

# Scaps

Retrieving Values with Types

What?

Why?

How?

What's next?



Who?

Lukas Wegmann



What?

Find Reusable Functionality in Libraries with Types



# Find Reusable Functionality in Libraries with Types

Public  
Values

Public  
Methods

Public  
Constructors

Snippets

Types

# Find Reusable Functionality in **Libraries** with Types





# Find Reusable Functionality in Libraries *with Types*

Query:

$A \Rightarrow B$

Potentially Helpful Signatures:

$A \Rightarrow B$

$A \Rightarrow X \Rightarrow B$

$A1 \Rightarrow B$  if  $A <: A1$

$A \Rightarrow B1$  if  $B1 <: B$

$A \Rightarrow X[B]$

$X[A] \Rightarrow B$

$(X \Rightarrow A) \Rightarrow B$

$A \Rightarrow (B \Rightarrow X) \Rightarrow Y$

$\text{Promise}[B1] \Rightarrow \text{Promise}[A1]$  if  $A <: A1, B1 <: B$

# Find Reusable Functionality in Libraries with Types and Keywords

```
print: String => _
```

# Find Reusable Functionality in Libraries with Types

Scaps

scala-library:2.11.7

**List[A].mkString(String): String**

Displays all elements of this list in a string using a separator string.

**params**

**sep**  
the separator string.

**returns**  
a string representation of this list. In the resulting string the string representations (w.r.t. the method `toString`) are separated by the string `sep`.

**example**

```
List(1, 2, 3).mkString("|") = "1|2|3"
```

**scala-library** scala.collection.immutable.List.mkString

[Doc](#) · [👍 This is what i've been looking for](#)

17 more results matching **mkString: \_ => \_ => \_**

Why?



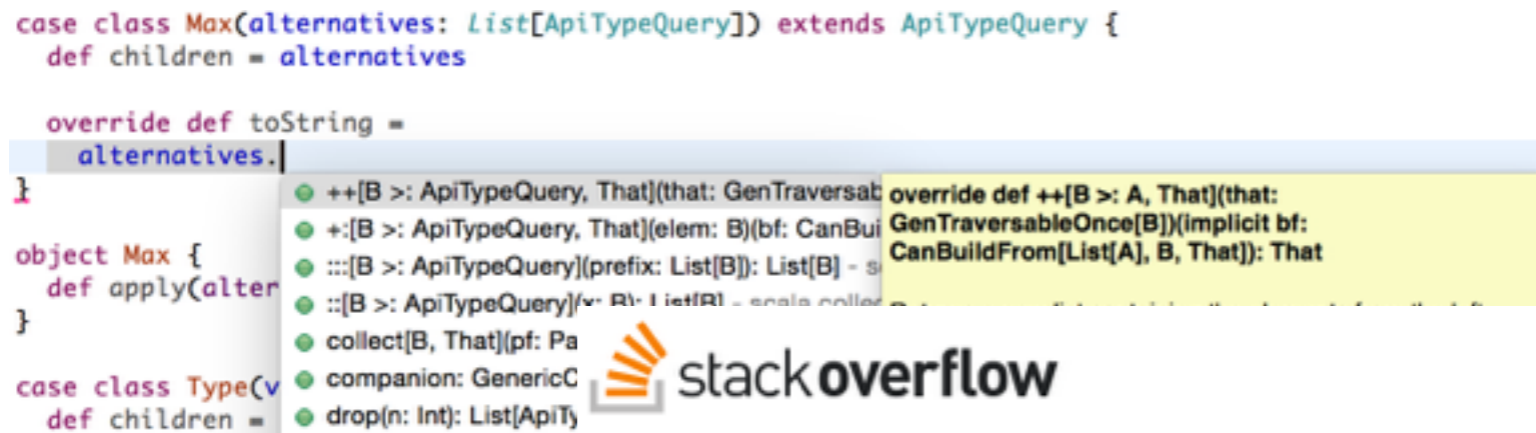
# Claim 1: Current tools don't match the expressiveness of Scala

```
case class Max(alternatives: List[ApiTypeQuery]) extends ApiTypeQuery {
  def children = alternatives

  override def toString =
    alternatives.
}

object Max {
  def apply(alternatives: List[ApiTypeQuery]) = Max(alternatives)
}

case class TypeQuery(children: List[ApiTypeQuery]) extends ApiTypeQuery {
  def children = children
}
```



Questions

## Scala: join an iterable of strings

▲ How do I "join" an iterable of strings by another string in Scala?

78

```
val thestrings = Array("a", "b", "c")
val joined = ???
println(joined)
```



I want this code to output `a,b,c` (join the elements by ",").

Claim 1: Current tools don't match the **expressiveness** of Scala

`java.util.List`  
32 Methods

`scala.collection.immutable.List`  
**177** Methods

`java.util.stream.Stream`  
**45** Methods

Claim 2: We are used to formulate problems with types

Scala: join an iterable of strings

▲ How to transform Scala collection of Option[X] to collection of X

78

▼ ▲ Un-optioning an optioned Option

43



2

▼ ▲ How to return an option when reading a vector

25



9

▼ ▲ Reading from a vector, I want to return none when trying to read an index that is out of bounds or some otherwise. Is there a standard method for this?

