Categorical Semantics and Adjoint Proto-Quipper

Max Gross

McGill University

COMEPLS, Sept. 17th

Outline

Background on Proto-Quipper

- Quipper is a functional, circuit describing programming language

Background on Proto-Quipper

- Quipper is a functional, circuit describing programming language
    - Composed of two levels:

Background on Proto-Quipper

- Quipper is a functional, circuit describing programming language
  - Composed of two levels:
  - First, building up circuits by applying gates

Background on Proto-Quipper

- Quipper is a functional, circuit describing programming language
  - Composed of two levels:
  - First, building up circuits by applying gates
  - Second, treating circuits as data.

Background on Proto-Quipper

- Quipper is a functional, circuit describing programming language
  - Composed of two levels:
  - First, building up circuits by applying gates
  - Second, treating circuits as data.
- Embedded within Haskell, thus has no type safety or formal semantics

Background on Proto-Quipper

- Quipper is a functional, circuit describing programming language
  - Composed of two levels:
  - First, building up circuits by applying gates
  - Second, treating circuits as data.
- Embedded within Haskell, thus has no type safety or formal semantics
- Proto-Quipper refers to a family of formally defined fragments of Quipper, starting with Proto-Quipper-S from Julien Ross' thesis

Past Work on Proto-Quipper-S

- Previously, worked on mechanizing the meta-theory of Proto-Quipper-S in Beluga using a linearity predicate on typing judgments

Past Work on Proto-Quipper-S

- Previously, worked on mechanizing the meta-theory of Proto-Quipper-S in Beluga using a linearity predicate on typing judgments

$$\begin{array}{rcl} a,b,c & ::= & x \mid q \mid (t,C,a) \mid \texttt{True} \mid \texttt{False} \mid \langle a,b\rangle \mid * \mid ab \mid \lambda x.a \mid \\ & & rev \mid unbox \mid box^T \mid \ \texttt{if}\ a\ \texttt{then}\ b\ \texttt{else}\ c \mid \texttt{let}\ * = a\ \texttt{in}\ b \mid \\ & & \texttt{let}\ \langle x,y\rangle = a\ \texttt{in}\ b. \end{array}$$

$$A,B ::= \texttt{qubit} \mid 1 \mid \texttt{bool} \mid A \otimes B \mid A \multimap B \mid !A \mid \texttt{Circ}(T,U)$$

Past Work on Proto-Quipper-S

- Previously, worked on mechanizing the meta-theory of Proto-Quipper-S in Beluga using a linearity predicate on typing judgments

$$\begin{array}{rcl} a,b,c & ::= & x \mid q \mid (t,C,a) \mid \texttt{True} \mid \texttt{False} \mid \langle a,b\rangle \mid * \mid ab \mid \lambda x.a \mid \\ & & rev \mid unbox \mid box^T \mid \ \texttt{if}\ a\ \texttt{then}\ b\ \texttt{else}\ c \mid \texttt{let}\ * = a\ \texttt{in}\ b \mid \\ & & \texttt{let}\ \langle x,y\rangle = a\ \texttt{in}\ b. \end{array}$$

$$A,B ::= \texttt{qubit} \mid 1 \mid \texttt{bool} \mid A \otimes B \mid A \multimap B \mid !A \mid \texttt{Circ}(T,U)$$

- The issue here was the difficulty of implementing these "quantum doo-dads" in Beluga, where Proto-Quipper constructs various operations on lists of quantum variables to implement appending circuits, creating new circuits, and reversing circuits in the operational semantics

