Shape Realization

Peng Fu

August 21, 2017

1 Linear T

The following definition of linear **T** is following Alves et. al. [1].

Definition 1.
Types T ::= **Unit** | **Qubit** | **Nat** | **List** T | $T \multimap T'$ | $T \otimes T'$
Terms e, n ::= $x \mid \lambda x.e \mid e\ e' \mid \mathsf{Z} \mid \mathsf{S} \mid \mathsf{iterN}_T \mid \mathsf{Nil}_T \mid \mathsf{Cons}_T \mid \mathsf{iterL}_{T,T'} \mid \mathsf{U} \mid \mathsf{elimU}_T \mid \mathsf{Pair}_{T,T'} \mid \mathsf{iterP}_{T,T',T''} \mid \mathsf{g}$
Contexts Γ ::= $\cdot \mid x : T, \Gamma$

Note that **Qubit** is an abstract type that can be manipulated linearly via various of *unitary gates* g of type **Qubit** $\multimap$ **Qubit** and controlled gate **Qubit** $\otimes$ **Qubit** $\multimap$ **Qubit** $\otimes$ **Qubit**. For example, we can add $\mathsf{hadamard}$: **Qubit** $\multimap$ **Qubit**.

We currently do not add non-reversible gates such as measurement $\mathsf{measure}$: **Qubit** $\multimap$ **Bool** and init : **Bool** $\multimap$ **Qubit** to the system. One reason is that later we will define a shape functor on linear **T**, in that case, we would have $\mathsf{Sh}\ \mathsf{measure}$: **Unit** $\multimap$ **Bool**. But there are at least two possible linear functions that has this type, i.e $\lambda x.\mathsf{elimU}\ x\ \mathsf{True}$ and $\lambda x.\mathsf{elimU}\ x\ \mathsf{False}$. While for any unitary gate g : **Qubit** $\multimap$ **Qubit**, we have $\mathsf{Sh}\ \mathsf{g}$: **Unit** $\multimap$ **Unit**, which conceptually is a unique linear function. The shape functor actually can accomondate init, as $\mathsf{Sh}\ \mathsf{init}$ is also a unique function. But for simplicity, we ignore both init and $\mathsf{measure}$.

We treat tensor product as a kind of inductive datatype, and we will explain why in Section 6.

Definition 2 (Typing rules of **T**)**.**

$$\frac{}{x : T \Vdash x : T}\ \mathit{var} \qquad \frac{\Gamma, x : T \Vdash e : T'}{\Gamma \Vdash \lambda x.e : T \multimap T'}\ \mathit{lam} \qquad \frac{\Gamma_1 \Vdash e : T \multimap T' \quad \Gamma_2 \Vdash e' : T}{\Gamma_1, \Gamma_2 \Vdash e\ e' : T'}\ \mathit{app}$$

$$\frac{}{\Vdash \mathsf{Pair}_{T,T'} : T \multimap T' \multimap T \otimes T'} \qquad \frac{}{\Vdash \mathsf{iterP}_{T,T',T''} : T \otimes T' \multimap (T \multimap T' \multimap T'') \multimap T''}$$

$$\frac{}{\Vdash \mathsf{Z} : \mathbf{Nat}} \qquad \frac{}{\Vdash \mathsf{S} : \mathbf{Nat} \multimap \mathbf{Nat}} \qquad \frac{}{\Vdash \mathsf{iterN}_T : \mathbf{Nat} \multimap T \multimap (T \multimap T) \multimap T}$$

$$\frac{}{\Vdash \mathsf{U} : \mathbf{Unit}} \qquad \frac{}{\Vdash \mathsf{elemU}_T : \mathbf{Unit} \multimap T \multimap T}$$

$$\frac{}{\Vdash \mathsf{Nil}_T : \mathbf{List}\ T} \qquad \frac{}{\Vdash \mathsf{Cons}_T : T \multimap \mathbf{List}\ T \multimap \mathbf{List}\ T} \qquad \frac{}{\Vdash \mathsf{iterL}_{T,T'} : \mathbf{List}\ T \multimap T' \multimap (T \multimap T' \multimap T') \multimap T'}$$

Note that typing ensure *syntactic* linearity, i.e. variables are used exactly once.

Definition 3 (Closed Reduction)**.**
$(\lambda x.e)\ e' \rightsquigarrow [e'/x]e$, *where* $\mathrm{FV}(e') = \emptyset$.
$\mathsf{iterN}\ \mathsf{Z}\ e_1\ e_2 \rightsquigarrow e_1$, *where* $\mathrm{FV}(e_2) = \emptyset$.
$\mathsf{iterN}\ (\mathsf{S}\ e')\ e_1\ e_2 \rightsquigarrow e_2\ (\mathsf{iterN}\ e'\ e_1\ e_2)$, *where* $\mathrm{FV}(e_2) = \emptyset$.
$\mathsf{iterL}\ \mathsf{Nil}\ e_1\ e_2 \rightsquigarrow e_1$, *where* $\mathrm{FV}(e_2) = \emptyset$.
$\mathsf{iterL}\ (\mathsf{Cons}\ a\ e')\ e_1\ e_2 \rightsquigarrow e_2\ a\ (\mathsf{iterL}\ e'\ e_1\ e_2)$, *where* $\mathrm{FV}(e_2) = \emptyset$.
$\mathsf{iterP}\ (\mathsf{Pair}\ e_1\ e_2)\ f \rightsquigarrow f\ e_1\ e_2$.
$\mathsf{elimU}\ \mathsf{U}\ e \rightsquigarrow e$

Note that closed reduction ensure *operational* linearity, i.e. syntactic linearity is an invariant under the closed reduction.

