Enriching a Linear/non-linear Lambda Calculus: A Programming Language for String Diagrams

Bert Lindenhovius, Michael Mislove and Vladimir Zamzhiev

Department of Computer Science
Tulane University

LICS’18
University of Oxford
9 July 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).
  • We wanted to emphasize its dependence on enrichment in 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).
  • We wanted to emphasize its dependence on enrichment in 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.
ECLNL is used to describe families of morphisms of an arbitrary, but fixed, symmetric monoidal category, which we denote $\mathcal{M}$.

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

Example
$\mathcal{M}$ 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 \( n \).

\[
\begin{array}{c}
|x_n\rangle \\
|x_{n-1}\rangle \\
\vdots \\
|x_i\rangle \\
\vdots \\
|x_1\rangle \\
\end{array}
\rightarrow
\begin{array}{c}
\text{QFT}_{n-1} \\
\text{QFT}_{n-2} \\
\vdots \\
\text{QFT}_1 \\
\end{array}
\rightarrow
\begin{array}{c}
\vdots \\
\vdots \\
\vdots \\
\vdots \\
|y_1\rangle \\
|y_2\rangle \\
\vdots \\
|y_n\rangle \\
\end{array}
\]

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 ECLNL calculus

The types of the language:

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

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

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

The term language:

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

Example

cubit-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.
Example

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

Example

two-hadamard : Circ($Q$, $Q$)
two-hadamard ≡ 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).
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 $V$.
- A symmetric monoidal closed category $C$.
- A symmetric monoidal adjunction:

$F: V \rightleftharpoons C: G$

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 $V$, enriched over itself.
- A $V$-enriched category $C$ with powers, copowers, finite products and finite coproducts.
- A $V$-enriched adjunction:

$$F : V \xrightarrow{\perp} C \xleftarrow{G} V$$

Theorem

Every LNL model with additives determines an EEC model.

An abstract model for ECLNL

A model of ECLNL 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: $\mathcal{V} \dashv \mathcal{C}$, 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 $E : \mathbf{M} \to \mathcal{C}$.

**Theorem:** Ignoring condition 4, an LNL model canonically induces a model of ECLNL.
Theorem (Soundness)

Every abstract model of ECLNL is computationally sound.
Concrete models of ECLNL

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

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

\[ M = [M^{op}, \text{Set}] \]

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

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

\[
\begin{array}{c}
\text{Set} \xrightarrow{\bot} \text{Fam}[M] \\
\text{Fam}[M](I, \_)
\end{array}
\]

A simpler model for the same language is given by:

\[
\begin{array}{c}
\text{Set} \xrightarrow{\bot} \overline{M} \\
\overline{M}(I, \_)
\end{array}
\]

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

\(^2\)Thanks 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 $\mathcal{M}$. Equipping $\mathcal{M}$ with the free $\mathbf{DCPO}$-enrichment yields another concrete (order-enriched) ECLNL model:

$$\begin{align*}
\mathbf{DCPO} & \quad \rotatebox{90}{\(\approx\)} \quad \mathcal{M}
\end{align*}$$

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

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

$$\mathcal{C}([\Phi], [\mathcal{T}] \rightarrow [U]) \cong \mathcal{V}(([\Phi]), \mathcal{M}([\mathcal{T}]_\mathcal{M}, [U]_\mathcal{M}))$$

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

\[
\begin{array}{c}
V \\
\downarrow \\
C
\end{array}
\]

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

---

$^3$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:

\[
\begin{array}{c}
\mathcal{V} \\
\downarrow \\
C,
\end{array}
\]

\[
\begin{array}{c}
- \\
\circ \\
\downarrow \\
\mathcal{C}(I, -)
\end{array}
\]

is parametrically algebraically compact.
Recursion

Extend the syntax:

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

Extend the operational semantics:

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

$$
\begin{array}{c}
\text{DCPO}_{\perp!} \\
\downarrow \quad \downarrow \quad \downarrow \\
M \\
\end{array}
\rightleftharpoons
\begin{array}{c}
\overline{M}_*(I, -) \\
\downarrow \quad \downarrow \quad \downarrow \\
\overline{M}(I, -) \\
\end{array}
\rightleftharpoons
\begin{array}{c}
\overline{M}_* \\
\downarrow \quad \downarrow \quad \downarrow \\
\text{DCPO}_{\perp!} \\
\end{array}
\rightleftharpoons
\begin{array}{c}
L \\
\downarrow \quad \downarrow \quad \downarrow \\
\overline{M}_*(I, -) \\
\end{array}
\rightleftharpoons
\begin{array}{c}
U \\
\downarrow \quad \downarrow \quad \downarrow \\
M \\
\end{array}
\rightleftharpoons
\begin{array}{c}
L \\
\downarrow \quad \downarrow \quad \downarrow \\
\overline{M}(I, -) \\
\end{array}
\rightleftharpoons
\begin{array}{c}
U \\
\downarrow \quad \downarrow \quad \downarrow \\
\text{DCPO} \\
\end{array}
$$

