Semantic Models of Quantum Programming Languages:  
Recursion in Categorical Models

Michael Mislove

Department of Computer Science  
Tulane University  
Work Supported by US AFOSR

Joint work with Bert Lindenhovius and Vladimir Zamdzhiev

Workshop on Higher Category Approach to Certifiably Correct Quantum Information Processing Systems  
February 4, 2019
2015 MURI Project
Semantics and Tools for
High Level Functional Quantum Programming Languages

- *Proto-Quipper* family of languages
  - Peter Selinger, Dalhousie\(^1\)
- Coq verification of *Qwire*
  - Steve Zdancewic (UPenn)
- Structure of qubits / Quantum information
  - Patrick Hayden (Stanford)

- Recursion in Linear / Nonlinear Models\(^2\) and Contextuality
  - Tulane component
- Dependent types
  - Aaron Stump (UIowa)
- Hoare logic and quantum languages
  - Xiaodi Wu (QICS, UMd)

\(^1\) Funded by other sources.
\(^2\) This talk
Prototypical Quantum Computer

- A *quantum computer* is a classical computer with a quantum co-processor.

![Diagram of Classical Computer and Quantum Co-processor with circuits and measurements](image)

- Circuit: sequence of unitary operators
Prototypical Quantum Computer

- We elide measurements and focus on a classical functional language for constructing circuits and a linear language for modeling them as linear morphisms.

- A quantum programming language is a classical functional language together with a linear language of quantum circuits:

  ![Diagram](Functional Language \rightarrow Linear language \leftarrow)

- We model circuit description languages using Linear / Nonlinear Models
Models of Functional Programming Languages

Classical Functional Language

Types $Q, R ::= \text{Bool} \mid \text{Nat} \mid Q + R \mid Q \times R \mid Q \to R$

Based on intuitionistic logic and typed lambda calculus

**Theorem** [Lambek] There’s a one-to-one correspondence between models of the typed lambda calculus and Cartesian closed categories
Models of Functional Programming Languages

Classical Functional Language

Types \( Q, R \ ::= \text{Bool} | \text{Nat} | Q + R | Q \times R | Q \rightarrow R \)

Based on intuitionistic logic and typed lambda calculus

**Theorem** [Lambek] There’s a one-to-one correspondence between models of the typed lambda calculus and Cartesian closed categories

Linear Functional Language

Types \( A, B \ ::= 0 | 1 | A + B | A \otimes B | A \multimap B | !A \)

Based on intuitionistic linear linear logic and linear lambda calculus

Models use symmetric monoidal closed categories.
A Linear/Non-Linear (LNL) model is given by the following data:

- A cartesian closed category $\mathbf{C}$.
- A symmetric monoidal closed category $\mathbf{L}$.
- A symmetric monoidal adjunction:

  $\mathbf{C} \Downarrow \mathbf{L}$

  $F(X \times Y) \simeq F(X) \otimes F(Y)$
  $F(X + Y) \simeq F(X) + F(Y)$
  $F(\emptyset) = 0$
  $F(1) = I$
  $F \circ G = !$ – the lift comonad

An LNL model is a model of Intuitionistic Linear Logic.\(^1\)

\(^1\)Nick Benton. *A mixed linear and non-linear logic: Proofs, terms and models.* CSL'94
Proto-Quipper-M (Rios and Selinger)

Types
\[ A, B ::= \alpha \mid 0 \mid A + B \mid I \mid A \otimes B \mid A \rightarrow B \mid !A \mid \text{Circ}(T, U) \]

Intuitionistic types
\[ P, R ::= 0 \mid P + R \mid I \mid P \otimes R \mid !A \mid \text{Circ}(T, U) \]

M-types
\[ T, U ::= \alpha \mid I \mid T \otimes U \]

