Semantics for a Lambda Calculus for String Diagrams

Bert Lindenhovius, Michael Mislove, Vladimir Zamdzhiev

Semantics for a Lambda Calculus for String Diagrams

Bert Lindenhovius
Department of Computer Science
Tulane University

Michael Mislove
Department of Computer Science
Tulane University

Vladimir Zamdzhiev
Université de Lorraine, CNRS, Inria, LORIA

Linear/non-linear (LNL) models, as described by Benton, soundly model a LNL term calculus and LNL logic closely related to intuitionistic linear logic. Every such model induces a canonical enrichment that we show soundly models a 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 which corresponds to a modified version of Benton and Wadler's adjoint calculus with recursion.

In keeping with the purpose of the special issue, we also describe the influence of Samson Abramsky's research on these results, and on the overall project of which this is a part.

1 Dedication

We are pleased to contribute to this volume honoring SAMSON ABRAMSKY's many contributions to logic and its applications in theoretical computer science. This contribution is an extended version of the paper [28], which concerns the semantics of high-level functional quantum programming languages. This research is part of a project on the same general theme. The project itself would not exist if it weren't for Samson's support, guidance and participation. And, as we document in the final section of this contribution, Samson's research has had a direct impact on this work, and also points the way for further results along the general line we are pursuing. We are now in the fifth year of the project, and Samson has been a continual inspiration, both directly by his participation in the Tulane team, and indirectly by including us in a number of related activities he helped organize that exposed us to a variety of topics, all of which have been interesting and many of which have a direct connection to our research. We're happy to acknowledge his contributions, and we hope he enjoys reading about the research we report here.

1.1 Some personal comments

The first author first learned about the work of Samson via [2]. This work captures many features of quantum physics in a categorical framework, which highly impressed the first author, who soon discovered that this was just the tip of Samson's work. There are not many scientists as versatile as Samson, given the number of different subjects he made significant contributions to, and his courage to explore new areas is very inspiring. This author was of course a bit nervous when he finally met Samson in real life at the Simons Institute in Berkeley, but it turned out that Samson is also very friendly, and we had several enjoyable conversations about mathematics and computer science, but also about the life beyond science. The author hopes to learn much more from Samson in the future. Thank you for being such an inspiration!

The second author first met Samson in 1984 while spending a sabbatical at Oxford, when Samson was a member of the semantics group at Imperial College. Our paths have crossed often since then, at scientific meetings and during visits at each other's institutions. While we have never co-authored a paper together, we did edit a volume on Information Flow [5], which grew out of Samson's Clifford Lectures at Tulane in 1998, and the follow-on meetings that we jointly helped organize. This author is reminded of the many important discussions that took place during such meetings, often occurring on the inevitable "conference outing". The first such chat this author can recall was during a walk in the dunes at Asilomar during the 1989 LiCS meeting, a meeting highlighted by Dana's Sunday morning sermon. But by far the most vivid memory is the night canopy tour on the outing following the Costa Rica Informatic Phenomena meeting, during which Samson let out a totally uncharacteristic yell as he threw himself into the dark clinging to a zip line. We also learned a lot about sloths on that trip. Those talks always were stimulating and informative, and often a lot of fun. Thank you, Samson!

The third author first met Samson at his PhD interview in Oxford in 2012. The interview went very well and he then became a PhD student of Samson for the next four years. Over the course of the PhD, Samson provided very valuable advice and support, which of course, the author appreciates very much. After the end of this author's PhD, we have met many times in numerous other scientific events which was very enjoyable as always. Samson's many contributions to many different parts of logic are clearly very impressive and Samson himself has been a big inspirational figure throughout this author's career. Over the years, we have also spent a lot of time in social events and outings in many different places and in this way the third author got to know Samson outside of professional settings which was always very interesting and very fun. Thank you, Samson!

2 Introduction

In recent years string diagrams have found applications across a range of areas in computer science and related fields: in concurrency theory, where they are used to model Petri nets [32]; in systems theory, where they are used in a calculus of signal flow diagrams [10]; and in quantum computing [23, 13] where they represent quantum circuits and have been used to completely axiomatize the Clifford+T segment of quantum mechanics [15].

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. For example, Verilog [44] and VHDL [45] are popular hardware description languages that are used to generate very large digital circuits. More recently, the PNBml language [42] was developed to generate Petri nets, and Quipper [22] and QWIRE [33] are quantum programming languages (among others) that are used to generate (and execute) 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. However, we do not fix a particular application domain. Our development only assumes that the string diagrams we are working with enjoy a symmetric monoidal structure. Our goal is to help lay a foundation for programming languages that generate string diagrams, and that support the addition of extensions for specific application domains along with the necessary language features.

More generally, we believe the use of formal methods could aid us in obtaining a better conceptual understanding of how to design languages that can be used to construct and analyze large and complicated (families) of string diagrams.

Our Results. We study several calculi in this paper, beginning with the *combined LNL* (CLNL) calculus, which is the diagram-free fragment of our main language. The CLNL calculus, described in Section 3, can be seen as a modified version of Benton's LNL calculus, first defined in [8]. The crucial difference is that in CLNL we allow the use of mixed contexts, so there is only one type of judgement. This reduces the number of typing rules, and allows us to extend the language to support the generation of string diagrams. We also present a categorical model for our language, which is given by an LNL model with finite coproducts, and prove its soundness.