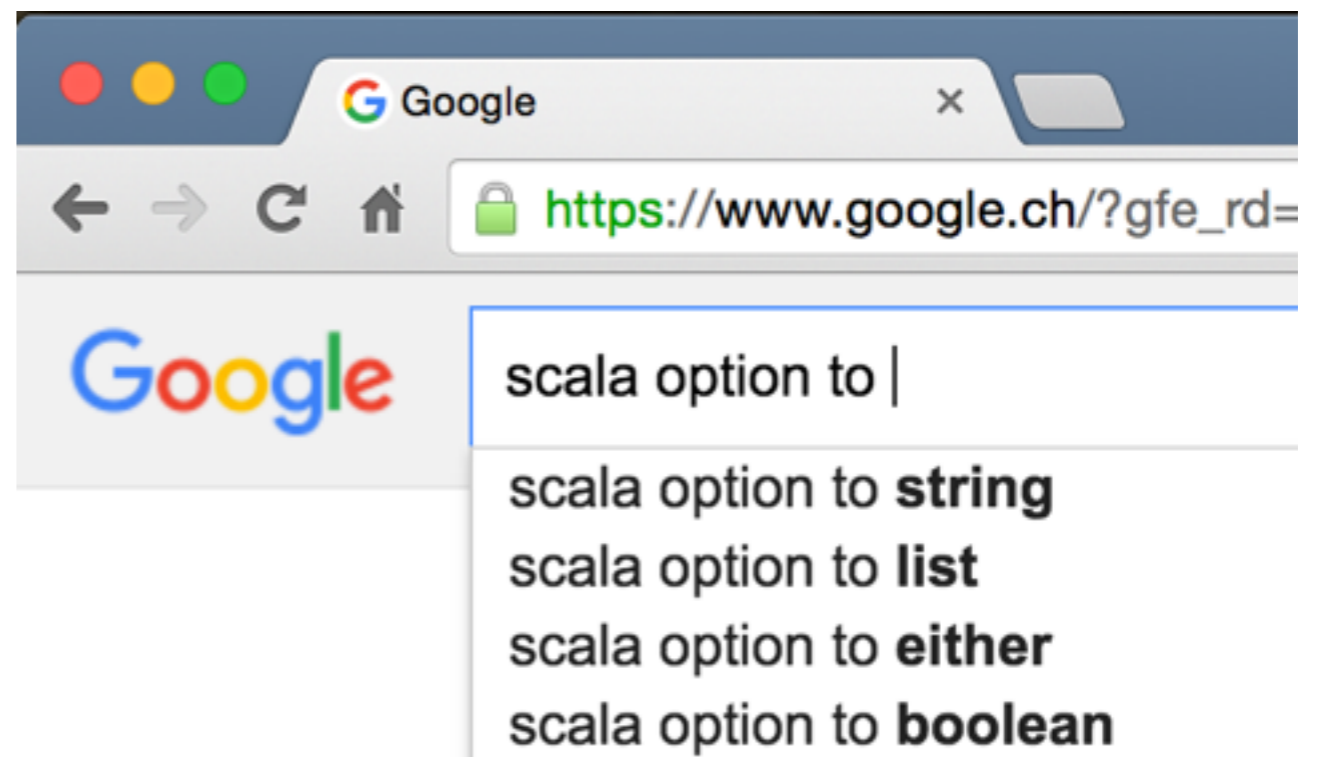
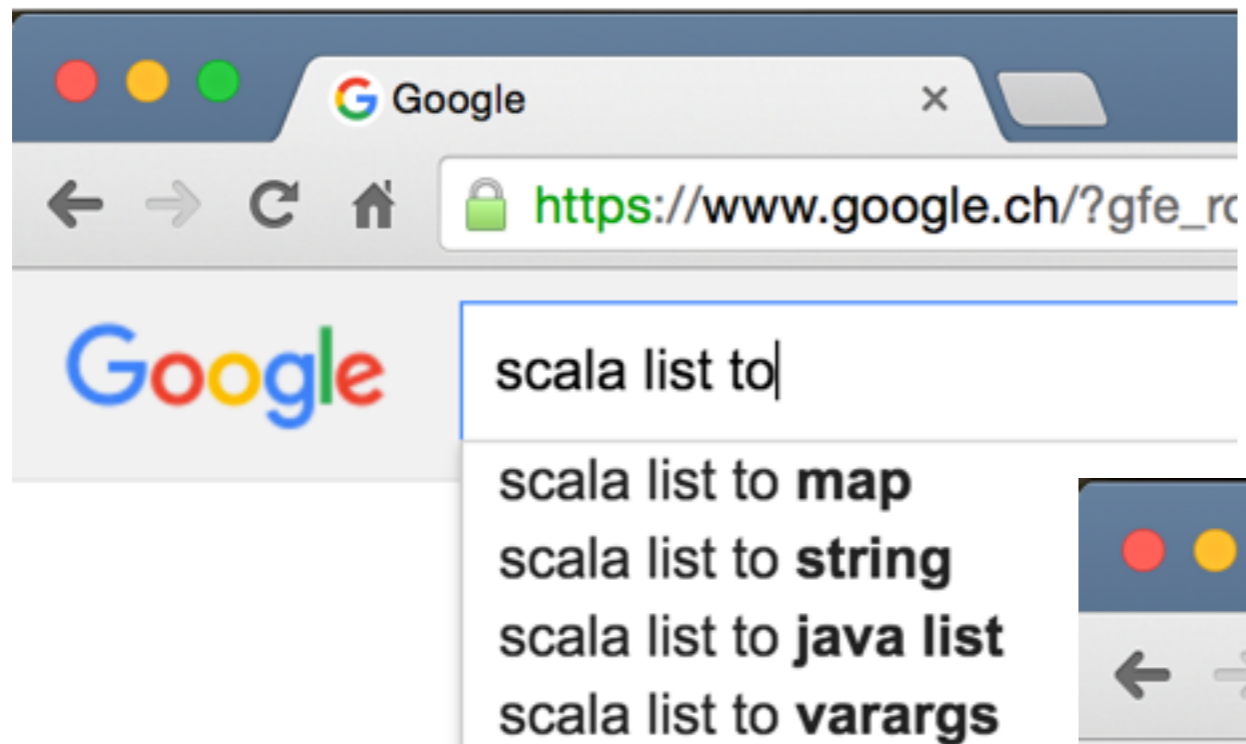
5



3

scala

Claim 2: We are used to **formulate problems with types**





# Claim 3: What works for Haskell wont work for Scala

Hoogle

[String] -> String -> String

**showCommandForUser** :: FilePath -> [String] -> String

process System.Process

Given a program p and arguments args, showCommandForUser p args returns a string suitable for pasting into sh (on POSIX OSs) or cmd.exe (on Windows).

**intercalate** :: [a] -> [[a]] -> [a]

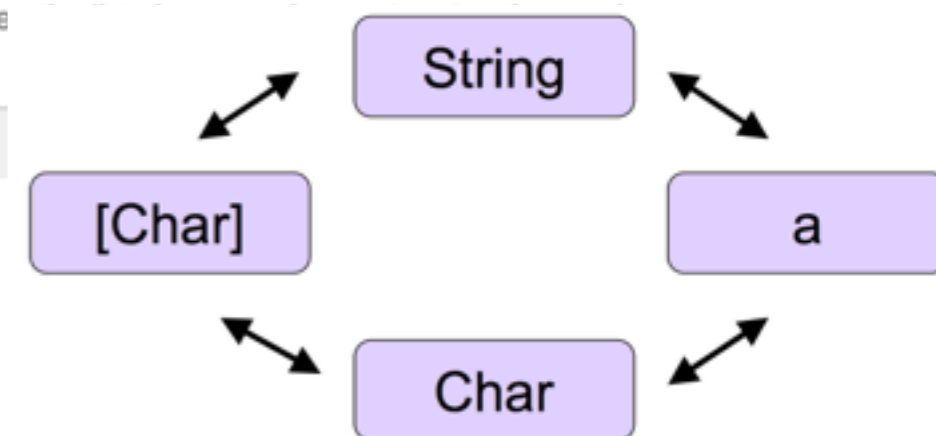
base Data.List

intercalate xs xss is equivalent to (concat (intersperse xs xss)). It inserts the list xs in between

**(<.)** :: FilePath -> String -> FilePath

filepath System.FilePath.Windows, filepath System.FilePath.Posix

Alias to addExtension, for people who like that sort of thing.



Claim 3: What works for Haskell **wont work for Scala**

GenTraversableOnce

~350 **Subtypes**

List

~50 **Supertypes**

List[\_] => GenTraversableOnce[GenTraversableOnce[\_]]  
> 6'000'000 **Subtypes**

How?

## Query by **Keywords**

```
sort: Array[Int] => Unit
```

```
object Sorting {  
  /** Sort an array of K where K is Ordered, preserving the existing order  
   * where the values are equal. */  
  def stableSort[K: ClassTag: Ordering](a: Array[K]): Unit  
}
```