Past Work on Proto-Quipper-S Cont'd

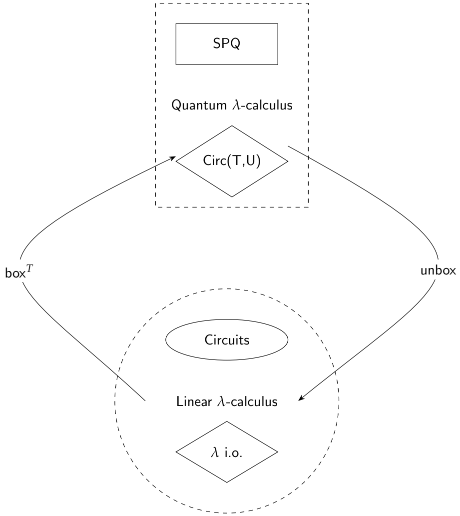
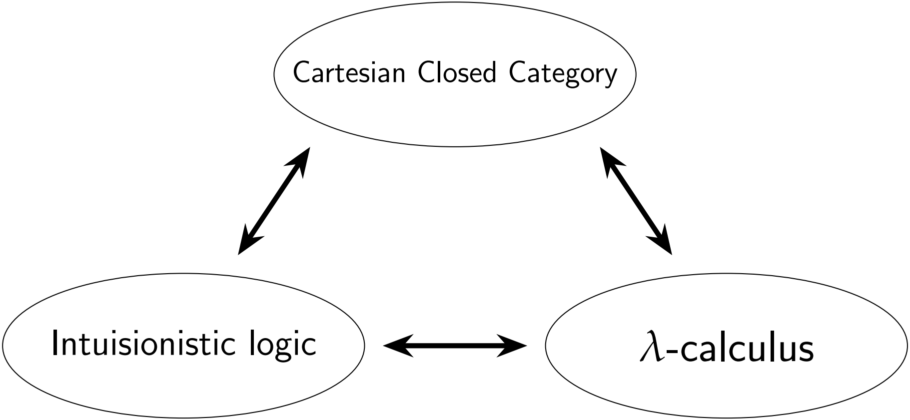
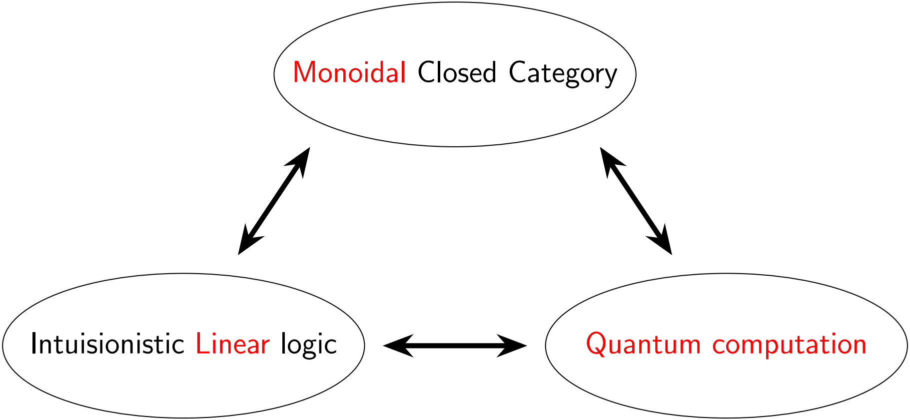


Figure: Two Levels of SPQ

Computational Trilogy



Quantum Computational Trilogy



Background on Category Theory

- A monoidal category is a category $\mathcal{M}$ equipped with:
  - Bifunctor $\otimes : \mathcal{M} \times \mathcal{M} \to \mathcal{M}$
  - Object I
  - Natural isomorphisms $\alpha : (X \otimes Y) \otimes Z \to X \otimes (Y \otimes Z)$, $l : I \otimes X \to X$, $r : X \otimes I \to X$

Background on Category Theory

- A monoidal category is a category $\mathcal{M}$ equipped with:
  - Bifunctor $\otimes : \mathcal{M} \times \mathcal{M} \to \mathcal{M}$
  - Object I
  - Natural isomorphisms $\alpha : (X \otimes Y) \otimes Z \to X \otimes (Y \otimes Z)$, $l : I \otimes X \to X$, $r : X \otimes I \to X$
- A symmetric monoidal category also has a natural transformation $\sigma : X \otimes Y \to Y \otimes X$

Background on Category Theory

- A monoidal category is a category $\mathcal{M}$ equipped with:
  - Bifunctor $\otimes : \mathcal{M} \times \mathcal{M} \to \mathcal{M}$
  - Object I
  - Natural isomorphisms $\alpha : (X \otimes Y) \otimes Z \to X \otimes (Y \otimes Z)$, $l : I \otimes X \to X$, $r : X \otimes I \to X$
- A symmetric monoidal category also has a natural transformation $\sigma : X \otimes Y \to Y \otimes X$
- A symmetric monoidal closed category is such that for every object B in $\mathcal{M}$, the functor $- \otimes B \to B$ has a specified right adjunction.

Background on Category Theory

- A monoidal category is a category $\mathcal{M}$ equipped with:
  - Bifunctor $\otimes : \mathcal{M} \times \mathcal{M} \to \mathcal{M}$
  - Object I
  - Natural isomorphisms $\alpha : (X \otimes Y) \otimes Z \to X \otimes (Y \otimes Z)$, $l : I \otimes X \to X$, $r : X \otimes I \to X$
- A symmetric monoidal category also has a natural transformation $\sigma : X \otimes Y \to Y \otimes X$
- A symmetric monoidal closed category is such that for every object B in $\mathcal{M}$, the functor $- \otimes B \to B$ has a specified right adjunction.
- In other words,for every object A, C in $\mathcal{M}$, there is an object $B \multimap C$ such that

$$\mathcal{M}(A \otimes B, C) \cong \mathcal{M}(A, B \multimap C)$$

Background on Category Theory

- A monoidal category is a category $\mathcal{M}$ equipped with:
  - Bifunctor $\otimes : \mathcal{M} \times \mathcal{M} \to \mathcal{M}$
  - Object I
  - Natural isomorphisms $\alpha : (X \otimes Y) \otimes Z \to X \otimes (Y \otimes Z)$, $l : I \otimes X \to X$, $r : X \otimes I \to X$
- A symmetric monoidal category also has a natural transformation $\sigma : X \otimes Y \to Y \otimes X$
- A symmetric monoidal closed category is such that for every object B in $\mathcal{M}$, the functor $- \otimes B \to B$ has a specified right adjunction.
- In other words,for every object A, C in $\mathcal{M}$, there is an object $B \multimap C$ such that

