Realizability at Work

Peng Fu

June 28, 2017

Abstract

In this note, I try to give an realizability interpretation of shape indexed dependent type. This answer several the questions I have before: 1. Programs in Gödel's **T** encodes its own termination proof, how to make it explicit. 2. How can one work on an index language without writing index annotation. This note shows that 1 is actually a special case of 2.

1 Dependent Types and Realizability à la Termination

1.1 Gödel's T

Definition 1.
Types $T \;::= \mathbf{Nat} \mid T \to T'$
Terms $e, n \;::= x \mid \lambda x.e \mid e\ e' \mid \mathsf{Z} \mid \mathsf{S} \mid \mathsf{elimN}_T$
Contexts $\Gamma \;::= \cdot \mid x : T, \Gamma$

We often write elimN whenever the context is clear.

Definition 2 (Reduction)**.**
$(\lambda x.e)\ e' \rightsquigarrow [e'/x]e$
$\mathsf{elimN}\ \mathsf{Z}\ e_1\ e_2 \rightsquigarrow e_1$
$\mathsf{elimN}\ (\mathsf{S}\ e')\ e_1\ e_2 \rightsquigarrow e_2\ e'\ (\mathsf{elimN}\ e'\ e_1\ e_2)$

Intuitively, $\mathsf{elimN}\ n\ e_1\ e_2$ means pattern matches on n, if n is Z, then evaluate e_1; if n is $\mathsf{S}\ n'$, then evaluate e_2 on the sub-component n' and the result of recursive call $\mathsf{elimN}\ n'\ e_1\ e_2$. Note that according to the type of elimN, e_2 always takes two arguments.

Definition 3 (Typing)**.**

$$\frac{(x : T) \in \Gamma}{\Gamma \vdash x : T}\ \mathit{var} \qquad \frac{\Gamma, x : T \vdash e : T'}{\Gamma \vdash \lambda x.e : T \to T'}\ \mathit{lam} \qquad \frac{\Gamma \vdash e : T \to T' \quad \Gamma \vdash e' : T}{\Gamma \vdash e\ e' : T'}\ \mathit{app}$$

$$\frac{}{\vdash \mathsf{Z} : \mathbf{Nat}} \qquad \frac{}{\vdash \mathsf{S} : \mathbf{Nat} \to \mathbf{Nat}} \qquad \frac{}{\vdash \mathsf{elimN}_T : \mathbf{Nat} \to T \to (\mathbf{Nat} \to T \to T) \to T}$$

Example 1.
Addition:
$\mathsf{add} : \mathbf{Nat} \to \mathbf{Nat} \to \mathbf{Nat}$
$\mathsf{add}\ n\ m = \mathsf{elimN}\ n\ m\ (\lambda x.\lambda r.\mathsf{S}\ r)$
Predecessor:
$\mathsf{pred} : \mathbf{Nat} \to \mathbf{Nat}$
$\mathsf{pred}\ n = \mathsf{elimN}\ n\ \mathsf{Z}\ (\lambda x.\lambda r.x)$

Exercise 1. *How to define multiplication?*

Exercise 2. *There is another recursive definition for addition:*
$\mathsf{add}\ \mathsf{Z}\ m = m$
$\mathsf{add}\ (\mathsf{S}\ n)\ m = \mathsf{add}\ n\ (\mathsf{S}\ m)$
Is this addition definable in Gödel's $\boldsymbol{T}$*?*

Exercise 3. *What kind of functions can be defined in Gödel's* $\boldsymbol{T}$*?*

Proposition 1. *If* $\Gamma \vdash e : T$*, then* e *is strongly normalizing[1].*

1.2 Realization à la Termination for Gödel's T

In this section, we are going to show each number in Gödel's $\mathbf{T}$ *realizes* its own termination proof, and more generally, each function *realizes* its own termination proof. For this, we are going to need to introduce a small proof system to talk about terminations in $\mathbf{T}$, more specifically, for each $e : T$, we introduce a formula T_e to assert that e is terminating. We will also need the usual logical apparatus such as quantification and implication. We call the resulting proof system Gödel's $\mathbf{T}\downarrow$.

Definition 4 (Gödel's $\mathbf{T}\downarrow$)**.**
Types/Formulas $F \ ::= T_e \mid F \to F' \mid \Pi x : T.F \mid \lambda x.F \mid F\ e$
Terms/Proofs $p \ ::= \alpha \mid \lambda\alpha.p \mid p\ p' \mid \lambda x.p \mid p\ e \mid \mathsf{indN}_T \mid \mathsf{pZ} \mid \mathsf{pS}$
Kinds $K \ ::= * \mid T \to K$
Assumptions $\Delta \ ::= \cdot \mid x : T, \Delta \mid \alpha : F, \Delta$

Note that we use $\Pi x : T.F$ to denote first order quantification, it means the formula F holds for any $x : T$ (it is also called a *implicit dependent type*). We call $\lambda x.F$ a property about x, and if e satisfies the property F, we write $F\ e$. Since now F can stand for either a formula or a property, we need a system of *kinding* to distinguish these two. If F has kind $*$, then F is a formula, otherwise F is a property.

Definition 5 (Kinding)**.**

$$\frac{\Delta, x : T \vdash F : K}{\Delta \vdash \lambda x.F : T \to K} \quad \frac{\Delta \vdash F : T \to K \quad \Delta \vdash e : T}{\Delta \vdash F\ e : K} \quad \frac{\Delta \vdash e : T}{\Delta \vdash T_e : *} \quad \frac{\Delta, x : T \vdash F : *}{\Delta \vdash \Pi x : T.F : *} \quad \frac{\Delta \vdash F : * \quad \Delta \vdash F' : *}{\Delta \vdash F \to F' : *}$$

Definition 6 (Typing)**.**

$$\frac{(\alpha : F) \in \Delta}{\Delta \vdash \alpha : F} \qquad \frac{\Delta, \alpha : F \vdash p : F'}{\Delta \vdash \lambda\alpha.p : F \to F'} \qquad \frac{\Delta \vdash p : F \to F' \quad \Delta \vdash p' : F}{\Delta \vdash p\ p' : F'}$$

$$\frac{\Delta, x : T \vdash p : F}{\Delta \vdash \lambda x.p : \Pi x : T.F} \qquad \frac{\Delta \vdash p : \Pi x : T.F \quad \Delta \vdash e : T}{\Delta \vdash p\ e : [e/x]F} \qquad \frac{\Delta \vdash p : F \quad F = F'}{\Delta \vdash p : F'}$$

$$\frac{}{\vdash \mathsf{pZ} : \mathbf{Nat}_{\mathsf{Z}}} \qquad \frac{}{\vdash \mathsf{pS} : \Pi n : \mathbf{Nat}.\mathbf{Nat}_n \to \mathbf{Nat}_{(\mathsf{S}\ n)}}$$

$$\frac{}{\vdash \mathsf{indN}_F : \Pi x : \mathbf{Nat}.\mathbf{Nat}_x \to F\ \mathsf{Z} \to (\Pi y : \mathbf{Nat}.\mathbf{Nat}_y \to F\ y \to F\ (\mathsf{S}\ y)) \to F\ x}$$

We call $\mathsf{pZ}, \mathsf{pS}, \mathsf{indN}_F$ axioms. The axiom pZ asserts Z is terminating. The axiom pS asserts for any n of type $\mathbf{Nat}$, if n is terminating, then $\mathsf{S}\ n$ is terminating. The axiom indN_F corresponds to induction principle: we can prove a property F holds for any terminating number x if the property F holds for Z and if the property holds for a terminating number y, then the property holds for $\mathsf{S}\ y$.