We write $\langle e_1, e_2\rangle$ as a shorthand for $\mathsf{Pair}\ e_1\ e_2$, and let $\langle x, y\rangle = e$ in e' as a shorthand for $\mathsf{iterP}\ e\ (\lambda x.\lambda y.e')$.

Definition 4 (Syntactic Linearity)**.**

- *x is linear.*
- *$\lambda x.e$ is linear iff $x \in \mathrm{FV}(e)$ and e is linear.*
- *$e\ e'$ is linear iff $\mathrm{FV}(e) \cap \mathrm{FV}(e') = \emptyset$ and e, e' are linear.*

Theorem 1. *If $\Gamma \Vdash e : T$, then $\mathrm{dom}(\Gamma) = \mathrm{FV}(e)$ and e is linear.*

Theorem 2. *If $\Gamma \Vdash e : T$, then e is strongly normalizing.*

The following examples are from [2].

Definition 5 (Quantum Fourier Transformation)**.**

```
$\mathsf{length} : \textbf{List Qubit} \multimap \textbf{Nat} \otimes \textbf{List Qubit}$
$\mathsf{length}\ l = \mathsf{iterL}\ l\ \langle \mathsf{Z}, \mathsf{Nil}\rangle\ (\lambda a.\lambda r.\mathrm{let}\ \langle n, l'\rangle = r\ \mathrm{in}\ \langle \mathsf{S}\ n, \mathsf{Cons}\ a\ l'\rangle)$

$\mathsf{qft} : \textbf{List Qubit} \multimap \textbf{List Qubit}$
$\mathsf{qft}\ l = \mathsf{iterL}\ l\ \mathsf{Nil}$
        $(\lambda x.\lambda xs'.\mathrm{let}\ \langle n, xs''\rangle = \mathsf{length}\ xs'\ \mathrm{in}$
          $\mathrm{let}\ \langle xs'', a\rangle = \mathsf{rotate}\ x\ n\ xs''\ \mathrm{in}\ \mathsf{Cons}\ (\mathsf{hadamard}\ a)\ xs'')$

$\mathsf{rotate} : \textbf{Qubit} \multimap \textbf{Nat} \multimap \textbf{List Qubit} \multimap \textbf{List Qubit} \otimes \textbf{Qubit}$
$\mathsf{rotate}\ c\ n\ l = \mathsf{iterL}\ l\ \langle \mathsf{Nil}, c\rangle$
        $(\lambda q.\lambda r.\mathrm{let}\ \langle qs', c'\rangle = r\ \mathrm{in}$
          $\mathrm{let}\ \langle n', qs''\rangle = \mathsf{length}\ qs'\ \mathrm{in}$
            $\mathrm{let}\ \langle q', c''\rangle = \mathsf{control}\ \langle \mathsf{rGate}\ (\mathsf{minus}\ (\mathsf{S}\ n)\ n')\ q, c'\rangle\ \mathrm{in}\ \langle \mathsf{Cons}\ q'\ qs'', c''\rangle)$
```

Note that rGate have the type $\textbf{Nat} \multimap \textbf{Qubit} \multimap \textbf{Qubit}$. Internally, it means $\mathsf{rGate}\ n$ is a shorthand to denote a gate of type $\textbf{Qubit} \multimap \textbf{Qubit}$ for a concrete n.

2 Shape Functor

We underline the cases where the shape functor makes a difference.

Definition 6 (Shape functor on Types)**.**

$\mathsf{Sh}\ \textbf{Unit} = \textbf{Unit}$
$\mathsf{Sh}\ \textbf{Nat} = \textbf{Nat}$
$\underline{\mathsf{Sh}\ \textbf{Qubit} = \textbf{Unit}}$
$\mathsf{Sh}\ (\textbf{List}\ T) = \textbf{List}\ (\mathsf{Sh}\ T)$
$\mathsf{Sh}\ (T \multimap T') = \mathsf{Sh}\ T \multimap \mathsf{Sh}\ T'$
$\mathsf{Sh}\ (T \otimes T') = \mathsf{Sh}\ T \otimes \mathsf{Sh}\ T'$

Definition 7.

$\mathsf{erase} : \textbf{Nat} \multimap (T \multimap T)$
$\mathsf{erase}\ n = \mathsf{iterN}\ n\ (\lambda x.x)\ (\lambda x.x)$

