USER GUIDE

SCOTT MORRISON AND DAVID SPIVAK

1. Scala

1.1. **Class hierarchies.**

This section contains an overview of the important types in the Scala library. You should read this section in conjunction with the Scaladocs.

The three most important types are

- `net.metaphor.api.Ontology`
  All database schemas have type `Ontology`.
- `net.metaphor.api.Translation`
  A `Translation` is a functor between two `Ontology`s
- `net.metaphor.api.Ontology#Dataset`
  A `Dataset` is a functor from an `Ontology` to `Set`. (Recall in Scala the # denotes an inner class — thus every `Dataset` is attached to a particular `Ontology` instance.)

`Ontology` is a subtype of a long sequence of more general classes of categories, illustrated in Figure 1.1.

1.2. **Categories.**

At the top of the hierarchy of categories is `Category`. It contains two abstract type members, `O` and `M`, which represent the types of objects and morphisms for the category. (If a type has an abstract type member, it cannot be instantiated — some subtype will override these type members, specifying concrete types.) Essentially the only other functionality in `Category` are the methods `identity`, `source`, `target` and `compose`, which provide the basic operations on objects and morphisms.

Below `Category` we have `SmallCategory`. For our purposes, a `SmallCategory` is a category for which we can talk about the category of functors to `Set`. In particular, `SmallCategory` contains an inner type `FunctorToSet`. (This will eventually be specialized to the inner type `Dataset` in `Ontology`.)

Below `SmallCategory` we have `LocallyFinitelyGeneratedCategory`. Mathematically, a locally finitely generated category is a category with a finite set of 'generators' between each pair of objects, such that every morphism can be obtained by composing some sequence of generators. Moreover, we insist that the set of generators with a fixed source but arbitrary target is finite (even when there are infinitely many objects), and similarly for a fixed target.

In Scala, we implement this by introducing a new abstract type member `G` to represent generators, and override the type `M`, defining it once and for all to be `PathEquivalenceClass`. Further, `LocallyFinitelyGeneratedCategory` provides definitions of `identity`, `source`, `target` and `compose`, the basic operations from `Category`. An implementation of a `LocallyFinitelyGeneratedCategory` must provide a number of new methods, instead. The most important of these is `def generators(s: O, t: O): List[G]`, specifying the generators between two objects. Further, the category must provide a method `def pathEquality(p1: Path, p2: Path): Boolean`, which determines whether two compositions of generators are equal. (Override the method `def pathHashCode(p: Path): Int` is highly recommended as well.)

Below `LocallyFinitelyGeneratedCategory` we have `FinitelyGeneratedCategory`. Now we insist that there are finitely many objects.

Below `FinitelyGeneratedCategory` we have `FinitelyPresentedCategory`, which provides a method `def relations(s: O, t: O): List[(Path, Path)]`. In principle at least, `FinitelyPresentedCategory` could provide an implementation of the method `pathEquality`, but since this is potentially a hard (!) problem we defer this to traits which use particular strategies to decide path equality.

Finally, an `Ontology` inherits from `FinitelyPresentedCategory`.

1.3. **Functors.** The basic `Functor` type is

```
trait Functor {
    val source: Category
    val target: Category

    def onObject(source.O): target.O
    def onMorphisms(source.M): target.M
}
```

It also comes with some convenience `apply` methods, so if `F` is a functor we can simply write `F(o)` or `F(m)` to apply it to an object or morphism. Implementations, however, should override `onObjects` and `onMorphisms`.

The type hierarchy is quite complicated, and two-dimensional rather than linear! Most of the relevant types are traits contained in the object `Functor`. In particular, we have

