Domains and 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

AchimFest  
University of Birmingham  
8 September 2018
• The Jung-Tix Problem

The troublesome probabilistic power domain, Jung & Tix, 1988
• The Jung-Tix Problem

*The troublesome probabilistic power domain*, Jung & Tix, 1988

And, I’m still working on that exit plan....
Prototypical Quantum Computer

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

```
Classical Computer  Quantum Co-processor
<table>
<thead>
<tr>
<th>circuits</th>
</tr>
</thead>
<tbody>
<tr>
<td>measurements</td>
</tr>
</tbody>
</table>
```

- Circuit: sequence of unitary operators
Prototypical Quantum Computer

• We’ll 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:

  ![Functional Language and Linear Language Diagram]

  - Functional Language → Linear language
  - Linear language ← Functional Language

• We study circuit description languages using Linear / Nonlinear Models
Proto-Quipper-M

• Proto-Quipper-M developed by Francisco Rios and Peter Selinger.

The types of the language:

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

The term language:

Terms \( M, N \) ::= \( x \mid \ell \mid c \mid \text{let} \ x = M \text{ in } N \)

\( \square_A M \mid \text{left}_{A,B} M \mid \text{right}_{A,B} M \mid \text{case } M \text{ of } \{ \text{left } x \rightarrow N \mid \text{right } y \rightarrow P \} \)

\( \ast \mid M; N \mid \langle M, N \rangle \mid \text{let } \langle x, y \rangle = M \text{ in } N \mid \lambda x^A. M \mid MN \)

\( \text{lift } M \mid \text{force } M \mid box_T M \mid \text{apply}(M, N) \mid (\rightarrow, C, \rightarrow') \)
Combined Typing Judgement

- There is only one form of type judgement.
- Typing contexts $\Phi, \Gamma, \ldots$ can be mixed.
- Typing contexts $Q$ are for circuit labels.

\[
\begin{align*}
\Phi, x : A ; \emptyset &\vdash x : A \quad \text{(var)} \\
\Phi ; \ell : \alpha &\vdash \ell : \alpha \quad \text{(label)} \\
\Phi ; \emptyset &\vdash c : A_c \quad \text{(const)} \\
\Gamma, x : A ; Q &\vdash M : B \quad \text{(abs)} \\
\Phi, \Gamma_1 ; Q &\vdash M : A \to B \quad \Phi, \Gamma_2 ; Q &\vdash N : A \quad \text{(app)} \\
\Phi ; \emptyset &\vdash M : A \quad \text{(lift)} \\
\Gamma ; Q &\vdash M : !A \quad \text{(force)} \\
\Gamma ; Q &\vdash M : !(T \to U) \quad \text{(box)} \\
\Phi, \Gamma_1 ; Q &\vdash M : \text{Circ}(T, U) \quad \Phi, \Gamma_2 ; Q &\vdash N : T \quad \text{(apply)} \\
\emptyset, Q &\vdash \ell : T \quad \emptyset, Q' &\vdash \ell' : U \\
C &\in \mathcal{M}_X(Q, Q') \quad \Phi, \emptyset &\vdash (\ell, C, \ell') : \text{Circ}(T, U) \quad \text{(circ)}
\end{align*}
\]

Table 3: The typing rules of Proto-Quipper-M (excerpt)
Assume $H : Q \rightarrow Q$ is a constant representing the Hadamard gate.

Example

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

This program creates a completed circuit consisting of two $H$ gates. The term is intuitionistic (can be copied, deleted).
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).\(^1\)

\(^1\)Figure source: https://commons.wikimedia.org/w/index.php?curid=14545612
Proto-Quipper-M is used to describe families of morphisms in an arbitrary, but fixed, symmetric monoidal category, $\mathcal{M}$.

**Example**

If $\mathcal{M} = \text{FdCStar}$, then a program in our language is a family of quantum circuits.

**Example**

$\mathcal{M}$ also could be a category of string diagrams that is freely generated.

- Model Verilog, VHDL, similar hardware description languages, Petri Nets, etc.
A Linear/Non-Linear (LNL) model as described by Benton is given by the following data:

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

$$F \circ G = ! \text{ – the lift comonad}$$

**Remark**

An LNL model is a model of Intuitionistic Linear Logic.

Nick Benton. *A mixed linear and non-linear logic: Proofs, terms and models.* CSL'94
Concrete models of Proto-Quipper-M

The original Proto-Quipper-M model is given by the LNL model:

\[
\begin{array}{ccc}
\text{Set} & \xrightarrow{- \circ I} & \text{Fam}[\overline{M}] \\
\downarrow & & \downarrow \\
\text{Fam}[\overline{M}](I, -) & & \\
\end{array}
\]

\overline{M} – closed, product complete category containing given SMC M

- **Fam[\overline{M}] = \{(X, A) | X \text{ discrete category, } A: X \rightarrow \overline{M} \text{ functor}\}**.

- \((f, \phi) \in \text{Fam}[\overline{M}][(X, A), (Y, B)]\) if \(f: X \rightarrow Y\) functor and \(\phi: A \rightarrow B \circ f\) natural transformation.

