all in all

guillaume, fred, conor, . . .

June 1, 2023

1   introduction

The humble cons-list was, for many years, the workhorse of functional pro-
gramming, being a means to capture free monoid structure in some (arbitrarily
biased) canonical form. (We write its values [] for them empty list and $x,\text{-}xs$ for
the list with head x and tail xs. Everyone else uses a *symmetric* symbol for ,-
— their teaching assistants weep as their students mistake it for a symmetric
operation.) With the advent of fancy (dependent or otherwise) type systems,
the world of list-like types has exploded, with many dimensions of variation,
e.g.,

dimensions two-dimensional matrices, and even higher-dimensional cuboids,
can be seen as nested lists, with types imposing varying degrees of

coherence meaning the extent to which we enforce non-raggedness, by what-
ever means, but notably

indexing with the over-deployed length-indexed vectors being but one example
(of a list with interesting elements with positions corresponding to those
in a list with dull elements, otherwise a 'natural number'), modelling sized
arrays and admitting safe

access which is not always left-to-right—right-to-left is also useful, but people
get up to all sorts of things to support random access or healthy amortized
complexity.

The archetype of these structures is (borrowing the language of propositions
to talk about data-bearing types) the 'predicate transformer'

$$\mathbf{All} \in (X \to \mathbf{Ty}) \to (\mathbf{List}\ X \to \mathbf{Ty})$$

where $\mathbf{All}\ P\ xs$ stores a 'proof' (possibly involving nontrivial data) of $P\ x$ for
each element x of xs. More specifically,

$$\mathbf{All}\ P\ [] \ni []\qquad \mathbf{All}\ P\ (x,\text{-}xs) \ni p,\text{-}ps \ \Leftarrow\ P\ x \ni p,\ \mathbf{All}\ P\ xs \ni ps$$

The type X gives some notion of *individual*; position in the list xs is a proxy for
the distinction between individuals; the way $P\ x$ varies with x tracks a notion

of *pertinence* to the individual. The length-indexed vectors is thus the special case where $X = 1$ — the list individuals are indistinct and the vector elements are impertinent. That is, **All** characterizes a kind of *table* where *xs* represents a row of *headers*, while the *cells* of the table have types dependent on the header which *governs* them.

Stepping onwards, we have at least three choices of direction.

- We might ask what sort of table has **All** P xs as its type of headers. Consider 'three-layer **All**'

$$\mathbf{Alll} \ \in \ ((x\!:\!X) \to P\,x \to \mathbf{Ty}) \to ((xs\!:\!\mathbf{List}\,X) \to \mathbf{All}\,P\,xs \to \mathbf{Ty})$$

  where data construction is in even tighter lock-step:

$$\mathbf{Alll}Q[][] \ni [] \qquad \mathbf{Alll}Q(x,\text{-}xs)(p,\text{-}ps) \ni q,\text{-}qs \ \Leftarrow \ Qxp \ni q, \mathbf{Alll}Qxsps \ni qs$$

- We might recognize **All** as the *predicate* lifting of **List** and ask what is the *relational* lifting.

$$\mathbf{ListR} \ \in \ (X \to Y \to \mathbf{Ty}) \to (\mathbf{List}\,X \to \mathbf{List}\,Y \to \mathbf{Ty})$$

  which requires that the two lists have the same length, or more pertinently a sensible notion of 'corresponding position', and that elements in corresponding positions are related:

$$\mathbf{ListR}R[][] \ni [] \qquad \mathbf{ListR}R(x,\text{-}xs)(y,\text{-}ys) \ni r,\text{-}rs \ \Leftarrow \ Rxy \ni r, \mathbf{ListR}Rxsys \ni rs$$

- We might reach into more dimensions, characterizing rectangular matrices with independent 'row' and 'column' headers, after the manner of a *spreadsheet*.

$$\mathbf{Matrix} \ \in \ (X \to Y \to \mathbf{Ty}) \to (\mathbf{List}\,X \to \mathbf{List}\,Y \to \mathbf{Ty})$$

  which is readily definable using **All**:

$$\mathbf{Matrix}\,R\,xs\,ys \ = \ \mathbf{All}\,(\lambda x \mapsto \mathbf{All}\,(\lambda y \mapsto R\,x\,y)\,ys)\,xs$$

The fact that **ListR** and **Matrix** have the same type but mean rather different things ought to prompt us to consider how to make the distinction plain. For **ListR**, the two lists are in the 'same dimension' and yoked together, with corresponding positions that are somehow cognate; for **Matrix**, the lists are in 'orthogonal dimensions', and any point in the cartesian product of their positions is in some way meaningful. Meanwhile, **ListR** and **Alll** both describe one-dimensional sequences yoked to two other one-dimensional sequences, but with differing patterns of type dependency.

