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

Achim

- *The Jung-Tix Problem*

  The troublesome probabilistic power domain, Jung & Tix, 1988

Achim

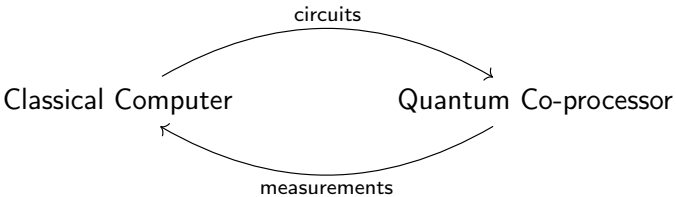
- *The Jung-Tix Problem*

  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



- 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*:

- 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:

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

The term language:

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

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.

$$\frac{}{\Phi, x : A; \emptyset \vdash x : A}\ (var) \qquad \frac{}{\Phi; \ell : \alpha \vdash \ell : \alpha}\ (label) \qquad \frac{}{\Phi; \emptyset \vdash c : A_c}\ (const)$$

$$\frac{\Gamma, x : A; Q \vdash M : B}{\Gamma; Q \vdash \lambda x^A.M : A \multimap B}\ (abs) \qquad \frac{\Phi, \Gamma_1; Q_1 \vdash M : A \multimap B \quad \Phi, \Gamma_2; Q_2 \vdash N : A}{\Phi, \Gamma_1, \Gamma_2; Q_1, Q_2 \vdash MN : B}\ (app)$$

$$\frac{\Phi; \emptyset \vdash M : A}{\Phi; \emptyset \vdash \text{lift}\, M : !A}\ (lift) \qquad \frac{\Gamma; Q \vdash M : !A}{\Gamma; Q \vdash \text{force}\, M : A}\ (force)$$

$$\frac{\Gamma; Q \vdash M : !(T \multimap U)}{\Gamma; Q \vdash \text{box}_T\, M : \text{Circ}(T, U)}\ (box) \qquad \frac{\Phi, \Gamma_1; Q_1 \vdash M : \text{Circ}(T, U) \quad \Phi, \Gamma_2; Q_2 \vdash N : T}{\Phi, \Gamma_1, \Gamma_2; Q_1, Q_2 \vdash \text{apply}(M, N) : U}\ (apply)$$

$$\frac{\emptyset; Q \vdash \vec{\ell} : T \quad \emptyset; Q' \vdash \vec{\ell'} : U \quad C \in \mathbf{M}_{\mathscr{L}}(Q, Q')}{\Phi; \emptyset \vdash (\vec{\ell}, C, \vec{\ell'}) : \text{Circ}(T, U)}\ (circ)$$

Table 3: The typing rules of Proto-Quipper-M (excerpt)

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.

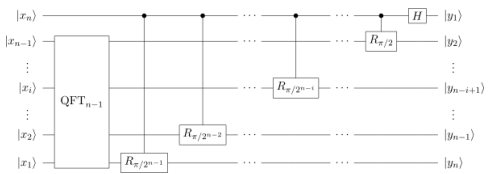


Figure: Quantum Fourier Transform on n qubits (subroutine in Shor's algorithm).[1]

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

Circuit Model

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

Example

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

Example

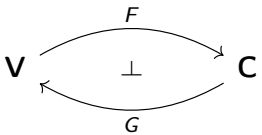
M also could be a category of string diagrams that is freely generated.

- Model Verilog, VHDL, similar hardware description languages, Petri Nets, etc.

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:



$F \circ G = !$ – the lift comonad

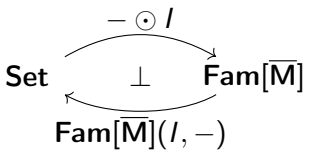
Remark

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:



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

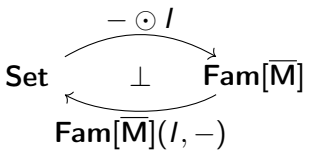
- $\mathbf{Fam}[\overline{\mathbf{M}}] = \{(X, A) \mid X$ discrete category, $A\colon X \to \overline{\mathbf{M}}$ functor$\}$.
- $(f, \phi) \in \mathbf{Fam}[\overline{\mathbf{M}}]((X, A), (Y, B))$ if $f\colon X \to Y$ functor and $\phi\colon A \to B \circ f$ natural transformation.
- $(g, \psi) \circ (f, \phi) = (g \circ f, \psi f \circ \phi)$.

