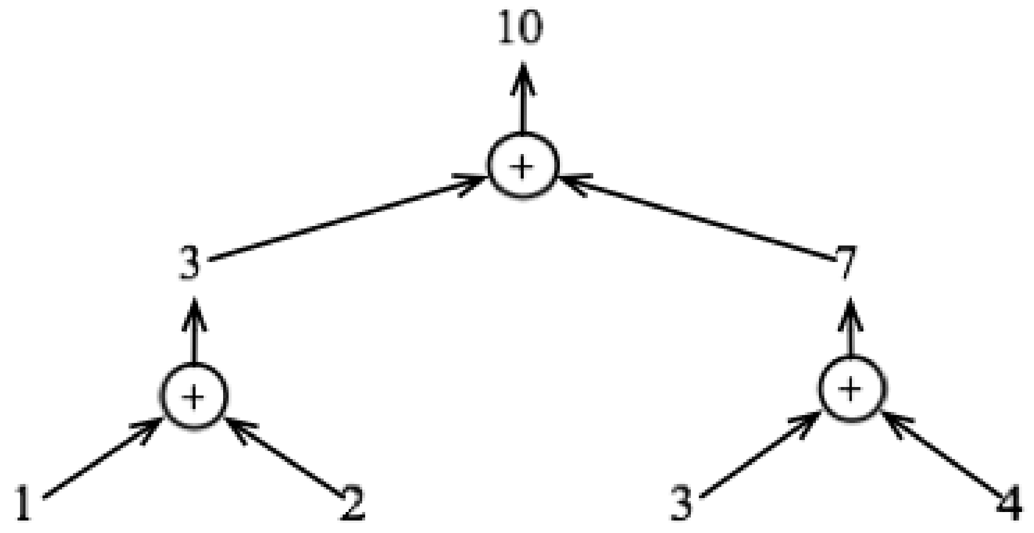
```
      Semigroup
trait ~~Adder~~[T] {
  def append(t1: T, t2: T): T
}
```

```
      Semigroup
trait ~~Adder~~[T] {
  def append(t1: T, t2: T): T
}
```



```scala
import org.scalacheck.Properties
import org.scalacheck.Prop.forAll

object SemgroupLaws extends Properties("semigroup") {

  val m = new IntSumSemigroup

  property("associativity") = forAll { (a: Int, b: Int, c: Int) =>
    m.append(m.append(a, b), c) == m.append(a, m.append(b, c))
  }
}
```

```
+ semigroup.associativity: OK, passed 100 tests.
```


10
+
3
7
+
+
1
2
3
4

“stri” + “ng” == “string”