Intuitively, $\Gamma \vdash p : F$ means p is a proof of F under the assumption Γ. From the typing we can see that p and e are really separate entities, but p can be *decorated* with e. Later we will see that these decorations can be erased, giving us the true structure of the proof.

Definition 7 (Equivalence).
$(\lambda x.F)\ e = [e/x]F$
$[e/x]F = [e'/x]F$ *if* $e = e'$.

The equality $e = e'$ means that there is some e'' such that $e \rightsquigarrow^* e''$ and $e' \rightsquigarrow^* e''$. The first equivalence corresponds to axiom of comprehension, and the second one corresponds to axiom of extensionality.

Definition 8 (Proof Reduction).
$e \rightsquigarrow e'$
$(\lambda x.p)\ e' \rightsquigarrow [e'/x]p$
$(\lambda \alpha.p)\ p' \rightsquigarrow [p'/\alpha]p$
$\mathsf{indN}\ \mathsf{Z}\ \mathsf{pZ}\ p_1\ p_2 \rightsquigarrow p_1$
$\mathsf{indN}\ (\mathsf{S}\ e')\ (\mathsf{pS}\ e'\ p')\ p_1\ p_2 \rightsquigarrow p_2\ e'\ p'\ (\mathsf{indN}\ e'\ p'\ p_1\ p_2)$

Proof reduction gives us a way to simplify the proofs in $\mathbf{T}\downarrow$. There are some peculiarities about the rules for indN: not only it performs a simultaneous pattern matching on its first two arguments, but also these patterns are *non-linear*, i.e. we have the same e' in $\mathsf{indN}\ (\mathsf{S}\ e')\ (\mathsf{pS}\ e'\ p')\ p_1\ p_2$.

Example 2. *Consider the following recursive definition of addition.*
$\mathsf{add} : \mathbf{Nat} \to \mathbf{Nat} \to \mathbf{Nat}$
$\mathsf{add}\ \mathsf{Z}\ m = m$
$\mathsf{add}\ (\mathsf{S}\ n)\ m = \mathsf{add}\ n\ (\mathsf{S}\ m)$
It is not straightforward to represent it in Gödel's $\mathbf{T}$. *But may be we can look at the problem from another angle: as we claim that each function in Gödel's* $\mathbf{T}$ *realizes its own termination proof, maybe we can prove the termination of this addition in* $\mathbf{T}\downarrow$, *this proof should gives us enough hints on how to implement the addition.*

So we do not know how to implement this addition, but whatever implementation add *that we come up, it should have this property:* $\mathsf{add}\ \mathsf{Z}\ m \rightsquigarrow^* m$ *and* $\mathsf{add}\ (\mathsf{S}\ n)\ m \rightsquigarrow^* \mathsf{add}\ n\ (\mathsf{S}\ m)$. *Now let us try to prove this* add *(without knowing how to implement it in* ***T****) is terminating (total) in* $\mathbf{T}\downarrow$. *This means we will need to find a* p *such that* $\vdash p : \Pi n : \mathbf{Nat}.\mathbf{Nat}_n \to \Pi m : \mathbf{Nat}.\mathbf{Nat}_m \to \mathbf{Nat}_{(\mathsf{add}\ n\ m)}$.

First let us see how to prove it informally. Let us first try this approach: Assume n, m *is terminating, we will prove* $\mathsf{add}\ n\ m$ *is terminating by induction on* n: *Suppose* $n = \mathsf{Z}$, *then* $\mathsf{add}\ n\ m \rightsquigarrow m$, *and by assumption we know that* m *is terminating. Thus* $\mathsf{add}\ \mathsf{Z}\ m$ *is terminating. Suppose* $n = \mathsf{S}\ n'$. *Since* $\mathsf{add}\ (\mathsf{S}\ n')\ m \rightsquigarrow \mathsf{add}\ n'\ (\mathsf{S}\ m)$, *we need to prove* $\mathsf{add}\ n'\ (\mathsf{S}\ m)$ *is terminating. Our inductive hypothesis in this case is* $\mathsf{add}\ n'\ m$ *is terminating. This means our inductive hypothesis is not general enough to be useful in this case.*

So now let us prove: Assume n *is terminating, then* $\mathsf{add}\ n\ m$ *is terminating for any terminating* m. *We again prove it by induction on* n: *When* $n = \mathsf{Z}$, *it is easy. When* $n = \mathsf{S}\ n'$, *we have the inductive hypothesis* $\mathsf{add}\ n'\ m$ *is terminating for any terminating* m. *We just need to prove* $\mathsf{add}\ n'\ (\mathsf{S}\ m)$ *is terminating for any terminating* m. *We can see our goal is just a special case of the inductive hypothesis this time around.*

Now let us construct a formal proof using our informal reasoning. Let $\Delta = n : \mathbf{Nat}, \alpha_1 : \mathbf{Nat}_n$, *we will find a* p *such that* $\Delta \vdash p : \Pi m : \mathbf{Nat}.\mathbf{Nat}_m \to \mathbf{Nat}_{(\mathsf{add}\ n\ m)}$. *Of course we will use induction, in this case let* $F = \lambda x.\Pi m : \mathbf{Nat}.\mathbf{Nat}_m \to \mathbf{Nat}_{(\mathsf{add}\ x\ m)}$. *We have*

$$\Delta \vdash \mathsf{indN}_F\ n\ \alpha_1 : F\ \mathsf{Z} \to (\Pi y : \mathbf{Nat}.\mathbf{Nat}_y \to F\ y \to F\ (\mathsf{S}\ y)) \to F\ n.$$

Now we replace all the F *by* $\lambda x.\Pi m : \mathbf{Nat}.\mathbf{Nat}_m \to \mathbf{Nat}_{(\mathsf{add}\ x\ m)}$, *by equivalence, we have the following.*

