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)

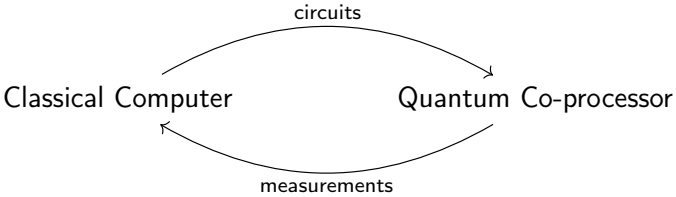
[1] Funded by other sources.

[2] This talk

Tulane University · The University of Iowa · Penn University of Pennsylvania · Stanford · Joint Center for Quantum Information and Computer Science

Prototypical Quantum Computer

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



- 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*:

$$\begin{tikzcd} \textit{Functional Language} \arrow[r, bend left] & \textit{Linear language} \arrow[l, bend left] \end{tikzcd}$$

- We model *circuit description languages* using Linear / Nonlinear Models

Models of Functional Programming Languages

Classical Functional Language

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

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 $\quad Q, R \quad ::= \quad \mathsf{Bool} \mid \mathsf{Nat} \mid Q + R \mid Q \times R \mid Q \to R$

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

Linear Functional Language

Types $\quad A, B \quad ::= \quad 0 \mid \mathsf{I} \mid A + B \mid A \otimes B \mid A \multimap B \mid !A$

Based on intuitionistic linear linear logic and linear lambda calculus

Models use symmetric monoidal closed categories.

Linear/Non-Linear models

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:

$$\begin{tikzcd} \mathbf{C} \arrow[r, bend left, "F"] \arrow[r, phantom, "\perp"] & \mathbf{L} \arrow[l, bend left, "G"] \end{tikzcd}$$

$$\begin{aligned} F(X \times Y) &\simeq F(X) \otimes F(Y) \\ F(X + Y) &\simeq F(X) + F(Y) \\ F(\emptyset) = 0 & \quad\;\; F(1) = I \\ F \circ G &= \; ! - \text{the lift comonad} \end{aligned}$$

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	$::=$	$\alpha \mid 0 \mid A + B \mid \mathsf{I} \mid A \otimes B \mid A \multimap B \mid !A \mid \mathsf{Circ}(T, U)$
Intuitionistic types	P, R	$::=$	$0 \mid P + R \mid \mathsf{I} \mid P \otimes R \mid !A \mid \mathsf{Circ}(T, U)$
M-types	T, U	$::=$	$\alpha \mid \mathsf{I} \mid T \otimes U$

$$
\begin{array}{lll}
\text{Terms} & M, N & ::= \; x \mid \ell \mid c \mid \mathsf{let}\ x = M\ \mathsf{in}\ N \\
& & \mid \Box_A M \mid \mathsf{left}_{A,B} M \mid \mathsf{right}_{A,B} M \mid \mathsf{case}\ M\ \mathsf{of}\ \{\mathsf{left}\ x \to N \mid \mathsf{right}\ y \to P\} \\
& & \mid * \mid M; N \mid \langle M, N \rangle \mid \mathsf{let}\ \langle x, y \rangle = M\ \mathsf{in}\ N \mid \lambda x^A.M \mid MN \\
& & \mid \mathsf{lift}\ M \mid \mathsf{force}\ M \mid \mathbf{box}_T M \mid \mathsf{apply}(M, N) \mid (\vec{\ell}, C, \vec{\ell'})
\end{array}
$$

- All types other than Intuitionistic types are *linear*
- M-types: morphisms from a symmetric monoidal category such as M = FdC*Alg
- Only use one (combined) form of type judgement

Example

Assume $H : Q \multimap Q$ is a constant representing the Hadamard gate.

Example

two-hadamard : $\mathrm{Circ}(Q, Q)$
two-hadamard $\equiv \mathrm{box}_Q$ lift $\lambda q^Q.HHq$

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.

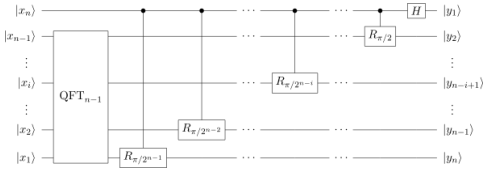


Figure: Quantum Fourier Transform on n qubits (subroutine in Shor's algorithm).[2]

Proto-Quipper-M is used to describe *families* of morphisms in an arbitrary, but fixed, symmetric monoidal category, **M**.

