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.

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.
- 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.
- Oirignal model does not support general recursion.
    - We extend the language with general recursion and prove soundness.

Circuit Model

Proto-Quipper-M is used to describe *families* of morphisms of an arbitrary, but fixed, symmetric monoidal category, which we denote $\mathbf{M}$.

Example

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

Example

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

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.

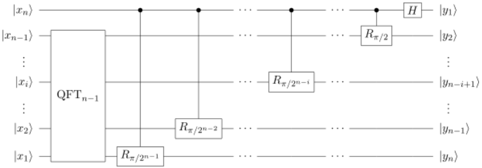

Figure: Quantum Fourier Transform on n qubits (subroutine in Shor's algorithm).[1]

[1] Figure source: `https://commons.wikimedia.org/w/index.php?curid=14545612`

Syntax of Proto-Quipper-M

The types of the language:

$$
\begin{array}{llcl}
\text{Types} & A, B & ::= & \alpha \mid 0 \mid A + B \mid I \mid A \otimes B \mid A \multimap B \mid\ !A \mid \mathbf{Circ(T, U)} \\
\text{Intuitionistic types} & P, R & ::= & \alpha \mid 0 \mid P + R \mid I \mid P \otimes R \mid\ !A \mid \mathbf{Circ(T, U)} \\
\text{M-types} & T, U & ::= & \alpha \mid I \mid T \otimes U
\end{array}
$$

The term language:

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

Example

Example

qubit-copy $\equiv \lambda q^{\textbf{qubit}}.\langle q, q \rangle$

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

Example

nat-copy $\equiv \lambda n^{\textbf{Nat}}.\langle n, n \rangle$

This is fine.

Example

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

Example

two-hadamard : $\mathrm{Circ}(Q, Q)$
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).

Example

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

Example

$$
\begin{aligned}
&\text{four-hadamard} : Q \multimap Q \\
&\text{four-hadamard} \equiv \text{let } hh = \text{two-hadamard in} \\
&\qquad\qquad\qquad\qquad \lambda q^{Q}.\text{apply}(hh, \text{apply}(hh, q))
\end{aligned}
$$

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.

Linear/Non-Linear models

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:

$$\begin{tikzcd} \mathbf{V} \arrow[rr, bend left, "F"] \arrow[rr, phantom, "\perp"] & & \mathbf{C} \arrow[ll, bend left, "G"] \end{tikzcd}$$

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:

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

Theorem

Every LNL model with additives determines an EEC model.

Egger, Møgelberg, Simpson. *The enriched effect calculus: syntax and semantics.* Journal of Logic and Computation 2012

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 $\mathbf{C}$ such that $\mathcal{C}$ has finite $\mathbf{V}$-coproducts.
3. A $\mathbf{V}$-symmetric monoidal adjunction: $$\begin{tikzcd} \mathcal{V} \arrow[rr, bend left, "- \odot I"] \arrow[rr, phantom, "\perp"] & & \mathcal{C}, \arrow[ll, bend left, "\mathcal{C}(I,-)"] \end{tikzcd}$$ where $(- \odot I)$ denotes the $\mathbf{V}$-copower of the tensor unit in $\mathcal{C}$.
4. A symmetric monoidal category $\mathbf{M}$ and a strong symmetric monoidal functor $E : \mathbf{M} \to \mathbf{C}$.

Theorem: Ignorning 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: [2]

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

[2] Thanks to Sam Staton for asking why do we need the **Fam** construction for this.

Concrete models of PQM

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

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

A simpler model for the same language is given by:

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

where in both cases $\overline{\mathbf{M}} = [\mathbf{M}^{op}, \mathbf{Set}]$.

[2] Thanks to Sam Staton for asking why do we need the **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 $\mathbf{DCPO}$-enrichment yields another concrete (order-enriched) Proto-Quipper-M model:

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

where $\overline{\mathbf{M}} = [\mathbf{M}^{\mathrm{op}}, \mathbf{DCPO}]$.

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:

$$\mathbf{C}(\llbracket \Phi \rrbracket, \llbracket T \rrbracket \multimap \llbracket U \rrbracket) \cong \mathbf{V}((\!| \Phi |\!), \mathcal{M}(\llbracket T \rrbracket_{\mathbf{M}}, \llbracket U \rrbracket_{\mathbf{M}}))$$

Therefore any well-typed term $\Phi; \emptyset \vdash m : T \multimap U$ 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 $T : \mathbf{C} \to \mathbf{C}$ is *parametrically algebraically compact*, if for every $A \in \mathrm{Ob}(\mathbf{C})$, 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:

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

where FG (or equivalently GF) is parametrically algebraically compact [3]*.*

[3] Benton & Wadler. *Linear logic, monads and the lambda calculus.* LiCS'96.