$\Delta \vdash \mathsf{indN}_F\ n\ \alpha_1 :$
$\quad (\Pi m : \mathbf{Nat}.\mathbf{Nat}_m \to \mathbf{Nat}_m) \to$
$\quad\quad (\Pi y : \mathbf{Nat}.\mathbf{Nat}_y \to (\Pi m : \mathbf{Nat}.\mathbf{Nat}_m \to \mathbf{Nat}_{(\mathsf{add}\ y\ m)}) \to (\Pi m : \mathbf{Nat}.\mathbf{Nat}_m \to \mathbf{Nat}_{(\mathsf{add}\ y\ (\mathsf{S}\ m))})) \to$
$\quad\quad\quad (\Pi m : \mathbf{Nat}.\mathbf{Nat}_m \to \mathbf{Nat}_{(\mathsf{add}\ n\ m)})$

The first argument for $\mathsf{indN}_F\ n\ \alpha_1$ *would be identity:* $\lambda m.\lambda\alpha.\alpha$*. So now we have:*

$$\begin{aligned}&\Delta \vdash \mathsf{indN}_F\ n\ \alpha_1\ (\lambda m.\lambda\alpha.\alpha) :\\ &\quad(\Pi y : \mathbf{Nat}.\mathbf{Nat}_y \to (\Pi m : \mathbf{Nat}.\mathbf{Nat}_m \to \mathbf{Nat}_{(\mathsf{add}\ y\ m)}) \to (\Pi m : \mathbf{Nat}.\mathbf{Nat}_m \to \mathbf{Nat}_{(\mathsf{add}\ y\ (\mathsf{S}\ m))})) \to\\ &\qquad(\Pi m : \mathbf{Nat}.\mathbf{Nat}_m \to \mathbf{Nat}_{(\mathsf{add}\ n\ m)}).\end{aligned}$$

Now we need to find a proof p' *such that*
$\Delta \vdash p' : (\Pi y : \mathbf{Nat}.\mathbf{Nat}_y \to (\Pi m : \mathbf{Nat}.\mathbf{Nat}_m \to \mathbf{Nat}_{(\mathsf{add}\ y\ m)}) \to (\Pi m : \mathbf{Nat}.\mathbf{Nat}_m \to \mathbf{Nat}_{(\mathsf{add}\ y\ (\mathsf{S}\ m))})$.
Then we would have $\Delta \vdash \mathsf{indN}_F\ n\ \alpha_1\ (\lambda m.\lambda\alpha.\alpha)\ p' : \Pi m : \mathbf{Nat}.\mathbf{Nat}_m \to \mathbf{Nat}_{(\mathsf{add}\ n\ m)}$*. Looking at the type of* p'*, the post-condition is an instance of pre-condition. So we have the following.*

$$\begin{gathered}\Delta, y : \mathbf{Nat}, \alpha_2 : \mathbf{Nat}_y, \alpha_3 : \Pi m : \mathbf{Nat}.\mathbf{Nat}_m \to \mathbf{Nat}_{(\mathsf{add}\ y\ m)} \vdash \lambda m.\alpha_3\ (\mathsf{S}\ m)\ (\mathsf{pS}\ m) :\\ \Pi m : \mathbf{Nat}.\mathbf{Nat}_m \to \mathbf{Nat}_{(\mathsf{add}\ y\ (\mathsf{S}\ m))}.\end{gathered}$$

So $p' = \lambda y.\lambda\alpha_2.\lambda\alpha_3.\lambda m.\alpha_3\ (\mathsf{S}\ m)\ (\mathsf{pS}\ m)$*. So* $p = \mathsf{indN}_F\ n\ \alpha_1\ (\lambda m.\lambda\alpha.\alpha)\ (\lambda y.\lambda\alpha_2.\lambda\alpha_3.\lambda m.\alpha_3\ (\mathsf{S}\ m)\ (\mathsf{pS}\ m))$*. Thus we have the following.*

$$\begin{gathered}\vdash \lambda n.\lambda\alpha_1.\mathsf{indN}_F\ n\ \alpha_1\ (\lambda m.\lambda\alpha.\alpha)\ (\lambda y.\lambda\alpha_2.\lambda\alpha_3.\lambda m.\alpha_3\ (\mathsf{S}\ m)\ (\mathsf{pS}\ m)) :\\ \Pi n : \mathbf{Nat}.\mathbf{Nat}_n \to \Pi m : \mathbf{Nat}.\mathbf{Nat}_m \to \mathbf{Nat}_{(\mathsf{add}\ n\ m)}\end{gathered}$$

Now that we have obtain a proof of $\Pi n : \mathbf{Nat}.\mathbf{Nat}_n \to \Pi m : \mathbf{Nat}.\mathbf{Nat}_m \to \mathbf{Nat}_{(\mathsf{add}\ n\ m)}$*, we can remove all the decorations, in* $\lambda n.\lambda\alpha_1.\mathsf{indN}_F\ n\ \alpha_1\ (\lambda m.\lambda\alpha.\alpha)\ (\lambda y.\lambda\alpha_2.\lambda\alpha_3.\lambda m.\alpha_3\ (\mathsf{S}\ m)\ (\mathsf{pS}\ m))$*, obtaining* $\lambda\alpha_1.\mathsf{indN}_F\ \alpha_1\ (\lambda\alpha.\alpha)\ (\lambda\alpha_2.\lambda\alpha_3.\alpha_3\ \mathsf{pS})$*, which is very similar to* $\lambda x_1.\mathsf{elimN}\ x_1\ (\lambda x.x)\ (\lambda x_2.\lambda x_3.x_3\ \mathsf{S})$*. We can quickly verify if we define* $\mathsf{add} = \lambda x_1.\mathsf{elimN}\ x_1\ (\lambda x.x)\ (\lambda x_2.\lambda x_3.x_3\ \mathsf{S})$*, whether we have* $\mathsf{add}\ \mathsf{Z}\ m \rightsquigarrow^* m$ *and* $\mathsf{add}\ (\mathsf{S}\ n)\ m \rightsquigarrow^* \mathsf{add}\ n\ (\mathsf{S}\ m)$*.*

Exercise 4. *Can we find a* p *such that* $\vdash p : \Pi x : \mathbf{Nat}.\mathbf{Nat}_x$*?*

1.2.1 Forgetful Mapping from $\mathbf{T}\downarrow$ to $\mathbf{T}$

In Example 2, we talked about removing decorations and extracting a program from a proof in $\mathbf{T}\downarrow$. We will formally define this process in this section.

The following definition $|p|$ erases all the decorations in p, and transforming the essential part of p to a term in $\mathbf{T}$.

Definition 9.
$|\alpha| = x_\alpha$
$|p\ p'| = |p|\ |p'|$
$|\lambda\alpha.p| = \lambda x_\alpha.|p|$
$|\lambda x.p| = |p|$
$|p\ e| = |p|$
$|\mathsf{pZ}| = \mathsf{Z}$
$|\mathsf{pS}| = \mathsf{S}$
$|\mathsf{indN}_F| = \mathsf{elimN}_{|F|}$

The following definition map a formula in $\mathbf{T}\downarrow$ to a type in $\mathbf{T}$.