Next, in Section 4, we describe our main language of interest, the *enriched CLNL* calculus, which we denote ECLNL. The ECLNL calculus adopts the syntax and operational semantics of Proto-Quipper-M, a circuit description language introduced by Rios and Selinger [38], but we develop our own categorical model. Ours is the first *abstract* categorical model for the language, which is again given by an LNL model, but endowed with an additional *enrichment* structure. The enrichment is the reason we chose to rename the language. By design, ECLNL is an extension of the CLNL calculus that adds language features for manipulating string diagrams. We show that our abstract model satisfies the soundness and constructivity requirements (see [38], Remark 4.1) of Rios and Selinger's original model. As special instances of our abstract model, we recover the original model of Rios and Selinger, and we also present a simpler concrete model, as well as one that is order enriched.

In Section 5 we resolve the open problem posed by Rios and Selinger of extending the language with general recursion. We show that all the relevant language properties are preserved, and then we prove soundness for both the CLNL and ECLNL calculi with recursion, after first extending our abstract models with some additional structure. We then present concrete models for the ECLNL calculus that support recursion and also support generating string diagrams from *any* symmetric monoidal category. We conclude the section with a concrete model for the CLNL calculus extended with recursion that we also prove is computationally adequate at non-linear types.

In Section 6, we conclude the paper and discuss further possible developments, such as adding inductive and recursive types, as well as a treatment of dependent types.

Related Work. Categorical models are fundamental for our results, and the ones we present rely on the LNL models first described by Benton in [8]. Our work also is inspired by the language Proto-Quipper-M [38] by Rios and Selinger, the latest of the circuit description languages Selinger and his group have been developing. Our ECLNL calculus has the same syntax and operational semantics as Proto-Quipper-M, but there are significant differences in the denotational models. Rios and Selinger start with a symmetric monoidal category $\mathbf{M}$, then they consider a fully faithful strong symmetric monoidal embedding of $\mathbf{M}$ into another category $\overline{\mathbf{M}}$ that has some suitable categorical structure (e.g. $\overline{\mathbf{M}} := [\mathbf{M}^{\text{op}}, \mathbf{Set}]$), so that the category $\mathbf{Fam}(\overline{\mathbf{M}})$ is symmetric monoidal closed and contains $\mathbf{M}$. Their model is then given by the symmetric monoidal adjunction between $\mathbf{Set}$ and $\mathbf{Fam}(\overline{\mathbf{M}})$, which allows them to distinguish "parameter" (non-linear) terms and "state" (linear) terms. They show their language is type safe, their semantics is sound, and they remark that it also is computationally adequate at observable types (there is no recursion, so all programs terminate). The semantics for our ECLNL calculus enjoys the same properties, but we present both an abstract model and a simpler concrete model that doesn't involve a $\mathbf{Fam}(-)$ construction. Moreover, we also describe an extension with recursion, based on ideas by Benton and Wadler [7], and present an adequacy result for the diagram-free fragment of the language.

QWIRE [33] also is a language for reasoning about quantum circuits. QWIRE is really two languages, a non-linear host language and a quantum circuits language. QWIRE led Rennela and Staton to consider a more general language Ewire [36, 37], which can be used to describe circuits that are not necessarily quantum. Ewire supports dynamic lifting, and they prove a soundness result assuming the reduction system for the non-linear language is normalizing. They also discuss extending Ewire with conditional branching and inductive types over the $\otimes$- and $\oplus$-connectives (but not $\multimap$). However, these extensions require imposing additional structure on the diagrams, such as the existence of coproducts and fold/unfold gates. In our approach, we assume only that the diagrams enjoy a symmetric monoidal structure. In addition, our language also supports general recursion, whereas Ewire does not. An important similarity is that Ewire also makes use of enriched category theory to describe the denotational model.

Aside from Ewire and Proto-Quipper-M, the other languages we mentioned cannot generate arbitrary string diagrams, and some of them do not have a formal denotational semantics.

3 An alternative LNL calculus

LNL models were introduced by Benton [8] as a means to soundly model an interesting LNL calculus together with a corresponding logic. The goal was to understand the relationship between intuitionistic logic and intuitionistic linear logic. In this section, we show that LNL models also soundly model a variant of the LNL calculus where, instead of having two distinct typing judgements (linear and non-linear), there is a single type of judgement whose context is allowed to be mixed. A similar idea was briefly discussed by Benton in his original paper [8]. The syntax and operational semantics for this language are derived as a special case of the language of Rios and Selinger [38]. We denote the resulting language by CLNL, which we call the "Combined LNL" calculus.

As with the other calculi we consider, we begin our discussion by first describing a categorical model for CLNL. This makes the presentation of the language easier to follow. A categorical model of the CLNL calculus is given by an LNL model with finite coproducts, as the next definition shows.

Definition 3.1 ([8])**.** A *model of the CLNL calculus* (CLNL model) is given by the following data: a cartesian closed category (CCC) with finite coproducts $(\mathbf{V}, \times, \to, 1, \coprod, \emptyset)$; a symmetric monoidal closed category (SMCC) with finite coproducts $(\mathbf{C}, \otimes, \multimap, I, +, 0)$; and a symmetric monoidal adjunction:

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

We also adopt the following notation:

- The comonad-endofunctor is $! := F \circ G$.
- The unit of the adjunction $F \dashv G$ is $\eta : \mathrm{Id} \stackrel{.}{\longrightarrow} G \circ F$.
- The counit of the adjunction $F \dashv G$ is $\varepsilon : ! \stackrel{.}{\longrightarrow} \mathrm{Id}$.

