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
{alindenh|mislove|vzamdzhi}@tulane.edu

Original version accepted for LICS 2018. For a preprint see [6].

Linear/non-linear (LNL) models, as described by Benton, soundly model an LNL term calculus and LNL logic closely related to intuitionistic linear logic. Every such model induces a canonical enrichment that we show soundly models an LNL lambda calculus for string diagrams, introduced by Rios and Selinger (with primary application in quantum computing). Our abstract treatment of this language leads to simpler concrete models compared to those presented so far. We also extend the language with general recursion and prove soundness. Finally, we present an adequacy result for the diagram-free fragment of the language that corresponds to a modified version of Benton and Wadler's adjoint calculus with recursion.

Introduction. String diagrams have found applications across a range of areas in computer science and related fields such as concurrency [7], systems theory [1], quantum computing [5, 2, 3] and others. But as the size of a system grows, constructing string diagram representations by hand quickly becomes intractable, and more advanced tools are needed to accurately represent and reason about the associated diagrams. In fact, just generating large diagrams is a difficult problem. One area where this has been addressed is in the development of circuit description languages [11, 12] that may be used to generate very large circuits. More recently, Quipper [4] and QWIRE [8] were designed as quantum programming languages which are used to generate quantum circuits.

In this paper we pursue a more abstract approach. We consider a lambda calculus for string diagrams whose primary purpose is to generate complicated diagrams from simpler components. Our development only assumes that the string diagrams we are working with enjoy a symmetric monoidal structure.

An enriched LNL calculus. We introduce the *enriched combined* LNL calculus, ECLNL, whose syntax and operational semantics coincide with those of Proto-Quipper-M [9]. We rename the language to emphasize its dependence on its abstract model, an LNL model with an associated *enrichment*.

Definition 1. An *enriched CLNL model* is given by the following data: (1) a cartesian closed category $\mathbf{V}$ together with its self-enrichment $\mathscr{V}$, such that $\mathscr{V}$ has finite $\mathbf{V}$-coproducts; (2) A $\mathbf{V}$-symmetric monoidal closed category $\mathscr{C}$ with underlying category $\mathbf{C}$ such that $\mathscr{C}$ has $\mathbf{V}$-copowers and finite $\mathbf{V}$-coproducts; (3) A $\mathbf{V}$-adjunction: $\mathscr{V} \underset{\mathscr{C}(I,-)}{\overset{-\odot I}{\underset{\bot}{\rightleftarrows}}} \mathscr{C}$. We also adopt the following notation: F and G are the underlying functors of $(-\odot I)$ and $\mathscr{C}(I,-)$ respectively and we set $! := F \circ G$.

By definition, enriched CLNL models are LNL models with some additional (enriched) structure. But as the next theorem shows, LNL models induce the additional enriched structure as well.

Submitted to:
QPL 2018

Theorem 2. *Every LNL model with finite coproducts induces an enriched CLNL model.*

The ECLNL calculus is designed to describe string diagrams. So we first explain exactly what kind of diagrams we have in mind. The morphisms of any symmetric monoidal category can be described using string diagrams [10]. So, we choose an arbitrary symmetric monoidal category $\mathbf{M}$, and then the string diagrams we will be working with are exactly those that correspond to the morphisms of $\mathbf{M}$. For example, if we set $\mathbf{M} = \mathbf{FdCStar}$, the category of finite-dimensional C*-algebras and completely positive maps, then we can use our calculus for quantum programming. Another interesting choice for quantum computing, in light of recent results [3], is setting $\mathbf{M}$ to be a suitable category of ZX-calculus diagrams.

Definition 3. An *ECLNL model* is given by: (1) an enriched CLNL model (Definition 1); (2) a symmetric monoidal category $(\mathbf{M}, \boxtimes, J)$ and a strong symmetric monoidal functor $E : \mathbf{M} \to \mathbf{C}$.

For space reasons, we omit the syntax and semantics, but they are available at [6]. There we define intuitionistic types, which are a subset of the types of our language. A type that is not intuitionistic is *linear*. The interpretation of a type A is an object $[\![A]\!]$ of $\mathbf{C}$, defined by induction in the usual way.

Proposition 4. *For every intuitionistic type P, there is a canonical isomorphism* $[\![P]\!] \cong F(X)$.

We write contexts as $\Gamma = x_1 : A_1, x_2 : A_2, \ldots, x_n : A_n$, where the x_i are variables and A_i are types. A variable in a context is intuitionistic (linear) if it is assigned an intuitionistic (linear) type. A context that contains only intuitionistic variables is called an *intuitionistic context*. The type system enforces that a linear variable is used exactly once, whereas a non-linear variable may be used any number of times.

Syntax and Semantics. A typing judgement has the form $\Gamma; Q \vdash m : A$. Γ is a term context and Q is a *label context* that is used to identify specific input/output wires of our string diagrams. Its interpretation is a morphism $[\![\Gamma]\!] \otimes [\![Q]\!] \to [\![A]\!]$ in $\mathbf{C}$. A *configuration* is a pair (S, m), where S is a labelled string diagram and m is a term. Operationally, we may think of S as the diagram that has been constructed so far, and m as the program that remains to be executed.

