Enriching a linear/non-linear lambda calculus: a programming language for string diagrams

Bert Lindenhovius, Michael Mislove and Vladimir Zamdzhiev

Department of Computer Science
Tulane University

7 June 2018
Proto-Quipper-M

- We will consider several variants of a functional programming language called Proto-Quipper-M (renamed to ECLNL in our LICS paper).
Proto-Quipper-M

- We will consider several variants of a functional programming language called Proto-Quipper-M (renamed to ECLNL in our LICS paper).
  - Multiple complains about the name.
We will consider several variants of a functional programming language called Proto-Quipper-M (renamed to ECLNL in our LICS paper).

- Multiple complaints about the name.

- Original language developed by Francisco Rios and Peter Selinger.
  - We present a more general abstract model.

- Language is equipped with formal denotational and operational semantics.

- Primary application is in quantum computing, but the language can describe arbitrary string diagrams.

- Original model does not support general recursion.
  - We extend the language with general recursion and prove soundness.
Proto-Quipper-M is used to describe families of morphisms of an arbitrary, but fixed, symmetric monoidal category, which we denote $M$.

**Example**

If $M = \text{FdCStar}$, the category of finite-dimensional $C^*$-algebras and completely positive maps, then a program in our language is a family of quantum circuits.

**Example**

$M$ could also be a category of string diagrams which is freely generated.
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 factorizing 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
Syntax of Proto-Quipper-M

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

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

The term language:

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

Example
qubit-copy \equiv \lambda q^{\text{qubit}}.\langle q, q \rangle

Not a well-typed program. Linear type checker will complain.

Example
nat-copy \equiv \lambda n^{\text{Nat}}.\langle n, n \rangle

This is fine.
Assume $H : Q \rightarrow Q$ is a constant representing the Hadamard gate.

**Example**

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

A program which creates a completed circuit consisting of two $H$ gates. The term is intuitionistic (can be copied, deleted).
Assume $H : Q \rightarrow Q$ is a constant representing the Hadamard gate.

Example

four-hadamard : $Q \rightarrow Q$

four-hadamard $\equiv$ let $hh = \text{two-hadamard in}$

\[
\lambda q^Q . \text{apply}(hh, \text{apply}(hh, q))
\]

A program which, given a qubit (wire), applies four hadamard gates to it.
Our approach

- Describe an abstract categorical model for the same language.
- Describe an abstract categorical model for the language extended with recursion.

Related work: Rennela and Staton describe a different circuit description language, called EWire (based on QWire), where they also use enriched category theory.
A Linear/Non-Linear (LNL) model as described by Benton is given by the following data:

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

\[ 
\begin{array}{ccc}
\mathcal{V} & \downarrow \& & \mathcal{C} \\
\downarrow & \quad & \quad & \downarrow \\
\mathcal{V} & \downarrow \& & \mathcal{C} \\
\end{array}
\]

**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
Models of the Enriched Effect Calculus

A model of the Enriched Effect Calculus (EEC) is given by the following data:

- A cartesian closed category $\mathbf{V}$, enriched over itself.
- A $\mathbf{V}$-enriched category $\mathbf{C}$ with powers, copowers, finite products and finite coproducts.
- A $\mathbf{V}$-enriched adjunction:

$$\begin{array}{ccc}
\mathbf{V} & \xrightarrow{\perp} & \mathbf{C} \\
F & \downarrow & G \\
\end{array}$$

**Theorem**

*Every LNL model with additives determines an EEC model.*

An abstract model for Proto-Quipper-M

A model of Proto-Quipper-M is given by the following data:

1. A cartesian closed category $\mathbf{V}$ together with its self-enrichment $\mathcal{V}$, such that $\mathcal{V}$ has finite $\mathbf{V}$-coproducts.

2. A $\mathbf{V}$-symmetric monoidal closed category $\mathcal{C}$ with underlying category $\mathcal{C}$ such that $\mathcal{C}$ has finite $\mathbf{V}$-coproducts.

3. A $\mathbf{V}$-symmetric monoidal adjunction: $\mathcal{V} \dashv \mathcal{C}$ with $(\cdot \odot I)$ denoting the $\mathbf{V}$-copower of the tensor unit in $\mathcal{C}$.

4. A symmetric monoidal category $\mathcal{M}$ and a strong symmetric monoidal functor $E : \mathcal{M} \to \mathcal{C}$.

Theorem: Ignoring condition 4, an LNL model canonically induces a model of PQM.
Soundness

Theorem (Soundness)

Every abstract model of Proto-Quipper-M is computationally sound.
Concrete models of PQM

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

\[ \text{Set} \xleftarrow{- \odot I} \text{Fam}[M] \xrightarrow{\perp} \text{Fam}[M](I, -) \]

\(^2\)Thanks to Sam Staton for asking why do we need the \text{Fam} construction for this.
Concrete models of PQM

The original Proto-Quipper-M model is given by the LNL model: ²

\[
\begin{array}{c}
\text{Set} \\
\downarrow \\
\text{Fam}[M] \\
\downarrow \\
\text{Fam}[M](I, -)
\end{array}
\]

\[
\begin{array}{c}
- \circ I \\
\text{Set} \\
\downarrow \\
\bar{M}(I, -)
\end{array}
\]

A simpler model for the same language is given by:

\[
\begin{array}{c}
\text{Set} \\
\downarrow \\
\bar{M}
\end{array}
\]

where in both cases \( \bar{M} = [M^{op}, \text{Set}] \).