Throughout the remainder of this section, we consider an arbitrary, but fixed, CLNL model. The CLNL calculus, which we introduce next, is interpreted in the category $\mathbf{C}$.

The syntax of the CLNL calculus is presented in Figure 1. It is exactly the diagram-free fragment of the ECLNL calculus, and because of space reasons, we only show the typing rules for ECLNL. However, the typing rules of the CLNL calculus can be easily derived from those for ECLNL by ignoring the Q label contexts (see the (pair) rule example below). Of course, ECLNL has some additional terms not in CLNL, so the corresponding typing rules should be ignored as well.

Observe that the non-linear types are a subset of the types of our language. Note also that there is no grammar which defines linear types. We say that a type that is not non-linear is *linear*. This definition is strictly speaking not necessary, but it helps to illustrate some concepts. In particular, any type $A \multimap B$ is therefore considered to be linear, even if A and B are non-linear. The interpretation of a type A is an object $[\![A]\!]$ of $\mathbf{C}$, defined by induction in the usual way (Figure 2).

Recall that in an LNL model with coproducts, we have:

$$I \cong F(1); \qquad 0 \cong F(\emptyset);$$

$$F(X) \otimes F(Y) \cong F(X \times Y); \qquad F(X) + F(Y) \cong F(X \coprod Y)$$

because F is strong (symmetric) monoidal and also a left adjoint. Based on these observations, we can define a non-linear interpretation $(\!|P|\!) \in \mathbf{V}$ of non-linear types P by induction in the following way:

$$\begin{aligned}(\!|I|\!) &:= 1;\\ (\!|0|\!) &:= \emptyset;\\ (\!|P \otimes Q|\!) &:= (\!|P|\!) \times (\!|Q|\!);\\ (\!|P + Q|\!) &:= (\!|P|\!) \coprod (\!|Q|\!);\\ (\!|!A|\!) &:= G[\![A]\!].\end{aligned}$$

Then a simple induction argument shows the next proposition.

Proposition 3.2. *For every non-linear type P, there is a canonical isomorphism* $\iota_P : [\![P]\!] \cong F(\!|P|\!)$.

A *context* is a function from a finite set of variables to types. 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. Its interpretation is as usual $[\![\Gamma]\!] = [\![A_1]\!] \otimes \cdots \otimes [\![A_n]\!]$. A variable in a context is non-linear (linear) if it is assigned a non-linear (linear) type. A context that contains only non-linear variables is called a *non-linear context*. Note, that we do not define linear contexts, because our typing rules refer only to contexts that either are non-linear or arbitrary (mixed).

A typing judgement has the form $\Gamma \vdash m : A$, where Γ is an (arbitrary) context, m is a term and A is a type. Its interpretation is a morphism $[\![\Gamma \vdash m : A]\!] : [\![\Gamma]\!] \to [\![A]\!]$ in $\mathbf{C}$, defined by induction on the derivation. For the typing rules of CLNL, the label contexts Q, Q', etc. from Figure 1 should be ignored. For example, the (pair) rule in CLNL becomes:

$$\frac{\Phi, \Gamma_1 \vdash m : A \qquad \Phi, \Gamma_2 \vdash n : B}{\Phi, \Gamma_1, \Gamma_2 \vdash \langle m, n \rangle : A \otimes B}\ \text{(pair)}$$

The type system enforces that a linear variable is used exactly once, whereas a non-linear variable may be used any number of times, including zero. Unlike Benton's LNL calculus, derivations in CLNL are in general not unique, because non-linear variables may be part of an arbitrary context Γ. For example, if P_1 and P_2 are non-linear types, then:

$$\frac{x : P_1 \vdash x : P_1 \qquad y : P_2 \vdash y : P_2}{x : P_1, y : P_2 \vdash \langle x, y \rangle : P_1 \otimes P_2}\ \text{(pair)}$$

$$\frac{x : P_1 \vdash x : P_1 \qquad x : P_1, y : P_2 \vdash y : P_2}{x : P_1, y : P_2 \vdash \langle x, y \rangle : P_1 \otimes P_2}\ \text{(pair)}$$

The CLNL Calculus

Variables	x,y,z		
Types	A,B,C	::=	$0 \mid A+B \mid I \mid A\otimes B \mid A \multimap B \mid !A$
Non-linear types	P,R	::=	$0 \mid P+R \mid I \mid P\otimes R \mid !A$
Variable contexts	Γ	::=	$x_1 : A_1, x_2 : A_2, \dots, x_n : A_n$
Non-linear variable contexts	Φ	::=	$x_1 : P_1, x_2 : P_2, \dots, x_n : P_n$
Terms	m,n,p	::=	$x \mid c \mid \text{let } x = m \text{ in } n \mid \Box_C m \mid \text{left}_{A,B} m \mid \text{right}_{A,B} m \mid$ $\text{case } m \text{ of } \{\text{left } x \to n \mid \text{right } y \to p\} \mid * \mid m;n \mid \langle m,n\rangle \mid$ $\text{let } \langle x,y\rangle = m \text{ in } n \mid \lambda x^A.m \mid mn \mid \text{lift } m \mid \text{force } m$
Values	v,w	::=	$x \mid c \mid \text{left}_{A,B} v \mid \text{right}_{A,B} v \mid * \mid \langle v,w\rangle \mid \lambda x^A.m \mid \text{lift } m$
Term Judgements	$\Gamma \vdash m : A$		(typing rules below - ignore Q contexts)