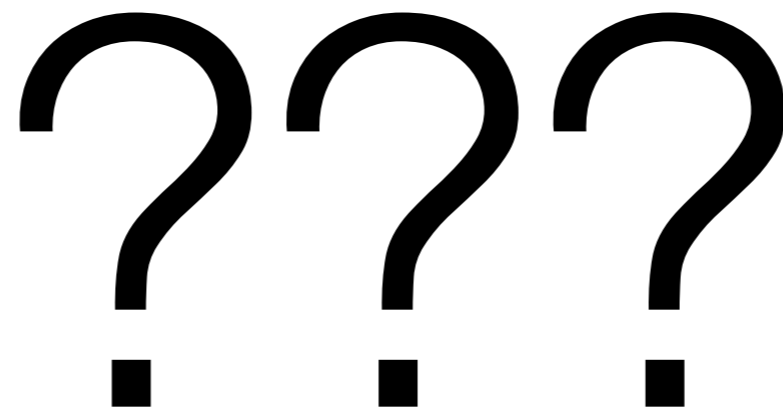




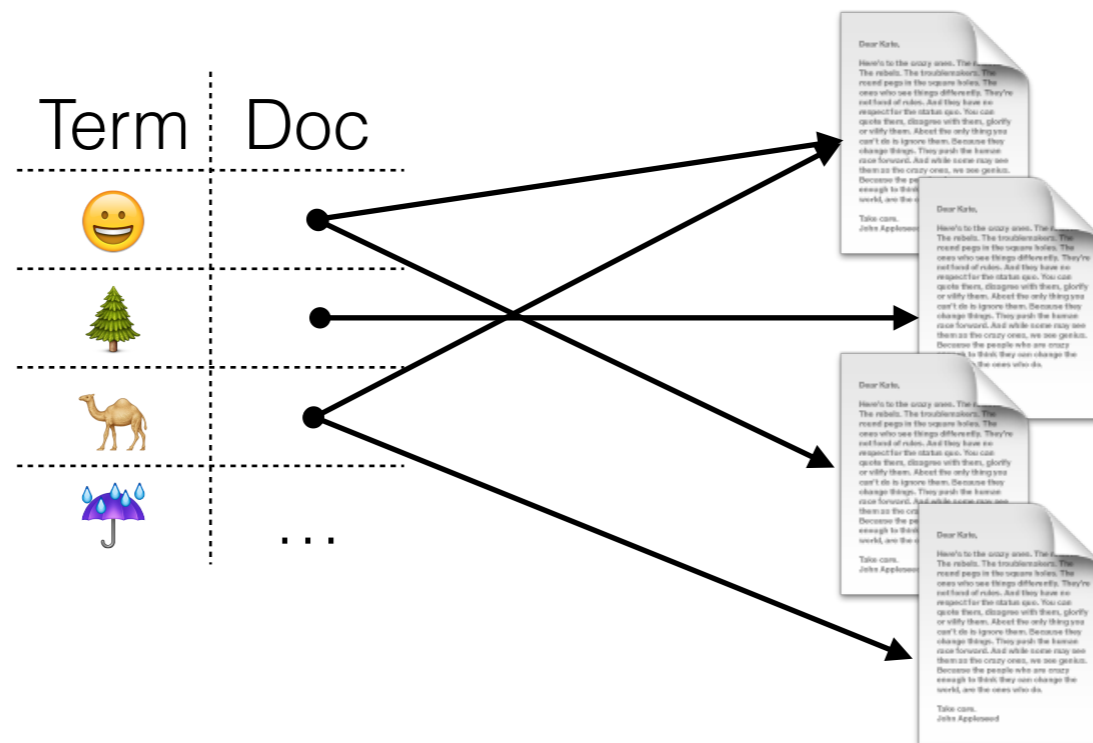
## Query by Types

sort: **Array[Int] => Unit**

```
object Sorting {  
  /** Sort an array of K where K is Ordered, preserving the existing order  
   * where the values are equal. */  
  def stableSort[K: ClassTag: Ordering](a: Array[K]): Unit  
}
```



Idea: Use **atomic terms** to characterize a type



## Step 1: Normalize Types

`(String, Int) => Person`

`new Person(String, Int)`

`String.(Int) => Person`



`String => Int => Person`

## Step 2: Add Polarity

`(String, Int) => Person`

`List[A] => A`

`Promise[String] => Unit`

`BitSet => (Int => A) => Set[A]`

**Covariant**

## Step 2: Add Polarity

`(String, Int) => Person`

`List[A] => A`

`Int => Promise[Char]`

`BitSet => (Int => A) => Set[A]`

Contravariant



## Step 2: Add Polarity

String => Array[Person]

java.util.List[A] => Int

# Invariant

## Step 2: Add Polarity

String => Array[Int] => Person



-String => -Array[\Int] => +Person

## Step 3: Substitute Type Parameters

Map[A, B] => List[(A, B)]

Map[B, A] => List[(A, B)]

Map[Key, Value] => List[(Key, Value)]

## Step 3: Substitute Type Parameters

`-List[-A] => -Array[\A] => +Option[+A]`



`-List[-Any] => -Array[\?] => +Option[+Nothing]`

Contravariant:  
Upper Bound

Invariant:  
???

Covariant:  
Lower Bound

## Step 4: Flatten

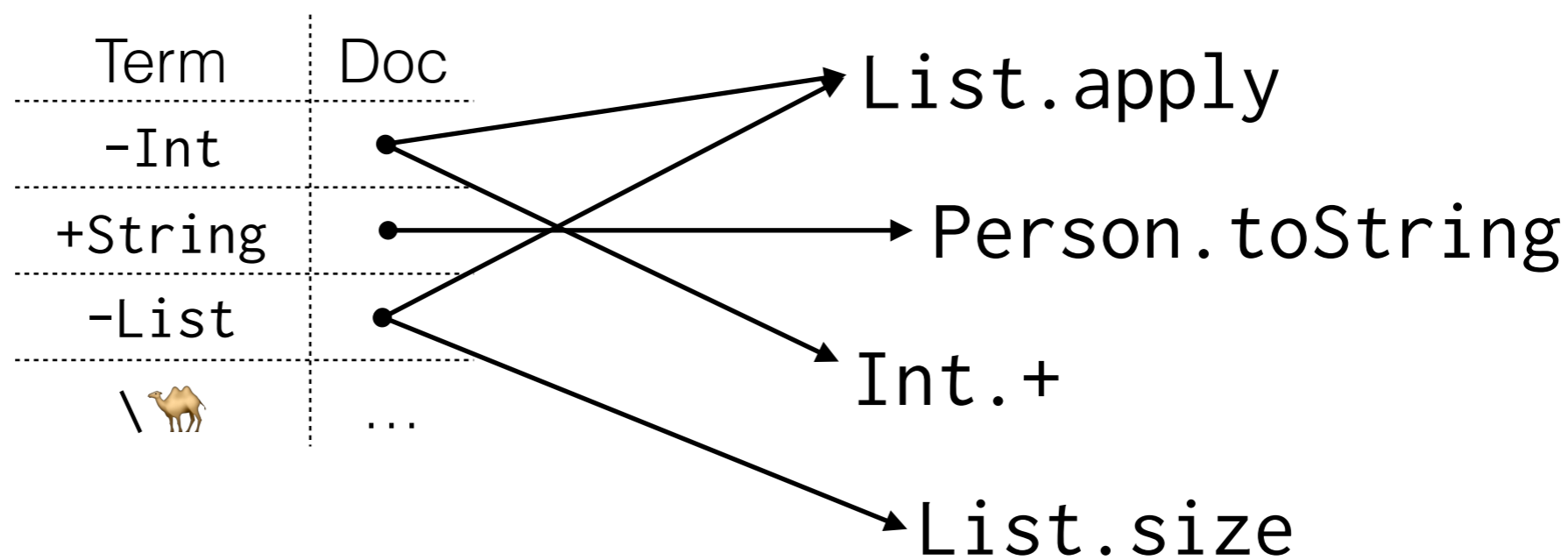
`-List[-Any] => -Array[\?] => +Option[+Nothing]`



`-List, -Any, -Array, \?, +Option, +Nothing`



What have we **won**?



What have we **lost**?

`scala.List`:

- 177 Members
- 118 Types
- 107 Fingerprints
- 7 **Collisions**
  - `foldLeft` / `foldRight`
  - `reduceLeft` / `reduceRight`
  - `equals` / `contains`
  - ...

Query Time!

`String => Set[Int]`

**Parse & Analyse**




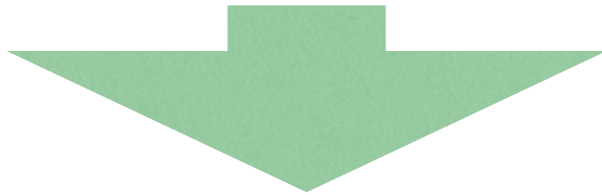
`java.lang.String => scala.collection.Set[scala.Int]`