[2] Figure 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:

$$\begin{tikzcd} \mathbf{Set} \arrow[r, bend left, "- \odot I"] \arrow[r, phantom, "\perp"] & \overline{\mathbf{M}} \arrow[l, bend left, "\overline{\mathbf{M}}(I, -)"] \end{tikzcd}$$

where $\overline{\mathbf{M}} = [\mathbf{M}^{\mathrm{op}}, \mathbf{Set}]$ is a closed, product complete category containing given SMC $\mathbf{M}$

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, $[\![P]\!] \in \overline{\mathbf{M}}$
- For intuitionistic types, also have $(\!|P|\!) \in \mathbf{Set}$

Theorem

For any intuitionistic type P, there exists a canonical isomorphism $\alpha_P : [\![P]\!] \to F(\!|P|\!)$.

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

$$\Delta_P := [\![P]\!] \xrightarrow{\alpha_P} F(\!|P|\!) \xrightarrow{F\langle id, id\rangle} F((\!|P|\!) \times (\!|P|\!)) \xrightarrow{\cong} F(\!|P|\!) \otimes F(\!|P|\!) \xrightarrow{\alpha_P^{-1} \otimes \alpha_P^{-1}} [\![P]\!] \otimes [\![P]\!]$$

$$\diamond_P := [\![P]\!] \xrightarrow{\alpha_P} F(\!|P|\!) \xrightarrow{F1} F1 \xrightarrow{\cong} I$$

where $FX = X \odot I$

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	$::=$	$X \mid \alpha \mid A + B \mid \mathsf{I} \mid A \otimes B \mid A \multimap B \mid !A \mid \mathsf{Circ}(T, U)$ $\mid \mu X.A$
Intuitionistic types	P, R	$::=$	$X \mid P + R \mid \mathsf{I} \mid P \otimes R \mid !A \mid \mathsf{Circ}(T, U) \mid \mu X.P$
M-types	T, U	$::=$	$\alpha \mid \mathsf{I} \mid T \otimes U$

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

Design Choice: Two kinds of type variables – intuitionistic and linear? Or just one kind (as above)?

Some useful recursive types

Example

$\mathsf{Nat} \equiv \mu X.I + X \qquad$ (intuitionistic)

Example

$\mathsf{List\ Nat} \equiv \mu X.I + X \otimes \mathsf{Nat} \qquad$ (intuitionistic)

Example

$\mathsf{List\ Qubit} \equiv \mu X.I + X \otimes \mathsf{Qubit} \qquad$ (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:

$$\begin{tikzcd} \mathbf{CPO} \arrow[rr, bend left, "F = -\odot I"] \arrow[rr, phantom, "\perp"] & & \mathcal{L}, \arrow[ll, bend left, "\mathcal{L}(I,-)"] \end{tikzcd}$$

3. The category $\mathcal{L}$ is $\mathbf{CPO}_{\perp!}$-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 $\mathbf{C}$ *be a category with an initial object* $\emptyset$ *and let* $T : \mathbf{C} \to \mathbf{C}$ *be an endofunctor. Assume further that the following* ω*-diagram*

$$\emptyset \xrightarrow{\iota} T\emptyset \xrightarrow{T\iota} T^2\emptyset \xrightarrow{T^2\iota} \cdots$$

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 ω*-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 $\multimap$?
Textbook Solution: **CPO**-enrichment and embedding-projection pairs.

Definition

Given a **CPO**-enriched category $\mathbf{C}$, an *embedding-projection* pair is a pair of morphisms $e : A \to B$ and $p : B \to A$, such that $p \circ e = \mathrm{id}$ and $e \circ p \leq \mathrm{id}$.

Theorem

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

Definition

The subcategory of $\mathbf{C}$ with the same objects, but whose morphisms are embeddings is denoted $\mathbf{C}_e$.

Interpretation of recursive types (contd.)