$$\mathcal{M}(A \otimes B, C) \cong \mathcal{M}(A, B \multimap C)$$

- Cartesian closed category is an SMCC where the tensor product is Cartesian

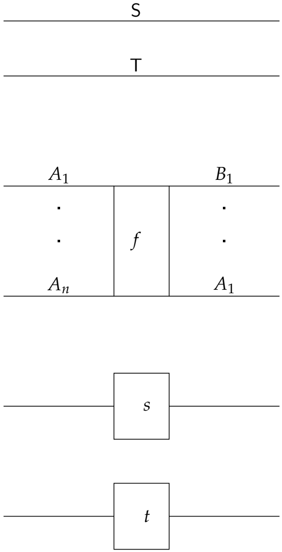
Quantum Circuits

$$S \otimes T$$

$$I$$

$$f : A_1 \otimes \cdots \otimes A_n \to B_1 \otimes \cdots \otimes B_m$$

$$s \otimes t$$



Mediating Between

- Stepping back: how can we mediate between the linear and the classical?

Mediating Between

- Stepping back: how can we mediate between the linear and the classical?
- Two schools of thought:

Mediating Between

- Stepping back: how can we mediate between the linear and the classical?
- Two schools of thought:
    1. Classical logic is just a part of linear logic!
    2. Classical logic and linear logic share a symmetric relationship with ways to get from one to the other
- In a way, they are equivalent

Benton's Linear Non-Linear Model

A linear non-linear model consists of:

1. A Cartesian closed category $(\mathcal{C}, 1, \times, \rightarrow)$
2. A symmetric monoidal closed category $(\mathcal{L}, I, \otimes, \multimap)$
3. symmetric monoidal adjunction between symmetric monoidal functors $(F, m) : \mathcal{C} \rightarrow \mathcal{L}$ and $(G, n) : \mathcal{L} \rightarrow \mathcal{C}$.

Benton's Linear Non-Linear Model

A linear non-linear model consists of:

1. A Cartesian closed category $(\mathcal{C}, 1, \times, \to)$
2. A symmetric monoidal closed category $(\mathcal{L}, I, \otimes, \multimap)$
3. symmetric monoidal adjunction between symmetric monoidal functors $(F, m) : \mathcal{C} \to \mathcal{L}$ and $(G, n) : \mathcal{L} \to \mathcal{C}$.

Here, a symmetric monoidal functor is a functor $F : \mathcal{M} \to \mathcal{M}'$ on monoidal categories equipped with a map $m_I : I' \to F(I)$ in $\mathcal{M}'$ and a natural transformation $m_{X,Y} : F(X) \otimes' F(Y) \to F(X \otimes Y)$, satisfying various coherence conditions. A symmetric monoidal adjunction is when the functors are symmetric monoidal.

Proto-Quipper-M (Benton-like)

- Proto-Quipper-M begins with a categorical model and builds up a programming language around it

Proto-Quipper-M (Benton-like)

- Proto-Quipper-M begins with a categorical model and builds up a programming language around it
- Separation between parameter (classic) and state (linear)

Proto-Quipper-M (Benton-like)

- Proto-Quipper-M begins with a categorical model and builds up a programming language around it
- Separation between parameter (classic) and state (linear)
- Takes a symmetric monoidal category $\mathbf{M}$ and Yoneda embeds it into a product closed SMC $\overline{\mathbf{M}}$, defining $\overline{\overline{\mathbf{M}}}$ as $\mathbf{Fam}(\overline{\mathbf{M}})$. States live in $\overline{\overline{\mathbf{M}}}$, parameters live in $\mathbf{Set}$.

Proto-Quipper-M (Benton-like)

- Proto-Quipper-M begins with a categorical model and builds up a programming language around it
- Separation between parameter (classic) and state (linear)
- Takes a symmetric monoidal category **M** and Yoneda embeds it into a product closed SMC $\overline{\mathbf{M}}$, defining $\overline{\overline{\mathbf{M}}}$ as $\mathbf{Fam}(\overline{\mathbf{M}})$. States live in $\overline{\overline{\mathbf{M}}}$, parameters live in **Set**.
- Symmetric monoidal adjunction $p \dashv \flat$ between $\overline{\overline{\mathbf{M}}}$ and **Set** forms the LNL model

Proto-Quipper-M (Non-Benton-like)

- Defines a comonad $! : p \circ \flat$ (boxing comonad) along with maps:

Proto-Quipper-M (Non-Benton-like)

- Defines a comonad $! : p \circ \flat$ (boxing comonad) along with maps:
  - force : $!A \rightarrow B$

Proto-Quipper-M (Non-Benton-like)

- Defines a comonad $! : p \circ \flat$ (boxing comonad) along with maps:
    - force $: !A \to B$
    - $$\frac{f : !A \to B}{\mathsf{lift}(f) : !A \to !B}$$

Proto-Quipper-M (Non-Benton-like)

- Defines a comonad $! : p \circ \flat$ (boxing comonad) along with maps:
  - force : $!A \to B$
  - $$\frac{f : !A \to B}{\text{lift}(f) : !A \to !B}$$
  - box : $!(TU) \to p(\mathbf{M}(T, U))$, inverse is unbox.