Definition 8 (Shape functor on Terms)**.**

$\mathsf{Sh}\ x = x$
$\underline{\mathsf{Sh}\ \mathsf{g} = \lambda x.x}$
$\underline{\mathsf{Sh}\ \mathsf{control} = \lambda x.x}$
$\mathsf{Sh}\ (\lambda x.e) = \lambda x.\mathsf{Sh}\ e$
$\mathsf{Sh}\ (e\ e') = (\mathsf{Sh}\ e)\ (\mathsf{Sh}\ e')$
$\mathsf{Sh}\ \mathsf{Z} = \mathsf{Z}$
$\mathsf{Sh}\ \mathsf{S} = \mathsf{S}$
$\mathsf{Sh}\ \mathsf{iterN}_T = \mathsf{iterN}_{(\mathsf{Sh}\ T)}$
$\mathsf{Sh}\ \mathsf{Nil}_T = \mathsf{Nil}_{(\mathsf{Sh}\ T)}$
$\mathsf{Sh}\ \mathsf{Cons}_T = \mathsf{Cons}_{(\mathsf{Sh}\ T)}$
$\mathsf{Sh}\ \mathsf{iterL}_{T,T'} = \mathsf{iterL}_{(\mathsf{Sh}\ T),(\mathsf{Sh}\ T')}$
$\mathsf{Sh}\ \mathsf{U} = \mathsf{U}$
$\mathsf{Sh}\ \mathsf{elimU}_T = \mathsf{elimU}_{(\mathsf{Sh}\ T)}$
$\mathsf{Sh}\ \mathsf{Pair}_{T,T'} = \mathsf{Pair}_{(\mathsf{Sh}\ T),(\mathsf{Sh}\ T')}$
$\mathsf{Sh}\ \mathsf{iterP}_{T,T',T''} = \mathsf{iterP}_{(\mathsf{Sh}\ T),(\mathsf{Sh}\ T'),(\mathsf{Sh}\ T'')}$
$\mathsf{Sh}\ \mathsf{rGate} = \lambda n.\mathsf{erase}\ n\ (\lambda x.x)$

Let $\mathsf{Sh}(\Gamma)$ denotes applying Sh to all the types in Γ.

Theorem 3. *If* $\Gamma \Vdash e : T$, *then* $\mathsf{Sh}(\Gamma) \Vdash \mathsf{Sh}\ e : \mathsf{Sh}\ T$.

Note that when we apply shape functor to linear **T**, we essentially obtain a copy of **T**.
Now let us apply the shape functor to the Fourier transformation function.

Definition 9.

$(\mathsf{Sh}\ \mathsf{length}) : \mathbf{List\ Unit} \multimap \mathbf{Nat} \otimes \mathbf{List\ Unit}$
$(\mathsf{Sh}\ \mathsf{length})\ l = \mathsf{iterL}\ l\ \langle \mathsf{Z}, \mathsf{Nil}\rangle\ (\lambda a.\lambda r.\text{let}\ \langle n, l'\rangle = r\ \text{in}\ \langle \mathsf{S}\ n, \mathsf{Cons}\ a\ l'\rangle)$

$(\mathsf{Sh}\ \mathsf{qft}) : \mathbf{List\ Unit} \multimap \mathbf{List\ Unit}$
$(\mathsf{Sh}\ \mathsf{qft})\ l = \mathsf{iterL}\ l\ \mathsf{Nil}$
$\quad\quad(\lambda x.\lambda xs'.\text{let}\ \langle n, xs''\rangle = (\mathsf{Sh}\ \mathsf{length})\ xs'\ \text{in}$
$\quad\quad\quad\text{let}\ \langle xs'', a\rangle = (\mathsf{Sh}\ \mathsf{rotate})\ x\ n\ xs''\ \text{in}\ \mathsf{Cons}\ a\ xs'')$

$(\mathsf{Sh}\ \mathsf{rotate}) : \mathbf{Unit} \multimap \mathbf{Nat} \multimap \mathbf{List\ Unit} \multimap \mathbf{List\ Unit} \otimes \mathbf{Unit}$
$(\mathsf{Sh}\ \mathsf{rotate})\ c\ n\ l = \mathsf{iterL}\ l\ \langle \mathsf{Nil}, c\rangle$
$\quad\quad(\lambda q.\lambda r.\text{let}\ \langle qs', c'\rangle = r\ \text{in}$
$\quad\quad\quad\text{let}\ \langle n', qs''\rangle = (\mathsf{Sh}\ \mathsf{length})\ qs'\ \text{in}$
$\quad\quad\quad\quad\text{let}\ \langle q', c''\rangle = \langle \mathsf{erase}\ (\mathsf{minus}\ (\mathsf{S}\ n)\ n')\ (\lambda x.x)\ q, c'\rangle\ \text{in}\ \langle \mathsf{Cons}\ q'\ qs'', c''\rangle)$

Note that we use the definition of Sh hadamard, Sh control *and* Sh rGate *in the above definition. We can see that* Sh qft *is an inefficient way to write identity function.*

3 Shape Realization

Definition 10. *We extend the Definition 1 with the followings.*

Types $T\ ::= ... \mid \mathbf{Qubit}_e \mid \mathbf{Nat}_e \mid (\mathbf{List}\ T)_e \mid \mathbf{Unit}_e \mid (T \otimes T')_e \mid \forall x : T.T' \mid \lambda x.T \mid T\ e$
Terms $e\ ::= ... \mid \mathsf{SNil}_T \mid \mathsf{SCons}_T \mid \mathsf{sIterL}_{T,T'} \mid \mathsf{SU} \mid \mathsf{SZ} \mid \mathsf{SS} \mid \mathsf{sIterN}_T \mid \mathsf{sElimU} \mid \mathsf{SPair}_{T,T'} \mid \mathsf{sIterP}_{T,T',T''}$
Kinds $K\ ::= * \mid T \to K$
Contexts $\Gamma\ ::= \cdot \mid x : T, \Gamma$

We write $T \in \mathbf{T}$ to mean the types defined in Definition 1. We use $\Gamma \Vdash e : T$ to mean it is a judgement derived by the rules in Definition 2.