Terms
\[ M, N ::= x \mid \ell \mid c \mid \text{let } x = M \text{ in } N \]
\[ \square_A M \mid \text{left}_{A,B} M \mid \text{right}_{A,B} M \mid \text{case } M \text{ of } \{ \text{left } x \rightarrow N \mid \text{right } y \rightarrow P \} \]
\[ \star \mid M; N \mid \langle M, N \rangle \mid \text{let } \langle x, y \rangle = M \text{ in } N \mid \lambda x^A. M \mid MN \]
\[ \text{lift } M \mid \text{force } M \mid \text{box}_T M \mid \text{apply}(M, N) \mid (\ell, C, \ell') \]

- All types other than Intuitionistic types are \textit{linear}
- M-types: morphisms from a symmetric monoidal category such as \( M = \text{FdC}^*\text{Alg} \)
- Only use one (combined) form of type judgement
Assume $H : Q \to Q$ is a constant representing the Hadamard gate.

Example

two-hadamard : Circ(Q, Q)
two-hadamard ≡ box_Q lift λq^Q.HHq

This program creates a completed circuit consisting of two $H$ gates. The term is intuitionistic (can be copied, deleted).
Circuit Model

Example

Shor’s algorithm for integer factorization may be seen as an infinite family of quantum circuits – each circuit is a procedure for factoring an $n$-bit integer, for a fixed $n$.

![Quantum Fourier Transform](https://commons.wikimedia.org/w/index.php?curid=14545612)

Figure: Quantum Fourier Transform on $n$ qubits (subroutine in Shor’s algorithm).²

Proto-Quipper-M is used to describe families of morphisms in an arbitrary, but fixed, symmetric monoidal category, $\mathcal{M}$.

²Figure source: https://commons.wikimedia.org/w/index.php?curid=14545612
Concrete model of Proto-Quipper-M

A simple Proto-Quipper-M model is given by the LNL model:

\[
\begin{align*}
\text{Set} & \xrightarrow{- \otimes I} M \\
\overline{M}(I, -) & \xleftarrow{\perp} \overline{M}
\end{align*}
\]

where \( \overline{M} = [M^{\text{op}}, \text{Set}] \) is a closed, product complete category containing given SMC \( M \)

**Theorem (Rios & Selinger)**

The simple categorical model of Proto-Quipper-M is type-safe, sound, and computationally adequate
Concrete model of Proto-Quipper-M

There are two semantic models:

- For all types, $\llbracket P \rrbracket \in \mathcal{M}$
- For intuitionistic types, also have $\llbracket P \rrbracket \in \text{Set}$

**Theorem**

For any intuitionistic type $P$, there exists a canonical isomorphism $\alpha_P : \llbracket P \rrbracket \rightarrow F(\llbracket P \rrbracket)$.

So we can define copy and discard morphisms for each intuitionistic type $P$:

\[
\begin{align*}
\Delta_P := \llbracket P \rrbracket & \xrightarrow{\alpha_P} F(\llbracket P \rrbracket) \xrightarrow{F(id,id)} F(\llbracket P \rrbracket \times \llbracket P \rrbracket) \\
& \xrightarrow{\cong} F(\llbracket P \rrbracket) \otimes F(\llbracket P \rrbracket) \xrightarrow{\alpha_P^{-1} \otimes \alpha_P^{-1}} \llbracket P \rrbracket \otimes \llbracket P \rrbracket
\end{align*}
\]

\[
\Diamond_P := \llbracket P \rrbracket \xrightarrow{\alpha_P} F(\llbracket P \rrbracket) \xrightarrow{F1} F1 \xrightarrow{\cong} I
\]

where $FX = X \otimes I$
Our Work: Adding Recursion

- Focus on adding recursive types.
  - Term recursion follows from recursive types.
- Main difficulty is with the categorical model.
- How can we copy/discard intuitionistic recursive types?
  - A list of qubits should be linear – cannot copy/discard.
  - A list of natural numbers should be intuitionistic – can implicitly copy/discard.
- For the rest of the talk we focus on the linear/non-linear type structure.
- How do we design a linear/non-linear FPC\(^3\) ?

