Presentation :

Scala Music Generation Project

Valérian Pittet

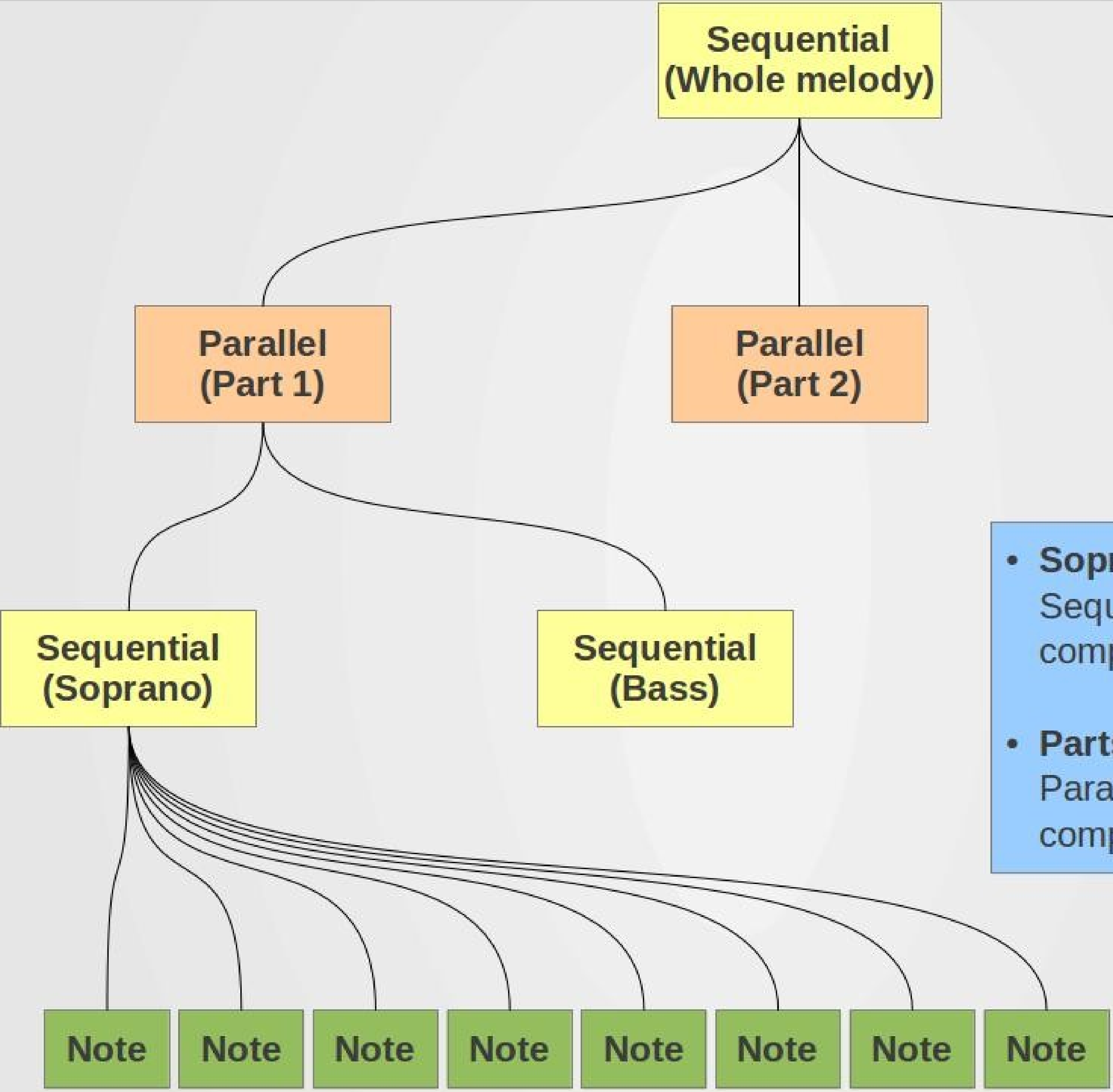
A Domain Specific Language for Music

- Music generation implies complex concepts
- Need support to build abstractions
- Base code to interpret and play described music
- DSL specification

Table of Contents

Tree structure

- Traditional music is multi-dimensional
- Code is linear
- Parallel and Sequential composition
- Flexible representation to match any kind of melody
- Represent melodies as syntax trees



- **Soprano** and **Bass** are Sequential segments composed in parallel
- **Parts 1, 2** and **3** are Parallel segments composed sequentially

Tree structure

```
trait MusicalSegment {
  def melody: List[MusicalSegment]
}

abstract class SequentialSegment(melody: List[MusicalSegment])
  implements MusicalSegment

abstract class ParallelSegment(melody: List[MusicalSegment])
  implements MusicalSegment

case class Note(tone: Tone, duration: BPM)
  implements MusicalSegment {
  def melody = this :: Nil
}
```

Tone representation

- Use traditional notes names ?

