1 Mutable Records

Up until now we have defined a very general form of a heap with mutable references and we have given the first concrete implementation that allows us to write and run programs that use these constructs; thanks to monads and some syntactic sugar we even obtain code that looks and feels very stateful and imperative. We now move to studying a way to define and manipulate records (which are no more than heterogeneous lists plus some definition of labels instead of Church Numeral based access) within our system.

We start by giving a new predicate to characterize record types$r$:

```
Heap h ref ⇒ Record ref h r
```

A record has a label type which takes as input an additional type parameter (the type of the selected field):

```
Label ref h r : ∗→ ∗
```

From a reference to a record and a label that selects a field of type α we can select a reference to a value of type α to a field inside the record:

```
⇐ : ref h r → Label ref h r α → ref h α
```

We can now say that our heterogeneous lists are records, and labels are a pair of functions similar to how references store a get and a set function:

```
Label r α =Label (r→ α) (r→ α → r)
```

We also need a function to convert a Church Numeral into a label:

```
CNum n ∧ HList h ⇒ labelAt : n → HAt h n
labelAt _ = Label (λh.hRead h) (_:n(λh.λv.hWrite v h (_:n)))
```

We can now instance the *Record* predicate:

```
Heap h ref ∧ HList r ⇒ Record ref h r
    Label ref h r α = Label r α
    (Reference get set) ⇐ (Label read write) =
            Reference (do r ← get
                              return (read r))
                          (λ v.do r ← get
                                  set (write r v))
```

The selection operator simply does some packing and unpacking of the label into a reference.

At this point we could consider a simple example that shows how we can use records mutably:

```
Person = String:String:Int:Nil

name : Label Person String

name = labelAt Z

surname : Label Person String

surname = labelAt (S Z)

age : Label Person Int

age = labelAt (S S Z)


ex₃ = do ‘‘John’’:’’Smith’’:27:Nil >>+ (λp.

                    do (p⇐age) := 25

                       let n = p⇐surname

                       in n*=(++’’ Jr’’))

runST Nil ex₃
```

In the example above we declare a record type (*Person*) and we name its fields with labels. At this point we dynamically allocate an instance of our record, we manipulate it by reassigning the age to 25 and adding the "*Jr.*" String to the surname.