\(^3\)FPC is an intuitionistic type theory studied by Fiore and Plotkin.
Adding Recursive Types

Type Variables \( X, Y \)

Types \( A, B \) ::=
\( X \mid \alpha \mid A + B \mid I \mid A \otimes B \mid A \rightarrow B \mid !A \mid \text{Circ}(T, U) \mid \mu X.A \)

Intuitionistic types \( P, R \) ::=
\( X \mid P + R \mid I \mid P \otimes R \mid !A \mid \text{Circ}(T, U) \mid \mu X.P \)

M-types \( T, U \) ::=
\( \alpha \mid I \mid T \otimes U \)

These types are accompanied by some formation rules, which we omit.

\textbf{Design Choice:} Two kinds of type variables – intuitionistic and linear? Or just one kind (as above)?
Some useful recursive types

Example
Nat $\equiv \mu X.l + X$ (intuitionistic)

Example
List Nat $\equiv \mu X.l + X \otimes \text{Nat}$ (intuitionistic)

Example
List Qubit $\equiv \mu X.l + X \otimes \text{Qubit}$ (linear)
A CPO-enriched model

CPO – ω-complete partial orders and monotone maps preserving suprema of ω-chains.

A CPO–enriched LNL model includes:

1. A CPO-symmetric monoidal closed category $\mathcal{L}$ with finite CPO-coproducts.
2. A CPO-symmetric monoidal adjunction:

$$F = - \otimes I$$

$\mathcal{L}$ is CPO$_{\perp I}$-enriched and has ω-colimits

Remark

1. and 3. imply $\mathcal{L}$ has a zero object and we can solve recursive domain equations.
Interpretation of recursive types

Interpreting recursive types requires finding initial (final) (co)algebras of certain CPO-endofunctors.

Lemma (Adámek)

Let \( C \) be a category with an initial object \( \emptyset \) and let \( T : C \to C \) be an endofunctor. Assume further that the following \( \omega \)-diagram

\[
\emptyset \xrightarrow{\iota} T\emptyset \xrightarrow{T\iota} T^2\emptyset \xrightarrow{T^2\iota} \ldots
\]

has a colimit and \( T \) preserves it. Then, the induced isomorphism is the initial \( T \)-algebra.

Corollary

In a symmetric monoidal closed category with finite coproducts and \( \omega \)-colimits, any endofunctor composed from constants, \( \otimes \) and \( + \) has an initial algebra.
Embedding-projection pairs

Problem: How do we interpret recursive types which also contain ! and \( \circ \) ?

Textbook Solution: CPO-enrichment and embedding-projection pairs.

Definition
Given a CPO-enriched category \( C \), an embedding-projection pair is a pair of morphisms \( e : A \rightarrow B \) and \( p : B \rightarrow A \), such that \( p \circ e = \text{id} \) and \( e \circ p \leq \text{id} \).

Theorem
If \( e \) is an embedding, then it has a unique projection, which we denote \( e^* \).

Definition
The subcategory of \( C \) with the same objects, but whose morphisms are embeddings is denoted \( C_e \).
Theorem (Smyth and Plotkin)

If $T : C \rightarrow D$ is a CPO-enriched functor and $C$ has $\omega$-colimits, then $T$ preserves $\omega$-colimits of embeddings. In other words, the restriction $T_e : C_e \rightarrow D_e$ is $\omega$-continuous.

Theorem

In our categorical model, any CPO-endofunctor $T : \mathcal{L} \rightarrow \mathcal{L}$ has an initial $T$-algebra, whose inverse is a final $T$-coalgebra.

Remark

The above theorem follows directly from results in Fiore’s PhD thesis.
Main Lemma

We define $\text{CPO}_{pe}$ to be the full-on-objects subcategory of $\text{CPO}$ whose morphisms $f$ are those satisfying $F(f) \in \mathcal{L}_e$. We call such $f$ pre-embeddings.