The ECLNL Calculus

Extend the CLNL syntax with:

Labels	ℓ, k		
Labelled string diagrams	S,D		
Types	A,B,C	::=	$\cdots \mid \alpha \mid \text{Diag}(T,U)$
Non-linear types	P,R	::=	$\cdots \mid \text{Diag}(T,U)$
M-types	T,U	::=	$\alpha \mid I \mid T\otimes U$
Label contexts	Q	::=	$\ell_1 : \alpha_1, \ell_2 : \alpha_2, \dots, \ell_n : \alpha_n$
Terms	m,n,p	::=	$\cdots \mid \ell \mid \text{box}_T m \mid \text{apply}(m,n) \mid (\vec{\ell}, S, \vec{\ell'})$
Label tuples	$\vec{\ell}, \vec{k}$	::=	$\ell \mid * \mid \langle \vec{\ell}, \vec{k}\rangle$
Values	v,w	::=	$\cdots \mid \ell \mid (\vec{\ell}, S, \vec{\ell'})$
Configurations	(S,m)		
Term Judgements	$\Gamma;Q \vdash m : A$		
Configuration Judgements	$Q \vdash (S,m) : A;Q'$		(see Definition 4.6)

The Typing Rules

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

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

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

$$\frac{\Phi,\Gamma_1;Q_1 \vdash m:I \qquad \Phi,\Gamma_2;Q_2 \vdash n:C}{\Phi,\Gamma_1,\Gamma_2;Q_1,Q_2 \vdash m;n : C}\text{(seq)} \qquad \frac{\Phi,\Gamma_1;Q_1 \vdash m:A \qquad \Phi,\Gamma_2,x:A;Q_2 \vdash n:B}{\Phi,\Gamma_1,\Gamma_2;Q_1,Q_2 \vdash \text{let } x = m \text{ in } n : B}\text{(let)}$$

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

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

$$\frac{\Gamma;Q \vdash m : !A}{\Gamma;Q \vdash \text{force } m : A}\text{(force)} \qquad \frac{\Gamma;Q \vdash m : !(T \multimap U)}{\Gamma;Q \vdash \text{box}_T m : \text{Diag}(T,U)}\text{(box)}$$

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

Figure 1: Syntax of the CLNL and ECLNL calculi.

$$[\![\alpha]\!] := E([\![\alpha]\!]_{\mathbf{M}}); \qquad [\![0]\!] := 0; \qquad [\![A+B]\!] := [\![A]\!] + [\![B]\!] \qquad [\![I]\!] := I \qquad [\![A \otimes B]\!] := [\![A]\!] \otimes [\![B]\!]$$

$$[\![A \multimap B]\!] := [\![A]\!] \multimap [\![B]\!] \qquad [\![!A]\!] := ![\![A]\!] \qquad [\![\mathrm{Diag}(T,U)]\!] := F(\mathscr{C}([\![T]\!],[\![U]\!]))$$

$$[\![\Phi, x : A; \emptyset \vdash x : A]\!] := [\![\Phi]\!] \otimes [\![A]\!] \xrightarrow{\diamond \otimes \mathrm{id}} I \otimes [\![A]\!] \xrightarrow{\cong} [\![A]\!]$$

$$[\![\Phi; \ell : \alpha \vdash \ell : \alpha]\!] := [\![\Phi]\!] \otimes [\![\alpha]\!] \xrightarrow{\diamond \otimes \mathrm{id}} I \otimes [\![\alpha]\!] \xrightarrow{\cong} [\![\alpha]\!]$$

$$[\![\Phi; \emptyset \vdash c : A_c]\!] := [\![\Phi]\!] \xrightarrow{\diamond} I \xrightarrow{[\![c]\!]} [\![A_c]\!]$$

$$\begin{aligned}[\![\Phi, \Gamma_1, \Gamma_2; Q_1, Q_2 \vdash \mathrm{let}\ x = m\ \mathrm{in}\ n : B]\!] &:= [\![\Phi]\!] \otimes [\![\Gamma_1]\!] \otimes [\![\Gamma_2]\!] \otimes [\![Q_1]\!] \otimes [\![Q_2]\!] \\ &\xrightarrow{\Delta \otimes \mathrm{id}} [\![\Phi]\!] \otimes [\![\Phi]\!] \otimes [\![\Gamma_1]\!] \otimes [\![\Gamma_2]\!] \otimes [\![Q_1]\!] \otimes [\![Q_2]\!] \xrightarrow{\cong} [\![\Phi]\!] \otimes [\![\Gamma_2]\!] \otimes [\![\Phi]\!] \otimes [\![\Gamma_1]\!] \otimes [\![Q_1]\!] \otimes [\![Q_2]\!] \\ &\xrightarrow{\mathrm{id} \otimes [\![m]\!] \otimes \mathrm{id}} [\![\Phi]\!] \otimes [\![\Gamma_2]\!] \otimes [\![A]\!] \otimes [\![Q_2]\!] \xrightarrow{[\![n]\!]} [\![B]\!]\end{aligned}$$

$$[\![\Gamma; Q \vdash \Box_C m : C]\!] := [\![\Gamma]\!] \otimes [\![Q]\!] \xrightarrow{[\![m]\!]} 0 \xrightarrow{!} C$$