Definition 5. A configuration is said to be *well-typed* with inputs Q, outputs Q' and type A, which we write as $Q \vdash (S, m) : A; Q'$, if there exists Q'' disjoint from Q', s.t. $S : Q \to Q'' \cup Q'$ is a labelled string diagram and $\emptyset; Q'' \vdash m : A$.

Thus, in a well-typed configuration, the term m has no free variables and its labels correspond to a subset of the outputs of S. We interpret a well-typed configuration $Q \vdash (S, m) : A; Q'$, by:

$$[\![(S, m)]\!] := [\![Q]\!] \xrightarrow{[\![S]\!]} [\![Q'']\!] \otimes [\![Q']\!] \xrightarrow{[\![\emptyset; Q'' \vdash m:A]\!] \otimes \mathrm{id}} [\![A]\!] \otimes [\![Q']\!]$$

Theorem 6 (Error freeness [9])**.** *If* $Q \vdash (S, m) : A; Q'$ *then* $(S, m) \not\Downarrow$ *Error.*

Theorem 7 (Subject reduction [9])**.** *If* $Q \vdash (S, m) : A; Q'$ *and* $(S, m) \Downarrow (S', v)$, *then* $Q \vdash (S', v) : A; Q'$.

With this in place, we can show our abstract model is sound.

Theorem 8. *(Soundness) If* $Q \vdash (S, m) : A; Q'$ *and* $(S, m) \Downarrow (S', v)$, *then* $[\![(S, m)]\!] = [\![(S', v)]\!]$.

A constructive property. If we assume, in addition, that $E : \mathbf{M} \to \mathbf{C}$ is fully faithful, then setting $\mathscr{M}(T, U) = \mathscr{C}(T, U)$ for $T, U \in \mathbf{M}$ defines a $\mathbf{V}$-enriched category $\mathscr{M}$ with the same objects as $\mathbf{M}$, and whose underlying category is isomorphic to $\mathbf{M}$. Moreover, E enriches to a fully faithful $\mathbf{V}$-functor $\underline{E} : \mathscr{M} \to \mathscr{C}$. As a consequence, our abstract model enjoys the following constructive property:

$$\mathbf{C}([\![\Phi]\!], [\![T]\!] \multimap [\![U]\!]) \cong \mathbf{V}(X, \mathscr{M}([\![T]\!], [\![U]\!]))$$

where $[\![\Phi]\!] \cong F(X)$. This means that 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 $\mathbf{V} = \mathbf{CPO}$), then we get precisely a (continuous) function from X to $\mathscr{M}([\![T]\!], [\![U]\!])$, or in other words, a family of string diagrams from $\mathbf{M}$.

Concrete Models. The original model of Rios and Selinger [9] is now easily recovered as an instance of our abstract model, but our abstract treatment of the language allows us to present a simpler model (left) together with an order-enriched one (right):

$$\mathbf{Set} \underset{[\mathbf{M}^{\mathrm{op}},\mathbf{Set}](I,-)}{\overset{-\odot I}{\underset{\longleftarrow}{\overset{\longrightarrow}{\bot}}}} [\mathbf{M}^{\mathrm{op}},\mathbf{Set}] \xleftarrow{Y} \mathbf{M} \qquad \mathbf{CPO} \underset{[\mathbf{M}^{\mathrm{op}},\mathbf{CPO}](I,-)}{\overset{-\odot I}{\underset{\longleftarrow}{\overset{\longrightarrow}{\bot}}}} [\mathbf{M}^{\mathrm{op}},\mathbf{CPO}] \xleftarrow{Y} \mathbf{M}$$

Extension with recursion. We extend the ECLNL calculus by adding the term $\mathrm{rec}\, x^{!A}.m$ and we add an additional typing rule (left) and an evaluation rule (right) as follows:

$$\frac{\Phi, x :!A; \emptyset \vdash m : A}{\Phi; \emptyset \vdash \mathrm{rec}\, x^{!A}.m : A}\,(\mathrm{rec}) \qquad \frac{(S, m[\mathrm{lift\ rec}\, x^{!A}.m \,/\, x]) \Downarrow (S', v)}{(S, \mathrm{rec}\, x^{!A}.m) \Downarrow (S', v)}$$

Definition 9. A model of the ECLNL calculus with recursion is given by a model of the ECLNL calculus for which the !-endofunctor is parametrically algebraically compact.