1 + 2 = 3

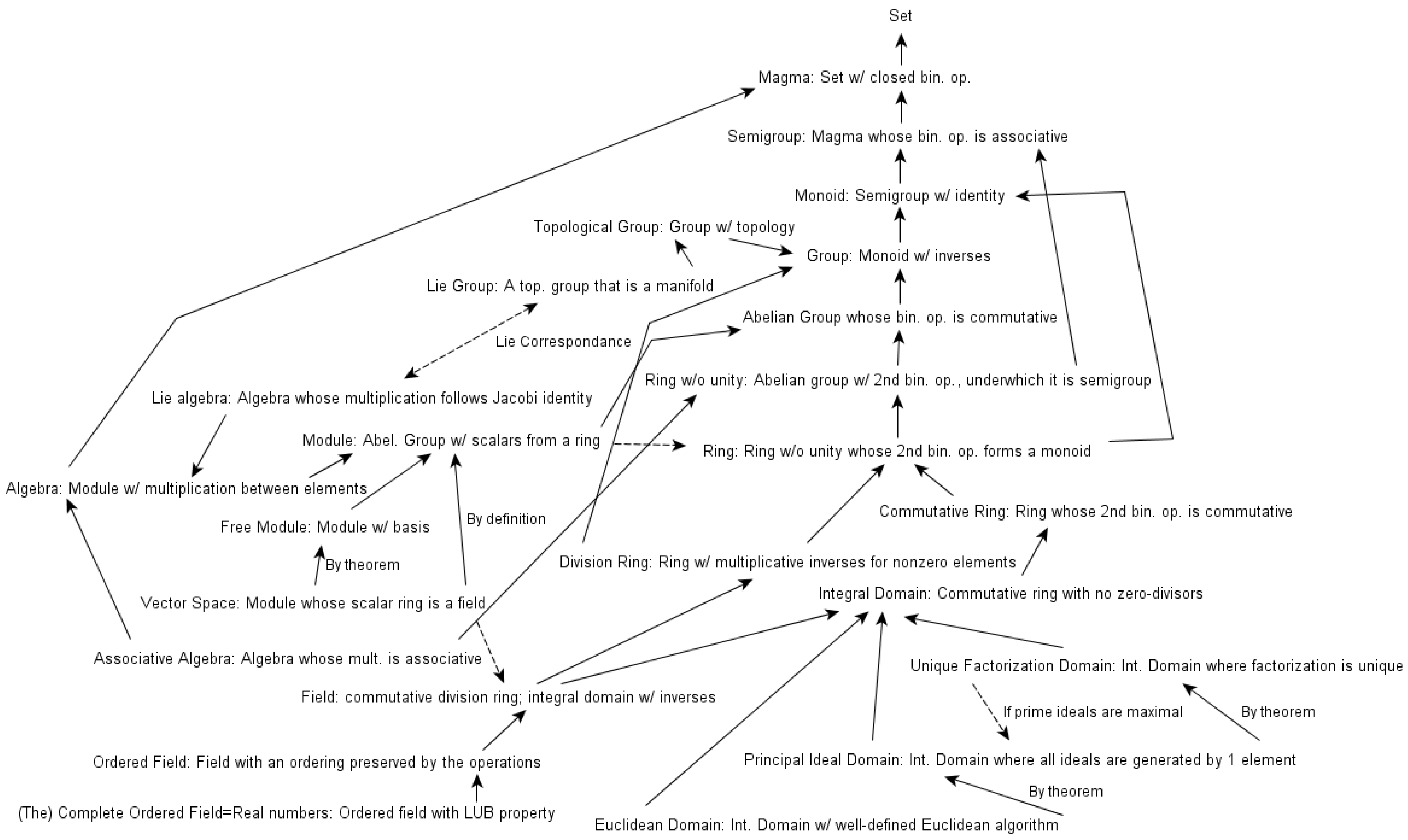
[1, 2] + [3, 4] = [1, 2, 3, 4]

3 hrs 2 mins + 5 mins = 3 hrs 7 mins

set(1, 2, 3) + set(2, 3, 4) = set(1, 2, 3, 4)

etc …

Finding the right, or most appropriate, abstractions is the most important part of engineering software systems […]They are the fundamental job of software engineering.


Set
Magma: Set w/ closed bin. op.
Semigroup: Magma whose bin. op. is associative
Monoid: Semigroup w/ identity
Topological Group: Group w/ topology
Group: Monoid w/ inverses
Lie Group: A top. group that is a manifold
Abelian Group whose bin. op. is commutative
Lie Correspondance
Ring w/o unity: Abelian group w/ 2nd bin. op., underwhich it is semigroup
Lie algebra: Algebra whose multiplication follows Jacobi identity
Module: Abel. Group w/ scalars from a ring
Ring: Ring w/o unity whose 2nd bin. op. forms a monoid
Algebra: Module w/ multiplication between elements
Commutative Ring: Ring whose 2nd bin. op. is commutative
Free Module: Module w/ basis
By definition
Division Ring: Ring w/ multiplicative inverses for nonzero elements
By theorem
Integral Domain: Commutative ring with no zero-divisors
Vector Space: Module whose scalar ring is a field
Associative Algebra: Algebra whose mult. is associative
Unique Factorization Domain: Int. Domain where factorization is unique
Field: commutative division ring; integral domain w/ inverses
If prime ideals are maximal
By theorem
Principal Ideal Domain: Int. Domain where all ideals are generated by 1 element
Ordered Field: Field with an ordering preserved by the operations
By theorem
(The) Complete Ordered Field=Real numbers: Ordered field with LUB property
Euclidean Domain: Int. Domain w/ well-defined Euclidean algorithm

```scala
trait Monoid[T] extends Semigroup[T] {
  def zero: T
}
```

```scala
trait Group[T] extends Monoid[T] {
  def inverse(t: T): T
}
```