$$[\![\Gamma; Q \vdash \mathrm{left}_{A,B} m : A+B]\!] := [\![\Gamma]\!] \otimes [\![Q]\!] \xrightarrow{[\![m]\!]} [\![A]\!] \xrightarrow{\mathrm{left}} [\![A]\!] + [\![B]\!] \xrightarrow{=} [\![A+B]\!]$$

$$[\![\Gamma; Q \vdash \mathrm{right}_{A,B} m : A+B]\!] := [\![\Gamma]\!] \otimes [\![Q]\!] \xrightarrow{[\![m]\!]} [\![B]\!] \xrightarrow{\mathrm{right}} [\![A]\!] + [\![B]\!] \xrightarrow{=} [\![A+B]\!]$$

$$[\![\Phi; \emptyset \vdash * : I]\!] := [\![\Phi]\!] \xrightarrow{\diamond} I$$

$$\begin{aligned}[\![\Phi, \Gamma_1, \Gamma_2; Q_1, Q_2 \vdash \mathrm{case}\ m\ \mathrm{of}\ \{\mathrm{left}\ x \to n \mid \mathrm{right}\ y \to p\} : C]\!] &:= [\![\Phi]\!] \otimes [\![\Gamma_1]\!] \otimes [\![\Gamma_2]\!] \otimes [\![Q_1]\!] \otimes [\![Q_2]\!] \\ &\xrightarrow{\Delta \otimes \mathrm{id}} [\![\Phi]\!] \otimes [\![\Phi]\!] \otimes [\![\Gamma_1]\!] \otimes [\![\Gamma_2]\!] \otimes [\![Q_1]\!] \otimes [\![Q_2]\!] \xrightarrow{\cong} [\![\Phi]\!] \otimes [\![\Gamma_2]\!] \otimes [\![\Phi]\!] \otimes [\![\Gamma_1]\!] \otimes [\![Q_1]\!] \otimes [\![Q_2]\!] \\ &\xrightarrow{\mathrm{id} \otimes [\![m]\!] \otimes \mathrm{id}} [\![\Phi]\!] \otimes [\![\Gamma_2]\!] \otimes [\![A+B]\!] \otimes [\![Q_2]\!] \xrightarrow{\cong} ([\![\Phi]\!] \otimes [\![\Gamma_2]\!] \otimes [\![A]\!] \otimes [\![Q_2]\!]) + ([\![\Phi]\!] \otimes [\![\Gamma_2]\!] \otimes [\![B]\!] \otimes [\![Q_2]\!]) \\ &\xrightarrow{[[\![n]\!],[\![p]\!]]} [\![C]\!]\end{aligned}$$

$$\begin{aligned}[\![\Phi, \Gamma_1, \Gamma_2; Q_1, Q_2 \vdash m; n : C]\!] &:= [\![\Phi]\!] \otimes [\![\Gamma_1]\!] \otimes [\![\Gamma_2]\!] \otimes [\![Q_1]\!] \otimes [\![Q_2]\!] \\ &\xrightarrow{\Delta \otimes \mathrm{id}} [\![\Phi]\!] \otimes [\![\Phi]\!] \otimes [\![\Gamma_1]\!] \otimes [\![\Gamma_2]\!] \otimes [\![Q_1]\!] \otimes [\![Q_2]\!] \xrightarrow{\cong} [\![\Phi]\!] \otimes [\![\Gamma_1]\!] \otimes [\![Q_1]\!] \otimes [\![\Phi]\!] \otimes [\![\Gamma_2]\!] \otimes [\![Q_2]\!] \\ &\xrightarrow{[\![m]\!] \otimes \mathrm{id}} I \otimes [\![\Phi]\!] \otimes [\![\Gamma_2]\!] \otimes [\![Q_2]\!] \\ &\xrightarrow{\cong} [\![\Phi]\!] \otimes [\![\Gamma_2]\!] \otimes [\![Q_2]\!] \xrightarrow{[\![n]\!]} [\![C]\!]\end{aligned}$$

$$\begin{aligned}[\![\Phi, \Gamma_1, \Gamma_2; Q_1, Q_2 \vdash \langle m, n \rangle : A \otimes B]\!] &:= [\![\Phi]\!] \otimes [\![\Gamma_1]\!] \otimes [\![\Gamma_2]\!] \otimes [\![Q_1]\!] \otimes [\![Q_2]\!] \\ &\xrightarrow{\Delta \otimes \mathrm{id}} [\![\Phi]\!] \otimes [\![\Phi]\!] \otimes [\![\Gamma_1]\!] \otimes [\![\Gamma_2]\!] \otimes [\![Q_1]\!] \otimes [\![Q_2]\!] \xrightarrow{\cong} [\![\Phi]\!] \otimes [\![\Gamma_1]\!] \otimes [\![Q_1]\!] \otimes [\![\Phi]\!] \otimes [\![\Gamma_2]\!] \otimes [\![Q_2]\!] \\ &\xrightarrow{[\![m]\!] \otimes [\![n]\!]} [\![A]\!] \otimes [\![B]\!]\end{aligned}$$

