From imperative code to recursion schemes

Alessandro Candolini

November 9, 2023

Agenda

1. Isomorphisms

Isomorphisms

```kotlin
sealed class Node(
   open val name: String,
   open val parent: Directory?
)

data class File(
  override val name: String,
  override val parent: Directory?,
  val size: Long): Node(name, parent)

data class Directory(
  override val name: String,
  override val parent: Directory?,
  val children: MutableList<Node>
) : Node(name, parent)
```

file system is represented by an *algebraic data type inductively defined*, which captures[1]

- the recursive nature of the filesystem
- separating attributes of files and attributes of directories (i. e., size)

[1]Wait next slides for cardinality questions and isomorphic and not isomorphic representations

- Blend of domain modelling and implementation details
- model design polluted by / coflated with elements to support the underlying implementation (i. e., leaking implementation choices to the consumer space)

```
data class Node(
  val name: String,
  val size: Long?,
  val children: MutableList<Node>?,
  val isFile : Boolean,
  val parent: Directory?
)
```

```
sealed interface Node

data class File(
  val name: String,
  val size: Long
) : Node

data class Directory(
  val name: String,
  val children: List<Node>
) : Node
```

```haskell
data FileSystem
  = File FileName Size
  | Directory DirectoryName [FileSystem]
  deriving (Eq, Show, Foldable, Functor)
```

```
enum FileSystem:
  case File(
    name: FileName,
    size: Size)
  case Directory(
    name: DirectoryName,
    children: List[FileSystem])
```

```
data Rose a = Rose a [Rose a]
data Tree1 a = Leaf1 a | Node1 [Tree1 a]
data Tree2 a = Leaf2 a | Node2 (NonEmpty (Tree2 a))
data Tree3 a = Leaf3 a | Branch (Tree3 a) (Tree3 a)
data Tree4 a = Empty | Node4 (a, Tree4 a , Tree4 a)
```

```haskell
data FileSystem
  = File Size
  | Directory (Map Name FileSystem)
  deriving (Eq, Show, Foldable, Functor)
```

Questions?