Theorem (Smyth and Plotkin)

*If $T : \mathbf{C} \to \mathbf{D}$ is a **CPO**-enriched functor and $\mathbf{C}$ has ω-colimits, then T preserves ω-colimits of embeddings. In other words, the restriction $T_e : \mathbf{C}_e \to \mathbf{D}_e$ is ω-continuous.*

Theorem

*In our categorical model, any **CPO**-endofunctor $T : \mathcal{L} \to \mathcal{L}$ 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 $\mathbf{CPO}_{pe}$ to be the full-on-objects subcategory of $\mathbf{CPO}$ 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, $[\![\Theta \vdash P]\!] \in \mathcal{L}$
- For intuitionistic types, also have $(\!|\Theta \vdash P|\!) \in \mathbf{CPO}_{pe}$

There exists a natural isomorphism

$$\alpha_{\Theta \vdash P} : [\![\Theta \vdash P]\!]_s \circ F^{\times n} \Longrightarrow F \circ (\!|\Theta \vdash P|\!)$$

Diagrammatically:

$$\begin{tikzcd}
\mathcal{L}_e^{|\Theta|} \arrow[rr, "{[\![\Theta \vdash P]\!]_e}"] & \arrow[d, phantom, "\overset{\alpha}{\Nearrow}"] & \mathcal{L}_e \\
\mathbf{CPO}_{pe}^{|\Theta|} \arrow[u, "F^{\times|\Theta|}"] \arrow[rr, "{(\!|\Theta \vdash P|\!)}"'] & & \mathbf{CPO}_{pe} \arrow[u, "F"']
\end{tikzcd}$$

Copy and Discard

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

We can define three maps:

Discard: $\diamond_P^\alpha := P \xrightarrow{\alpha} F(X) \xrightarrow{F(1_X)} F(1) \xrightarrow{\cong} I$;

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

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

Given two intuitionistic objects P_1 and P_2, a morphism $f : P_1 \to P_2$ is called *intuitionistic*, if there exists a morphism $f' \in \mathbf{CPO}(X, Y)$ and two isomorphisms

α and β, such that $f = P_1 \xrightarrow{\alpha} F(X) \xrightarrow{F(f')} F(Y) \xrightarrow{\beta} P_2$.

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

- $\diamond_{P_2} \circ f = \diamond_{P_1}$;
- $\Delta_{P_2} \circ f = (f \otimes f) \circ \Delta_{P_1}$;
- $\mathbf{lift}_{P_2} \circ f = !f \circ \mathbf{lift}_{P_1}$.

Thank You!

Questions??

Syntax

$$\frac{}{\Phi, x : A; \emptyset \vdash x : A}\ (\text{var}) \qquad \frac{}{\Phi; \ell : \alpha \vdash \ell : \alpha}\ (\text{label}) \qquad \frac{}{\Phi; \emptyset \vdash c : A_c}\ (\text{const}) \qquad \frac{\Phi, \Gamma_1; Q_1 \vdash m : A \qquad \Phi, \Gamma_2, x : A; Q_2 \vdash n : B}{\Phi, \Gamma_1, \Gamma_2; Q_1, Q_2 \vdash \text{let } x = m \text{ in } n : B}\ (\text{let})$$

$$\frac{\Gamma; Q \vdash m : 0}{\Gamma; Q \vdash \square_C m : C}\ (\text{initial}) \qquad \frac{\Gamma; Q \vdash m : A}{\Gamma; Q \vdash \text{left}_{A,B} m : A + B}\ (\text{left}) \qquad \frac{\Gamma; Q \vdash m : B}{\Gamma; Q \vdash \text{right}_{A,B} m : A + B}\ (\text{right}) \qquad \frac{}{\Phi; \emptyset \vdash * : I}\ (*)$$

$$\frac{\Phi, \Gamma_1; Q_1 \vdash m : A + B \qquad \Phi, \Gamma_2, x : A; Q_2 \vdash n : C \qquad \Phi, \Gamma_2, y : B; Q_2 \vdash p : C}{\Phi, \Gamma_1, \Gamma_2; Q_1, Q_2 \vdash \text{case } m \text{ of } \{\text{left } x \to n \mid \text{right } y \to p\} : C}\ (\text{case}) \qquad \frac{\Phi, \Gamma_1; Q_1 \vdash m : I \qquad \Phi, \Gamma_2; Q_2 \vdash n : C}{\Phi, \Gamma_1, \Gamma_2; Q_1, Q_2 \vdash m; n : C}\ (\text{seq})$$