$$\begin{aligned}[\![\Phi, \Gamma_1, \Gamma_2; Q_1, Q_2 \vdash \mathrm{let}\ \langle x, y \rangle = m\ \mathrm{in}\ n : C]\!] &:= [\![\Phi]\!] \otimes [\![\Gamma_1]\!] \otimes [\![\Gamma_2]\!] \otimes [\![Q_1]\!] \otimes [\![Q_2]\!] \\ &\xrightarrow{\Delta \otimes \mathrm{id}} [\![\Phi]\!] \otimes [\![\Phi]\!] \otimes [\![\Gamma_1]\!] \otimes [\![\Gamma_2]\!] \otimes [\![Q_1]\!] \otimes [\![Q_2]\!] \xrightarrow{\cong} [\![\Phi]\!] \otimes [\![\Gamma_1]\!] \otimes [\![Q_1]\!] \otimes [\![\Phi]\!] \otimes [\![\Gamma_2]\!] \otimes [\![Q_2]\!] \\ &\xrightarrow{[\![m]\!] \otimes \mathrm{id}} [\![A \otimes B]\!] \otimes [\![\Phi]\!] \otimes [\![\Gamma_2]\!] \otimes [\![Q_2]\!] \xrightarrow{\cong} [\![\Phi]\!] \otimes [\![\Gamma_2]\!] \otimes [\![A]\!] \otimes [\![B]\!] \otimes [\![Q_2]\!] \xrightarrow{[\![n]\!]} [\![C]\!]\end{aligned}$$

$$[\![\Gamma; Q \vdash \lambda x^A . m : A \multimap B]\!] := [\![\Gamma]\!] \otimes [\![Q]\!] \xrightarrow{\mathbf{curry}([\![m]\!] \circ \cong)} [\![A]\!] \multimap [\![B]\!]$$

Figure 2: Denotational semantics of the ECLNL calculus

$$\llbracket \Phi,\Gamma_1,\Gamma_2;Q_1,Q_2 \vdash mn : B\rrbracket := \llbracket\Phi\rrbracket\otimes\llbracket\Gamma_1\rrbracket\otimes\llbracket\Gamma_2\rrbracket\otimes\llbracket Q_1\rrbracket\otimes\llbracket Q_2\rrbracket$$
$$\xrightarrow{\Delta\otimes\mathrm{id}} \llbracket\Phi\rrbracket\otimes\llbracket\Phi\rrbracket\otimes\llbracket\Gamma_1\rrbracket\otimes\llbracket\Gamma_2\rrbracket\otimes\llbracket Q_1\rrbracket\otimes\llbracket Q_2\rrbracket \xrightarrow{\cong} \llbracket\Phi\rrbracket\otimes\llbracket\Gamma_1\rrbracket\otimes\llbracket Q_1\rrbracket\otimes\llbracket\Phi\rrbracket\otimes\llbracket\Gamma_2\rrbracket\otimes\llbracket Q_2\rrbracket$$
$$\xrightarrow{\llbracket m\rrbracket\otimes\llbracket n\rrbracket} \llbracket A\multimap B\rrbracket\otimes\llbracket A\rrbracket \xrightarrow{\mathrm{ev}} \llbracket B\rrbracket$$

$$\llbracket \Phi;\emptyset \vdash \mathrm{lift}\, m : !A\rrbracket := \llbracket\Phi\rrbracket \xrightarrow{\textbf{lift}} !\llbracket\Phi\rrbracket \xrightarrow{!\llbracket m\rrbracket} !\llbracket A\rrbracket$$

$$\llbracket \Gamma;Q \vdash \mathrm{force}\, m : A\rrbracket := \llbracket\Gamma\rrbracket\otimes\llbracket Q\rrbracket \xrightarrow{\llbracket m\rrbracket} !\llbracket A\rrbracket \xrightarrow{\varepsilon} \llbracket A\rrbracket$$

$$\llbracket \Gamma;Q \vdash \mathrm{box}_T m : \mathrm{Diag}(T,U)\rrbracket := \llbracket\Gamma\rrbracket\otimes\llbracket Q\rrbracket \xrightarrow{\llbracket m\rrbracket} !(\llbracket T\rrbracket\multimap\llbracket U\rrbracket) \xrightarrow{\cong} \llbracket\mathrm{Diag}(T,U)\rrbracket$$

$$\llbracket \Phi,\Gamma_1,\Gamma_2;Q_1,Q_2 \vdash \mathrm{apply}(m,n) : U\rrbracket := \llbracket\Phi\rrbracket\otimes\llbracket\Gamma_1\rrbracket\otimes\llbracket\Gamma_2\rrbracket\otimes\llbracket Q_1\rrbracket\otimes\llbracket Q_2\rrbracket$$
$$\xrightarrow{\Delta\otimes\mathrm{id}} \llbracket\Phi\rrbracket\otimes\llbracket\Phi\rrbracket\otimes\llbracket\Gamma_1\rrbracket\otimes\llbracket\Gamma_2\rrbracket\otimes\llbracket Q_1\rrbracket\otimes\llbracket Q_2\rrbracket \xrightarrow{\cong} \llbracket\Phi\rrbracket\otimes\llbracket\Gamma_1\rrbracket\otimes\llbracket Q_1\rrbracket\otimes\llbracket\Phi\rrbracket\otimes\llbracket\Gamma_2\rrbracket\otimes\llbracket Q_2\rrbracket$$
$$\xrightarrow{\llbracket m\rrbracket\otimes\llbracket n\rrbracket} \llbracket\mathrm{Diag}(T,U)\rrbracket\otimes\llbracket T\rrbracket \xrightarrow{\cong} !(\llbracket T\rrbracket\multimap\llbracket U\rrbracket)\otimes\llbracket T\rrbracket \xrightarrow{\varepsilon\otimes\mathrm{id}} (\llbracket T\rrbracket\multimap\llbracket U\rrbracket)\otimes\llbracket T\rrbracket \xrightarrow{\mathrm{ev}} \llbracket U\rrbracket$$

$$\llbracket \Phi;\emptyset \vdash (\vec{\ell},S,\vec{\ell'}) : \mathrm{Diag}(T,U)\rrbracket := \llbracket\Phi\rrbracket \xrightarrow{\diamond} I \xrightarrow{\cong} F(1) \xrightarrow{F(\Psi(\phi(\vec{\ell},S,\vec{\ell'})))} \llbracket\mathrm{Diag}(T,U)\rrbracket$$