Integers:	3 – 3 = 0
Rotations:	left + right = nothing
DB operations:	insert + delete = nothing

```scala
trait Ring[T] {
  def addition: Group[T]
  def multiplication: Monoid[T]
}

trait Field[T] {
  def addition: Group[T]
  def multiplication: Group[T]
}
```

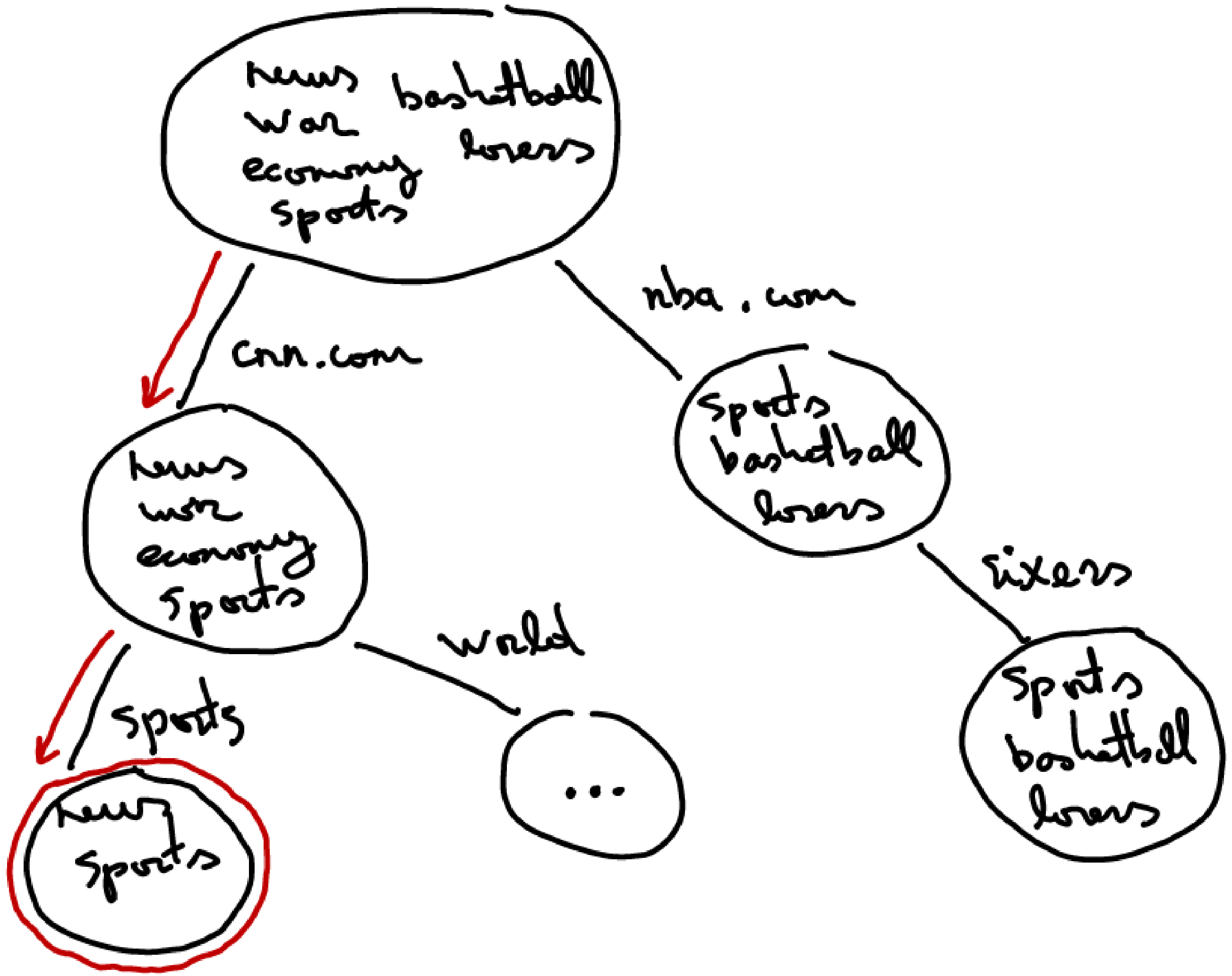
```scala
class Matrix[T](private val items: Seq[Seq[T]]) {

  def add(m: Matrix[T])(implicit sg: Semigroup[T]): Matrix[T] = /* excercise */

  def times(m: Matrix[T])(implicit rg: Ring[T]): Matrix[T] = /* excercise */

  def inverse(implicit fl: Field[T]): Matrix[T] = /* excercise */

}
```