Proto-Quipper-M (Non-Benton-like)

- Defines a comonad $! : p \circ \flat$ (boxing comonad) along with maps:
  - force : $!A \to B$
  - $$\frac{f : !A \to B}{\mathsf{lift}(f) : !A \to !B}$$
  - box : $!(TU) \to p(\mathbf{M}(T, U))$, inverse is unbox.
  - Gives us apply : $p(M(T, U)) \otimes T \to U$
- Always staying within $\overline{\overline{\mathbf{M}}}$

Proto-Quipper-M (Non-Benton-like)

- Defines a comonad $! : p \circ \flat$ (boxing comonad) along with maps:
  - force : $!A \to B$
  - $$\frac{f : !A \to B}{\mathsf{lift}(f) : !A \to !B}$$
  - box : $!(TU) \to p(\mathbf{M}(T, U))$, inverse is unbox.
  - Gives us apply : $p(M(T, U)) \otimes T \to U$
- Always staying within $\overline{\overline{\mathbf{M}}}$
- 'The category $\overline{\overline{\mathbf{M}}}$, together with the adjunction given by p and b forms a linear-non-linear model in the sense of Benton'

Proto-Quipper-Adj

- Equivalent to Proto-Quipper-M, but seeks to make the adjoint structure more explicit

Proto-Quipper-Adj

- Equivalent to Proto-Quipper-M, but seeks to make the adjoint structure more explicit
    - Foreground the adjunction instead of understanding it as a consequence

Proto-Quipper-Adj

- Equivalent to Proto-Quipper-M, but seeks to make the adjoint structure more explicit
  - Foreground the adjunction instead of understanding it as a consequence
  - Clear separation of functional programming layer and circuit layer

Proto-Quipper-Adj

- Equivalent to Proto-Quipper-M, but seeks to make the adjoint structure more explicit
  - Foreground the adjunction instead of understanding it as a consequence
  - Clear separation of functional programming layer and circuit layer
- Also, explicit syntax for circuits (easier for reasoning)

Proto-Quipper-Adj

- Equivalent to Proto-Quipper-M, but seeks to make the adjoint structure more explicit
  - Foreground the adjunction instead of understanding it as a consequence
  - Clear separation of functional programming layer and circuit layer
- Also, explicit syntax for circuits (easier for reasoning)
- Thus, Proto-Quipper-Adj is composed of two programming languages with three modes of use

Types

$$A_{\mathrm{U}}, B_{\mathrm{U}} ::= 1_{\mathrm{U}} \mid A_{\mathrm{U}} \otimes_{\mathrm{U}} B_{\mathrm{U}} \mid A_{\mathrm{U}} \multimap_{\mathrm{U}} B_{\mathrm{U}} \mid \uparrow^{\mathrm{U}}_{\mathrm{L}} A_{\mathrm{L}}$$

$$A_{\mathrm{L}}, B_{\mathrm{L}} ::= 1_{\mathrm{L}} \mid A_{\mathrm{L}} \otimes_{\mathrm{L}} B_{\mathrm{L}} \mid A_{\mathrm{L}} \multimap_{\mathrm{L}} B_{\mathrm{L}} \mid \uparrow^{\mathrm{L}}_{\mathrm{Q}} A_{\mathrm{Q}} \mid \downarrow^{\mathrm{U}}_{\mathrm{L}} A_{\mathrm{U}}$$

$$A_{\mathrm{Q}}, B_{\mathrm{Q}} ::= 1_{\mathrm{Q}} \mid A_{\mathrm{Q}} \otimes_{\mathrm{Q}} B_{\mathrm{Q}} \mid A_{\mathrm{Q}} \multimap_{\mathrm{Q}} B_{\mathrm{Q}} \mid \mathsf{qubit} \mid \downarrow^{\mathrm{L}}_{\mathrm{Q}} A_{\mathrm{L}}$$

Terms

$$
\begin{aligned}
M, N ::= {} & x \mid \lambda x.M \mid MN \\
& \mid \langle\rangle \mid let\ \langle\rangle = M\ in\ N \\
& \mid \langle M, N\rangle \mid let\ \langle x, y\rangle = M\ in\ N \\
& \mid \mathrm{susp}\, M \mid \mathrm{susp}\, C \mid \mathrm{force}\, M \\
& \mid \mathrm{down}\, M \mid let\ down\ x = M\ in\ N \mid let\ down\ x = C\ in\ M \\
C, D ::= {} & x \mid \lambda x.C \mid CD \\
& \mid \langle\rangle \mid let\ \langle\rangle = C\ in\ D \\
& \mid \langle C, D\rangle \mid let\ \langle x, y\rangle : A \otimes B = C\ in\ D \\
& \mid \mathrm{force}\, M \mid \mathrm{down}\, M \mid let\ down\ x = C\ in\ D \mid g \\
P ::= {} & M \mid C
\end{aligned}
$$