Figure 2: Denotational semantics of the ECLNL calculus (cont'd)

are two different derivations of the same judgement. While this might seem to be a disadvantage, it leads to a reduction in the number of rules, it allows a language extension that supports describing string diagrams (see Section 4), and it allows us to easily add general recursion (see Section 5). Moreover, the interpretation of any two derivations of the same judgement are equal (see Theorem 4.5).

Definition 3.3. A morphism $f : \llbracket P_1\rrbracket \to \llbracket P_2\rrbracket$ is called *non-linear*, if

$$f = \llbracket P_1\rrbracket \xrightarrow{\iota} F(\!| P_1 |\!) \xrightarrow{F(f')} F(\!| P_2 |\!) \xrightarrow{\iota^{-1}} \llbracket P_2\rrbracket,$$

for some $f' \in \mathbf{V}((\!| P_1 |\!),(\!| P_2 |\!))$.

Definition 3.4. We define maps on non-linear types as follows:

Discard: $\diamond_P := \llbracket P\rrbracket \xrightarrow{\iota} F(\!| P |\!) \xrightarrow{F1} F1 \xrightarrow{\cong} I;$

Copy: $\Delta_P := \llbracket P\rrbracket \xrightarrow{\iota} F(\!| P |\!) \xrightarrow{F(\langle \mathrm{id},\mathrm{id}\rangle)} F((\!| P |\!)\times(\!| P |\!)) \xrightarrow{\cong} F(\!| P |\!)\otimes F(\!| P |\!) \xrightarrow{\iota^{-1}\otimes\iota^{-1}} \llbracket P\rrbracket\otimes\llbracket P\rrbracket;$

Lift: $\mathbf{lift}_P := \llbracket P\rrbracket \xrightarrow{\iota} F(\!| P |\!) \xrightarrow{F\eta} !F(\!| P |\!) \xrightarrow{!\iota^{-1}} !\llbracket P\rrbracket.$

Proposition 3.5. *If $f : \llbracket P_1\rrbracket \to \llbracket P_2\rrbracket$ is non-linear, then:*

- $\diamond_{P_2}\circ f = \diamond_{P_1};$
- $\Delta_{P_2}\circ f = (f\otimes f)\circ\Delta_{P_1};$
- $\boldsymbol{lift}_{P_2}\circ f = !f\circ \boldsymbol{lift}_{P_1}.$

The operational and denotational semantics for the languages we discuss are presented in Figures 2 and 3. The rules for CLNL are obvious special cases of those for ECLNL (which we discuss in the next section). The evaluation rules for CLNL can be derived from those of ECLNL (Figure 3) by ignoring the diagram components. For example, the evaluation rule for (pair) is given by:

$$\frac{m \Downarrow v \qquad n \Downarrow v'}{\langle m,n\rangle \Downarrow \langle v,v'\rangle}$$

Similarly, the denotational interpretations of terms in CLNL can be derived from those of ECLNL (Figure 2) by ignoring the Q label contexts. For example, the interpretation of $[\![\Phi,\Gamma_1,\Gamma_2 \vdash \langle m,n\rangle : A\otimes B]\!]$ is given by the composition:

$$[\![\Phi]\!]\otimes[\![\Gamma_1]\!]\otimes[\![\Gamma_2]\!] \xrightarrow{\Delta\otimes\mathrm{id}} [\![\Phi]\!]\otimes[\![\Phi]\!]\otimes[\![\Gamma_1]\!]\otimes[\![\Gamma_2]\!] \xrightarrow{\cong} [\![\Phi]\!]\otimes[\![\Gamma_1]\!]\otimes[\![\Phi]\!]\otimes[\![\Gamma_2]\!] \xrightarrow{[\![m]\!]\otimes[\![n]\!]} [\![A]\!]\otimes[\![B]\!].$$

Theorem 3.6. *Theorems 4.5 – 4.10 also hold true when restricted to the CLNL calculus in the obvious way.*

4 Enriching the CLNL calculus

In this section we introduce the *enriched* CLNL calculus, ECLNL, whose syntax and operational semantics coincide with those of Proto-Quipper-M [38]. We rename the language in order to emphasize its dependence on its abstract categorical model, an LNL model with an associated *enrichment*. The categorical enrichment provides a natural framework for formulating the models we use, and for stating the constructivity properties (see Subsection 4.3) that we want our concrete models to satisfy.