Definition 10.
$|T_e| = T$
$|F \to F'| = |F| \to |F'|$
$|\Pi x : T.F| = |F|$
$|\lambda x.F| = |F|$
$|F\ e| = |F|$

The following definition apply the erasure to the assumption Δ.

Definition 11.
$|x : T, \Delta| = |\Delta|$
$|\alpha : F, \Delta| = x_\alpha : |F|, |\Delta|$

The theorem below is proved inductively, it gives us an algorithm to do the transformation.

Theorem 1. *If $\Delta \vdash p : F$ then $|\Delta| \vdash |p| : |F|$.*

Proof. By induction on the derivation of $\Delta \vdash p : F$. □

Abstractly, the above theorem means that there is a map from $\vdash p : \mathbf{Nat}_n$ to $\vdash n : \mathbf{Nat}$ for a normalized n. In the next section, we will show the other map from $\vdash n : \mathbf{Nat}$ to $\vdash p : \mathbf{Nat}_n$.

1.2.2 Annotated Mapping from T to T ↓

We mentioned before each function in **T** realizes its own termination proof, this also means that if we manage to define a function in **T**, we should be able to obtain its termination proof *automatically.* In this section, we show how to obtain such proof.

The following annotation maps a type in **T** to a formula in **T** ↓. this annotation, together with Theorem 3, are often called *realizability* in the literature.

Definition 12.
$[\mathbf{Nat}]_e = \mathbf{Nat}_e$
$[T \to T']_e = \Pi y : T.[T]_y \to [T']_{(e\ y)}$, *where y is fresh.*

Theorem 2. $|[T]_e| = T$

Definition 13.
$[\cdot] = \cdot$
$[x : T, \Gamma] = x : T, \alpha : [T]_x, [\Gamma]$, *where α is fresh.*

Theorem 3. *If $\Gamma \vdash e : T$ then there exists a p such that $[\Gamma] \vdash p : [T]_e$. We often denote such p as p_e.*

Proof. By induction on derivation of $\Gamma \vdash e : T$. All the cases are straightforward, except the following case.

- Case. $\vdash \mathsf{elimN}_T : \mathbf{Nat} \to T \to (\mathbf{Nat} \to T \to T) \to T$.

  We know that $[\mathbf{Nat} \to T \to (\mathbf{Nat} \to T \to T) \to T]_{\mathsf{elimN}_T} =$

  $\Pi y : \mathbf{Nat}.\mathbf{Nat}_y \to \Pi y_1 : T.[T]_{y_1} \to \Pi y_2 : \mathbf{Nat} \to T \to T.$

  $(\Pi y_3 : \mathbf{Nat}.\mathbf{Nat}_{y_3} \to \Pi y_4 : T.[T]_{y_4} \to [T]_{(y_2\ y_3\ y_4)}) \to [T]_{(\mathsf{elimN}\ y\ y_1\ y_2)}$.

  We just need to show that this type is inhabited using indN. Let $\Delta = y : \mathbf{Nat}, \alpha_1 : \mathbf{Nat}_y, y_1 : T, \alpha_2 : [T]_{y_1}, y_2 : \mathbf{Nat} \to T \to T, \alpha_3 : (\Pi y_3 : \mathbf{Nat}.\mathbf{Nat}_{y_3} \to \Pi y_4 : T.[T]_{y_4} \to [T]_{(y_2\ y_3\ y_4)})$. We need to find a p such that $\Delta \vdash p : [T]_{(\mathsf{elimN}\ y\ y_1\ y_2)}$. Let $F = \lambda x.[T]_{(\mathsf{elimN}\ x\ y_1\ y_2)}$. Then we have the following.

  $\Delta \vdash \mathsf{indN}_F\ y :$

  $\mathbf{Nat}_y \to [T]_{(\mathsf{elimN}\ \mathsf{Z}\ y_1\ y_2)} \to$

  $(\Pi z : \mathbf{Nat}.\mathbf{Nat}_z \to [T]_{(\mathsf{elimN}\ z\ y_1\ y_2)} \to [T]_{(\mathsf{elimN}\ (\mathsf{S}\ z)\ y_1\ y_2)}) \to [T]_{(\mathsf{elimN}\ y\ y_1\ y_2)}$

  By type equivalence, we have the following.

  $\Delta \vdash \mathsf{indN}_F\ y :$

  $\mathbf{Nat}_y \to [T]_{y_1} \to (\Pi z : \mathbf{Nat}.\mathbf{Nat}_z \to [T]_{(\mathsf{elimN}\ z\ y_1\ y_2)} \to [T]_{(y_2\ z\ y_1 (\mathsf{elimN}\ z\ y_1\ y_2))}) \to [T]_{(\mathsf{elimN}\ y\ y_1\ y_2)}$.

  Hence the following hold.

  $\Delta \vdash \mathsf{indN}_F\ y\ \alpha_1\ \alpha_2 :$

  $(\Pi z : \mathbf{Nat}.\mathbf{Nat}_z \to [T]_{(\mathsf{elimN}\ z\ y_1\ y_2)} \to [T]_{(y_2\ z\ y_1 (\mathsf{elimN}\ z\ y_1\ y_2))}) \to [T]_{(\mathsf{elimN}\ y\ y_1\ y_2)}$.

  Moreover, $\Delta \vdash \lambda z.\lambda\alpha.\alpha_3\ z\ \alpha\ (\mathsf{elimN}\ z\ y_1\ y_2) : \Pi z : \mathsf{Nat}.\mathsf{Nat}_z.[T]_{(\mathsf{elimN}\ z\ y_1\ y_2)} \to [T]_{(y_2\ z\ y_1 (\mathsf{elimN}\ z\ y_1\ y_2))}$.
  So $\Delta \vdash \mathsf{indN}_F\ y\ \alpha_1\ \alpha_2\ (\lambda z.\lambda\alpha.\alpha_3\ z\ \alpha\ (\mathsf{elimN}\ z\ y_1\ y_2)) : [T]_{(\mathsf{elimN}\ y\ y_1\ y_2)}$. Thus we are done.

□

Again, Theorem 3 gives us an algorithm to obtain proofs from functions. An example is that for a normalized number $n : \mathbf{Nat}$, we have a proof of its termination $p : \mathbf{Nat}_n$.

Observing the proof of Theorem 3, we can see the map from $\mathbf{T}$ to $\mathbf{T}\downarrow$ is essentially an eta-expansion, i.e. for $|p_e|$ is an eta-expansion of e.

2 Dependent Types and Realizability à la Shape

The moral of last section is the function and the proof of its termination is the same thing, similar phenominon appears when we indexed container data type by its shape. For example, the vector append function behaves the same as list append function. This suggests we could adopt the same realizability approach to understand shape indexing.