NOT IMPRESSED

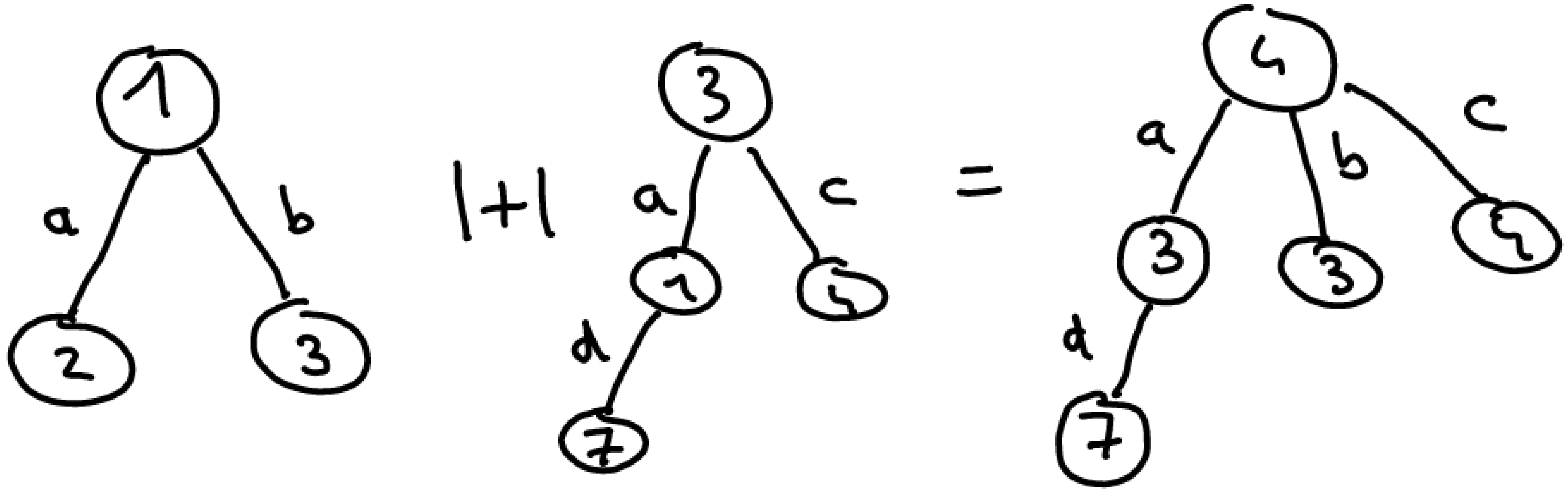
Real Life Example: Classifying URLs

Problem

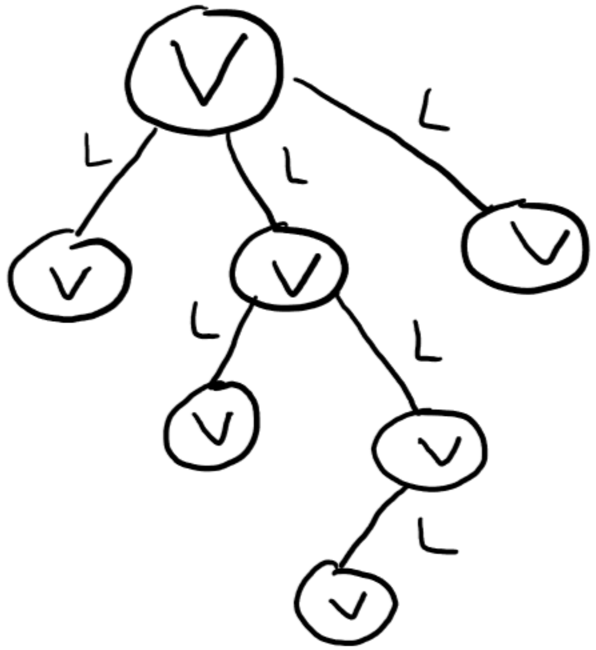
I know that http://www.cnn.com/sports/ufc/cormier-new-lhw-champ is about sports, what about http://www.cnn.com/sports/nba/cavs-advance-to-finals ?


