Chapter 5

Relational algebraic ornaments

The three datatypes Nat, $\mathsf{List}\ A$, and $\mathit{Vec}\ A$ are evidently related: a list is a natural number whose cons nodes are decorated with elements of A, and a vector is a list enriched with length information. Such relationship can be seen by "overlaying" one datatype declaration on the other: for example, the declaration of $\mathsf{List}\ A$ differs from that of Nat only in an extra field $(a\ :\ A)$ in the cons constructor, and the declaration of $\mathit{Vec}\ A$ differs from that of $\mathsf{List}\ A$ in that (i) the index set is changed from $\top$ to Nat, (ii) the cons constructor has two extra fields, and (iii) the index of the recursive position is specified to be m. Such differences between datatype declarations are encoded as *ornaments*. Whenever there is an ornament between two datatypes, there is a forgetful function from the more informative datatype to the other, erasing information according to the ornament's specification of datatype differences. For example, we have a forgetful function from lists to natural numbers that discards elements associated with cons nodes — i.e., it computes the length of a list — and another one from vectors to lists which removes all length information from a vector and returns the underlying list.

Ornaments constitute the second underlying universe:

$$\mathit{Orn}\ :\ \{I\ J\ :\ \mathsf{Set}\}\ (e\ :\ J \to I)\ (D\ :\ \mathsf{Desc}\ I)\ (E\ :\ \mathsf{Desc}\ J) \to \mathsf{Set}_1$$

An ornament $O\ :\ \mathit{Orn}\ e\ D\ E$ specifies the difference between the more informative description E and the basic description D, and is parametrised by an "index erasure" function e from the index set of E to that of D. The ornament gives rise to a forgetful function

$$\mathit{forget}\ O\ :\ \mu\ E\ \rightrightarrows\ (\mu\ D \circ e)$$

For example, there are families of ornaments

$$\mathit{NatD} - \mathit{ListD}\ :\ (A\ :\ \mathsf{Set}) \to \mathit{Orn}\ !\ \mathit{NatD}\ (\mathit{ListD}\ A)$$

and

$$\mathit{ListD} - \mathit{VecD}\ :\ (A\ :\ \mathsf{Set}) \to \mathit{Orn}\ !\ (\mathit{ListD}\ A)\ (\mathit{VecD}\ A)$$

(where $! = \mathit{const}\ \mathit{tt}$) that encode the differences between the list-like datatypes. The function

$$\mathit{forget}\ (\mathit{NatD} - \mathit{ListD}\ A)\ \{\mathit{tt}\}\ :\ \mathsf{List}\ A \to \mathsf{Nat}$$

computes the length of a list, and the function

$$\mathit{forget}\ (\mathit{ListD} - \mathit{VecD}\ A)\ :\ \forall\ \{n\} \to \mathit{Vec}\ A\ n \to \mathsf{List}\ A$$

computes the underlying list of a vector.

Ornamental descriptions. Ornaments arise between existing datatype descriptions. The typical scenario of using ornaments, however, is first modifying a base description into a more informative one and then specifying an ornament between the two descriptions. *Ornamental descriptions* are introduced to combine the two steps into one:

$$\mathit{OrnDesc}\ :\ \{I\ :\ \mathsf{Set}\}\ (J\ :\ \mathsf{Set})\ (e\ :\ J \to I)\ (D\ :\ \mathsf{Desc}\ I) \to \mathsf{Set}_1$$

An ornamental description

$$\mathit{OD}\ :\ \mathit{OrnDesc}\ J\ e\ D$$

is like a new description of type $\mathsf{Desc}\ J$, but is written relative to a base description D such that not only can we extract the new description

$$\lfloor\ \mathit{OD}\ \rfloor\ :\ \mathsf{Desc}\ J$$

but we can also extract an ornament from the base description D to the new description

$$\lceil\ \mathit{OD}\ \rceil\ :\ \mathit{Orn}\ e\ D\ \lfloor\ \mathit{OD}\ \rfloor$$

An ornamental description is a convenient way to specify a new datatype that has an ornamental relationship with an existing one; it might be thought of as simultaneously denoting the new description and the ornament — the floor and ceiling brackets $\lfloor \_ \rfloor$ and $\lceil \_ \rceil$ are added to resolve ambiguity.
Example. Let $\_ \leqslant A\_\ :\ A \to A \to \mathsf{Set}$ be an ordering on A and declare a datatype of ordered lists (parametrised by A and $\_ \leqslant A\_$) indexed by a lower bound under this ordering:

$$\begin{array}{l}
\textbf{indexfirst data}\ \mathit{OrdList}\ A\ _ \leqslant A_\ :\ A \to \mathsf{Set}\ \textbf{where} \\
\quad \mathit{OrdList}\ A\ _ \leqslant A_\ b \\
\qquad \mathit{accepts}\ \mathit{nil} \\
\qquad \mathit{or}\qquad \mathit{cons}\ (a\ :\ A)\ (\mathit{leq}\ :\ b \leqslant A\ a)\ (\mathit{as}\ :\ \mathit{OrdList}\ A\ _ \leqslant A_\ a)
\end{array}$$

This datatype can be thought of as being decoded from an ornamental description

$$\mathit{OrdListOD}\ A\ \_ \leqslant A\_\ :\ \mathit{OrnDesc}\ A\ !\ (\mathit{ListD}\ A)$$

which inserts the field *leq* and refines the index of the recursive position to a. That is, the underlying description for *OrdList* is

$$\lfloor\ \mathit{OrdListOD}\ A\ \_ \leqslant A\_\ \rfloor\ :\ \mathsf{Desc}\ A$$

(so *OrdList* A _ $\leqslant A$_ b desugars to $\mu \lfloor$ *OrdListOD* A _ $\leqslant A$_ $\rfloor$ b), and

$$\lceil \mathit{OrdListOD}\ A\ \_ \leqslant A\_\ \rceil : \mathit{Orn}\ !\ (\mathit{ListD}\ A)\ \lfloor \mathit{OrdListOD}\ A\ \_ \leqslant A\_\ \rfloor$$

is the ornament from lists to ordered lists.