Then there are two semantic models:

- For all types, $\llbracket \Theta \vdash P \rrbracket \in \mathcal{L}$
- For intuitionistic types, also have $\llbracket \Theta \vdash P \rrbracket \in \text{CPO}_{pe}$

There exists a natural isomorphism

$$\alpha_{\Theta \vdash P} : \llbracket \Theta \vdash P \rrbracket_s \circ F^{\times n} \cong F \circ \llbracket \Theta \vdash P \rrbracket$$

Diagrammatically:
Let $P$ be an intuitionistic object and $\alpha : P \rightarrow F(X)$ an isomorphism.

We can define three maps:

**Discard:** $\diamond P : P \xrightarrow{\alpha} F(X) \xrightarrow{F(1_X)} F(1) \xrightarrow{\cong} I$;

**Copy:** $\Delta P : P \xrightarrow{\alpha} F(X) \xrightarrow{F(\langle id, id \rangle)} F(X \times X) \xrightarrow{\cong} F(X) \otimes F(X) \xrightarrow{\alpha^{-1} \otimes \alpha^{-1}} P \otimes P$;

**Lift:** $\text{lift} P : P \xrightarrow{\alpha} F(X) \xrightarrow{F(\eta_X)} !F(X) \xrightarrow{!(\alpha^{-1})} !P$.

Given two intuitionistic objects $P_1$ and $P_2$, a morphism $f : P_1 \rightarrow P_2$ is called *intuitionistic*, if there exists a morphism $f' \in \mathcal{CPO}(X, Y)$ and two isomorphisms $\alpha$ and $\beta$, such that $f = P_1 \xrightarrow{\alpha} F(X) \xrightarrow{F(f')} F(Y) \xrightarrow{\beta} P_2$.

If $f : P_1 \rightarrow P_2$ is intuitionistic, then:

- $\diamond P_2 \circ f = \diamond P_1$;
- $\Delta P_2 \circ f = (f \otimes f) \circ \Delta P_1$;
- $\text{lift} P_2 \circ f = !f \circ \text{lift} P_1$. 
Thank You!

Questions??
Operational semantics

\[
\begin{align*}
(S, m) \downarrow (S', v) & \quad (S', n) \downarrow (S'', v') & \\
(S, \langle m, n \rangle) \downarrow (S'', \langle v, v' \rangle) & \quad (S, m) \downarrow (S', \langle v, v' \rangle) & \quad (S', n[v/x, v'/y]) \downarrow (S'', w) \\
(S, \text{let } \langle x, y \rangle = m \text{ in } n) \downarrow (S'', w) & \\
(S, \text{lift } m) \downarrow (S, \text{lift } m) & \quad (S, m) \downarrow (S', \text{lift } m') & \quad (S', m') \downarrow (S'', v) \\
(S, \text{force } m) \downarrow (S'', v) & \\
(S, m) \downarrow (S', \text{lift } n) & \quad \text{freshlabels}(T) = (Q, \tilde{\ell}) & \quad (\text{id}_Q, n\tilde{\ell}) \downarrow (D, \tilde{\ell}') \\
(S, \text{box}_T m) \downarrow (S', (\tilde{\ell}, D, \tilde{\ell}')) & \\
(S, m) \downarrow (S', (\tilde{\ell}, D, \tilde{\ell}')) & \quad (S', n) \downarrow (S'', \tilde{k}) & \quad \text{append}(S'', \tilde{k}, \tilde{\ell}, D, \tilde{\ell}') = (S''', \tilde{k}') \\
(S, \text{apply}(m, n)) \downarrow (S''', \tilde{k}') & \\
(S, m) \downarrow (S', (\tilde{\ell}, D, \tilde{\ell}')) & \quad (S', n) \downarrow (S'', \tilde{k}) & \quad \text{append}(S'', \tilde{k}, \tilde{\ell}, D, \tilde{\ell}') \text{ undefined} \\
(S, \text{apply}(m, n)) \downarrow \text{Error} & \\
(S, \text{apply}(m, n)) \downarrow (S', (\tilde{\ell}, D, \tilde{\ell}')) & \\
(S, (\tilde{\ell}, D, \tilde{\ell}')) \downarrow (S, (\tilde{\ell}, D, \tilde{\ell}'))
\end{align*}
\]