The fact that **Matrix** is definable in terms of **All** should prompt us to ask whether we can manage the same for **Alll** and **ListR**. To avoid undue suspense,

category theorists can but type theorists cannot. By dependent pairing, we may form

$$\textbf{ListR}\; xs\; ys \;\; = \;\; (rs \!:\! \textbf{All}\,(\lambda x \mapsto (y \!:\! Y) \times R\, x\, y)\, xs) \times \; \textbf{map fst}\; rs = ys$$

That is, we forget that we know in advance the $\textbf{List}\,Y$ we want, yoke each x to any old y such that $R\,x\,y$, then, as an afterthought, demand the outrageous coincidence that the ys we collected turned out to be exactly the ys we now remember that we wanted. Such inverse image constructions create troublesome pedantry when one's audience is too mechanical to be readily intimidated. Similarly,

$$\textbf{Alll}\; Q\; xs\; ps \;\; = \;\; (qs \!:\! \textbf{All}\,(\lambda x \mapsto (p \!:\! P\, x) \times Q\, x\, p)\, xs) \times \textbf{map fst}\; qs = ps$$

where again the prior indexing is rendered as posterior projection. If we want to retain the intrinsic coherence that comes from knowing the indices in advance, the current state of the art requires us to invent $\textbf{Alll}$ and $\textbf{ListR}$ afresh, as if they have nothing to do with one another.

We have seen five structures — $\textbf{List}$, $\textbf{All}$, $\textbf{Alll}$, $\textbf{ListR}$ and $\textbf{Matrix}$ — but we might imagine many others, e.g., rectangular collections of cells where the length and height are not explicit in the type but their uniformity is enforced, or cuboids of data indexed by three matrices on the three faces meeting at the origin themselves indexed by two of three lists on their axes. Let us seek a common scheme for describing these m-dimensional data 'bricks' indexed over various n-dimensional 'bricks', where the n are selected from the m. The humble cons-list is then the simplest form of one-dimensional brick, in which the cell type is unindexed and so is the container type.

2 telescopes with explicit dependency

Let us start in *one* dimension, seeking to characterize the distinctions between $\textbf{List}$, $\textbf{All}$, $\textbf{Alll}$ and $\textbf{ListR}$. Look at how the cell types are indexed by 'headers' in each case

type former	cell indices
$\textbf{List}$	
$\textbf{All}$	$x \!:\! X$
$\textbf{Alll}$	$x \!:\! X, p \!:\! P\, x$
$\textbf{ListR}$	$x \!:\! X, y \!:\! Y$

These indices form *telescopes*, in the sense of de Bruijn, in which every entry type can depend on the entry values to its left. For $\textbf{List}$, $\textbf{All}$ and $\textbf{Alll}$, we see fully dependent telescopes of lengths 0, 1 and 2, respectively. But for $\textbf{ListR}$, we note that the type Y may *not* depend on the value of x. The pattern of permitted dependencies determines the type of container brick from which the cell indices are drawn. Above, the non-dependent xs and y are drawn from $\textbf{List}$s, while the dependent p must be drawn from an $\textbf{All}$. We may therefore specify the headers of a one-dimensional brick by giving a telescope with explicit dependency permissions.

Definition 2.1 (telescope with explicit dependency) *Inductively, the empty telescope ε is a telescope with explicit dependency, and an existing such telescope Δ may be extended to the right by some $\Delta, y : \vec{x}.T$ where y is a name fresh for Δ, and $\vec{x}$ is a dependency-closed selection of names bound in Δ i(n the order that Δ binds them), and T is a type with respect to the subtelescope $\vec{x} \downarrow \Delta$ of Δ selected by $\vec{x}$. Such a selection is dependency-closed if every x selected in $\vec{x}$ has all its dependencies included in $\vec{x}$. Dependency closure ensures the existence of the selected subtelescope.*

$$
\begin{aligned}
\downarrow\varepsilon &= \varepsilon \\
(\vec{z}y)\downarrow(\Delta, y:\vec{x}.T) &= (\vec{z} \downarrow \Delta), y:\vec{x}.T \quad noting\ \vec{x} \subseteq \vec{z} \\
(\vec{z}z)\downarrow(\Delta, y:\vec{x}.T) &= (\vec{z}z) \downarrow \Delta \qquad\quad if\ y \neq z
\end{aligned}
$$

Remark 2.1.1 (fresh names) *Of course, telescopes should be considered invariant under α-equivalence. Formally, telescopes are given in some nameless representation, with selections being given as bit vectors.*