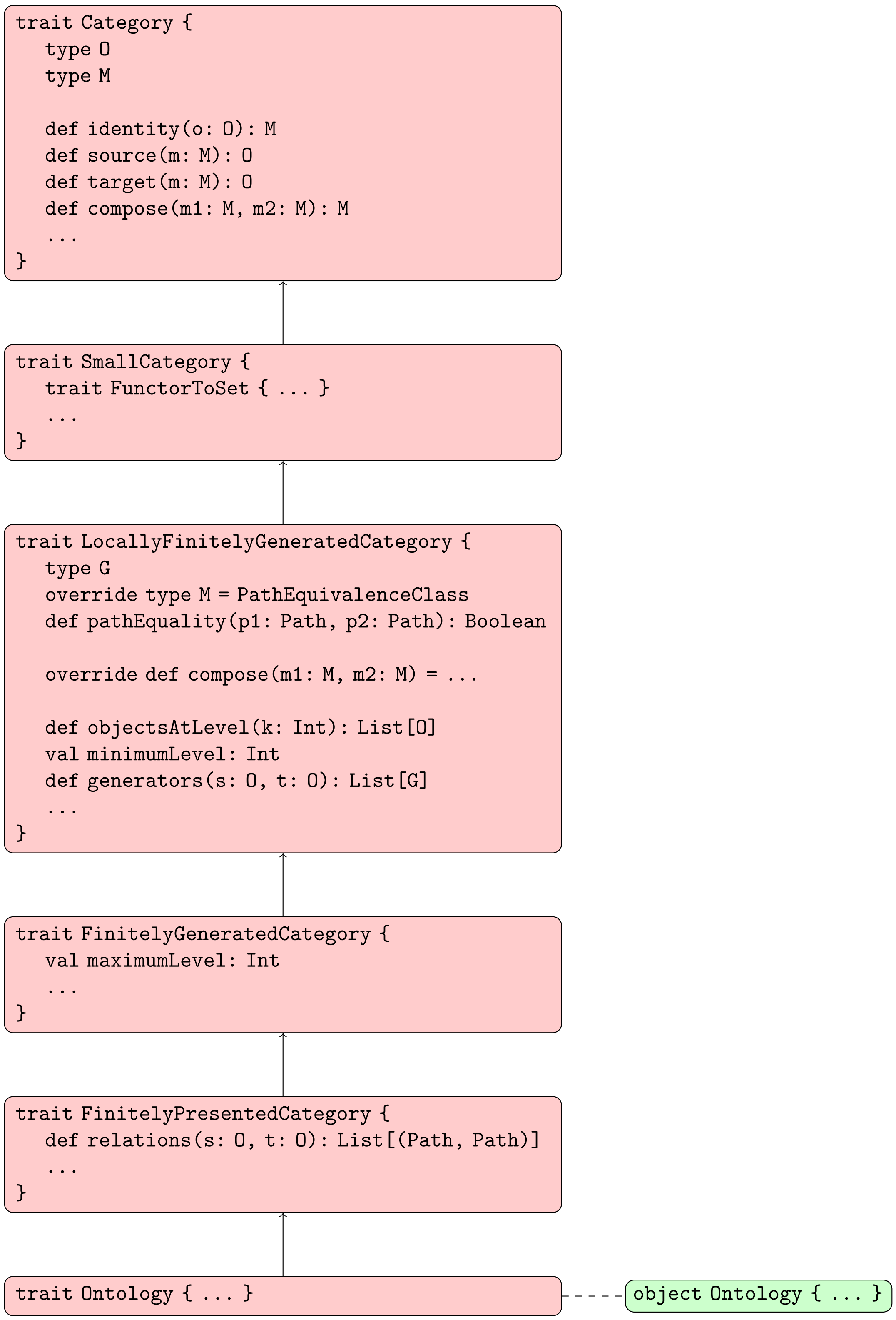
- `Functor.withSmallSource`
- `Functor.withLocallyFinitelyGeneratedSource`
- `Functor.withFinitelyGeneratedSource`
- `Functor.withFinitelyPresentedSource`

and

- `Functor.withSmallTarget`
- `Functor.withLocallyFinitelyGeneratedTarget`
- `Functor.withFinitelyGeneratedTarget`
- `Functor.withFinitelyPresentedTarget`

along with the combined types `Functor.withXXXSource.withYYYTarget`, for each of the possible values of `XXX` and `YYY`.

1.4. **Functors to `Set`.**



```
trait Category {
   type O
   type M

   def identity(o: O): M
   def source(m: M): O
   def target(m: M): O
   def compose(m1: M, m2: M): M
   ...
}
```

```
trait SmallCategory {
   trait FunctorToSet { ... }
   ...
}
```

```
trait LocallyFinitelyGeneratedCategory {
   type G
   override type M = PathEquivalenceClass
   def pathEquality(p1: Path, p2: Path): Boolean

   override def compose(m1: M, m2: M) = ...

   def objectsAtLevel(k: Int): List[O]
   val minimumLevel: Int
   def generators(s: O, t: O): List[G]
   ...
}
```

```
trait FinitelyGeneratedCategory {
   val maximumLevel: Int
   ...
}
```

```
trait FinitelyPresentedCategory {
   def relations(s: O, t: O): List[(Path, Path)]
   ...
}
```

```
trait Ontology { ... }
```

```
object Ontology { ... }
```

Figure 1. The `Ontology` type hierarchy.