- \((g, \psi) \circ (f, \phi) = (g \circ f, \psi f \circ \phi)\).

**Theorem (Rios & Selinger)**

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

The original Proto-Quipper-M model is given by the LNL model:

\[
\begin{array}{c}
\text{Set} \xrightarrow{\bot} \text{Fam}[M]
\end{array}
\]

Sam Staton asked why the \text{Fam} construction is needed – it’s not:

A simpler model for Proto-Quipper-M satisfying the same properties is given by:

\[
\begin{array}{c}
\text{Set} \xrightarrow{\bot} \overline{M}
\end{array}
\]

where in both cases \(\overline{M} = [M^{\text{op}}, \text{Set}]\).
Our Work: Adding Recursion

- Rename the language to \textit{ECLNL}
  - Emphasizes \textit{Enrichment}, \textit{Combined typing judgement} and \textit{LNL models}.
  - Doesn't tie the language to quantum programming \textit{per se}.

- Describe an \textit{abstract} categorical model for the same language.

- Extend language and abstract categorical model to support recursion.

- Prove soundness for abstract models, and computational adequacy for \textit{concrete model}.

\textbf{Related work:} Rennela and Staton describe a different circuit description language, called EWire (based on QWire), for which they also use enriched category theory.
An abstract model for ECLNL

An *ECLNL model* is given by the following data:

1. A cartesian closed category $\mathbf{V}$ together with its self-enrichment $\mathbf{V}$ having finite $\mathbf{V}$-coproducts.

2. A $\mathbf{V}$-symmetric monoidal closed category $\mathbf{C}$ having finite $\mathbf{V}$-coproducts.

3. A $\mathbf{V}$-symmetric monoidal adjunction: $\mathbf{V} \xrightarrow{\bot} \mathbf{C}$

   \[
   \begin{array}{c}
   \mathbf{V} \\
   \downarrow \\
   \mathbf{C}
   \end{array}
   \xrightarrow{\mathbf{C}(I,-)} \mathbf{C}
   \]

   where $(- \odot I)$ denotes the $\mathbf{V}$-copower of the tensor unit in $\mathbf{C}$.

4. A symmetric monoidal category $\mathbf{M}$ and a strong symmetric monoidal functor $E: \mathbf{M} \to \mathbf{C}$, the underlying category of $\mathbf{C}$.

**Theorem:** Absent condition 4, an LNL model canonically induces an ECLNL model.\(^2\)

Soundness

Theorem (Soundness)

*Every abstract model of ECLNL is computationally sound.*
Concrete models of the base language

Fix an arbitrary symmetric monoidal category $\mathcal{M}$. Equipping $\mathcal{M}$ with the free $\text{DCPO}$-enrichment yields a concrete (order-enriched) ECLNL model:

\[
\begin{array}{ccc}
\text{DCPO} & \xrightarrow{\bot} & \overline{\mathcal{M}} \\
\leftarrow & \circ & \rightarrow \\
\overline{\mathcal{M}}(I, -) & \xleftarrow{\bot} & \text{DCPO}
\end{array}
\]

where $\overline{\mathcal{M}} = [\mathcal{M}^{\text{op}}, \text{DCPO}]$. 
Abstract models with recursion

Definition
An endofunctor $T : C \to C$ is parametrically algebraically compact, if for every $A \in \text{Ob}(C)$, the endofunctor $A \otimes T(-)$ has an initial algebra and a final coalgebra whose carriers coincide.

Theorem
A categorical model of a linear/non-linear lambda calculus extended with recursion is given by an LNL model:

$$V \vdash C F G$$

where $FG$ (or equivalently $GF$) is parametrically algebraically compact $^3$.

$^3$Benton & Wadler. Linear logic, monads and the lambda calculus. LiCS’96.
ECLNL extended with general recursion

Definition
A categorical model of ECLNL extended with general recursion is given by a model of ECLNL, where in addition:

5. The comonad endofunctor:

\[ V \xrightarrow{- \odot I} C \]

is parametrically algebraically compact.
Recursion

Extend the syntax:

\[
\Phi, x : !A; \emptyset \vdash m : A \\
\Phi; \emptyset \vdash \text{rec } x^{!A} m : A
\]

(\text{rec})

Extend the operational semantics:

\[
(C, m[\text{lift rec } x^{!A} m/x]) \Downarrow (C', v) \\
(C, \text{rec } x^{!A} m) \Downarrow (C', v)
\]
Soundness

Theorem (Soundess)

*Every model of ECLNL extended with recursion is computationally sound.*
Concrete model of ECLNL extended with recursion

Let $M_*$ be the free $\text{DCPO}_{\perp!}$-enrichment of $M$ and $\overline{M}_* = [M_*^{\text{op}}, \text{DCPO}_{\perp!}]$ be the associated enriched functor category.