  A, A#, B, C, C#, D, …

- Irregularities

- Context dependent

- Use scale steps ?

  I, II, III, IV, V, VI, VII

- Regular

- Context independent

- Captures differences

  between C# and D♭

Tone representation

```
trait Tone {
  val octave: Int
  val alteration: Option[Boolean]
}
// musical rest
case object O implements Tone {
  val octave = 0
  val alteration = None
}
case class I(octave: Int, alteration: Option[Boolean] extends Tone
case class II(octave: Int, alteration: Option[Boolean] extends Tone
case class III(octave: Int, alteration: Option[Boolean] extends Tone
case class IV(octave: Int, alteration: Option[Boolean] extends Tone
case class V(octave: Int, alteration: Option[Boolean] extends Tone
case class VI(octave: Int, alteration: Option[Boolean] extends Tone
case class VII(octave: Int, alteration: Option[Boolean] extends Tone
```

Basic composition

- Avoid call for constructors
- Parallel and sequential composition
- Base operators (+ and |)
- Control tree shape (++ and ||)
- Parenthesis capture

Advanced composition

- Sequential repetition

```
// in trait MusicalSegment

def *(repetition: Int): SequentialSegment
```

- Repetition with transformations

```
def fillSeq(
  trans: (MusicalSegment => MusicalSegment)*): SequentialSegment

def fillPar(
  trans: (MusicalSegment => MusicalSegment)*): ParallelSegment
```

Style enhancement

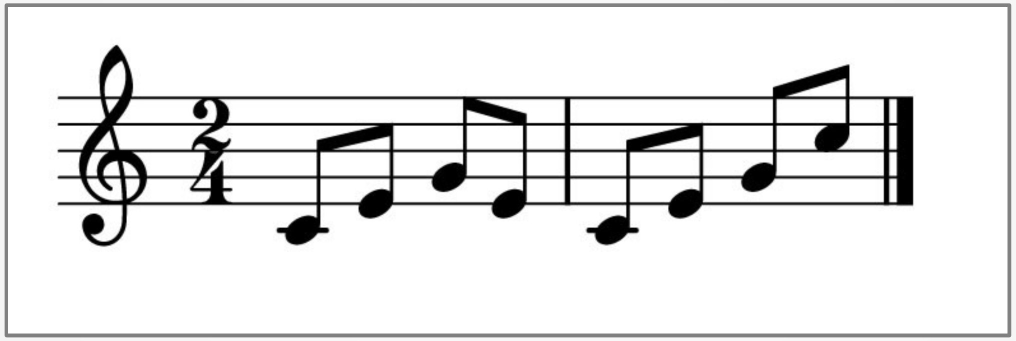
- DSL should be concise
- Avoid explicit instantiation
- Implicit conversions : Tone to Note
- Implicit conversions : Tone to Note builder

Note mapping

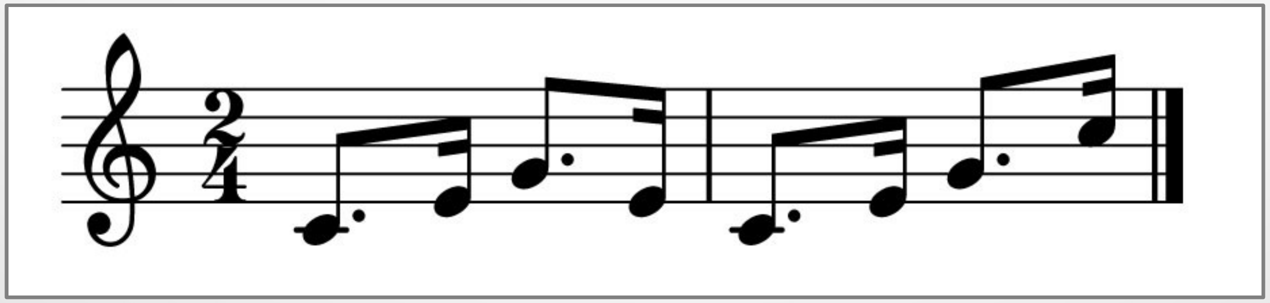
- Manipulate notes independently
- Generate complex melody from a basic one
- Specify multiple transformations
- Cyclic application