**Normalize & Polarize**

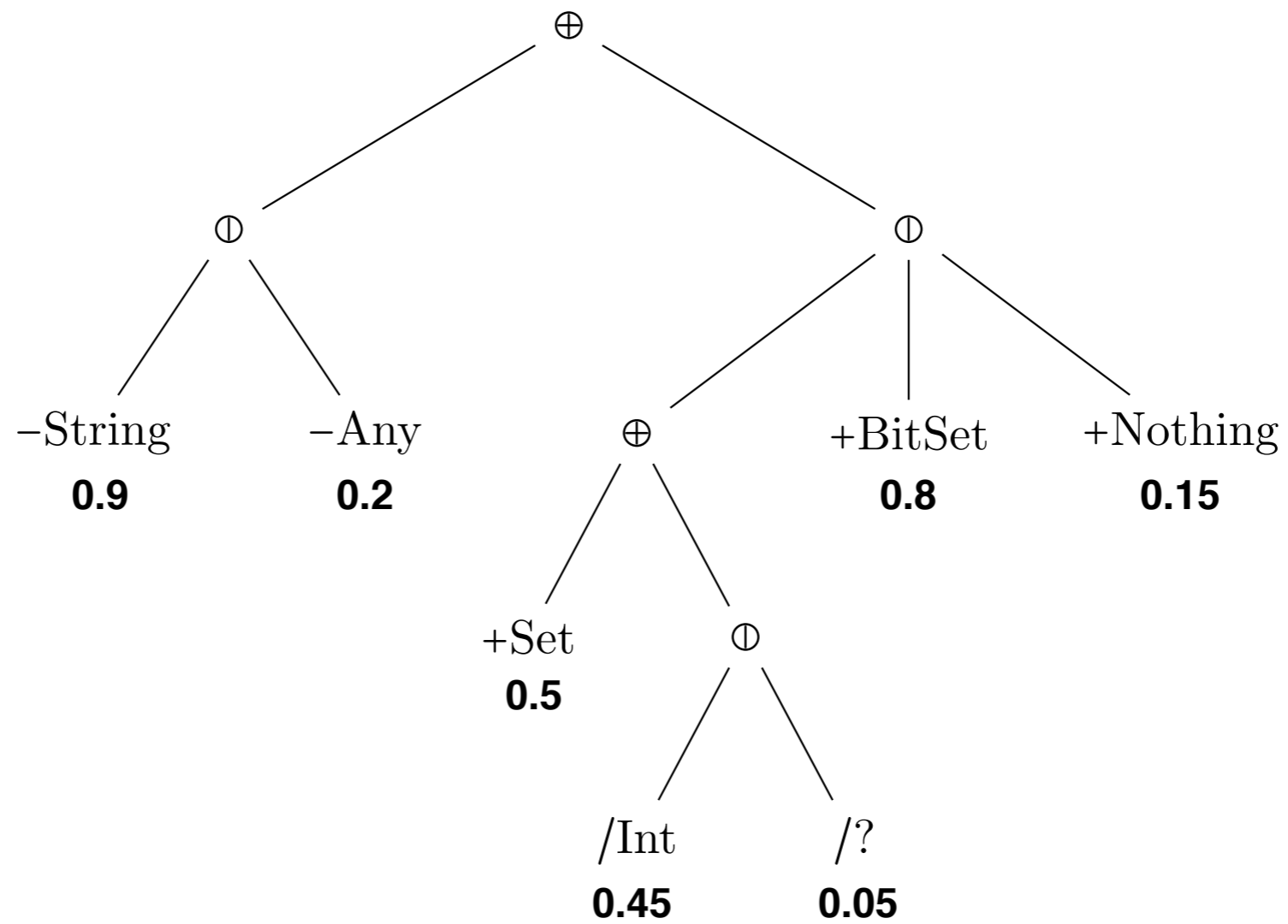


`-String => +Set[\Int]`

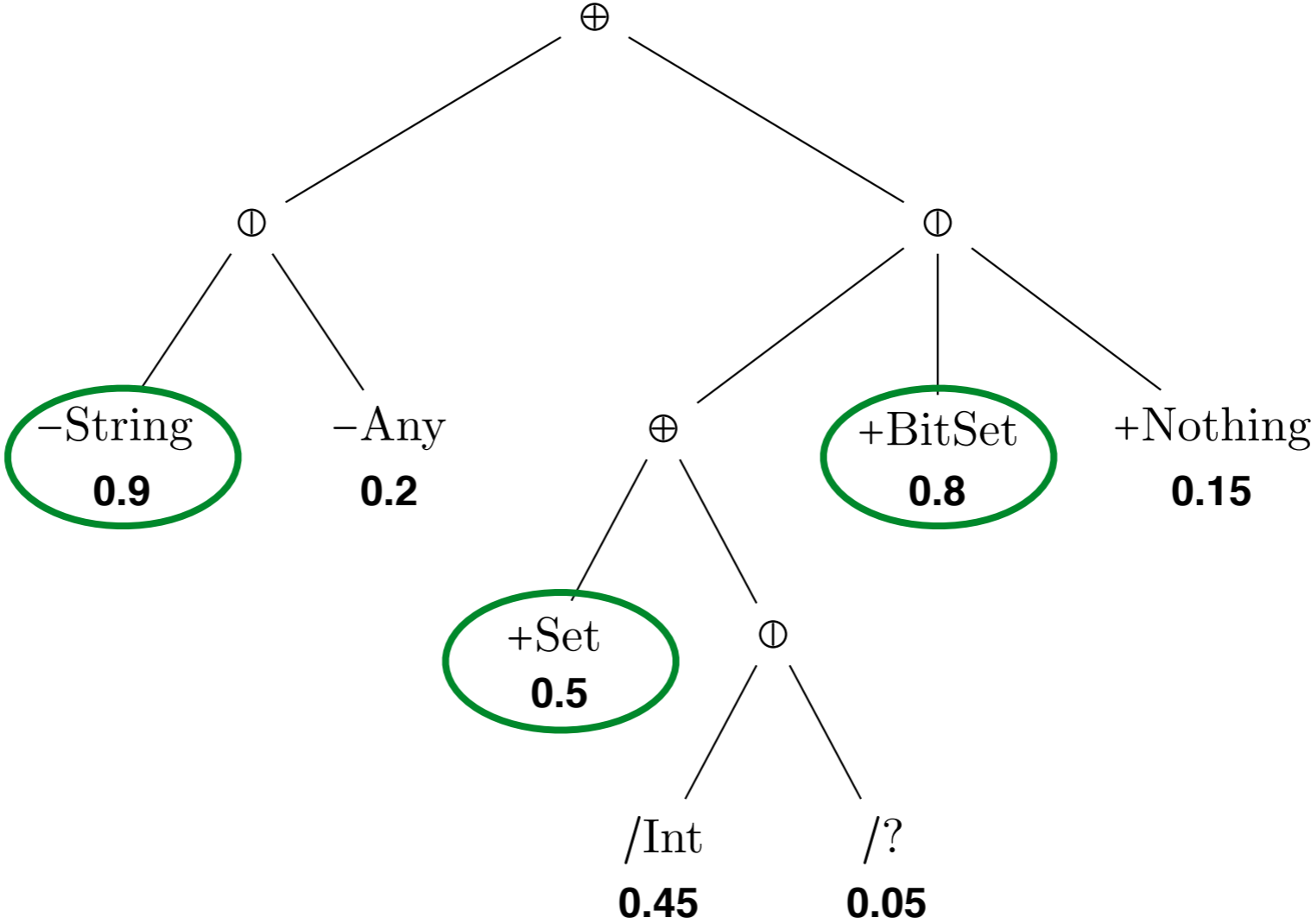




-String => +Set[\Int]