Definition 11 (Kinding)**.**

$$\frac{\Gamma, x : T \vdash T' : K \quad T \in \mathbf{T}}{\Gamma \vdash \lambda x.T' : T \to K} \qquad \frac{\Gamma \vdash T' : T \to K \quad \Gamma' \Vdash e : T}{\Gamma \cup \Gamma' \vdash T'\ e : K} \qquad \frac{\Gamma \Vdash e : \mathbf{List}\ (\mathsf{Sh}\ T)}{\Gamma \vdash (\mathbf{List}\ T)_e : *} \qquad \frac{\Gamma \Vdash e : \mathbf{Nat}}{\Gamma \vdash \mathbf{Nat}_e : *}$$

$$\frac{\Gamma, x : T \vdash T' : * \quad T \in \mathbf{T}}{\Gamma \vdash \forall x : T.T' : *} \qquad \frac{\Gamma \Vdash e : (\mathsf{Sh}\ T) \otimes (\mathsf{Sh}\ T')}{\Gamma \vdash (T \otimes T')_e : *} \qquad \frac{T \in \mathbf{T}}{\Gamma \vdash T : *} \qquad \frac{\Gamma \Vdash e : \mathbf{Unit}}{\Gamma \vdash \mathbf{Qubit}_e : *}$$

$$\frac{\Gamma \vdash T : * \quad \Gamma \vdash T' : *}{\Gamma \vdash T \multimap T' : *}$$

Note that kinding prevents a self-dependent case like $x : \mathbf{Nat}_{\mathsf{sIterN}\ x\ \mathsf{Z}\ (\lambda y.\mathsf{S}\ y)} \vdash x : \mathbf{Nat}_{\mathsf{sIterN}\ x\ \mathsf{Z}\ (\lambda y.\mathsf{S}\ y)}$ with $x : \mathbf{Nat}_{\mathsf{sIterN}\ x\ \mathsf{Z}\ (\lambda y.\mathsf{S}\ y)} \vdash \mathsf{sIterN}\ x\ \mathsf{Z}\ (\lambda y.\mathsf{S}\ y) : \mathbf{Nat}$, where x occurs at its types(which happens in self type).

We write $e = e'$ if there exist a e'' such that $e \rightsquigarrow^* e''$ and $e' \rightsquigarrow^* e''$.

Definition 12 (Type equivalence)**.**
$(\lambda x.T)\ e \rightsquigarrow [e/x]T$
$[e/x]T \rightsquigarrow [e'/x]T$ *if* $e = e'$.

We write $T = T'$ if there exist a T'' such that $T \rightsquigarrow^* T''$ and $T' \rightsquigarrow^* T''$. All kindable types are strongly normalizable, so we only need to work with normal form of types and they enjoy the usual inversion property, i.e. If $\Gamma \vdash T' : T \to K$, then the normal form of T' must be of the form $\lambda x.T''$ such that $\Gamma, x : T \vdash T'' : K$ (we use this property in Lemma 1).

Definition 13 (Well-formed context)**.**

$$\frac{}{\cdot \vdash \mathsf{wf}} \qquad \frac{\Gamma \vdash \mathsf{wf} \quad \Gamma \vdash T : *}{\Gamma, x : T \vdash \mathsf{wf}}$$

Lemma 1. *If* $\Gamma \vdash \mathsf{wf}$*, then for any* $x : T \in \Gamma$, $\mathrm{FV}(T) \subseteq \mathrm{dom}(\Gamma)$ *and* $x \notin \mathrm{FV}(T)$.

Proof. We prove this by induction on $\Gamma \vdash \mathsf{wf}$. The base case is trivial as Γ is empty. We now consider the step case. Suppose $\Gamma \vdash \mathsf{wf}$, $\Gamma, x : T \vdash \mathsf{wf}$ and $\Gamma \vdash T : *$. We just need to show that $\mathrm{FV}(T) \subseteq \mathrm{dom}(\Gamma)$, and this will imply that $x \notin \mathrm{FV}(T)$. We prove this by induction on the length of derivation of $\Gamma \vdash T : *$. The base cases are proved using Theorem 1. The only nontrivial case is the following.

$$\frac{\Gamma \vdash T' : T \to * \quad \Gamma' \Vdash e : T}{\Gamma \cup \Gamma' \vdash T'\ e : *}$$

Since $\Gamma \vdash T' : T \to *$, it must be the case that $T' \equiv \lambda z.T''$ such that $\Gamma, z : T \vdash T'' : *$. By IH (as in this case the length of derivation for $\Gamma, z : T \vdash T'' : *$ is smaller than $\Gamma \cup \Gamma' \vdash T'\ e : *$), we have $\mathrm{FV}(T'') \subseteq \mathrm{dom}(\Gamma, z : T)$. Thus $\mathrm{FV}(T') = \mathrm{FV}(\lambda z.T'') \subseteq \mathrm{dom}(\Gamma)$. Thus $\mathrm{FV}(T'\ e) \subseteq \mathrm{dom}(\Gamma \cup \Gamma')$.

