1 Inheritance

We wish to add support for inheritance to our system. Since inheritance is a relation between types, we give a type predicate that expresses the fact that one type inherits another:

```
Inherits t t'
  get_base : t -> t'
  set_base : t -> t' -> t'
```

we wish to be able to both obtain the value of the inherited value from inside the inheriting value, and also to modify the inherited value inside the inheriting value. We give a first instance that allows us to express the fact that the inheriting relation is transitive:

```
Inherits t t', Inherits t' t'' ⇒ Inherits t t''
  get_base = get_base . get_base
  set_base t t'' =
    let t' = get_base t
        t'_mod = set_base t' t''
    in set_base t t'_mod
```

We now instance our predicate for the simple mutable implementation. We reserve the first item inside an object for the value of the base class, which will neither be marked as *Field* nor *Method*; instead we define a new "placeholder" type that will mark the fact that at that slot there is the instance of the base object:

```
Base a = Base a
```

We can now add to the inheritance relation the notion that whenever the first slot of an heterogeneous list is of type *Basea* then the heterogeneous list itself inherits *a*:

```
Inherits (Base a : t) b
    get_base ((Base h) : tl) = h
    set_base (_ : tl) h' = h' : tl
```

We now link the inheritance relation with the subtyping relation for references, so that whenever a reference points to something that inherits something else, then this reference can be downcast to a reference to the inherited value:

```
Inherits t t' ⇒ Reference h t <: Reference h t'
    downcast self_ref =
```

```
Reference(do self <- eval self_ref
              return get_base self)
         (λbase'.
            do self <- eval self_ref
               self' = set_base self base'
               self_ref := self'
               return ())
```

2 Selection and Subtyping

At this point we can safely and easily downcast from a reference to a value of type t to a reference to any other type t' that inherits from t.

A feature often encountered in programming languages that support inheritance is the possibility of selecting fields and methods that belong to the base class without any need for explicit casting. For this reason we give a predicate for selection:

```
Heap h ref ⇒ Selectable t (Label t' a)
    SelectionResult t a
    (<=) : ref h t -> Label t' a -> SelectionResult t a
```

which holds both the *SelectionResult* type function that tells us what results from the selection of a value of type a from an instance of type t, and also the selection operator (<=) itself.

We instance this predicate for all the kinds of selection: once for fields, another for methods and the last one for subtyping; the first two instances are filled exactly as explained in the "SELECTION" section:

```
Selectable t (Label t (Field a))
Recursive r, Rec t = t_rec ⇒ Selectable t (Label t (Method a b t_rec))
```

now we can instance our predicate so that it deals with selection on a subtype of the type of the value on which the selection is being performed:

```
Selectable t' a ∧ t <: t' ⇒ Selectable t a
    SelectionResult t a = SelectionResult t' a
    self <= l = downcast self <= l
```

where both operations of the predicate simply delegate the task of actual selection to the cast value.