Note mapping

```
implicit val noteDuration = E // eight note
val melody = (I + III + V) *2 withScale Major(C)
```



```
melody mapNotes (_/(3/2), _ /2) // duration * 1.5, duration / 2
```



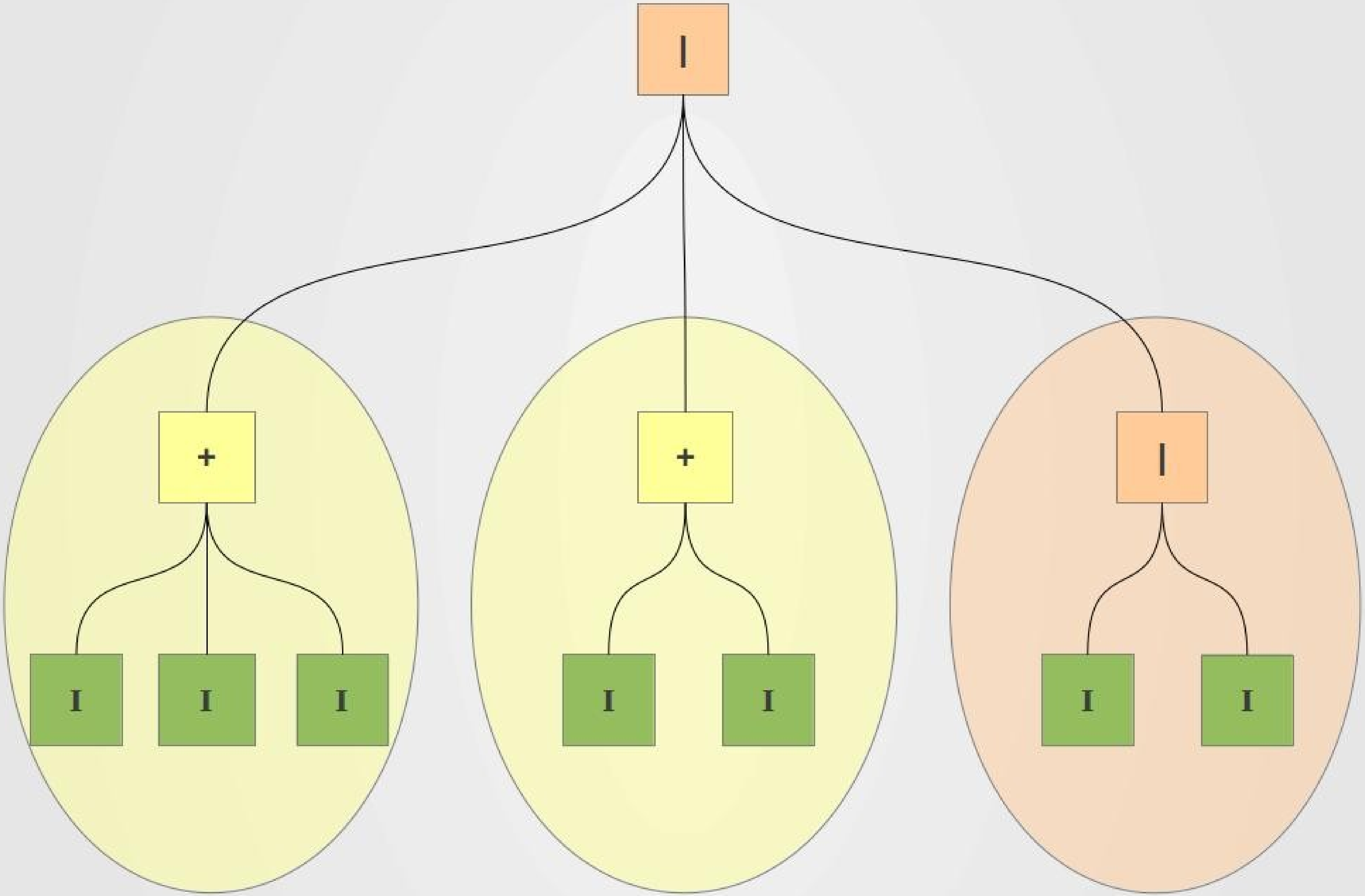
General mapping

- Target different composition modes
- Use boolean predicates
- Typed transformations
- Period and ranges of application
- Type inference in anonymous function