2.1 Gödel's List T

Gödel's **List T** allows us to define terminating functions that operate on list of things. Note that we do not include **Nat** at the moment just to simplify the representation, adding type **Nat** will not be a problem.

Definition 14.
Types $T \ ::= \mathbf{Unit} \mid \mathbf{Bool} \mid \mathbf{Bit} \mid \mathbf{List}\ T \mid T \to T'$
Terms $e, n \ ::= x \mid \lambda x.e \mid e\ e' \mid \mathsf{U} \mid \mathsf{True} \mid \mathsf{False} \mid \mathsf{Nil}_T \mid \mathsf{Cons}_T \mid \mathsf{if}_T \mid \mathsf{elimL}_{T,T'} \mid \mathsf{init}$
Contexts $\Gamma \ ::= \cdot \mid x : T, \Gamma$

Definition 15 (Typing).

$$\frac{(x:T)\in\Gamma}{\Gamma\vdash x:T}\ \mathit{var} \qquad \frac{\Gamma, x:T\vdash e:T'}{\Gamma\vdash \lambda x.e : T\to T'}\ \mathit{lam} \qquad \frac{\Gamma\vdash e:T\to T' \quad \Gamma\vdash e':T}{\Gamma\vdash e\ e' : T'}\ \mathit{app}$$

$$\frac{}{\vdash \mathsf{U} : \mathbf{Unit}} \qquad \frac{}{\vdash \mathsf{True} : \mathbf{Bool}} \qquad \frac{}{\vdash \mathsf{False} : \mathbf{Bool}}$$

$$\frac{}{\vdash \mathsf{if}_T : \mathsf{Bool}\to T\to T\to T} \qquad \frac{}{\vdash \mathsf{Nil}_T : \mathbf{List}\ T} \qquad \frac{}{\vdash \mathsf{Cons}_T : T\to \mathbf{List}\ T\to \mathbf{List}\ T}$$

$$\frac{}{\vdash \mathsf{elimL}_{T,T'} : \mathbf{List}\ T\to T'\to (T\to \mathbf{List}\ T\to T'\to T')\to T'} \qquad \frac{}{\vdash \mathsf{init} : \mathbf{Bool}\to \mathbf{Bit}}$$

Note that there is no constructors for **Bit**, it is treated as abstract data type, one can add more primitive functions such as $\mathsf{and} : \mathbf{Bit}\to\mathbf{Bit}\to\mathbf{Bit}$, but we will not consider these as we can extend the approach accordingly.

Definition 16.
$(\lambda x.e)\ e' \rightsquigarrow [e'/x]e$
$\mathsf{if}\ \mathsf{True}\ e_1\ e_2 \rightsquigarrow e_1$
$\mathsf{if}\ \mathsf{False}\ e_1\ e_2 \rightsquigarrow e_2$
$\mathsf{elimL}\ \mathsf{Nil}_T\ e_1\ e_2 \rightsquigarrow e_1$
$\mathsf{elimL}\ (\mathsf{Cons}_T\ e'\ e'')\ e_1\ e_2 \rightsquigarrow e_2\ e'\ e''\ (\mathsf{elimL}\ e''\ e_1\ e_2)$

Note that elimL pattern matches on the first argument, if it is Nil_T, then it returns e_1, otherwise it called e_2, supplying it with sub-component e', e'' and the recursive argument $(\mathsf{elimL}\ e''\ e_1\ e_2)$.

Example 3.
Append:
$\mathsf{app} : \mathbf{List\ Bit}\to\mathbf{List\ Bit}\to\mathbf{List\ Bit}$
$\mathsf{app}\ l_1\ l_2 = \mathsf{elimL}\ l_1\ l_2\ (\lambda x.\lambda l.\lambda r.\mathsf{Cons}\ x\ r)$
Tail:

$\mathsf{tail} : \textbf{List Bit} \to \textbf{List Bit}$
$\mathsf{tail}\ l = \mathsf{elimL}\ l\ \mathsf{Nil}\ (\lambda x.\lambda l.\lambda r.l)$

2.2 Realization à la Shape

We should distinguish the notion of shape and size. It is often customary to use natural numbers as indexes to denote the size of a container, but number may not be indicate the shape. For example consider a list of list $[[1, 1]]$, while it does have size 1, but its shape is definitely not 1.

We first specify shape as a endo-function from types to types in **List T**. This definition of shape is due to Peter Selinger.

Definition 17 (Shape)**.**
$\mathsf{Sh}\ \textbf{Unit} = \textbf{Unit}$
$\mathsf{Sh}\ \textbf{Bool} = \textbf{Bool}$
$\mathsf{Sh}\ \textbf{Bit} = \textbf{Unit}$
$\mathsf{Sh}\ (\textbf{List}\ T) = \textbf{List}\ (\mathsf{Sh}\ T)$
$\mathsf{Sh}\ (T \to T') = \mathsf{Sh}\ T \to \mathsf{Sh}\ T'$

We write $\mathsf{Sh}(\Gamma)$ to mean applying the shape function to all the types in Γ. As a matter of fact, shape function has a functorial behaviour.

Theorem 4. *If $\Gamma \vdash e : T$, then there exists an e' such that $\mathsf{Sh}(\Gamma) \vdash e' : \mathsf{Sh}\ T$, we denote such e' as $\mathsf{Sh}\ e$.*

Proof. By induction on derivation of $\Gamma \vdash e : T$. □

Note that $\mathsf{Sh}\ \mathsf{init} = \lambda x.\mathsf{U}$ and $\mathsf{Sh}\ (e\ e') = (\mathsf{Sh}\ e)\ (\mathsf{Sh}\ e')$. Now we are ready to define Gödel's Sh (**List T**), a realization of **List T** by shape.

Definition 18 (Gödel's Sh (**List T**))**.**
Types $F\ ::=\ T_e \mid F \to F' \mid \Pi x : T.F \mid \lambda x.F \mid F\ e$
Terms $p\ ::= \alpha \mid \lambda\alpha.p \mid p\ p' \mid \lambda x.p \mid p\ e \mid \mathsf{VNil}_T \mid \mathsf{VCons}_T \mid \mathsf{elimV}_{T,F} \mid \mathsf{pU} \mid \mathsf{pT} \mid \mathsf{pF} \mid \mathsf{elimIf}_F \mid \mathsf{pInit}$
Kinds $K\ ::= * \mid T \to K$
Contexts $\Delta\ ::= \cdot \mid x : T, \Delta \mid \alpha : F, \Delta$

While we can still read T_e as e of type $\mathsf{Sh}\ T$ is terminating, it is more helpful to understand T_e as a data T index by a shape data e.