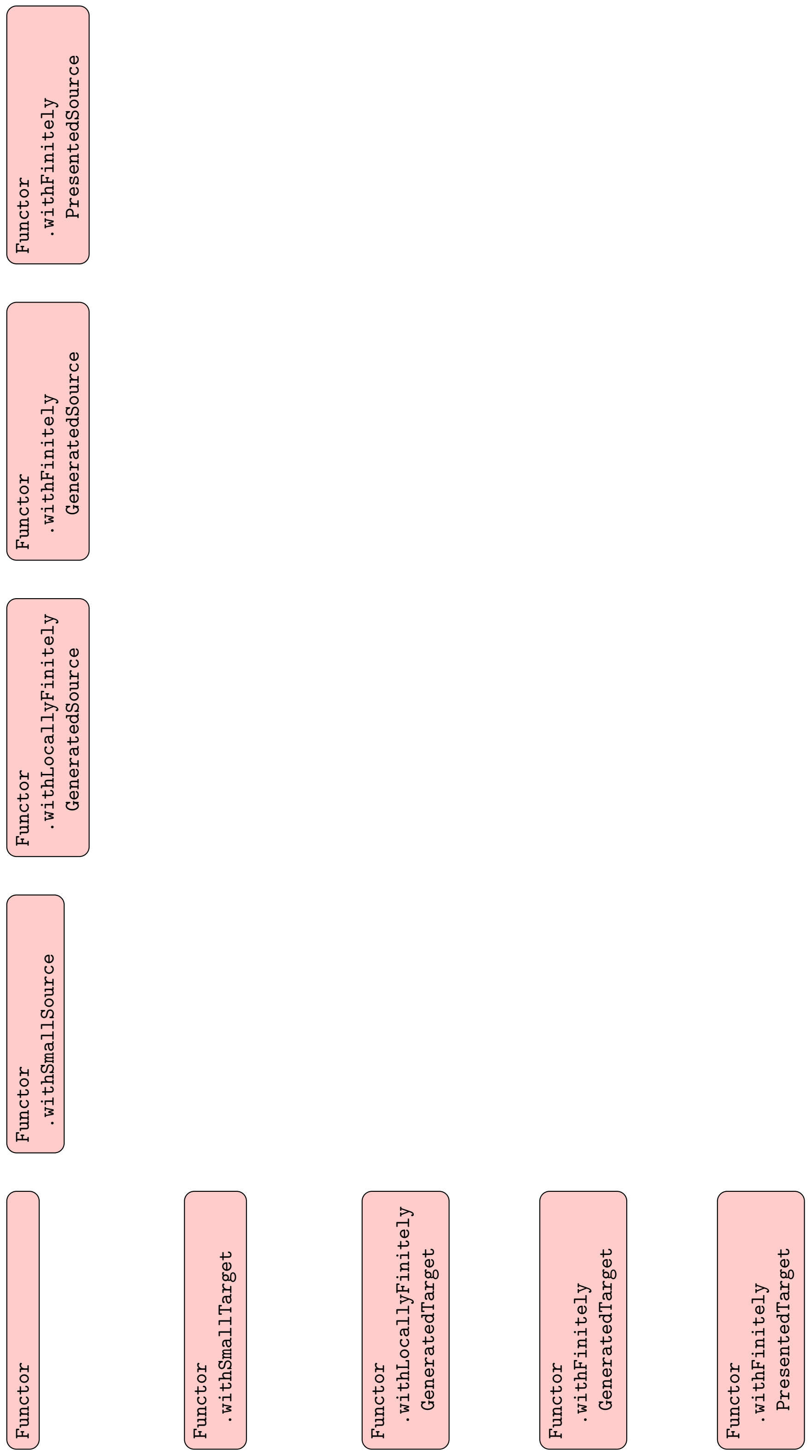

Functor
Functor .withSmallTarget
Functor .withLocallyFinitely GeneratedTarget
Functor .withFinitely GeneratedTarget
Functor .withFinitely PresentedTarget
Functor .withSmallSource
Functor .withLocallyFinitely GeneratedSource
Functor .withFinitely GeneratedSource
Functor .withFinitely PresentedSource

FIGURE 2. The `Functor` type hierarchy.