Some Typing Rules

$$\frac{\Delta \vdash M : \uparrow^{\mathsf{L}}_{\mathsf{Q}} A_{\mathsf{Q}}}{\Delta \vdash \mathsf{force}\ M : A_{\mathsf{Q}}}\ (\text{Q/FORCE}) \qquad \frac{\Delta \vdash M : \uparrow^{\mathsf{U}}_{\mathsf{L}} A_{\mathsf{L}}}{\Delta \vdash \mathsf{force}\ M : A_{\mathsf{L}}}\ (\text{F/FORCE})$$

$$\frac{\Delta_{\mathsf{C}}, \Delta_1 \vdash M : \downarrow^{\mathsf{U}}_{\mathsf{L}} A_{\mathsf{U}} \qquad \Delta_{\mathsf{C}}, \Delta_2, x : A_{\mathsf{U}} \vdash N : B_{\mathsf{L}}}{\Delta_{\mathsf{C}}, \Delta_1, \Delta_2 \vdash \mathsf{let\ down}\ x = M\ \mathsf{in}\ N : B_{\mathsf{L}}}\ (\text{F/LETD/F})$$

$$\frac{(g : U \multimap_{\mathsf{Q}} S) \in \Sigma}{\Delta_{\mathsf{W}} \vdash g : A_{\mathsf{Q}} \multimap_{\mathsf{Q}} B_{\mathsf{Q}}}\ (\text{Q/GATE})$$

Categorical Model

- To draw comparison to Proto-Quipper-M, we focus solely on the blue, functional language which operates over classical and linear modes.

Categorical Model

- To draw comparison to Proto-Quipper-M, we focus solely on the blue, functional language which operates over classical and linear modes.
- We take a Cartesian category $(\mathcal{C}, 1, \times, \rightarrow)$ and a symmetric monoidal closed category $(\mathcal{L}, I, \otimes, \multimap)$ with symmetric monoidal functors $\uparrow : \mathcal{L} \rightarrow \mathcal{C}$ and $\downarrow : \mathcal{C} \rightarrow \mathcal{L}$ forming a symmetric monoidal adjunction $\downarrow \dashv \uparrow$

Categorical Model

- To draw comparison to Proto-Quipper-M, we focus solely on the blue, functional language which operates over classical and linear modes.
- We take a Cartesian category $(\mathcal{C}, 1, \times, \rightarrow)$ and a symmetric monoidal closed category $(\mathcal{L}, I, \otimes, \multimap)$ with symmetric monoidal functors $\uparrow : \mathcal{L} \rightarrow \mathcal{C}$ and $\downarrow : \mathcal{C} \rightarrow \mathcal{L}$ forming a symmetric monoidal adjunction $\downarrow \dashv \uparrow$
- i.e. there is a natural isomorphism Φ such that

$$\mathcal{L}(\downarrow A, Y) \cong \mathcal{C}(A, \uparrow Y)$$

Type Interpretation

- In Adjoint Proto-Quipper, types are interpreted in C or L depending on their mode.

Type Interpretation

- In Adjoint Proto-Quipper, types are interpreted in C or L depending on their mode.
- A type A_U in mode U is interpreted as $[\![A_U]\!]_U \in C$, while a type A_L in mode L (or a type $A_{\geq L}$ in linear or unrestricted mode) is interpreted as $[\![A_{\geq L}]\!]_L \in L$.

Type Interpretation

- In Adjoint Proto-Quipper, types are interpreted in C or L depending on their mode.
- A type A_U in mode U is interpreted as $[\![A_U]\!]_U \in C$, while a type A_L in mode L (or a type $A_{\geq L}$ in linear or unrestricted mode) is interpreted as $[\![A_{\geq L}]\!]_L \in L$.

1. $[\![1_U]\!]_U = 1, \quad [\![1_L]\!]_L = I$
2. $[\![A_U \otimes B_U]\!]_U = [\![A_U]\!]_U \times [\![B_U]\!]_U,$
   $[\![A_L \otimes B_L]\!]_L = [\![A_L]\!]_L \otimes [\![B_L]\!]_L$
3. $[\![A_U \multimap B_U]\!]_U = [\![A_U]\!]_U \to [\![B_U]\!]_U,$
   $[\![A_L \multimap B_L]\!]_L = [\![A_L]\!]_L \multimap [\![B_L]\!]_L$
4. $[\![\uparrow A_L]\!]_U = \uparrow [\![A_L]\!]_L, \quad [\![\downarrow A_U]\!]_L = \downarrow [\![A_U]\!]_U$
5. $[\![B_U]\!]_L = \downarrow [\![B_U]\!]_U \stackrel{(4)}{=} [\![\downarrow B_U]\!]_L$

Context Interpretation

Sequents are interpreted as morphisms in the category corresponding to the mode of their succedent:

Context Interpretation

Sequents are interpreted as morphisms in the category corresponding to the mode of their succedent:

$$[\![x : C_U, \ldots, y : C'_U \vdash P : A_U]\!]_U : [\![C_U]\!]_U \times \cdots \times [\![C'_U]\!]_U \to [\![A_U]\!]_U \tag{1}$$

$$[\![x : C_{\geq L}, \ldots, y : C'_{\geq L} \vdash P : A_L]\!]_L : [\![C_{\geq L}]\!]_L \otimes \cdots \otimes [\![C'_{\geq L}]\!]_L \to [\![A_L]\!]_L \tag{2}$$

Suspend and Force

$$\frac{\Delta_U \vdash P : A_L}{\Delta_U \vdash \mathsf{susp}\ P :\uparrow A_L}$$

Suspend and Force

$$\frac{\Delta_U \vdash P : A_L}{\Delta_U \vdash \mathsf{susp}\ P :\uparrow A_L}$$

Given a morphism $P : [\![\Delta_U]\!]_L \multimap [\![A_L]\!]_L =\downarrow [\![\Delta_U]\!]_U \multimap [\![A_L]\!]_L$.

Suspend and Force

$$\frac{\Delta_U \vdash P : A_L}{\Delta_U \vdash \mathsf{susp}\ P :\uparrow A_L}$$

Given a morphism $P : [\![\Delta_U]\!]_L \multimap [\![A_L]\!]_L =\downarrow [\![\Delta_U]\!]_U \multimap [\![A_L]\!]_L$.
By the adjunction Φ

$$\mathsf{susp}\ P = \Phi(P) : [\![\Delta_U]\!]_U \to\uparrow [\![A_L]\!]_L = [\![\Delta_U]\!]_U \to [\![\uparrow A_L]\!]_U$$

Suspend and Force

$$\frac{\Delta_U \vdash P : A_L}{\Delta_U \vdash \mathsf{susp}\ P :\uparrow A_L}$$

Given a morphism $P : [\![\Delta_U]\!]_L \multimap [\![A_L]\!]_L =\downarrow [\![\Delta_U]\!]_U \multimap [\![A_L]\!]_L$.
By the adjunction Φ

$$\mathsf{susp}\ P = \Phi(P) : [\![\Delta_U]\!]_U \to\uparrow [\![A_L]\!]_L = [\![\Delta_U]\!]_U \to [\![\uparrow A_L]\!]_U$$

$$\frac{\Delta_U \vdash M :\uparrow A_L}{\Delta_U \vdash \mathsf{force}\ M : A_L}$$

Suspend and Force

$$\frac{\Delta_U \vdash P : A_L}{\Delta_U \vdash \mathsf{susp}\ P :\uparrow A_L}$$

Given a morphism $P : [\![\Delta_U]\!]_L \multimap [\![A_L]\!]_L =\downarrow [\![\Delta_U]\!]_U \multimap [\![A_L]\!]_L$.
By the adjunction Φ

$$\mathsf{susp}\ P = \Phi(P) : [\![\Delta_U]\!]_U \to\uparrow [\![A_L]\!]_L = [\![\Delta_U]\!]_U \to [\![\uparrow A_L]\!]_U$$

$$\frac{\Delta_U \vdash M :\uparrow A_L}{\Delta_U \vdash \mathsf{force}\ M : A_L}$$

Given a morphism $M : [\![\Delta_U]\!]_U \to [\![\uparrow A_L]\!]_U = [\![\Delta_U]\!]_U \to\uparrow [\![A_L]\!]_L$

Suspend and Force

$$\frac{\Delta_U \vdash P : A_L}{\Delta_U \vdash \mathsf{susp}\ P :\uparrow A_L}$$

Given a morphism $P : [\![\Delta_U]\!]_L \multimap [\![A_L]\!]_L =\downarrow [\![\Delta_U]\!]_U \multimap [\![A_L]\!]_L$.
By the adjunction Φ

$$\mathsf{susp}\ P = \Phi(P) : [\![\Delta_U]\!]_U \to\uparrow [\![A_L]\!]_L = [\![\Delta_U]\!]_U \to [\![\uparrow A_L]\!]_U$$

$$\frac{\Delta_U \vdash M :\uparrow A_L}{\Delta_U \vdash \mathsf{force}\ M : A_L}$$

Given a morphism $M : [\![\Delta_U]\!]_U \to [\![\uparrow A_L]\!]_U = [\![\Delta_U]\!]_U \to\uparrow [\![A_L]\!]_L$
By the adjunction Φ

$$\mathsf{force}\ M = \Phi^{-1}(M) :\downarrow [\![\Delta_U]\!]_U \to [\![A_L]\!]_L = [\![\Delta_U]\!]_L \to [\![A_L]\!]_L$$

Down

$$\frac{\Delta_U \vdash M : A_U}{\Delta_U \vdash \mathsf{down}\ M : \downarrow A_U}$$

Down

$$\frac{\Delta_U \vdash M : A_U}{\Delta_U \vdash \mathsf{down}\ M :\downarrow A_U}$$

gives us a morphism $M : [\![\Delta_U]\!]_U \to [\![A_U]\!]_U$.