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.
Theorem

The following LNL model:

\[
\begin{array}{ccc}
DCPO & \downarrow & DCPO_{\perp!} \\
\downarrow & & \downarrow \\
\perp & & \perp \\
U & &
\end{array}
\]

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

• Concrete models indicate good prospects for additional features.
Thank you for your attention!
### Syntax

- **Var:** \( \Phi, x : A; \emptyset \vdash x : A \)  
- **Label:** \( \Phi; \ell : \alpha \vdash \ell : \alpha \)  
- **Const:** \( \Phi; \emptyset \vdash c : A_c \)  
- **Let:** \( \Phi, \Gamma_1; Q \vdash m : A \quad \Phi, \Gamma_2, x : A; Q_2 \vdash n : B \)  
  \( \Phi, \Gamma_1, \Gamma_2; Q_1, Q_2 \vdash \text{let } x = m \text{ in } n : B \)

- **Initial:** \( \Gamma; Q \vdash m : 0 \)  
- **Left:** \( \Gamma; Q \vdash \text{left}_{A,B} m : A + B \)  
- **Right:** \( \Gamma; Q \vdash \text{right}_{A,B} m : A + B \)  
- **Case:** \( \Phi, \Gamma_1, \Gamma_2; Q_1, Q_2 \vdash \text{case } m \text{ of } \{ \text{left } x \rightarrow n | \text{right } y \rightarrow p \} : C \)  
- **Seq:** \( \Phi, \Gamma_1; Q_1 \vdash m : I \quad \Phi, \Gamma_2; Q_2 \vdash n : C \)  
  \( \Phi, \Gamma_1, \Gamma_2; Q_1, Q_2 \vdash m ; n : C \)

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

- **Abs:** \( \Gamma, x : A; Q \vdash \lambda x^A.m : A \rightarrow B \)  
- **App:** \( \Phi, \Gamma_1; Q_1 \vdash m : A \rightarrow B \quad \Phi, \Gamma_2; Q_2 \vdash n : A \)  
  \( \Phi, \Gamma_1, \Gamma_2; Q_1, Q_2 \vdash mn : B \)

- **Box:** \( \Gamma; Q \vdash \text{box}_T m : \text{Diag}(T, U) \)  
- **Diag:** \( \Phi, \Gamma_1; Q_1 \vdash m : \text{Diag}(T, U) \quad \Phi, \Gamma_2; Q_2 \vdash n : T \)  
  \( \Phi, \Gamma_1, \Gamma_2; Q_1, Q_2 \vdash \text{apply}(m, n) : U \)

- **Force:** \( \Phi; \emptyset \vdash m : !A \)  
- **Lift:** \( \Phi; \emptyset \vdash \text{lift } m : !A \)

- **Diagonal:** \( \emptyset; Q \vdash \tilde{\ell} : T \quad \emptyset; Q' \vdash \tilde{\ell}' : U \quad S \in M_L(Q, Q') \)  
  \( \Phi; \emptyset \vdash (\tilde{\ell}, S, \tilde{\ell}') : \text{Diag}(T, U) \)
Operational semantics

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

Extend the denotational semantics: \( [[\Phi; \emptyset \vdash \text{rec } x^A \ m : A]] := \sigma[m] \circ \gamma[\Phi] \).

\[
\begin{align*}
[[\Phi]] \otimes ![\Phi] & \xrightarrow{\text{id} \otimes \text{lift}} [[\Phi]] \otimes [[\Phi]] \xrightarrow{\Delta} [[\Phi]] \\
[[\Phi]] \otimes ![\Omega_{[\Phi]}] & \xrightarrow{\omega^{-1}_{[\Phi]}} \Omega_{[\Phi]} \\
[[\Phi]] \otimes ![\Omega_{[\Phi]}] & \xrightarrow{\omega_{[\Phi]}} \Omega_{[\Phi]} \\
[[\Phi]] \otimes ![\sigma[m]] & \xrightarrow{\text{id}} \Omega_{[\Phi]} \\
[[\Phi]] \otimes ![A] & \xrightarrow{[m]} [[A]]
\end{align*}
\]