Remark

If $M = 1$, then the above model degenerates to the left vertical adjunction, which is a model of a LNL lambda calculus with general recursion.
Computational adequacy

Theorem
The following LNL model:

\[ DCPO \vdash DCPO \bot \]

is computationally adequate at intuitionistic types for the circuit-free fragment of ECLNL.

- Use logical relations for proof.
- Problem with adding circuits is that structural induction over logical relations breaks down on tensors from \( \mathbf{M} \).
- Need more assumptions about \( \mathbf{M} \) for "traditional" approach to work.
Ongoing / Future work

1. Inductive / recursive types.
   - We can support inductive types, since both $\mathcal{C}$ and $\mathcal{V}$ are algebraically complete for endofunctors preserving $\omega$-colimits.
   - $\mathcal{C}$ is algebraically compact for endofunctors preserving $\omega$-colimits, but $\mathcal{V}$ is not.
   - Problem is identifying which parametrically algebraic compact bifunctors $T : \mathcal{C}^{\text{op}} \times \mathcal{C} \to \mathcal{C}$ are intuitionistic. We believe we have solved this.
     Note: e-p pairs arise here!

2. Dependent types (Fam/CFam constructions are well-behaved w.r.t. current models).

3. Dynamic lifting.
Conclusions

• One can construct a model of ECLNL by categorically enriching certain denotational models.

• We described a sound abstract model for ECLNL (with general recursion).

• Systematic construction for concrete models that works for any circuit (string diagram) model described by a symmetric monoidal category.

• The "domain theory" is at the most general level – DCPO, DCPO⊥,!.
Thanks for your attention!

And

Happy Birthday, Achim!
Thanks for your attention!

And

Happy Birthday, Achim!
(S, m) is a configuration if S is a (partially completed) labeled circuit, and m is a term.

\[
(S, m) \Downarrow (S', v) \quad (S', n) \Downarrow (S'', v')
\]
\[
(S, \langle m, n \rangle) \Downarrow (S'', \langle v, v' \rangle)
\]
\[
(S', (v, v')) \quad (S', n[v/x,v'/y]) \Downarrow (S'', w)
\]
\[
(S, \text{let} \langle x,y \rangle = m \text{ in } n) \Downarrow (S'', w)
\]
\[
(S, \text{lift } m) \Downarrow (S', \text{lift } m)
\]
\[
(S, \text{force } m) \Downarrow (S'', v)
\]
\[
(S, m) \Downarrow (S', \text{lift } n) \quad \text{freshlabels}(T) = (Q, \tilde{\ell}) \quad \text{id}_{Q, n\tilde{\ell}} \Downarrow (D, \ell')
\]
\[
(S, \text{box}_T m) \Downarrow (S', (\ell, D, \ell'))
\]
\[
(S, m) \Downarrow (S', (\ell, D, \ell')) \quad (S', n) \Downarrow (S'', \tilde{k}) \quad \text{append}(S'', \tilde{k}, \ell, D, \ell') = (S''', \tilde{k}')
\]
\[
(S, \text{apply}(m, n)) \Downarrow (S''', \tilde{k}')
\]
\[
(S, m) \Downarrow (S', (\ell, D, \ell')) \quad (S', n) \Downarrow (S'', \tilde{k}) \quad \text{append}(S'', \tilde{k}, \ell, D, \ell') \text{ undefined}
\]
\[
(S, \text{apply}(m, n)) \Downarrow \text{Error}
\]
\[
(S, (\ell, D, \ell')) \Downarrow (S, (\ell, D, \ell'))
\]
Recursion (contd.)

Extend the denotational semantics:

\[
\llbracket \Phi; \emptyset \vdash \text{rec } x^!A \ m : A \rrbracket := \sigma[m] \circ \gamma[\Phi].
\]

\[
\begin{array}{c}
\llbracket \Phi \rrbracket \otimes \llbracket \Phi \rrbracket \xleftarrow{id \otimes \text{lift}} \llbracket \Phi \rrbracket \otimes \llbracket \Phi \rrbracket \xrightarrow{\Delta} \llbracket \Phi \rrbracket \\
\downarrow \quad \downarrow \gamma[\Phi] \\
\llbracket \Phi \rrbracket \otimes !\Omega[\Phi] \xleftarrow{\omega[\Phi]} \Omega[\Phi] \xrightarrow{id} \Omega[\Phi] \\
\downarrow \quad \downarrow id \\
\llbracket \Phi \rrbracket \otimes !\Omega[\Phi] \xrightarrow{\omega^{-1}[\Phi]} \llbracket \Phi \rrbracket \otimes !\Omega[\Phi] \\
\downarrow \quad \downarrow id \\
\llbracket \Phi \rrbracket \otimes !\llbracket A \rrbracket \xrightarrow{id \otimes \gamma[\Phi]} \llbracket \Phi \rrbracket \otimes !\llbracket A \rrbracket \\
\downarrow \quad \downarrow \sigma[m] \\
\llbracket \Phi \rrbracket \otimes !\llbracket A \rrbracket \xrightarrow{\sigma[m]} \llbracket m \rrbracket \xrightarrow{[m]} \llbracket A \rrbracket
\end{array}
\]