Theorem 10. *Theorems 6 – 8 remain true for the ECLNL calculus extended with recursion.*

Concrete models supporting recursion. Let **CPO** be the category of cpo's (possibly without bottom) and Scott-continuous functions, and let $\mathbf{CPO}_{\bot!}$ be the category of *pointed* cpo's and *strict* Scott-continuous functions. We present a concrete model for an arbitrary symmetric monoidal **M**. Let $\mathscr{M}$ be the free **CPO**-enrichment of **M**. $\mathscr{M}$ is then a **CPO**-symmetric monoidal category with the same monoidal structure as **M**. Let $\mathscr{M}_\bot$ be the free $\mathbf{CPO}_{\bot!}$-enrichment of **M**. Then, $\mathscr{M}_\bot$ has the same objects as **M** and hom-cpo's $\mathscr{M}_\bot(A,B) = \mathscr{M}(A,B)_\bot$, where $(-)_\bot : \mathscr{CPO} \to \mathscr{CPO}_{\bot!}$ is the domain-theoretic lifting functor. By using the enriched Yoneda lemma together with the Day convolution monoidal structure, we see that the enriched functor category $[\mathscr{M}_\bot^{\mathrm{op}}, \mathscr{CPO}_{\bot!}]$ is $\mathbf{CPO}_{\bot!}$-symmetric monoidal closed.

Theorem 11. *The following data:*

$$\mathscr{CPO} \underset{[\mathscr{M}_\bot^{op},\mathscr{CPO}_{\bot!}](I,-)}{\overset{-\odot I}{\underset{\longleftarrow}{\overset{\longrightarrow}{\bot}}}} [\mathscr{M}_\bot^{op},\mathscr{CPO}_{\bot!}] \xleftarrow{Y} \mathscr{M}_\bot \longleftarrow \mathscr{M}$$

is a sound model of the ECLNL calculus extended with recursion.

Moreover, the concrete model enjoys a constructive property similar to the one above:

$$[\mathscr{M}_\bot^{\mathrm{op}}, \mathscr{CPO}_{\bot!}](\llbracket\Phi\rrbracket, \llbracket T\rrbracket \multimap \llbracket U\rrbracket) \cong \mathscr{CPO}(X, \mathscr{M}_\bot(\llbracket T\rrbracket, \llbracket U\rrbracket))$$

Therefore, the interpretation of $\Phi; \emptyset \vdash m : T \multimap U$ corresponds to a Scott-continuous function from X to $\mathscr{M}_\bot(\llbracket T\rrbracket, \llbracket U\rrbracket)$. In other words, this is a family of *string diagram computations*, in the sense that every element is either a string diagram of **M** or a non-terminating computation.

Theorem 12. *The CLNL model* $\mathbf{CPO} \underset{U}{\overset{(-)_\bot}{\underset{\longleftarrow}{\overset{\longrightarrow}{\bot}}}} \mathbf{CPO}_{\bot!}$ *, where U is the forgetful functor, is a sound model for the diagram-free fragment of the ECLNL calculus with recursion. Moreover, the model is also computationally adequate. If $\emptyset \vdash m : P$ is a well-typed term, where P is an intuitionistic type, then:*

$$m \Downarrow \ \textit{iff}\ \ \llbracket \emptyset \vdash m : P \rrbracket \neq \bot\,.$$

References

[1] F. Bonchi, P. Sobocinski & F. Zanasi (2015): *Full Abstraction for Signal Flow Graphs*. In: *POPL*, ACM, pp. 515–526.

[2] B. Coecke & R. Duncan (2008): *Interacting Quantum Observables*. In: *ICALP (2)*, *Lecture Notes in Computer Science* 5126, Springer, pp. 298–310.

[3] S. Perdrix E. Jeandel & R. Vilmart (2017): *A Complete Axiomatisation of the ZX-Calculus for Clifford+T Quantum Mechanics*.

[4] A. S. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger & B. Valiron (2013): *Quipper: a scalable quantum programming language*. In: *PLDI*, ACM, pp. 333–342.

[5] A. Hadzihasanovic (2015): *A Diagrammatic Axiomatisation for Qubit Entanglement*. In: *LICS*, IEEE Computer Society, pp. 573–584.

[6] B. Lindenhovius, M. Mislove & V. Zamdzhiev (2018): *Enriching a Linear/Non-linear Lambda Calculus: A Programming Language for String Diagrams*. Available at `http://cs.tulane.edu/~vzamdzhi/preprint/lics18.pdf`. To appear in LICS 2018.

[7] J. Meseguer & U. Montanari (1988): *Petri Nets Are Monoids: A New Algebraic Foundation for Net Theory*. In: *LICS*, IEEE Computer Society, pp. 155–164.

[8] J. Paykin, R. Rand & S. Zdancewic (2017): *QWIRE: a core language for quantum circuits*. In: *POPL*, ACM, pp. 846–858.

[9] F. Rios & P. Selinger (2017): *A categorical model for a quantum circuit description language*. To appear in QPL 2017.

[10] P. Selinger (2011): *A Survey of Graphical Languages for Monoidal Categories*. *New Structures for Physics*.

[11] D. Thomas & P. Moorby (2008): *The Verilog Hardware Description Language*. Springer Science & Business Media.

[12] N. Zainalabedin (1997): *VHDL: Analysis and modeling of digital systems*. McGraw-Hill, Inc.