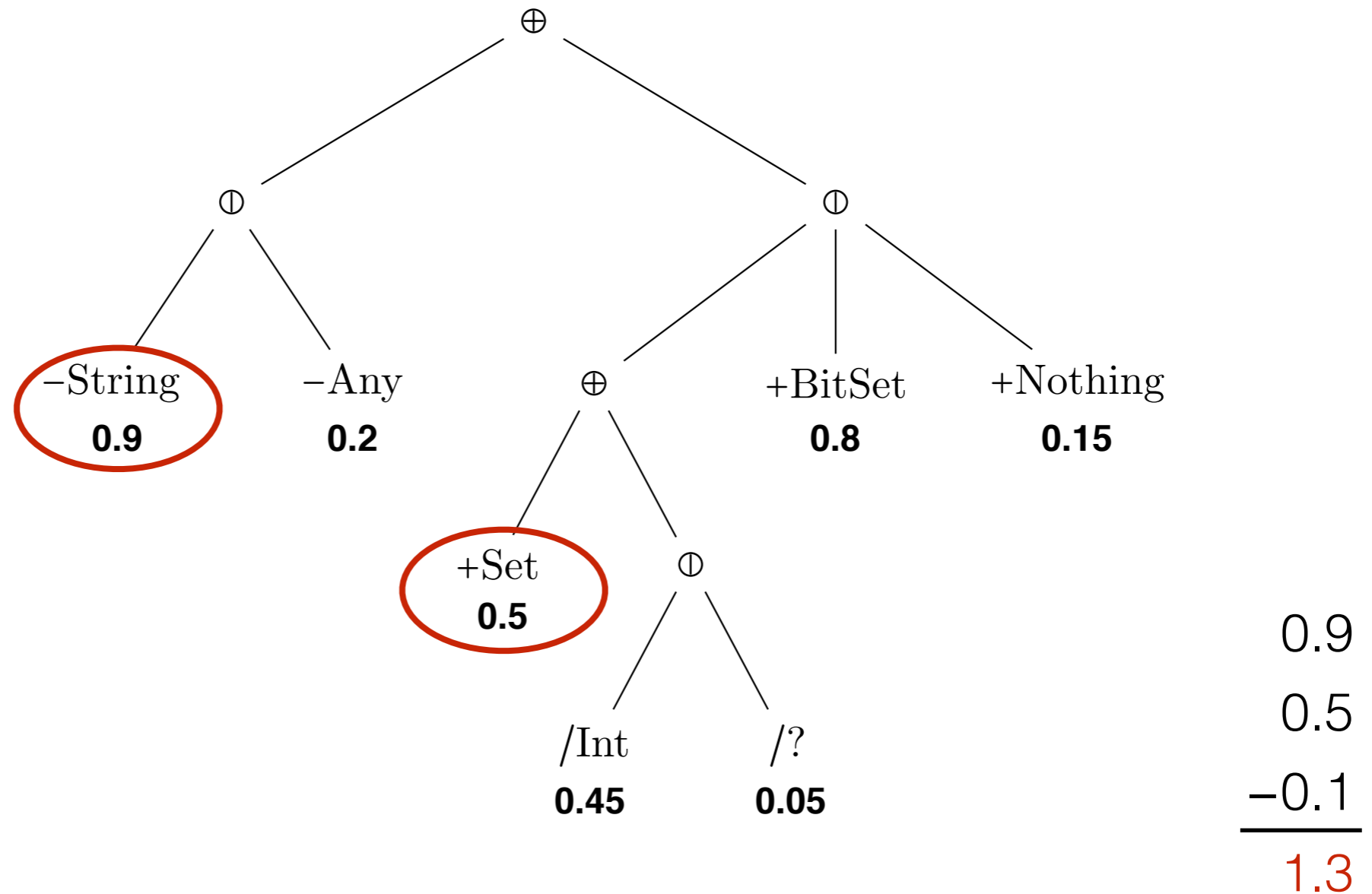
# Fetch Fingerprints with Dominant Terms