Theorem (Rios & Selinger)

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

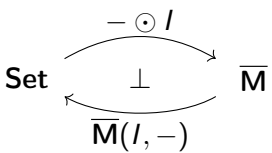
Concrete models of Proto-Quipper-M

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



Sam Staton asked why the **Fam** construction is needed – it's not:

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



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

Our Work: Adding Recursion

- Rename the language to *ECLNL*
  - Emphasizes *Enrichment*, *Combined typing judgement* and *LNL models*.
  - Doesn't tie the language to quantum programming *per se*.
- Describe an *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 *concrete model*.

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 $\mathcal{V}$ having finite $\mathbf{V}$-coproducts.
2. A $\mathbf{V}$-symmetric monoidal closed category $\mathcal{C}$ having finite $\mathbf{V}$-coproducts.
3. A $\mathbf{V}$-symmetric monoidal adjunction: $\mathcal{V} \underset{\mathcal{C}(I,-)}{\overset{-\odot I}{\perp}} \mathcal{C}$,

   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 $E : \mathbf{M} \to \mathbf{C}$, the underlying category of $\mathcal{C}$.

Theorem: Absent condition 4, an LNL model canonically induces an ECLNL model.[2]

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

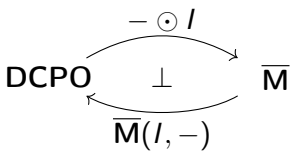
Soundness

Theorem (Soundness)

Every abstract model of ECLNL is computationally sound.

Concrete models of the base language

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



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

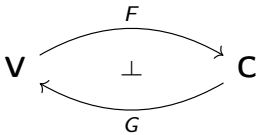
Abstract models with recursion

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:



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

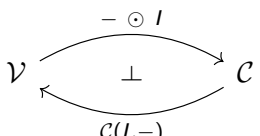
[3] Benton & Wadler. *Linear logic, monads and the lambda calculus*. LiCS'96.

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:



is parametrically algebraically compact.

Recursion

Extend the syntax:

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

Extend the operational semantics:

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

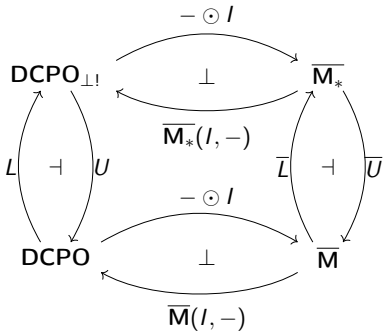
Soundness

Theorem (Soundess)

Every model of ECLNL extended with recursion is computationally sound.

Concrete model of ECLNL extended with recursion

Let $\mathbf{M}_*$ 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.



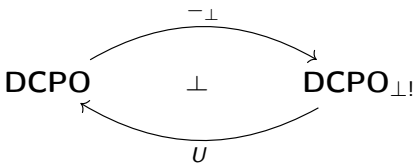
Remark

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:



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 ω-colimits.
   - $\mathcal{C}$ is algebraically compact for endofunctors preserving ω-colimits, but $\mathcal{V}$ is not.
   - Problem is identifying which parametrically algebraic compact bifunctors $T : \mathcal{C}^{op} \times \mathcal{C} \to \mathcal{C}$ 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, $\mathrm{DCPO}_{\perp,!}$.

Thanks for your attention!

And

Happy Birthday, Achim!

Thanks for your attention!

And

Happy Birthday, Achim!

Operational semantics

(S, m) is a *configuration* if S is a (partially completed) labeled circuit, and m is a term.

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

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

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

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

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

Recursion (contd.)

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