General mapping

- On a simple melody

```
val melody = (I + I) || (I + I) | (I | I)
```

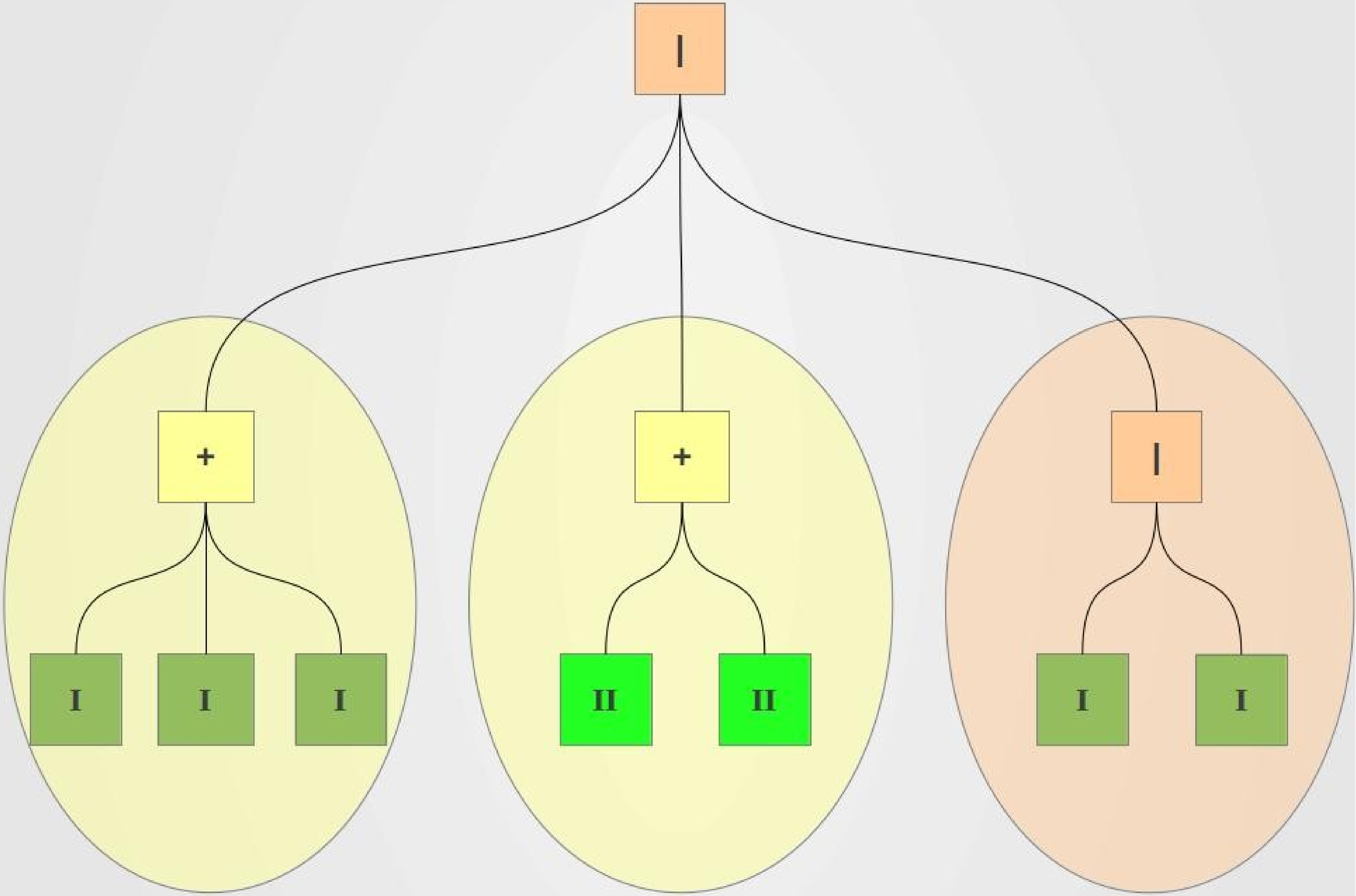

|
+
+
|
I
I
I
I
I
I
I

General mapping

- On a simple melody

```
val melody = (I + I + I) || (I + I) | (I | I)

melody mapIf (isSeq given (_.length == 2) thenDo (_ + 1))
```


|
+
+
|
I
I
I
II
II
I
I

Case study demonstration

- Recuerdos de la Alhambra *(Francisco Tarrega)*
- Tremolo, repetitions
- Stable rhythm
- Let's see (listen to) the code

Thank you for your attention !