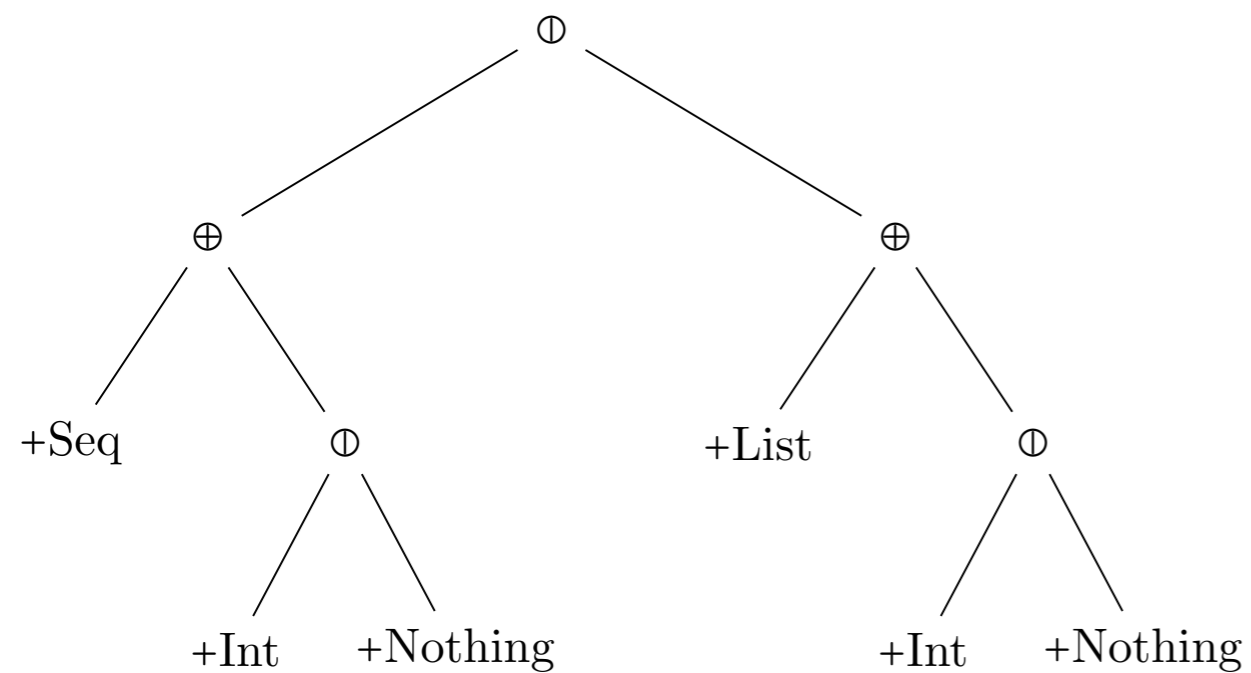


# Score Retrieved Fingerprints

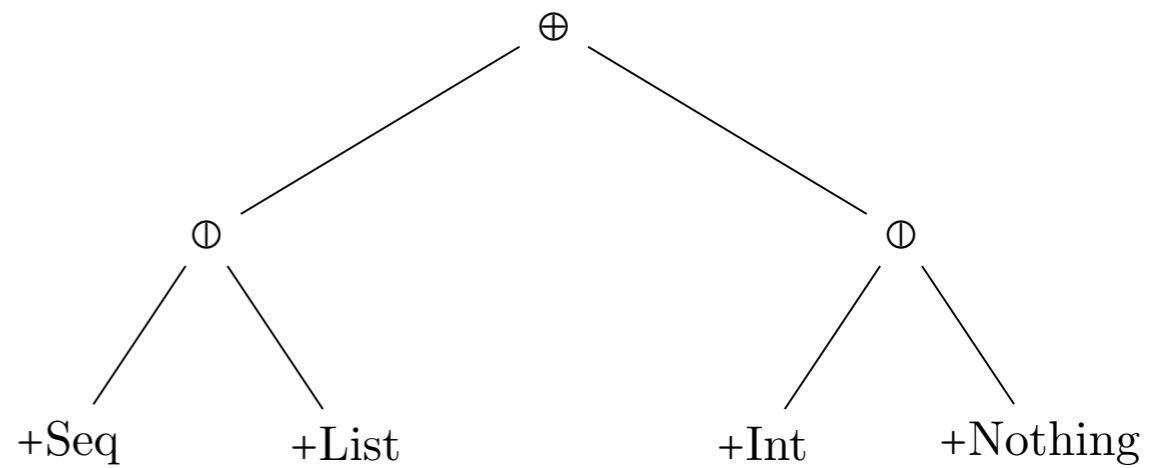
-String, -Any, +Set, /String



## + Query Expression **Rewriting**

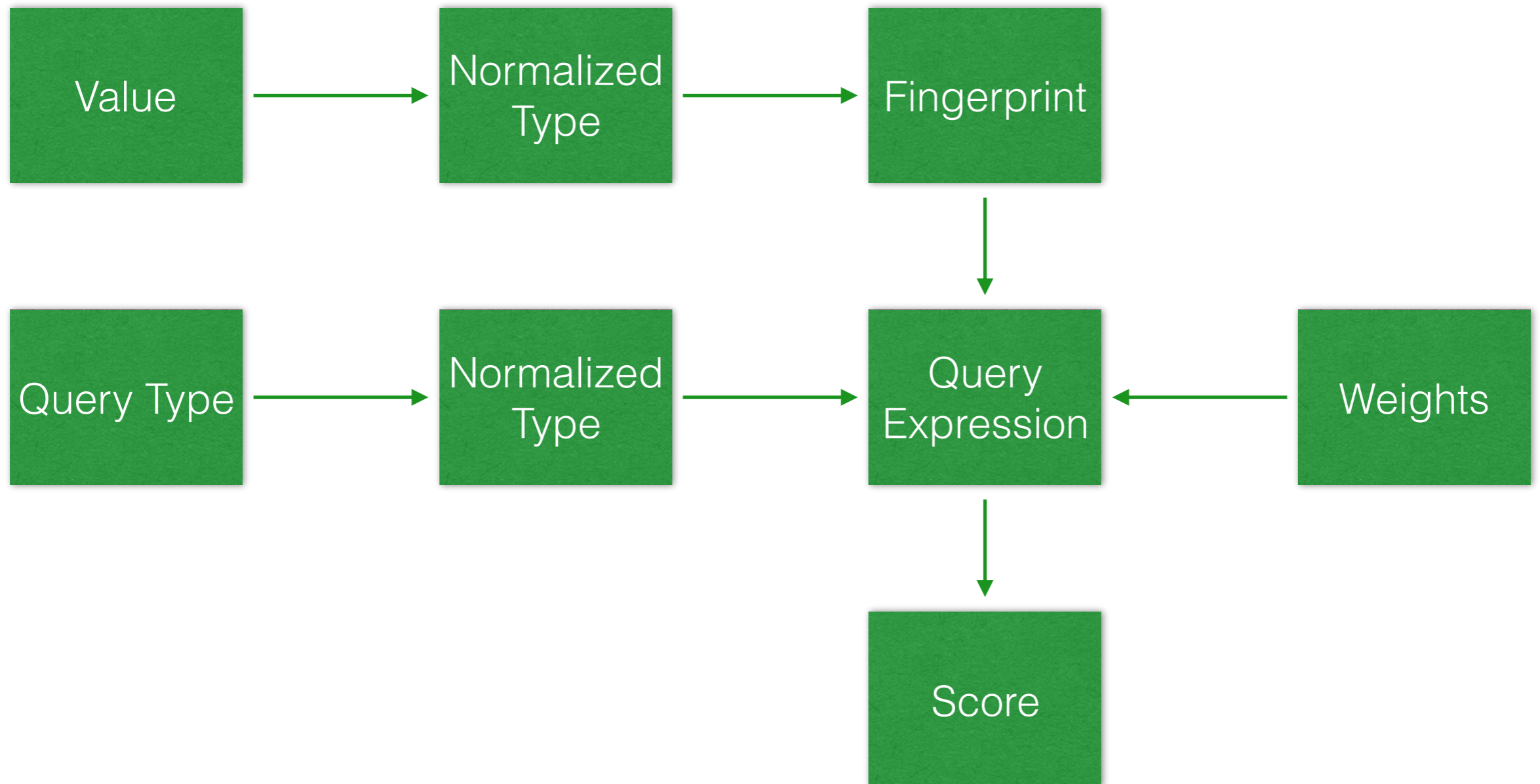


Size:  $n*m$

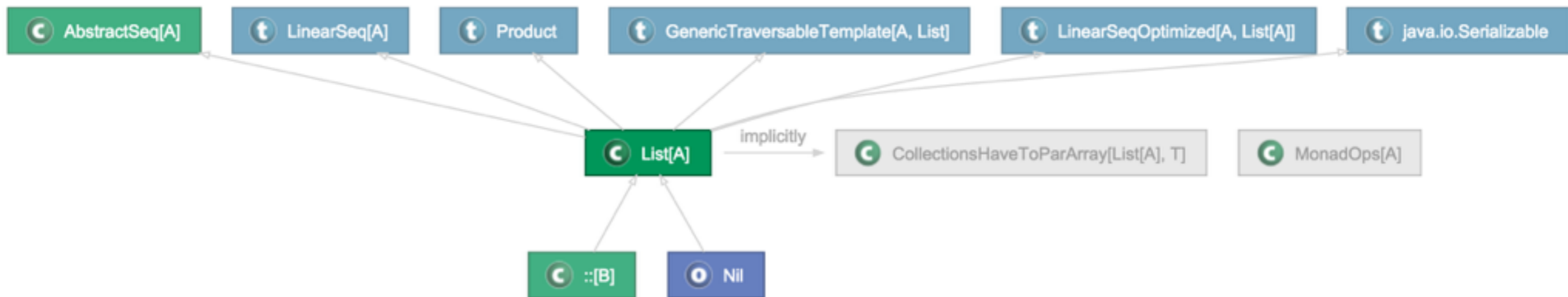


Size:  $n+m$

## Recap



+ Incorporates **Subtyping** and **Implicit Conversions**



with **Type Views**

-List[A] ▷ -LinearSeq[A]

-List[A] ▷ -MonadOps[A]

+List[A] ▷ +Nil

## + Isomorphisms

$$(A, B) \Rightarrow (C, D)$$

$$(B, A) \Rightarrow (D, C)$$

$$A \Rightarrow B \Rightarrow (C, D)$$

$$A \Rightarrow B \Rightarrow (C \Rightarrow 1) \Rightarrow D$$

$$(C \Rightarrow D \Rightarrow 1) \Rightarrow (A \Rightarrow B \Rightarrow 1)$$

+ Also works with Higher-kinded Parameters

Scaps

scala-library:2.11.7  scalajs-dom\_sjs0.6\_2.11:0.8.0  scalajs-library\_2.11:0.6.2  scalaz-core\_2.11:7.1.1

```
scala.concurrent.Future.sequence[A, M <: scala.collection.TraversableOnce[X]](M[Future[A]])(implicit
CanBuildFrom[M[Future[A]], A, M[A]], implicit ExecutionContext): Future[M[A]]
```

0.50700617

Simple version of `Future.traverse`. Transforms a `TraversableOnce[Future[A]]` into a `Future[TraversableOnce[A]]`. Useful for reducing many `Future`s into a single `Future`.

scala-library scala.concurrent.Future.sequence

[Doc](#) · [This is what i've been looking for](#)

-List[A] ▷ -<Any1>[A]

+List[A] ▷ +<Nothing1>[A]