□

Definition 14. *Let* $T \in \mathbf{T}$*, we define* $[T]_e$ *as the following.*

$[\mathbf{Unit}]_e = \mathbf{Unit}_e$
$[\mathbf{Nat}]_e = \mathbf{Nat}_e$
$[\mathbf{Qubit}]_e = \mathbf{Qubit}_e$
$[\mathbf{List}\ T]_e = (\mathbf{List}\ T)_e$
$[T \otimes T']_e = (T \otimes T')_e$
$[T \multimap T']_e = \forall y : \mathsf{Sh}\ T.[T]_y \multimap [T']_{(e\ y)}$*, where* y *is fresh.*

We use FV as a function that gives a set of free variables. We use $(\Gamma_1 \cap \Gamma_2)\#\mathrm{FV}(e\ e')$ as a short hand for $(\mathrm{dom}(\Gamma_1) \cap \mathrm{dom}(\Gamma_2))\#\mathrm{FV}(e\ e')$. The following are the typing for the shape indexed types.

Definition 15 (Typing rule of $\mathbf{T}_{\mathsf{Sh}}$)**.**

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

$$\frac{\Gamma_1 \vdash e : T \multimap T' \quad \Gamma_2 \vdash e' : T \quad (\Gamma_1 \cap \Gamma_2)\#\mathrm{FV}(e\ e')}{\Gamma_1 \cup \Gamma_2 \vdash e\ e' : T'}\ \mathit{app} \qquad \frac{\Gamma, x : T \vdash e : T' \quad x \notin \mathrm{FV}(e) \quad T \in \mathbf{T}}{\Gamma \vdash e : \forall x : T.T'}$$

$$\frac{\Gamma_1 \vdash e : \forall x : T.T' \quad \Gamma_2 \Vdash e' : T}{\Gamma_1 \cup \Gamma_2 \vdash e : [e'/x]T'} \qquad \frac{\Gamma \vdash e : T \quad T = T'}{\Gamma \vdash e : T'}$$

$$\frac{}{\vdash \mathsf{SU} : \mathbf{Unit}_{\mathsf{U}}} \qquad \frac{}{\vdash \mathsf{SZ} : \mathbf{Nat}_{\mathsf{Z}}}$$

$$\frac{}{\vdash \mathsf{SS} : \forall x : \mathbf{Nat}.\mathbf{Nat}_x \multimap \mathbf{Nat}_{(\mathsf{S}\ x)}}$$

$$\frac{\Gamma \vdash T : \mathbf{Nat} \to *}{\Gamma \vdash \mathsf{sIterN}_T : \forall x : \mathbf{Nat}.\mathbf{Nat}_x \multimap T\ \mathsf{Z} \multimap (\forall y : \mathbf{Nat}.T\ y \multimap T\ (\mathsf{S}\ y)) \multimap T\ x}$$

$$\frac{\Gamma \vdash T : \mathbf{Unit} \to *}{\Gamma \vdash \mathsf{sElimU}_T : \forall x : \mathbf{Unit}.\mathbf{Unit}_x \multimap T\ \mathsf{U} \multimap T\ x}$$

$$\frac{}{\vdash \mathsf{SPair}_{T,T'} : \forall x : \mathsf{Sh}\ T.T \multimap \forall y : \mathsf{Sh}\ T'.T' \multimap (T \otimes T')_{\mathsf{Pair}\ x\ y}}$$

$$\frac{\Gamma \vdash T'' : \mathsf{Sh}\ T \otimes \mathsf{Sh}\ T' \to *}{\Gamma \vdash \mathsf{sIterP}_{T,T',T''} : \forall x : \mathsf{Sh}\ T \otimes \mathsf{Sh}\ T'.(T \otimes T')_x \multimap (\forall y : \mathsf{Sh}\ T.[T]_y \multimap \forall z : \mathsf{Sh}\ T'.[T']_z \multimap T''\ (\mathsf{Pair}\ y\ z)) \multimap T''\ x}$$

$$\frac{}{\vdash \mathsf{SNil}_T : (\mathbf{List}\ T)_{\mathsf{Nil}_{(\mathsf{Sh}\ T)}}}$$

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

$$\frac{\Gamma \vdash T' : \mathbf{List}\ (\mathsf{Sh}\ T') \to *}{\Gamma \vdash \mathsf{sIterL}_{T,T'} : A}$$

Note that $A = \forall x : \mathsf{List}\ (\mathsf{Sh}\ T).(\mathsf{List}\ T)_x \multimap T'\ \mathsf{Nil}_{(\mathsf{Sh}\ T)} \multimap$
$(\forall z : \mathsf{Sh}\ T.[T]_z \multimap \forall y : \mathbf{List}\ (\mathsf{Sh}\ T).T'\ y \multimap T'\ (\mathsf{Cons}\ z\ y)) \multimap T'\ x$

For each unitary gate $\mathsf{g} : \mathbf{Qubit} \multimap \mathbf{Qubit}$, we define a new gate $\mathsf{Sg} : \forall x : \mathbf{Unit}.\mathbf{Qubit}_x \multimap \mathbf{Qubit}_x$. In the implementation, Sg will be the same as g.

Note that the reduction rules for the shape indexed iterator is the same as its monomorphic version, so we elide them here.

Lemma 2. *If* $\Gamma \vdash e : T$ *and* $\Gamma \vdash \mathsf{wf}$*, then* $\Gamma \vdash T : *$.