news war economy sports basketball lovers
cnn.com
nba.com
news war economy sports
sports basketball lovers
sports
world
sixers
news sports
...
sports basketball lovers

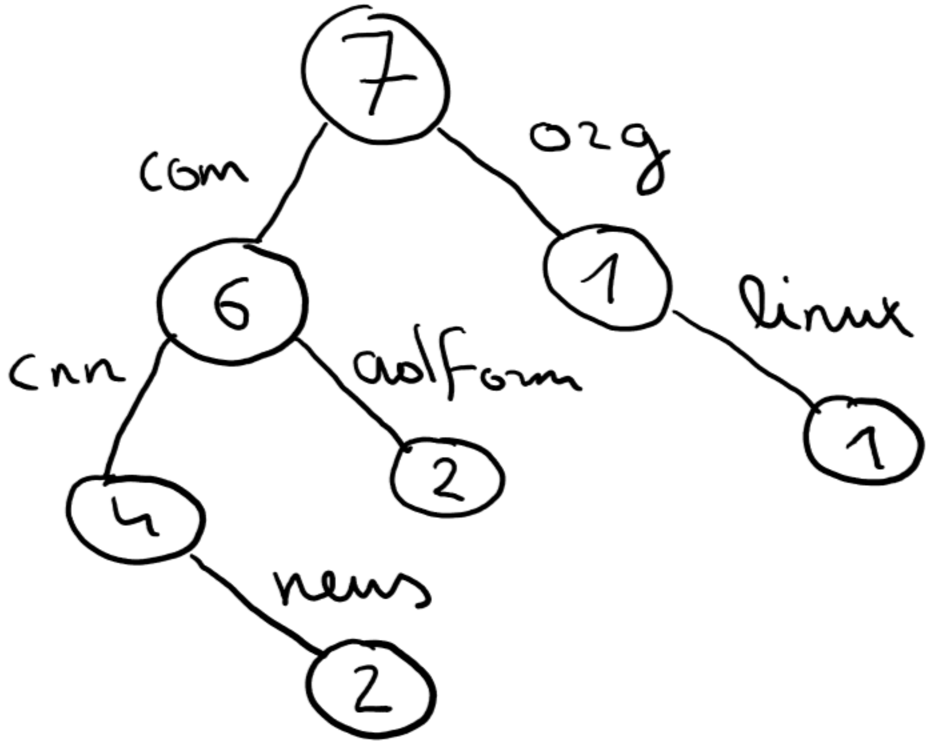
NOT SURE IF MONOID OR ...