Definition

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

5. The comonad endofunctor:

$$\begin{tikzcd} \mathcal{V} \arrow[r, bend left, "- \odot I"] \arrow[r, phantom, "\perp"] & \mathcal{C}, \arrow[l, bend left, "\mathcal{C}(I,-)"] \end{tikzcd}$$

is parametrically algebraically compact.

Recursion

Extend the syntax:

$$\frac{\Phi, x :!A; \emptyset \vdash m : A}{\Phi; \emptyset \vdash \mathrm{rec}\ x^{!A}\ m : A}\ (\mathrm{rec})$$

Extend the operational semantics:

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

Example

Example

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

The simplest nonterminating program of an arbitrary type A.

Example

hadamards : Nat $\multimap Q \multimap Q$
hadamards $\equiv$ rec $hs^{!(\mathrm{Nat} \multimap Q \multimap Q)} \lambda n^{\mathrm{Nat}} \lambda q^{Q}$

$$\begin{aligned} &\text{if } n = 0 \text{ then } q \\ &\text{else } H \text{ (force hs) } n - 1 \; q \end{aligned}$$

A program which given a natural number n composes n Hadamard gates.

Recursion (contd.)

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

$$\begin{tikzcd}
{[\![\Phi]\!] \otimes ![\![\Phi]\!]} \arrow[d, dashed, "\mathrm{id} \otimes !\gamma_{[\![\Phi]\!]}"'] & & {[\![\Phi]\!] \otimes [\![\Phi]\!]} \arrow[ll, "\mathrm{id} \otimes \mathbf{lift}"'] & {[\![\Phi]\!]} \arrow[l, "\Delta"'] \arrow[d, dashed, "\gamma_{[\![\Phi]\!]}"] \\
{[\![\Phi]\!] \otimes !\Omega_{[\![\Phi]\!]}} \arrow[d, "\mathrm{id}"'] & & & \Omega_{[\![\Phi]\!]} \arrow[lll, "\omega^{-1}_{[\![\Phi]\!]}"'] \arrow[d, "\mathrm{id}"] \\
{[\![\Phi]\!] \otimes !\Omega_{[\![\Phi]\!]}} \arrow[rrr, "\omega_{[\![\Phi]\!]}"] \arrow[d, dashed, "\mathrm{id} \otimes !\sigma_{[\![m]\!]}"'] & & & \Omega_{[\![\Phi]\!]} \arrow[d, dashed, "\sigma_{[\![m]\!]}"] \\
{[\![\Phi]\!] \otimes ![\![A]\!]} \arrow[rrr, "{[\![m]\!]}"'] & & & {[\![A]\!]}
\end{tikzcd}$$

Soundness

Theorem (Soundess)

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

Concrete model of Proto-Quipper-M extended with recursion

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

$$
\begin{tikzcd}
\mathbf{DCPO}_{\perp!} \arrow[rr, bend left, "- \odot I"] \arrow[rr, phantom, "\perp"] \arrow[d, bend left, "U"] \arrow[d, phantom, "\dashv"] & & \overline{\mathbf{M}_*} \arrow[ll, bend left, "\overline{\mathbf{M}_*}(I, -)"] \arrow[d, bend left, "\overline{U}"] \arrow[d, phantom, "\dashv"] \\
\mathbf{DCPO} \arrow[u, bend left, "L"] \arrow[rr, bend left, "- \odot I"] \arrow[rr, phantom, "\perp"] & & \overline{\mathbf{M}} \arrow[ll, bend left, "\overline{\mathbf{M}}(I, -)"] \arrow[u, bend left, "\overline{L}"]
\end{tikzcd}
$$

Remark

If $\mathbf{M} = \mathbf{1}$, 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:

$$\begin{tikzcd} \mathbf{DCPO} \arrow[rr, bend left, "\perp"] \arrow[rr, phantom, "\perp"] & & \mathbf{DCPO}_{\perp !}, \arrow[ll, bend left, "U"] \end{tikzcd}$$

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!

Syntax

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

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

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

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

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

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

Operational semantics

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

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

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

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

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