+ Integrates well with Text Retrieval

`score(di, sort: Array[Int] => Unit)`

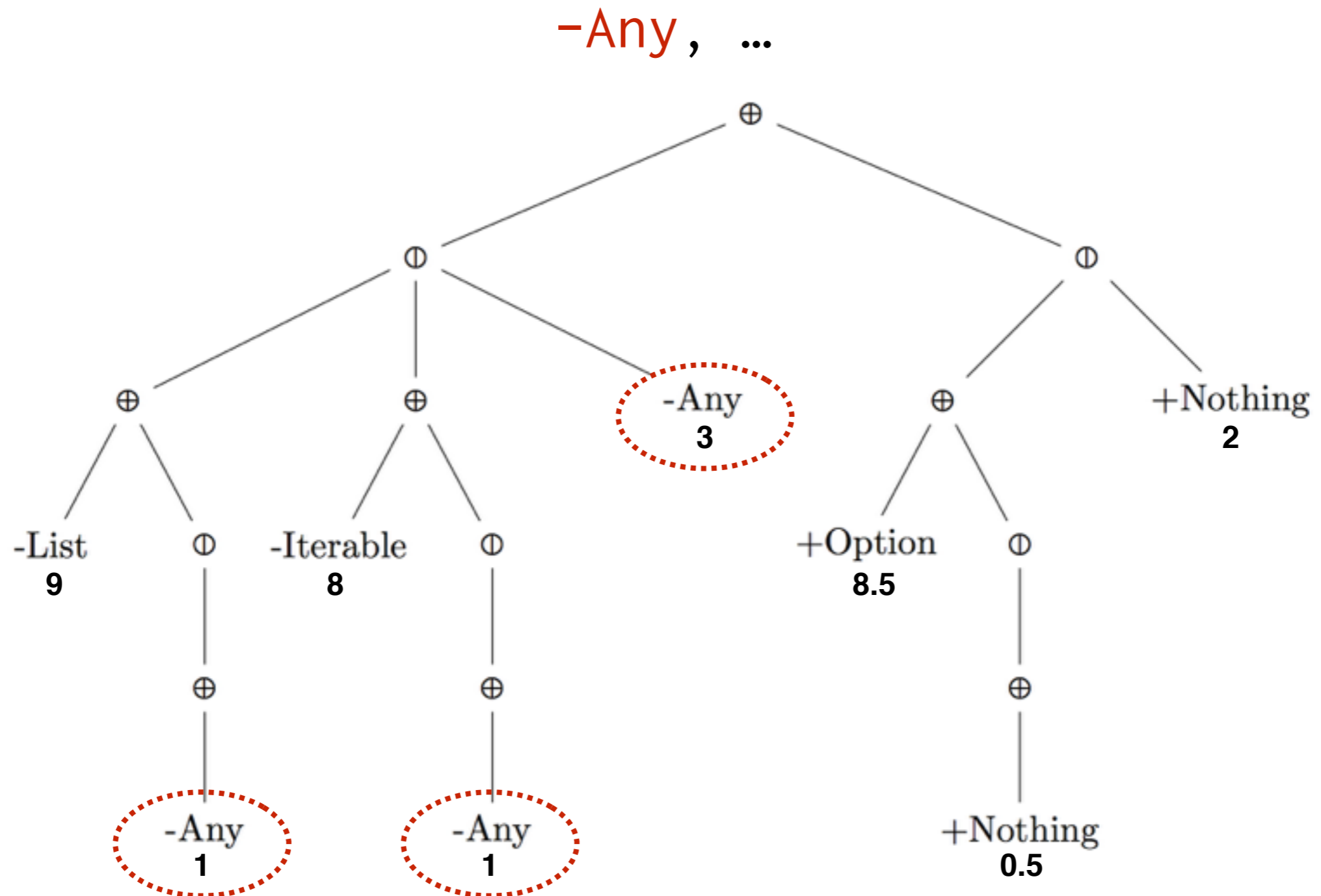
`=`

`score(di, sort)`

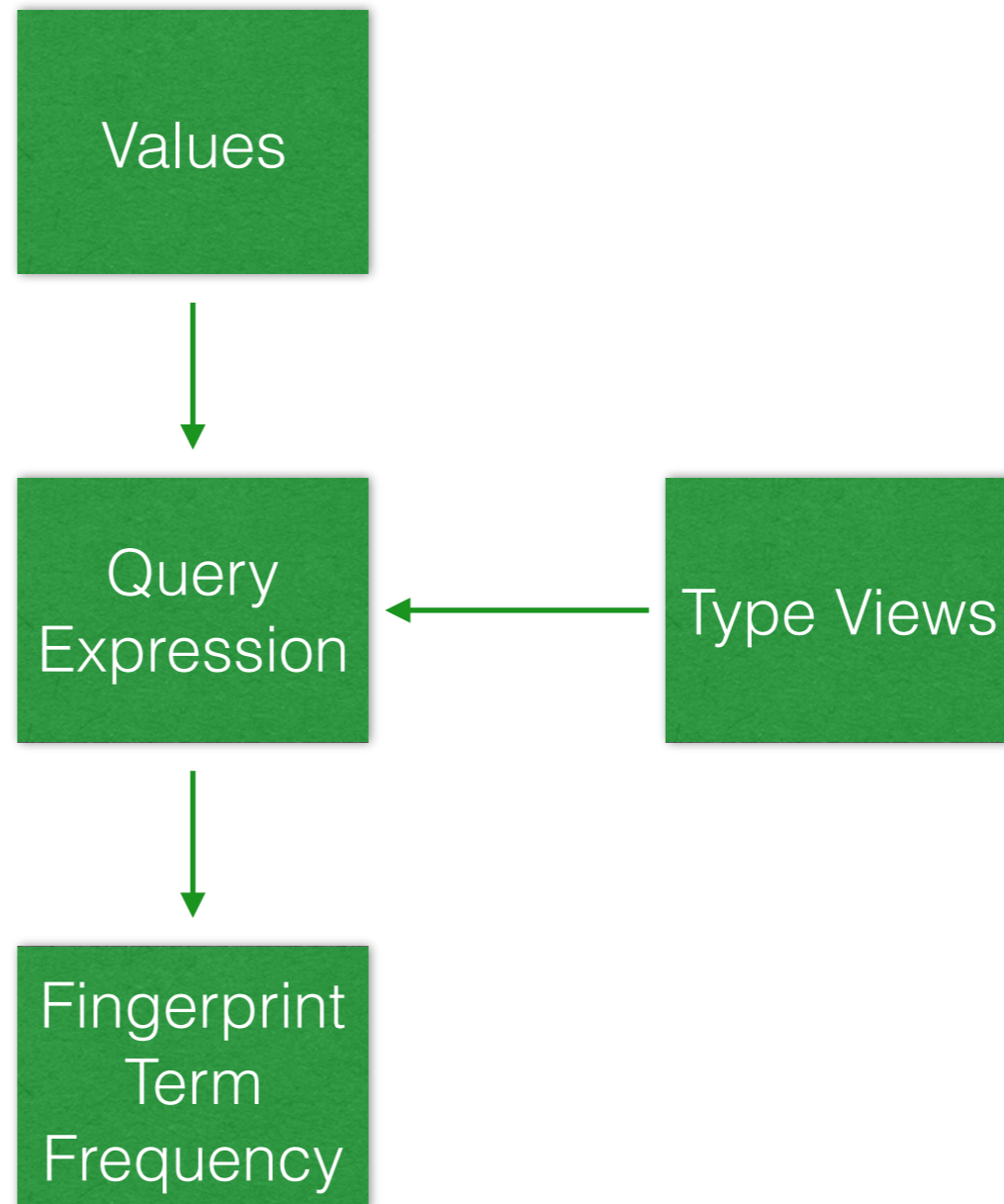
`+`

`score(di, Array[Int] => Unit)`

- Query Expression **Evaluation** is not Trivial



- Requires **Frequency Stats** for Scoring Terms



- Requires to **balance** Search Dimensions

String => Int => Set[Int]

**Structure**



String => Set[Int]

**Hierarchy**



String => BitSet

- Struggles with the **Type Class** Pattern

Query:

`List[Int] => Int`

Should Match:

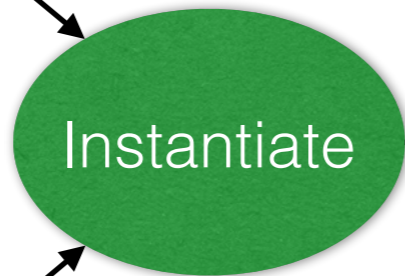
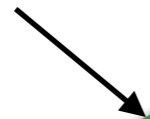
`List[A] => Numeric[A] => A`

What's next?

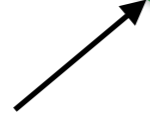


# Support Type Classes

List[A] => Numeric[A] => A



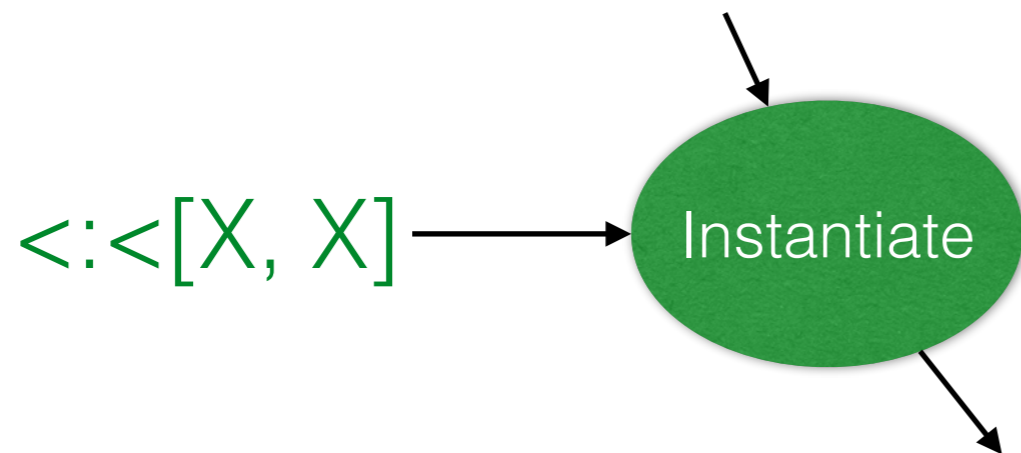
List[Int] => Numeric[Int] => Int



Numeric[Int]

## Support Type Classes and Other Implicits

$\text{Option}[A] \Rightarrow \text{<::<}[A, \text{Option}[B]] \Rightarrow \text{Option}[B]$



$\text{Option}[\text{Option}[B]] \Rightarrow \text{<::<}[\text{Option}[B], \text{Option}[B]] \Rightarrow \text{Option}[B]$

# Improve Query Performance

150'000 Documents

(scala, scalaz & scala-refactoring)