1
a
b
2
3
|+|
3
a
c
1
4
d
7
=
4
a
b
c
3
3
4
d
7

```scala
case class LabeledBranchTree[L, V](value: V, branches: Map[L, LabeledBranchTree[L, V]])



implicit def monoid[L, V: Monoid]: Monoid[LabeledBranchTree[L, V]] = new Monoid[LabeledBranchTree[L, V]] {

  override def zero: LabeledBranchTree[L, V] =
     LabeledBranchTree(implicitly[Monoid[V]].zero, implicitly[Monoid[Map[L, LabeledBranchTree[L, V]]]].zero)

  override def append(f1: LabeledBranchTree[L, V], f2: => LabeledBranchTree[L, V]): LabeledBranchTree[L, V] =
     LabeledBranchTree(f1.value |+| f2.value, f1.branches |+| f2.branches)
}
```

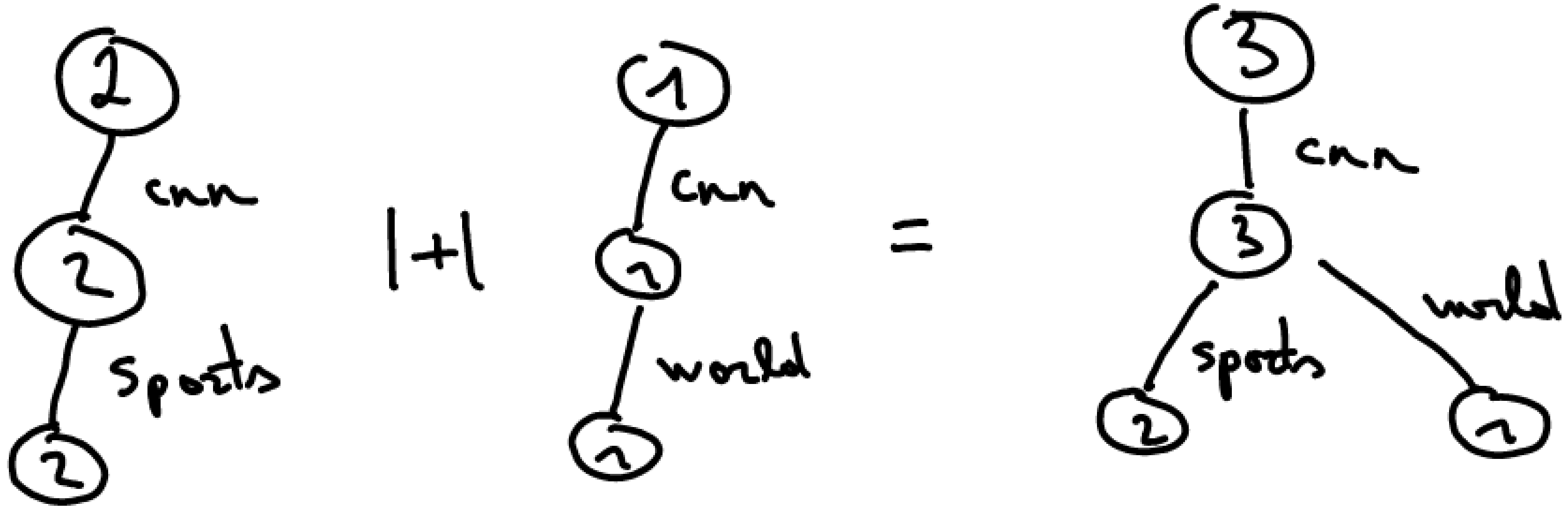


(V : Monoid)