Definition 19 (Reduction)**.**
$e \rightsquigarrow e'$
$(\lambda x.p)\ e' \rightsquigarrow [e'/x]p$
$(\lambda\alpha.p)\ p' \rightsquigarrow [p'/\alpha]p$
$\mathsf{elimIf}\ \mathsf{True}\ \mathsf{pT}\ p_1\ p_2 \rightsquigarrow p_1$
$\mathsf{elimIf}\ \mathsf{False}\ \mathsf{pF}\ p_1\ p_2 \rightsquigarrow p_2$
$\mathsf{elimV}\ \mathsf{Nil}\ (\mathsf{VNil}\ \mathsf{Nil})\ p_1\ p_2 \rightsquigarrow p_1$
$\mathsf{elimV}\ (\mathsf{Cons}\ a\ a')\ (\mathsf{VCons}\ a\ p'\ a'\ p'')\ p_1\ p_2 \rightsquigarrow p_2\ a\ p'\ a'\ p''\ (\mathsf{elimV}\ a'\ p''\ p_1\ p_2)$

Notice the simultaneous pattern maching and non-linearity for elimV and elimIf.

Definition 20 (Type Equivalence)**.**
$(\lambda x.F)\ e = [e/x]F$
$[e/x]F = [e'/x]F$ *if* $e = e'$.

Definition 21 (Kinding)**.**

$$\frac{\Delta, x : T \vdash F : K}{\Delta \vdash \lambda x.F : T \to K} \qquad \frac{\Delta \vdash F : T \to K \quad \Delta \vdash e : T}{\Delta \vdash F\ e : K} \qquad \frac{\Delta \vdash e : \mathsf{Sh}\ T}{\Delta \vdash T_e : *} \qquad \frac{\Delta, x : T \vdash F : *}{\Delta \vdash \Pi x : T.F : *} \qquad \frac{\Delta \vdash F : * \quad \Delta \vdash F' : *}{\Delta \vdash F \to F' : *}$$

The following definition shows how to map types in **List T** to Sh (**List T**).

Definition 22.
$[\mathbf{Unit}]_e = \mathbf{Unit}_e$
$[\mathbf{Bool}]_e = \mathbf{Bool}_e$
$[\mathbf{Bit}]_e = \mathbf{Bit}_e$
$[\mathbf{List}\ T]_e = (\mathbf{List}\ T)_e$
$[T \to T']_e = \Pi y : \mathsf{Sh}\ T.[T]_y \to [T']_{(e\ y)}$, *where y is fresh.*

Definition 23 (Typing)**.**

$$\frac{(\alpha : F) \in \Delta}{\Delta \vdash \alpha : F} \qquad \frac{\Delta, \alpha : F \vdash p : F'}{\Delta \vdash \lambda\alpha.p : F \to F'} \qquad \frac{\Delta \vdash p : F \to F' \quad \Delta \vdash p' : F}{\Delta \vdash p\ p' : F'}$$

$$\frac{\Delta, x : T \vdash p : F}{\Delta \vdash \lambda x.p : \Pi x : T.F} \qquad \frac{\Delta \vdash p : \Pi x : T.F \quad \Delta \vdash e : T}{\Delta \vdash p\ e : [e/x]F} \qquad \frac{\Delta \vdash p : F \quad F = F'}{\Delta \vdash p : F'}$$

$$\frac{}{\vdash \mathsf{pU} : \mathbf{Unit}_{\mathsf{U}}} \qquad \frac{}{\vdash \mathsf{pT} : \mathbf{Bool}_{\mathsf{True}}} \qquad \frac{}{\vdash \mathsf{pF} : \mathbf{Bool}_{\mathsf{False}}}$$

$$\frac{}{\vdash \mathsf{pInit} : \Pi x : \mathbf{Bool}.\mathbf{Bool}_x \to \mathbf{Bit}_{\mathsf{U}}}$$

$$\frac{}{\vdash \mathsf{elimIf}_F : \Pi y : \mathbf{Bool}.\mathbf{Bool}_y \to F\ \mathsf{True} \to F\ \mathsf{False} \to F\ y}$$

$$\frac{}{\vdash \mathsf{VNil}_T : \Pi x : \mathbf{List}\ (\mathsf{Sh}\ T).(\mathbf{List}\ T)_x}$$

$$\frac{}{\vdash \mathsf{VCons}_T : \Pi y : \mathsf{Sh}\ T.[T]_y \to \Pi x : \mathbf{List}\ (\mathsf{Sh}\ T).(\mathbf{List}\ T)_x \to (\mathbf{List}\ T)_{(\mathsf{Cons}\ y\ x)}}$$

$$\frac{}{\vdash \mathsf{elimV}_{T,F} : A}$$

Note that $A = \Pi x : \mathsf{List}\ (\mathsf{Sh}\ T).(\mathsf{List}\ T)_x \to F\ \mathsf{Nil}_{(\mathsf{Sh}\ T)} \to$
$(\Pi z : \mathsf{Sh}\ T.[T]_z \to \Pi y : \mathbf{List}\ (\mathsf{Sh}\ T).(\mathbf{List}\ T)_y \to F\ y \to F\ (\mathsf{Cons}\ z\ y)) \to F\ x$

Example 4. *Recalled that we have the usual list append function:*
$\mathsf{app} : \mathbf{List\ Bit} \to \mathbf{List\ Bit} \to \mathbf{List\ Bit}$
$\mathsf{app}\ l_1\ l_2 = \mathsf{elimL}\ l_1\ l_2\ (\lambda x.\lambda l.\lambda r.\mathsf{Cons}\ x\ r)$
Note that $\mathsf{Sh}\ \mathsf{app} : \mathbf{List\ Unit} \to \mathbf{List\ Unit} \to \mathbf{List\ Unit} = \mathsf{app}$. *We are going define a vector append function, which is the following.*
$\mathsf{vApp} : \Pi x : \mathbf{List\ Unit}.(\mathbf{List\ Bit})_x \to \Pi y : \mathbf{List\ Unit}.(\mathbf{List\ Bit})_y \to (\mathbf{List\ Bit})_{(\mathsf{app}\ x\ y)}$
$\mathsf{vApp}\ n_1\ \alpha_1\ n_2\ \alpha_2 = \mathsf{elimV}_{\mathbf{Bit},F}\ n_1\ \alpha_1\ \alpha_2\ (\lambda z.\lambda\beta.\lambda y.\lambda\beta_1.\mathsf{VCons}\ z\ \beta\ (\mathsf{app}\ y\ n_2)\ \beta_1)$.
Note that $F = \lambda x.(\mathbf{List\ Bit})_{(\mathsf{app}\ x\ n_2)}$. *We can see that if we remove all the terms that are in* **List T***, the* vApp *is essentially the same as* app*. Wouldn't it be nice if we just need to write* app *and let the compiler derive* vApp*? In next section we will show exactly how to do this.*

2.2.1 Annotated Mapping from List T to Sh (List T)

In this section, we will show how to transform programs in **List T** to programs in Sh (**List T**).

