1 Objects

We now start with the characterization of objects in our system. The *Object* predicate says that an object will be a record which supports methods and inheritance; of course all the object operators are expected to work in conjunction with the rest of the system. Since objects will reference themselves, to avoid recursive type definitions we give a predicate that allows us to freely add or remove some constructor from an object:

```
class Recursive o where
  type Rec o :: *
  to :: o -> Rec o
  from :: Rec o -> o
```

An object is required to support the *Recursive* predicate:

```
class (Record o ref st s, Recursive o, ro~Rec o) => Object o ref st s where
  type Base o :: *
  get_base :: ref s o -> ref s (Base o)

  type Method ro ref st :: * -> * -> *
  (<=|) :: ref s r -> Label r (Method ro ref st a b) -> a -> st s b
```

With respect to inheritance, it looks clear how we can instance coercion to take advantage (and make access more uniform) of the *base* operations:

```
instance Object o ref st s => Coercible (ref s o) (ref s (Base o)) where
  coerce = get_base
```

We also add a further predicate that characterizes inheritance:

```
class Inherits a b

instance (Object o ref st s) => Inherits o (Base o)
```

Notice that methods are defined with a different operator than the selection operator for records. This can be addressed as follows: we define a new predicate for selecting something from a reference through a label:

```
class Selectable ref st t s a where
  type Selection s t a :: *
  (<=) :: ref s t -> Label t a -> Selection s t a
```

We instance this predicate twice, one for field selection and one for method selection. Before doing so, though, we must be careful to disambiguate the last parameter in the record definition:

```
class (RefSt ref st s) => Record r ref st s where
  type Label r :: * -> *
  type Field a :: *
  (<==) :: ref s r -> Label r (Field a) -> ref s a
```

In the case of simple mutable records we can easily add a trivial constructor for *Field* such as:

```
data Id a = Id a
```

Now we can instance the selection predicate:

```
instance Record ref st r s => Selectable ref st r s a where
  type Selection s r a = ref s a
  (<=) = (<==)

instance (Object ref st o s, ro~Rec o) => Selectable ref st o s (Label ro a b) where
  type Selection s o a = a -> st s b
  (<=) = (<=|)
```

The final operation we wish to support is that of selecting a method or a field directly from any of the inherited classes of an object without explicit coercions:

```
instance (Object ref st o s, bo~Base o, Selectable ref st bo s a) => Selectable ref st o s
  type Selection s o a = Selection s bo a
  (<=) = get_base . (<=)
```