Down

$$\frac{\Delta_U \vdash M : A_U}{\Delta_U \vdash \text{down } M :\downarrow A_U}$$

gives us a morphism $M : [\![\Delta_U]\!]_U \to [\![A_U]\!]_U$.
We down this to a morphism $\downarrow M :\downarrow [\![\Delta_U]\!]_U \to\downarrow [\![A_U]\!]_U = [\![\Delta_U]\!]_L \to [\![\downarrow A_L]\!]_L$ as required.

Apply (Classic)

$$\frac{\Delta_U, \Delta_U^1 \vdash M : A_U \multimap B_U \qquad \Delta_U, \Delta_U^2 \vdash N : A_U}{\Delta_U, \Delta_U^1, \Delta_U^2 \vdash M\, N : B_U}$$

Apply (Classic)

$$\frac{\Delta_U, \Delta_U^1 \vdash M : A_U \multimap B_U \qquad \Delta_U, \Delta_U^2 \vdash N : A_U}{\Delta_U, \Delta_U^1, \Delta_U^2 \vdash M\,N : B_U}$$

Two morphisms in C:

Tools: $C_{[\![A_U]\!]_U} : [\![A_U]\!]_U \to [\![A_U]\!]_U \times [\![A_U]\!]_U$ and
$\mathsf{eval}_{[\![A_U]\!]_U, [\![B_U]\!]_U} : ([\![A_U]\!]_U \to [\![B_U]\!]_U) \times [\![A_U]\!]_U \to [\![B_U]\!]_U$

Apply (Classic)

$$\frac{\Delta_U, \Delta_U^1 \vdash M : A_U \multimap B_U \qquad \Delta_U, \Delta_U^2 \vdash N : A_U}{\Delta_U, \Delta_U^1, \Delta_U^2 \vdash M\,N : B_U}$$

Two morphisms in C:

Tools: $C_{[\![A_U]\!]_U} : [\![A_U]\!]_U \to [\![A_U]\!]_U \times [\![A_U]\!]_U$ and
$\mathsf{eval}_{[\![A_U]\!]_U, [\![B_U]\!]_U} : ([\![A_U]\!]_U \to [\![B_U]\!]_U) \times [\![A_U]\!]_U \to [\![B_U]\!]_U$

$$(M \times N) \circ (\mathsf{Id}_{[\![\Delta_U]\!]_U} \times \sigma_{[\![\Delta_U]\!]_U, [\![\Delta_U^1]\!]_U} \times \mathsf{Id}_{[\![\Delta_U^2]\!]_U}) \circ (C_{[\![\Delta_U]\!]_U} \times \mathsf{Id}_{[\![\Delta_U^1]\!]_U} \times \mathsf{Id}_{[\![\Delta_U^2]\!]_U})$$
$$: ([\![A_U]\!]_U \to [\![B_U]\!]_U) \times [\![B_U]\!]_U$$

Apply (Classic)

$$\frac{\Delta_U, \Delta_U^1 \vdash M : A_U \multimap B_U \qquad \Delta_U, \Delta_U^2 \vdash N : A_U}{\Delta_U, \Delta_U^1, \Delta_U^2 \vdash M\, N : B_U}$$

Two morphisms in C:

Tools: $C_{[\![A_U]\!]_U} : [\![A_U]\!]_U \to [\![A_U]\!]_U \times [\![A_U]\!]_U$ and
$\mathsf{eval}_{[\![A_U]\!]_U, [\![B_U]\!]_U} : ([\![A_U]\!]_U \to [\![B_U]\!]_U) \times [\![A_U]\!]_U \to [\![B_U]\!]_U$

$$(M \times N) \circ (\mathsf{Id}_{[\![\Delta_U]\!]_U} \times \sigma_{[\![\Delta_U]\!]_U, [\![\Delta_U^1]\!]_U} \times \mathsf{Id}_{[\![\Delta_U^2]\!]_U}) \circ (C_{[\![\Delta_U]\!]_U} \times \mathsf{Id}_{[\![\Delta_U^1]\!]_U} \times \mathsf{Id}_{[\![\Delta_U^2]\!]_U})$$
$$: ([\![A_U]\!]_U \to [\![B_U]\!]_U) \times [\![B_U]\!]_U$$

Apply eval to get the desired result.

Apply (Linear)

$$\frac{\Delta_U, \Delta^1_{\geq L} \vdash M : A_L \multimap B_L \qquad \Delta_U, \Delta^2_{\geq L} \vdash N : A_L}{\Delta_U, \Delta^1_{\geq L}, \Delta^2_{\geq L} \vdash M\,N : B_L}$$

Apply (Linear)

$$\frac{\Delta_U, \Delta^1_{\geq L} \vdash M : A_L \multimap B_L \qquad \Delta_U, \Delta^2_{\geq L} \vdash N : A_L}{\Delta_U, \Delta^1_{\geq L}, \Delta^2_{\geq L} \vdash M\,N : B_L}$$

Similar to unrestricted version, but how to reuse Δ_U?