Definition 24.
$[\cdot] = \cdot$
$[x : T, \Gamma] = x : \mathsf{Sh}\ T, \alpha : [T]_x, [\Gamma]$, *where α is fresh.*

Note that the above definition exhibit a map from $x : T$ to a dependent pair $x : \mathsf{Sh}\ T, \alpha : [T]_x$. We will show another map from $x : \mathsf{Sh}\ T, \alpha : [T]_x$ to $x : T$ in next section.

The proof of the following theorem gives an algorithm to systematically annotate programs in **List T**.

Theorem 5. *If $\Gamma \vdash e : T$, then there exists a p such that $[\Gamma] \vdash p : [T]_{(\mathsf{Sh}\ e)}$.*

Proof. By induction on derivation of $\Gamma \vdash e : T$. Theorem 4 is needed, and the tricky cases are the base cases. The cases for $\mathsf{init}, \mathsf{U}, \mathsf{True}, \mathsf{False}, \mathsf{Nil}_T$ are straightforward.

- Case. $\vdash \mathsf{if}_T : \mathbf{Bool} \to T \to T \to T$

  We need to find a p such that $\vdash p : \Pi y : \mathbf{Bool}.\mathbf{Bool}_y \to \Pi y_1 : \mathsf{Sh}\ T.[T]_{y_1} \to \Pi y_2 : \mathsf{Sh}.[T]_{y_2} \to [T]_{(\mathsf{if}\ y\ y_1\ y_2)}$. Let $\Delta = y : \mathbf{Bool}, \alpha_1 : \mathbf{Bool}_y, y_1 : \mathsf{Sh}\,T, \alpha_2 : [T]_{y_1}, y_2 : \mathsf{Sh}, \alpha_3 : [T]_{y_2}$. We just need to find a p such that $\Delta \vdash p : [T]_{(\mathsf{if}\ y\ y_1\ y_2)}$. We have $\vdash \mathsf{elimIf} : \Pi y : \mathbf{Bool}.\mathbf{Bool}_y \to F\ \mathsf{True} \to F\ \mathsf{False} \to F\ y$. Let $F = \lambda y.[T]_{(\mathsf{if}\ y\ y_1\ y_2)}$. We have $\Delta \vdash \mathsf{elimIf}\ y\ \alpha_1 : [T]_{(\mathsf{if}\ \mathsf{True}\ y_1\ y_2)} \to [T]_{(\mathsf{if}\ \mathsf{False}\ y_1\ y_2)} \to [T]_{(\mathsf{if}\ y\ y_1\ y_2)}$. By type equivalence, $\Delta \vdash \mathsf{elimIf}\ y\ \alpha_1 : [T]_{y_1} \to [T]_{y_2} \to [T]_{(\mathsf{if}\ y\ y_1\ y_2)}$. Thus $\Delta \vdash \mathsf{elimIf}\ y\ \alpha_1\ \alpha_2\ \alpha_3 : [T]_{(\mathsf{if}\ y\ y_1\ y_2)}$. So we are done.

- Case. $\vdash \mathsf{Cons}_T : T \to \mathbf{List}\ T \to \mathbf{List}\ T$.

  We need to find a p such that $\vdash p : \Pi y : \mathsf{Sh}\ T.[T]_y \to \Pi y_1 : \mathbf{List}\ (\mathsf{Sh}\ T).(\mathbf{List}\ T)_{y_1} \to (\mathbf{List}\ T)_{(\mathsf{Cons}\ y\ y_1)}$. Let $\Delta = y : \mathsf{Sh}\ T, \alpha_1 : [T]_y, y_1 : \mathbf{List}\ (\mathsf{Sh}\ T), \alpha_2 : (\mathbf{List}\ T)_{y_1}$. We just need to find a p such that $\Delta \vdash p : (\mathbf{List}\ T)_{(\mathsf{Cons}\ y\ y_1)}$. We know that $\vdash \mathsf{VCons}_T : \Pi y : \mathsf{Sh}\ T.[T]_y \to \Pi x : \mathbf{List}\ (\mathsf{Sh}\ T).(\mathbf{List}\ T)_x \to (\mathbf{List}\ T)_{(\mathsf{Cons}\ y\ x)}$. Thus $\Delta \vdash \mathsf{VCons}_T\ y\ \alpha_1\ y_1\ \alpha_2 : (\mathbf{List}\ T)_{(\mathsf{Cons}\ y\ y_1)}$.

- Case. $\vdash \mathsf{elimL}_{T,T'} : \mathbf{List}\ T \to T' \to (T \to \mathbf{List}\ T \to T' \to T') \to T'$

  We need to find a p such that $\vdash p : [\mathbf{List}\ T \to T' \to (T \to \mathbf{List}\ T \to T' \to T') \to T']_{\mathsf{elimL}_{(\mathsf{Sh}\ T),(\mathsf{Sh}\ T')}}$. We know that:

  $[\mathbf{List}\ T \to T' \to (T \to \mathbf{List}\ T \to T' \to T') \to T']_{\mathsf{elimL}_{(\mathsf{Sh}\ T),(\mathsf{Sh}\ T')}} =$

  $\Pi x : \mathbf{List}\ (\mathsf{Sh}\ T).(\mathbf{List}\ T)_x \to \Pi y_1 : \mathsf{Sh}\ T'.[T']_{y_1} \to \Pi y_2 : \mathsf{Sh}\ T \to \mathbf{List}\ (\mathsf{Sh}\ T) \to \mathsf{Sh}\ T' \to \mathsf{Sh}\ T'.$

  $(\Pi y_3 : \mathsf{Sh}\ T.[T]_{y_3} \to \Pi y_4 : \mathbf{List}\ (\mathsf{Sh}\ T).(\mathbf{List}\ T)_{y_4} \to \Pi y_5 : \mathsf{Sh}\ T'.[T']_{y_5} \to [T']_{y_2\ y_3\ y_4\ y_5})$

  $\to [T']_{(\mathsf{elimL}\ x\ y_1\ y_2)}$.

  Let $\Delta = x : \mathbf{List}\ (\mathsf{Sh}\ T), \alpha_1 : (\mathbf{List}\ T)_x, y_1 : \mathsf{Sh}\ T', \alpha_2 : [T']_{y_1}, y_2 : \mathsf{Sh}\ T \to \mathbf{List}\ (\mathsf{Sh}\ T) \to \mathsf{Sh}\ T' \to \mathsf{Sh}\ T', \alpha_3 : \Pi y_3 : \mathsf{Sh}\ T.[T]_{y_3} \to \Pi y_4 : \mathbf{List}\ (\mathsf{Sh}\ T).(\mathbf{List}\ T)_{y_4} \to \Pi y_5 : \mathsf{Sh}\ T'.[T']_{y_5} \to [T']_{y_2\ y_3\ y_4\ y_5}$.