Theorem 4 (Coherence)**.** *If* $\Gamma \vdash e : T$ *and* $\Gamma \vdash \mathsf{wf}$*, then* e *is linear and* $\mathrm{FV}(e) \subseteq \mathrm{dom}(\Gamma)$.

4 Annotation Mapping

In this section, we will show how to automatically transform programs in $\mathbf{T}$ to the programs in $\mathbf{T}_{\mathsf{Sh}}$.

Definition 16.
$[\cdot] = \cdot$
$[z : T, \Gamma] = y : \mathsf{Sh}\ T, z : [T]_y, [\Gamma]$, *where y is a fresh variable.*

Note that the above definition exhibit a map from $z : T$ to a dependent pair $y : \mathsf{Sh}\ T, z : [T]_y$. We will show another map from $y : \mathsf{Sh}\ T, z : [T]_y$ to $z : T$ 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 $\Gamma \Vdash e : T$. Here we show a few selected cases.

- Case. $\Vdash \mathsf{iterP}_{T,T',T''} : T \otimes T' \multimap (T \multimap T' \multimap T'') \multimap T''$

  We need to find a p such that $\vdash p : \forall x : \mathsf{Sh}\ T \otimes \mathsf{Sh}\ T'.(T \otimes T')_x \multimap \forall y : \mathsf{Sh}\ T \multimap \mathsf{Sh}\ T' \multimap \mathsf{Sh}\ T''.(\forall y_1 : \mathsf{Sh}\ T.[T]_{y_1} \multimap \forall y_2 : \mathsf{Sh}\ T'.[T']_{y_2} \multimap [T'']_{y\ y_1\ y_2}) \multimap [T'']_{\mathsf{iterP\ x\ y}}$. This is equivalence to finding a p such that $x : \mathsf{Sh}\ T \otimes \mathsf{Sh}\ T', z_1 : (T \otimes T')_x, y : \mathsf{Sh}\ T \multimap \mathsf{Sh}\ T' \multimap \mathsf{Sh}\ T'', z_2 : \forall y_1 : \mathsf{Sh}\ T.[T]_{y_1} \multimap \forall y_2 : \mathsf{Sh}\ T'.[T']_{y_2} \multimap [T'']_{y\ y_1\ y_2} \vdash p : [T'']_{\mathsf{iterP\ x\ y}}$.

  Let $T'' = \lambda x.[T'']_{\mathsf{iterP}\ x\ y}$. So $y : \mathsf{Sh}\ T \multimap \mathsf{Sh}\ T' \multimap \mathsf{Sh}\ T'' \vdash \mathsf{sIterP} : \forall x : \mathsf{Sh}\ T \otimes \mathsf{Sh}\ T'.(T \otimes T')_x \multimap (\forall y' : \mathsf{Sh}\ T.[T]_{y'} \multimap \forall z : \mathsf{Sh}\ T'.[T']_z \multimap [T'']_{\mathsf{iterP}\ (\mathsf{Pair}\ y'\ z)\ y}) \multimap [T'']_{\mathsf{iterP}\ x\ y}$. By type equivalence, we know that $y : \mathsf{Sh}\ T \multimap \mathsf{Sh}\ T' \multimap \mathsf{Sh}\ T'' \vdash \mathsf{sIterP} : \forall x : \mathsf{Sh}\ T \otimes \mathsf{Sh}\ T'.(T \otimes T')_x \multimap (\forall y' : \mathsf{Sh}\ T.[T]_{y'} \multimap \forall z : \mathsf{Sh}\ T'.[T']_z \multimap [T'']_{y\ y'\ z}) \multimap [T'']_{\mathsf{iterP}\ x\ y}$.

  This imples $x : \mathsf{Sh}\ T \otimes \mathsf{Sh}\ T', z_1 : (T \otimes T')_x, y : \mathsf{Sh}\ T \multimap \mathsf{Sh}\ T' \multimap \mathsf{Sh}\ T'', z_2 : \forall y_1 : \mathsf{Sh}\ T.[T]_{y_1} \multimap \forall y_2 : \mathsf{Sh}\ T'.[T']_{y_2} \multimap [T'']_{y\ y_1\ y_2} \vdash \mathsf{sIterP}\ [x]\ z_1\ z_2 : [T'']_{\mathsf{iterP}\ x\ y}$.

- Case. $\Vdash \mathsf{iterL}_{T,T'} : \textbf{List}\ T \multimap T' \multimap (T \multimap T' \multimap T') \multimap T'$

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

  $[\textbf{List}\ T \multimap T' \multimap (T \multimap T' \multimap T') \multimap T']_{\mathsf{iterL}_{(\mathsf{Sh}\ T),(\mathsf{Sh}\ T')}} =$

  $\forall x : \textbf{List}\ (\mathsf{Sh}\ T).(\textbf{List}\ T)_x \multimap \forall y_1 : \mathsf{Sh}\ T'.[T']_{y_1} \multimap \forall y_2 : \mathsf{Sh}\ T \multimap \mathsf{Sh}\ T' \multimap \mathsf{Sh}\ T'.$
  $(\forall y_3 : \mathsf{Sh}\ T.[T]_{y_3} \multimap \forall y_4 : \mathsf{Sh}\ T'.[T']_{y_4} \multimap [T']_{y_2\ y_3\ y_4}) \multimap [T']_{(\mathsf{iterL}\ x\ y_1\ y_2)}$.