Apply (Linear)

$$\frac{\Delta_U, \Delta^1_{\geq L} \vdash M : A_L \multimap B_L \qquad \Delta_U, \Delta^2_{\geq L} \vdash N : A_L}{\Delta_U, \Delta^1_{\geq L}, \Delta^2_{\geq L} \vdash M\, N : B_L}$$

Similar to unrestricted version, but how to reuse Δ_U?
Given $C_{[\![A_U]\!]_U} : [\![A_U]\!]_U \to [\![A_U]\!]_U \times [\![A_U]\!]_U$,

$$\begin{aligned} \downarrow C_{[\![A_U]\!]_U} :& \downarrow [\![A_U]\!]_U \to \downarrow ([\![A_U]\!]_U \times [\![A_U]\!]_U) \\ =& \downarrow [\![A_U]\!]_U \multimap \downarrow [\![A_U]\!]_U \otimes \downarrow [\![A_U]\!]_U \\ =& [\![\downarrow A_U]\!]_L \multimap [\![\downarrow A_U]\!]_L \otimes [\![\downarrow A_U]\!]_L \end{aligned}$$

Apply (Linear)

$$\frac{\Delta_U, \Delta^1_{\geq L} \vdash M : A_L \multimap B_L \qquad \Delta_U, \Delta^2_{\geq L} \vdash N : A_L}{\Delta_U, \Delta^1_{\geq L}, \Delta^2_{\geq L} \vdash M\,N : B_L}$$

Similar to unrestricted version, but how to reuse Δ_U?
Given $C_{[\![A_U]\!]_U} : [\![A_U]\!]_U \to [\![A_U]\!]_U \times [\![A_U]\!]_U$,

$$\begin{aligned}\downarrow C_{[\![A_U]\!]_U} :& \downarrow [\![A_U]\!]_U \to \downarrow ([\![A_U]\!]_U \times [\![A_U]\!]_U) \\ =& \downarrow [\![A_U]\!]_U \multimap \downarrow [\![A_U]\!]_U \otimes \downarrow [\![A_U]\!]_U \\ =& [\![\downarrow A_U]\!]_L \multimap [\![\downarrow A_U]\!]_L \otimes [\![\downarrow A_U]\!]_L\end{aligned}$$

$$(M \otimes N) \circ (\mathsf{Id}_{\downarrow [\![\Delta_U]\!]_U} \otimes \sigma_{[\![\Delta_U]\!]_L, [\![\Delta^1_{\geq L}]\!]_L} \otimes \mathsf{Id}_{[\![\Delta^2_{\geq L}]\!]_L}) \circ (\downarrow C_{[\![\Delta_U]\!]_U} \otimes \mathsf{Id}_{[\![\Delta^1_{\geq L}]\!]_L} \otimes \mathsf{Id}_{[\![\Delta^2_{\geq L}]\!]_L})$$

Cut for Time

- Categorical semantics for quantum mode (interesting double adjunction)

Cut for Time

- Categorical semantics for quantum mode (interesting double adjunction)
- Recovery of Proto-Quipper-M's programming abstractions

Cut for Time

- Categorical semantics for quantum mode (interesting double adjunction)
- Recovery of Proto-Quipper-M's programming abstractions
- Any actual Beluga code

Further Work

- Finishing up some proofs (confluence?)

Further Work

- Finishing up some proofs (confluence?)
- 'Foundational calculus'... for what?

Further Work

- Finishing up some proofs (confluence?)
- 'Foundational calculus'... for what?
- Mechanizing all the proofs

Further Work

- Finishing up some proofs (confluence?)
- 'Foundational calculus'... for what?
- Mechanizing all the proofs
- Looking at newer members of the Proto-Quipper family

Bibliography

Benton, P. N. (1994). “A Mixed Linear and Non-Linear Logic: Proofs, Terms and Models (Extended Abstract)”. In: *Selected Papers from the 8th International Workshop on Computer Science Logic*. CSL ’94. Berlin, Heidelberg: Springer-Verlag, pp. 121–135. ISBN: 3540600175.

Lee, Dongho et al. (Nov. 2021). “Concrete Categorical Model of a Quantum Circuit Description Language with Measurement”. In: *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2021)*. Ed. by Mikołaj Bojańczy and Chandra Chekuri. Vol. 213. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 51, 51:1–51:20. DOI: 10.4230/LIPIcs.FSTTCS.2021.51.

Rios, Francisco and Peter Selinger (Feb. 2018). “A Categorical Model for a Quantum Circuit Description Language (Extended Abstract)”. In: *Proceedings of the 14th International Conference on Quantum Physics and Logic (QPL), Nijmegen, the Netherlands, July 3–7, 2017*. Ed. by Bob Coecke and Aleks Kissinger. Vol. 266. Electronic Proceedings in Theoretical Computer Science. Waterloo, NSW, Australia: Open Publishing Association, pp. 164–178. DOI: 10.4204/EPTCS.266.11.

Questions?