  We just need to find a p such that $\Delta \vdash p : [T']_{(\mathsf{elimL}\ x\ y_1\ y_2)}$. We know that:

  $\vdash \mathsf{elimV}_{T,F}$:

  $\Pi x : \mathsf{List}\ (\mathsf{Sh}\ T).(\mathsf{List}\ T)_x \to F\ \mathsf{Nil}_{(\mathsf{Sh}\ T)} \to$

  $(\Pi z : \mathsf{Sh}\ T.T_z \to \Pi y : \mathbf{List}\ (\mathsf{Sh}\ T).(\mathbf{List}\ T)_y \to F\ y \to F\ (\mathsf{Cons}\ z\ y)) \to F\ x.$

  Let $F = \lambda x.[T']_{(\mathsf{elimL}\ x\ y_1\ y_2)}$. We have the following:

  $\Delta \vdash \mathsf{elimV}\ x\ \alpha_1$:

  $[T']_{(\mathsf{elimL}\ \mathsf{Nil}\ y_1\ y_2)} \to$

  $(\Pi z : \mathsf{Sh}\ T.T_z \to \Pi y : \mathbf{List}\ (\mathsf{Sh}\ T).(\mathbf{List}\ T)_y \to [T']_{(\mathsf{elimL}\ y\ y_1\ y_2)} \to [T']_{(\mathsf{elimL}\ (\mathsf{Cons}\ z\ y)\ y_1\ y_2)}) \to$

  $[T']_{(\mathsf{elimL}\ x\ y_1\ y_2)}$

  By type equivalence, we know that:

  $\Delta \vdash \mathsf{elimV}\ x\ \alpha_1$:

  $[T']_{y_1} \to (\Pi z : \mathsf{Sh}\,T.T_z \to \Pi y : \mathbf{List}\ (\mathsf{Sh}\,T).(\mathbf{List}\,T)_y \to [T']_{(\mathsf{elimL}\ y\ y_1\ y_2)} \to [T']_{(y_2\ z\ y\ (\mathsf{elimL}\ y\ y_1\ y_2))}) \to$

  $[T']_{(\mathsf{elimL}\ x\ y_1\ y_2)}$.

  By application, we have

  $\Delta \vdash \mathsf{elimV}\ x\ \alpha_1\ \alpha_2$:

  $(\Pi z : \mathsf{Sh}\ T.T_z \to \Pi y : \mathbf{List}\ (\mathsf{Sh}\ T).(\mathbf{List}\ T)_y \to [T']_{(\mathsf{elimL}\ y\ y_1\ y_2)} \to [T']_{(y_2\ z\ y\ (\mathsf{elimL}\ y\ y_1\ y_2))}) \to$

  $[T']_{(\mathsf{elimL}\ x\ y_1\ y_2)}$.

On the other hand, we know that

$\Delta, z : \mathsf{Sh}\ T, \beta : T_z, y : \textbf{List}\ (\mathsf{Sh}\ T), \beta_1 : (\textbf{List}\ T)_y \vdash \alpha_3\ z\ \beta : \Pi y_5 : \mathsf{Sh}\ T'.[T']_{y_5} \to [T']_{y_2\ z\ y\ y_5}$.

Thus we have

$\Delta, z : \mathsf{Sh}\ T, \beta : T_z, y : \textbf{List}\ (\mathsf{Sh}\ T), \beta_1 : (\textbf{List}\ T)_y \vdash \alpha_3\ z\ \beta\ (\mathsf{elimL}\ y\ y_1\ y_2) :$
$[T']_{(\mathsf{elimL}\ y\ y_1\ y_2)} \to [T']_{y_2\ z\ y\ (\mathsf{elimL}\ y\ y_1\ y_2)}$.

This implies:

$\Delta \vdash \lambda z.\lambda\beta.\lambda y.\lambda\beta_1.\alpha_3\ z\ \beta\ (\mathsf{elimL}\ y\ y_1\ y_2) :$
$\Pi z : \mathsf{Sh}\ T.T_z \to \Pi y : \textbf{List}\ (\mathsf{Sh}\ T).(\textbf{List}\ T)_y \to [T']_{(\mathsf{elimL}\ y\ y_1\ y_2)} \to [T']_{y_2\ z\ y\ (\mathsf{elimL}\ y\ y_1\ y_2)}$.

So $\Delta \vdash \mathsf{elimV}\ x\ \alpha_1\ \alpha_2\ (\lambda z.\lambda\beta.\lambda y.\lambda\beta_1.\alpha_3\ z\ \beta\ (\mathsf{elimL}\ y\ y_1\ y_2)) : [T']_{(\mathsf{elimL}\ x\ y_1\ y_2)}$. So we are done.

□

2.2.2 Forgetful Mapping from Sh (List T) to List T

Definition 25.

$|T_e| = T$
$|F \to F'| = |F| \to |F'|$
$|\Pi x : T.F| = |F|$
$|\lambda x.F| = |F|$
$|F\ e| = |F|$

Definition 26.

$|\alpha| = x_\alpha$
$|p\ p'| = |p|\ |p'|$
$|\lambda\alpha.p| = \lambda x_\alpha.|p|$
$|\lambda x.p| = |p|$
$|p\ e| = |p|$
$|\mathsf{pT}| = \mathsf{True}$
$|\mathsf{pF}| = \mathsf{False}$
$|\mathsf{pInit}| = \mathsf{init}$
$|\mathsf{pU}| = \mathsf{U}$
$|\mathsf{elimIf}_F| = \mathsf{if}_{|F|}$
$|\mathsf{VNil}_T| = \mathsf{Nil}_T$
$|\mathsf{VCons}_T| = \mathsf{Cons}_T$
$|\mathsf{elimV}_{T,F}| = \mathsf{elimL}_{T,|F|}$

Theorem 6. $|[T]_e| = T$

Definition 27.

$|x : T, \Delta| = |\Delta|$
$|\alpha : F, \Delta| = x_\alpha : |F|, |\Delta|$

Theorem 7. *If* $\Delta \vdash p : F$ *then* $|\Delta| \vdash |p| : |F|$.

Proof. By induction on the derivation of $\Delta \vdash p : F$. □

Exercise 5. *Can we define a head function for list in* Sh (**List T**) *and* **List T**? *What kind of extension will be required? What is the implications?*

Acknowlegement

This note is inspired by the discussion I have with Peter Selinger and Francisco Rios on adding dependent type to Proto-quipper. Maybe the particular systems I describe in this note is not a solution for all the problems, but I still think it is worthwhile to write it up and post it somewhere on the web.

References

[1] J.-Y. Girard, P. Taylor, and Y. Lafont. *Proofs and types*, volume 7. Cambridge University Press Cambridge, 1989.