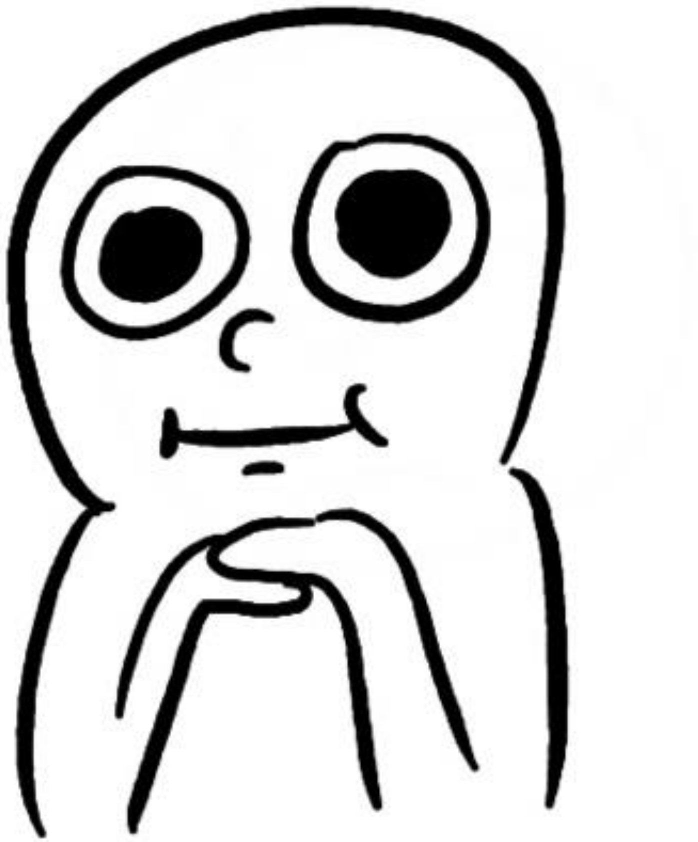


(L = String ; V = Int)

```
TextLine("data/pageviews.txt")
  .map(LogPageview.parse)
  .collect { case LogPageview(time, Some(url)) =>
     LabeledBranchTree.fromUrl(url -> 1)
  }
  .sum
  .write(TypedTsv("data/visits.txt"))
```

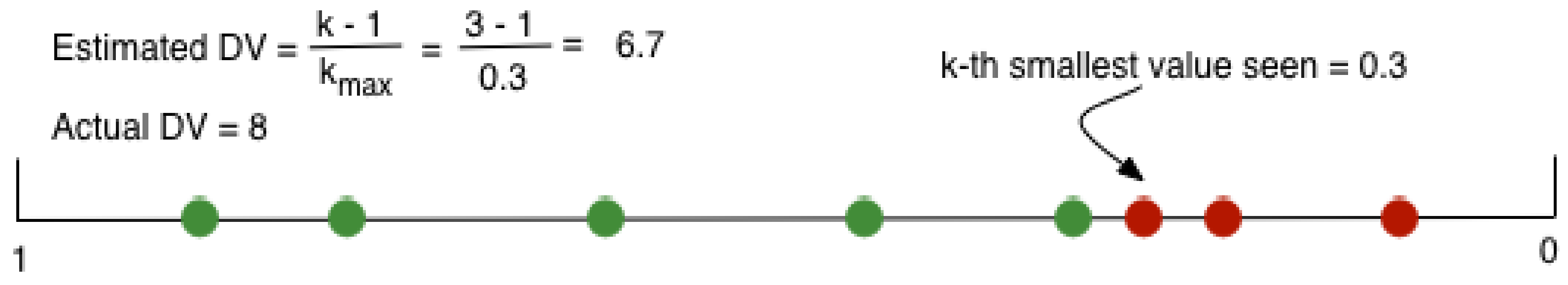

2
cnn
2
sports
2
|+|
1
cnn
1
world
1
=
3
cnn
3
sports
world
2
1

```
LabeledBranchTree[String, Int] counts visits

LabeledBranchTree[String, Set[Category]] tracks categories

LabeledBranchTree[String, Set[User]] tracks unique users

LabeledBranchTree[String, (Set[User], Set[Category], Int)] does all of the above
```