²Thanks to Sam Staton for asking why do we need the \text{Fam} construction for this.
Concrete models of the base language (contd.)

Fix an arbitrary symmetric monoidal category $\mathbf{M}$. Equipping $\mathbf{M}$ with the free $\textbf{DCPO}$-enrichment yields another concrete (order-enriched) Proto-Quipper-$\mathbf{M}$ model:

\[
\begin{array}{ccc}
\text{DCPO} & - \circ \bot & \overline{\mathbf{M}} \\
\text{\overline{\mathbf{M}}} & \text{\overline{\mathbf{M}}} & \text{\overline{\mathbf{M}}} \\
\end{array}
\]

where $\overline{\mathbf{M}} = [\mathbf{M}^\text{op}, \textbf{DCPO}]$. 
A constructive property

Assuming there is a full and faithful embedding of $E : \mathbf{M} \to \mathbf{C}$, then the model enjoys the following property:

$$\mathbf{C}(\llbracket \Phi \rrbracket, \llbracket T \rrbracket \to \llbracket U \rrbracket) \cong \mathbf{V}(\llbracket \Phi \rrbracket; \mathcal{M}(\llbracket T \rrbracket_{\mathbf{M}}; \llbracket U \rrbracket_{\mathbf{M}}))$$

Therefore any well-typed term $\Phi; \emptyset \vdash m : T \to U$ corresponds to a $\mathbf{V}$-parametrised family of string diagrams. For example, if $\mathbf{V} = \mathbf{Set}$ (or $\mathbf{V} = \mathbf{DCPO}$), then we get precisely a (Scott-continuous) function from $X$ to $\mathcal{M}(\llbracket T \rrbracket_{\mathbf{M}}; \llbracket U \rrbracket_{\mathbf{M}})$ or in other words, a (Scott-continuous) family of string diagrams from $\mathbf{M}$. 
Abstract model with recursion?

Definition
An endofunctor $T : C \to C$ is \textit{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:

$$
\begin{array}{c}
V \\
\downarrow \\
\bigcirc \\
\downarrow \\
C
\end{array}
$$

where $FG$ (or equivalently $GF$) is \textit{parametrically algebraically compact} \footnote{Benton & Wadler. \textit{Linear logic, monads and the lambda calculus.} LiCS’96.}.
Proto-Quipper-M extended with general recursion

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

5. The comonad endofunctor:

\[ \forall \downarrow I \quad C, \]

is parametrically algebraically compact.
Recursion

Extend the syntax:

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

(rec)

Extend the operational semantics:

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

nonterminate: $A$
nonterminate $\equiv \text{rec } x^{!A} \text{ force } x$

The simplest nonterminating program of an arbitrary type $A$.

Example

hadamards : $\text{Nat} \rightarrow Q \rightarrow Q$

hadamards $\equiv \text{rec } hs^{!(\text{Nat} \rightarrow Q \rightarrow Q)} \lambda n^{\text{Nat}} \lambda q^{Q}$

\[
\begin{align*}
\text{if } n = 0 & \text{ then } q \\
\text{else } H \text{ (force hs) } n - 1 & \text{ q}
\end{align*}
\]

A program which given a natural number $n$ composes $n$ Hadamard gates.
Recursion (contd.)

Extend the denotational semantics: $[\Phi; \emptyset \vdash \text{rec } x^A m : A] := \sigma_{[m]} \circ \gamma_{[\Phi]}$.

$$
\begin{align*}
[\Phi] \otimes ! [\Phi] & \xleftarrow{\text{id} \otimes \text{lift}} \ [\Phi] \otimes [\Phi] \xrightarrow{\Delta} [\Phi] \\
\text{id} \otimes ! \gamma_{[\Phi]} & \downarrow \ [\Phi] \otimes ! \Omega_{[\Phi]} \xleftarrow{\omega^{-1}_{[\Phi]}} \Omega_{[\Phi]} \\
\text{id} & \downarrow \ [\Phi] \otimes ! \Omega_{[\Phi]} \xrightarrow{\omega_{[\Phi]}} \Omega_{[\Phi]} \\
\text{id} \otimes ! \sigma_{[m]} & \downarrow \ [\Phi] \otimes ! [A] \xrightarrow{[m]} [A] \\
\end{align*}
$$
Soundness

Theorem (Soundness)

*Every model of Proto-Quipper-M extended with recursion is computationally sound.*
Concrete model of Proto-Quipper-M extended with recursion

Let $M_*$ be the free $\text{DCPO}_\bot!$-enrichment of $M$ and $\overline{M}_* = [M_*^{\text{op}}, \text{DCPO}_\bot!]$ 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:

\[
\begin{array}{c}
\text{DCPO} \\
\text{⊥} \\
\text{U} \\
\end{array} \quad \begin{array}{c}
\text{DCPO}_{⊥!} \\
\text{⊥} \\
\end{array}
\]

is computationally adequate at intuitionistic types for the diagram-free fragment of Proto-Quipper-M.
Future work

• Inductive / recursive types (model appears to have sufficient structure).

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

• Dynamic lifting.
Conclusion

- One can construct a model of PQM by categorically enriching certain denotational models.

- We described a sound abstract model for PQM (with general recursion).

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

- Concrete models indicate good prospects for additional features.
Thank you for your attention!
Operational semantics

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