  Let $\Delta = x : \textbf{List}\ (\mathsf{Sh}\ T), y_1 : \mathsf{Sh}\ T', y_2 : \mathsf{Sh}\ T \multimap \mathsf{Sh}\ T' \multimap \mathsf{Sh}\ T', \Gamma = z_1 : (\textbf{List}\ T)_x, z_2 : [T']_{y_1}, z_3 : \forall y_3 : \mathsf{Sh}\ T.[T]_{y_3} \multimap \forall y_4 : \mathsf{Sh}\ T'.[T']_{y_4} \multimap [T']_{y_2\ y_3\ y_4}$.

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

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

  $\forall x : \mathsf{List}\ (\mathsf{Sh}\ T).(\mathsf{List}\ T)_x \multimap F\ \mathsf{Nil}_{(\mathsf{Sh}\ T)} \multimap$
  $(\forall z : \mathsf{Sh}\ T.T_z \multimap \forall y : \textbf{List}\ (\mathsf{Sh}\ T).F\ y \multimap F\ (\mathsf{Cons}\ z\ y)) \multimap F\ x.$

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

  $x : \textbf{List}\ (\mathsf{Sh}\ T), z_1 : (\textbf{List}\ T)_x, y_1 : \mathsf{Sh}\ T', y_2 : \mathsf{Sh}\ T \multimap \mathsf{Sh}\ T' \multimap \mathsf{Sh}\ T' \vdash \mathsf{sIterL}\ x\ z_1$:
  $[T']_{(\mathsf{iterL\ Nil}\ y_1\ y_2)} \multimap$
  $(\forall z : \mathsf{Sh}\ T.T_z \multimap \forall y : \textbf{List}\ (\mathsf{Sh}\ T).[T']_{(\mathsf{iterL}\ y\ y_1\ y_2)} \multimap [T']_{(\mathsf{iterL}\ (\mathsf{Cons}\ z\ y)\ y_1\ y_2)}) \multimap [T']_{(\mathsf{iterL}\ x\ y_1\ y_2)}$

  By type equivalence, we know that:

  $x : \textbf{List}\ (\mathsf{Sh}\ T), z_1 : (\textbf{List}\ T)_x, y_1 : \mathsf{Sh}\ T', y_2 : \mathsf{Sh}\ T \multimap \mathsf{Sh}\ T' \multimap \mathsf{Sh}\ T' \vdash \mathsf{sIterL}\ [x]\ z_1$:
  $[T']_{y_1} \multimap (\forall z : \mathsf{Sh}\ T.T_z \multimap \forall y : \textbf{List}\ (\mathsf{Sh}\ T).[T']_{(\mathsf{iterL}\ y\ y_1\ y_2)} \multimap [T']_{(y_2\ z\ (\mathsf{iterL}\ y\ y_1\ y_2))}) \multimap$
  $[T']_{(\mathsf{iterL}\ x\ y_1\ y_2)}$.

  By application, we have

  $x : \textbf{List}\ (\mathsf{Sh}\ T), z_1 : (\textbf{List}\ T)_x, y_1 : \mathsf{Sh}\ T', y_2 : \mathsf{Sh}\ T \multimap \mathsf{Sh}\ T' \multimap \mathsf{Sh}\ T', z_2 : [T']_{y_1} \vdash \mathsf{sIterL}\ [x]\ z_1\ z_2$:

$(\forall z : \mathsf{Sh}\ T.T_z \multimap \forall y : \mathbf{List}\ (\mathsf{Sh}\ T).[T']_{(\mathsf{iterL}\ y\ y_1\ y_2)} \multimap [T']_{(y_2\ z\ (\mathsf{iterL}\ y\ y_1\ y_2))}) \multimap$
$[T']_{(\mathsf{iterL}\ x\ y_1\ y_2)}$.

On the other hand, we know that

$y_2 : \mathsf{Sh}\ T \multimap \mathsf{Sh}\ T' \multimap \mathsf{Sh}\ T', z_3 : \forall y_3 : \mathsf{Sh}\ T.[T]_{y_3} \multimap \forall y_4 : \mathsf{Sh}\ T'.[T']_{y_4} \multimap [T']_{y_2\ y_3\ y_4}, z : \mathsf{Sh}\ T, z_4 : [T]_z \vdash z_3\ [z]\ z_4 : \forall y_4 : \mathsf{Sh}\ T'.[T']_{y_4} \multimap [T']_{y_2\ z\ y_4}$.

Thus we have

$y_1 : \mathsf{Sh}\ T', y_2 : \mathsf{Sh}\ T \multimap \mathsf{Sh}\ T' \multimap \mathsf{Sh}\ T', z_3 : \forall y_3 : \mathsf{Sh}\ T.[T]_{y_3} \multimap \forall y_4 : \mathsf{Sh}\ T'.[T']_{y_4} \multimap [T']_{y_2\ y_3\ y_4}, z : \mathsf{Sh}\ T, z_4 : [T]_z, y : \mathbf{List}\ (\mathsf{Sh}\ T) \vdash z_3\ [z]\ z_4\ [\mathsf{iterL}\ y\ y_1\ y_2] : [T']_{(\mathsf{iterL}\ y\ y_1\ y_2)} \multimap [T']_{y_2\ z\ (\mathsf{iterL}\ y\ y_1\ y_2)}$.