TELL ME MORE

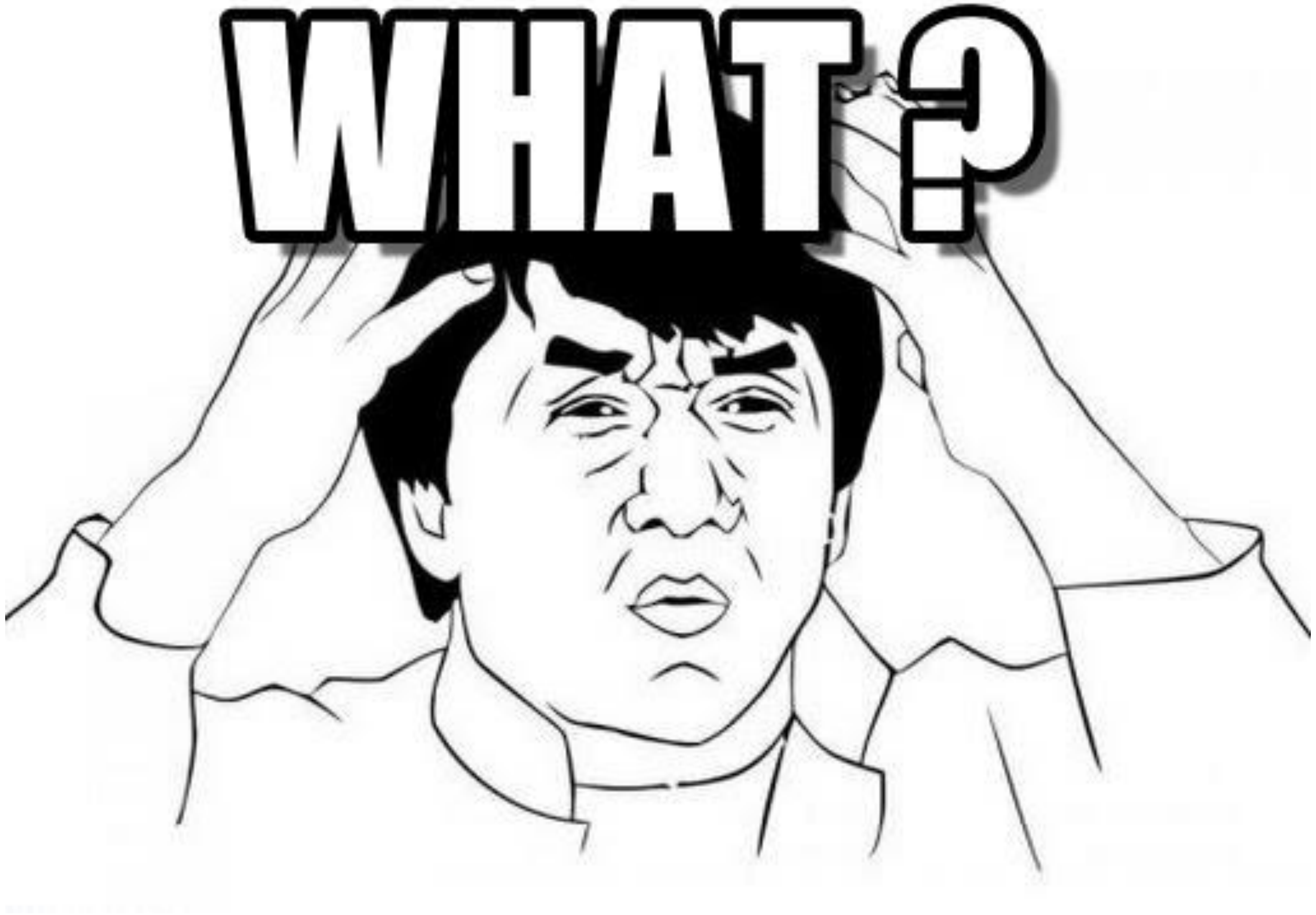
Approximate Counting Monoid



Estimated DV = $\frac{k - 1}{k_{max}} = \frac{3 - 1}{0.3} = 6.7$
Actual DV = 8
k-th smallest value seen = 0.3
1
0

`LabeledBranchTree[String, HyperLogLog] tracks unique users efficiently`

TL;DR: good abstractions make your life super easy


WHAT ?