$$\frac{\Phi, \Gamma_1; Q_1 \vdash m : A \qquad \Phi, \Gamma_2; Q_2 \vdash n : B}{\Phi, \Gamma_1, \Gamma_2; Q_1, Q_2 \vdash \langle m, n \rangle : A \otimes B}\ (\text{pair}) \qquad \frac{\Phi, \Gamma_1; Q_1 \vdash m : A \otimes B \qquad \Phi, \Gamma_2, x : A, y : B; Q_2 \vdash n : C}{\Phi, \Gamma_1, \Gamma_2; Q_1, Q_2 \vdash \text{let } \langle x, y \rangle = m \text{ in } n : C}\ (\text{let-pair})$$

$$\frac{\Gamma, x : A; Q \vdash m : B}{\Gamma; Q \vdash \lambda x^A.m : A \multimap B}\ (\text{abs}) \qquad \frac{\Phi, \Gamma_1; Q_1 \vdash m : A \multimap B \qquad \Phi, \Gamma_2; Q_2 \vdash n : A}{\Phi, \Gamma_1, \Gamma_2; Q_1, Q_2 \vdash mn : B}\ (\text{app}) \qquad \frac{\Phi; \emptyset \vdash m : A}{\Phi; \emptyset \vdash \text{lift } m : !A}\ (\text{lift}) \qquad \frac{\Gamma; Q \vdash m : !A}{\Gamma; Q \vdash \text{force } m : A}\ (\text{force})$$

$$\frac{\Gamma; Q \vdash m : !(T \multimap U)}{\Gamma; Q \vdash \text{box}_T m : \text{Diag}(T, U)}\ (\text{box}) \qquad \frac{\Phi, \Gamma_1; Q_1 \vdash m : \text{Diag}(T, U) \quad \Phi, \Gamma_2; Q_2 \vdash n : T}{\Phi, \Gamma_1, \Gamma_2; Q_1, Q_2 \vdash \text{apply}(m, n) : U}\ (\text{apply}) \qquad \frac{\emptyset; Q \vdash \vec{\ell} : T \quad \emptyset; Q' \vdash \vec{\ell'} : U \quad S \in \mathbf{M}_L(Q, Q')}{\Phi; \emptyset \vdash (\vec{\ell}, S, \vec{\ell'}) : \text{Diag}(T, U)}\ (\text{diag})$$

Operational semantics

$$\frac{(S, m) \Downarrow (S', v) \quad (S', n) \Downarrow (S'', v')}{(S, \langle m, n \rangle) \Downarrow (S'', \langle v, v' \rangle)} \qquad \frac{(S, m) \Downarrow (S', \langle v, v' \rangle) \quad (S', n[v \,/\, x, v' \,/\, y]) \Downarrow (S'', w)}{(S, \mathrm{let}\ \langle x, y \rangle = m\ \mathrm{in}\ n) \Downarrow (S'', w)}$$

$$\frac{}{(S, \mathrm{lift}\ m) \Downarrow (S, \mathrm{lift}\ m)} \qquad \frac{(S, m) \Downarrow (S', \mathrm{lift}\ m') \quad (S', m') \Downarrow (S'', v)}{(S, \mathrm{force}\ m) \Downarrow (S'', v)}$$

$$\frac{(S, m) \Downarrow (S', \mathrm{lift}\ n) \quad \mathrm{freshlabels}(T) = (Q, \vec{\ell}) \quad (\mathrm{id}_Q, n\vec{\ell}) \Downarrow (D, \vec{\ell}')}{(S, \mathrm{box}_T\, m) \Downarrow (S', (\vec{\ell}, D, \vec{\ell}'))}$$

$$\frac{(S, m) \Downarrow (S', (\vec{\ell}, D, \vec{\ell}')) \quad (S', n) \Downarrow (S'', \vec{\kappa}) \quad \mathrm{append}(S'', \vec{\kappa}, \vec{\ell}, D, \vec{\ell}') = (S''', \vec{\kappa}')}{(S, \mathrm{apply}(m, n)) \Downarrow (S''', \vec{\kappa}')}$$

$$\frac{(S, m) \Downarrow (S', (\vec{\ell}, D, \vec{\ell}')) \quad (S', n) \Downarrow (S'', \vec{\kappa}) \quad \mathrm{append}(S'', \vec{\kappa}, \vec{\ell}, D, \vec{\ell}')\ \mathrm{undefined}}{(S, \mathrm{apply}(m, n)) \Downarrow \mathrm{Error}} \qquad \frac{}{(S, (\vec{\ell}, D, \vec{\ell}')) \Downarrow (S, (\vec{\ell}, D, \vec{\ell}'))}$$