This implies:

$y_1 : \mathsf{Sh}\ T', y_2 : \mathsf{Sh}\ T \multimap \mathsf{Sh}\ T' \multimap \mathsf{Sh}\ T', z_3 : \forall y_3 : \mathsf{Sh}\ T.[T]_{y_3} \multimap \forall y_4 : \mathsf{Sh}\ T'.[T']_{y_4} \multimap [T']_{y_2\ y_3\ y_4} \vdash [\lambda z.]\lambda z_4.[\lambda y.]z_3\ [z]\ z_4\ [\mathsf{iterL}\ y\ y_1\ y_2] :$

$\forall z : \mathsf{Sh}\ T.T_z \multimap \forall y : \mathbf{List}\ (\mathsf{Sh}\ T).[T']_{(\mathsf{iterL}\ y\ y_1\ y_2)} \multimap [T']_{y_2\ z\ (\mathsf{iterL}\ y\ y_1\ y_2)}$.

So $\Gamma, \Delta \vdash \mathsf{sIterL}\ [x]\ z_1\ z_2\ ([\lambda z.]\lambda z_4.[\lambda y.]z_3\ [z]\ z_4\ [\mathsf{iterL}\ y\ y_1\ y_2]) : [T']_{(\mathsf{iterL}\ x\ y_1\ y_2)}$. So we are done.

□

5 Forgetful Mapping

Definition 17.
$|(\mathbf{List}\ T)_e| = \mathbf{List}\ T$
$|(\mathbf{Qubit})_e| = \mathbf{Qubit}$
$|(\mathbf{Nat})_e| = \mathbf{Nat}$
$|(\mathbf{Unit})_e| = \mathbf{Unit}$
$|(T \otimes T')_e| = T \otimes T'$
$|T \multimap T'| = |T| \multimap |T'|$
$|\forall x : T.T'| = |T'|$
$|\lambda x.T| = |T|$
$|T\ e| = |T|$

Definition 18. $|\mathsf{SNil}_T| = \mathsf{Nil}_T$
$|\mathsf{SCons}_T| = \mathsf{Cons}_T$
$|\mathsf{sIterL}_{T,T'}| = \mathsf{iterL}_{T,T'}$
$|\mathsf{SU}| = \mathsf{U}$
$|\mathsf{SZ}| = \mathsf{Z}$
$|\mathsf{SS}| = \mathsf{S}$
$|\mathsf{sIterN}_T| = \mathsf{iterN}_T$
$|\mathsf{sElimU}| = \mathsf{elimU}$
$|\mathsf{SPair}_{T,T'}| = \mathsf{Pair}_{T,T'}$
$|\mathsf{sIterP}_{T,T',T''}| = \mathsf{iterP}_{T,T',T''}$
For all the other cases, $|e| = e$

Definition 19. $|.|_{\mathrm{FV}(e)} = .$
$|x : T, \Gamma|_{\mathrm{FV}(e)} = |\Gamma|_{\mathrm{FV}(e)}$ *if* $x \notin \mathrm{FV}(e)$.
$|x : T, \Gamma|_{\mathrm{FV}(e)} = x : |T|, |\Gamma|_{\mathrm{FV}(e)}$ *if* $x \in \mathrm{FV}(e)$.

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

Theorem 7. *If* $\Gamma \vdash e : T$ *then* $|\Gamma|_{\mathrm{FV}(e)} \Vdash e : |T|$.

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

6 Why tensor product is an inductive type?

Suppose the tensor product is not an inductive type, then we would have to define $[T \otimes T']_e = [T]_e \otimes [T']_e$, this does not make sense. We would need to somehow specify the normal form of e in this case, i.e. $[T \otimes T']_{\langle e, e' \rangle} = [T]_e \otimes [T']_{e'}$. But this means we can not have a general annotation theorem such as Theorem 5 on non-normal form.

7 Discussion

We can see that the so-call shape index program is essentially **T** programs, it does not really extend the program we can write, for example, there is no way to use the shape index type to define the safe head function for list of qubits, i.e. a function of type $\forall n : \mathbf{List\ Unit}.(\mathbf{List\ Qubit})_{\mathsf{Cons\ U}\ n} \multimap (\mathbf{Qubit} \otimes \mathbf{List\ Qubit})_{\mathsf{Pair\ U}\ n}$ is not definable. This is becase the erasure of this type is: $\mathbf{List\ Qubit} \multimap \mathbf{Qubit} \otimes \mathbf{List\ Qubit}$, and without initialization, there is no way to define such function in linear **T**(no total function in linear **T** can have the type $\mathbf{List\ Qubit} \multimap \mathbf{Qubit} \otimes \mathbf{List\ Qubit}$).

References

[1] Sandra Alves, Maribel Fernández, Mário Florido, and Ian Mackie. Gödels system T revisited. *Theoretical Computer Science*, 411(11-13):1484–1500, 2010.

[2] Alexander S Green, Peter LeFanu Lumsdaine, Neil J Ross, Peter Selinger, and Benoît Valiron. An introduction to quantum programming in quipper. In *International Conference on Reversible Computation*, pages 110–124. Springer, 2013.