~500ms per Query



All Scala Libraries\*

<100ms per Query

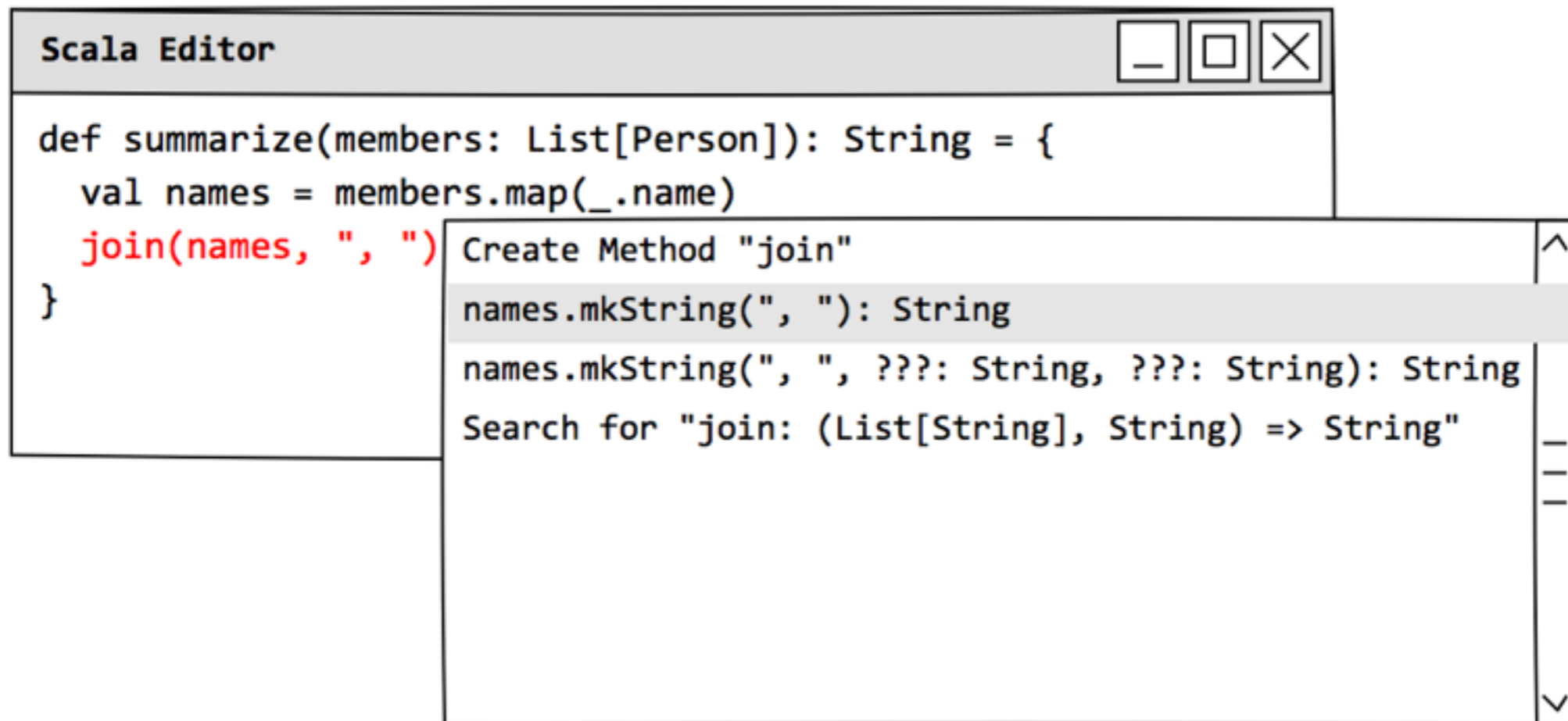
Integrate with IDEs



**ENSIME**



## Search with Context



The image shows a screenshot of a Scala Editor window. The editor contains the following code:

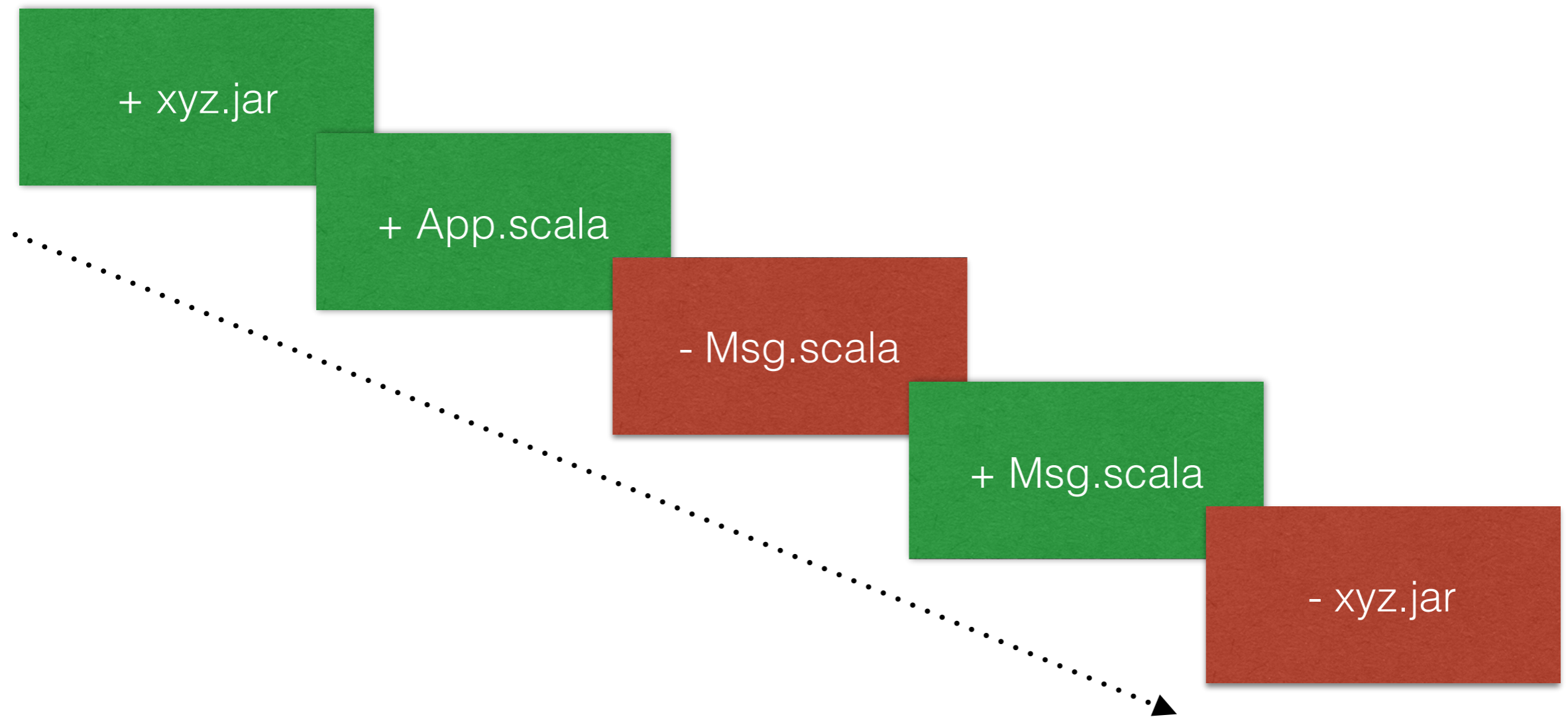
```
def summarize(members: List[Person]): String = {  
  val names = members.map(_.name)  
  join(names, ", ")  
}
```

A code completion popup is visible over the `join` method call. The popup lists the following options:

- Create Method "join"
- `names.mkString(", "): String` (highlighted)
- `names.mkString(", ", ???: String, ???: String): String`
- Search for "join: (List[String], String) => String"

The popup also includes a search bar and navigation arrows (up and down) on the right side.

# Incremental Indexing



Etc.

- Property Filters (`implicit`, `val`)
- Type Aliases
- ...



More Languages?



Thank *you*!

[scala-search.org](http://scala-search.org)

[github.com/scala-search/scaps](https://github.com/scala-search/scaps)

[twitter.com/Luegg1](https://twitter.com/Luegg1)