We begin by briefly recalling the main ingredients of categories enriched over a symmetric monoidal closed category $(\mathbf{V},\otimes,\multimap,I)$:

- A $\mathbf{V}$-*enriched* category (briefly, a $\mathbf{V}$-category) $\mathscr{A}$ consists of a collection of objects; for each pair of objects A,B there is a 'hom' object $\mathscr{A}(A,B)\in\mathbf{V}$; for each object A, there is a 'unit' morphism $u_A : I\to\mathscr{A}(A,A)$ in $\mathbf{V}$; and given objects A,B,C, there is a 'composition' morphism $c_{ABC} : \mathscr{A}(A,B)\otimes\mathscr{A}(B,C)\to\mathscr{A}(A,C)$ in $\mathbf{V}$.
- A $\mathbf{V}$-*functor* $F:\mathscr{A}\to\mathscr{B}$ between $\mathbf{V}$-categories assigns to each object $A\in\mathscr{A}$ an object $FA\in\mathscr{B}$, and to each pair of objects $A,A'\in\mathscr{A}$ a $\mathbf{V}$-morphism $F_{AA'} : \mathscr{A}(A,A')\to\mathscr{B}(FA,FA')$;
- A $\mathbf{V}$-*natural transformation* between $\mathbf{V}$-functors $F,G:\mathscr{A}\to\mathscr{B}$ consists of $\mathbf{V}$-morphisms $\alpha_A : I\to\mathscr{B}(FA,GA)$ for each $A\in\mathscr{A}$;
- A $\mathbf{V}$-functor $F:\mathscr{A}\to\mathscr{B}$ has a right $\mathbf{V}$-*adjoint* $G:\mathscr{B}\to\mathscr{A}$ if there is a $\mathbf{V}$-isomorphism, $\mathscr{B}(FA,B)\cong\mathscr{A}(A,GB)$ that is $\mathbf{V}$-natural in both A and B;

The $\mathbf{V}$-morphisms that occur in these definitions are all subject to additional conditions expressed in terms of commuting diagrams in $\mathbf{V}$; for these we refer to [11, Chapter 6], which provides a detailed exposition on enriched category theory. We denote the category of $\mathbf{V}$-categories by $\mathbf{V}$-$\mathbf{Cat}$.

The first example of a $\mathbf{V}$-enriched category is the category $\mathscr{V}$ that has the same objects as $\mathbf{V}$ and whose hom objects are given by $\mathscr{V}(A,B)=A\multimap B$. We refer to this category as the *self-enrichment* of $\mathbf{V}$. If $\mathscr{A}$ is a $\mathbf{V}$-category, then the $\mathbf{V}$-*copower* of an object $A\in\mathscr{A}$ by an object $X\in\mathbf{V}$ is an object $X\odot A\in\mathscr{A}$ together with an isomorphism $\mathscr{A}(X\odot A,B)\cong\mathscr{V}(X,\mathscr{A}(A,B))$, which is $\mathbf{V}$-natural in B.

Any (lax) monoidal functor $G:\mathbf{C}\to\mathbf{V}$ between symmetric monoidal closed categories induces a *change of base* functor $G_*:\mathbf{C}\text{-}\mathbf{Cat}\to\mathbf{V}\text{-}\mathbf{Cat}$ assigning to each $\mathbf{C}$-category $\mathscr{A}$ a $\mathbf{V}$-category $G_*\mathscr{A}$ with the same objects as $\mathscr{A}$, but with hom objects given by $(G_*\mathscr{A})(A,B)=G\mathscr{A}(A,B)$. In particular, if $\mathbf{V}$ is locally small (which we always assume), then the functor $\mathbf{V}(I,-):\mathbf{V}\to\mathbf{Set}$ is a monoidal functor; the corresponding change of base functor assigns to each $\mathbf{V}$-category $\mathscr{A}$ its *underlying category*, which we denote with $\mathbf{A}$, i.e., the same letter but in boldface. We note that the underlying category of $\mathscr{V}$ is isomorphic to $\mathbf{V}$. Moreover, if the monoidal functor G above has a strong monoidal left adjoint, then the corresponding change of base functor maps $\mathbf{C}$-categories to $\mathbf{V}$-categories with isomorphic underlying categories, and $\mathbf{C}$-functors to $\mathbf{V}$-functors with the same underlying functors (up to the isomorphisms between the underlying categories). If $\mathbf{V}$ has all coproducts, then $\mathbf{V}(I,-)$ has a left adjoint $V : \mathbf{Set} \to \mathbf{V}$ that is monoidal [11, Proposition 6.4.6]. Applying the corresponding change of base functor to a locally small category equips this category with the *free* $\mathbf{V}$-enrichment.

Symmetric monoidal categories can be generalized to $\mathbf{V}$-*symmetric monoidal* categories, where the monoidal structure is also enriched over $\mathbf{V}$ [31, §4]. It follows from [31, Proposition 6.3] that the functor G_* above maps $\mathbf{C}$-symmetric monoidal categories to $\mathbf{V}$-symmetric monoidal categories. If for each fixed $A \in \mathbf{V}$, the $\mathbf{V}$-functor $(- \otimes A)$ has a right $\mathbf{V}$-adjoint, denoted $(A \multimap -)$, then we call $\mathscr{A}$ a $\mathbf{V}$-symmetric monoidal *closed* category. We note that the $(- \otimes -)$ and $(- \multimap -)$ bifunctors on $\mathbf{V}$ can be *enriched* to $\mathbf{V}$-bifunctors on $\mathscr{V}$ (i.e., such that their underlying functors correspond to the original functors) such that $\mathscr{V}$ becomes a $\mathbf{V}$-symmetric monoidal closed category.

Finally, if $\mathbf{V}$ has finite products, a $\mathbf{V}$-category $\mathscr{A}$ is said to have $\mathbf{V}$-coproducts if it has an object 0 and for each $A,B \in \mathscr{A}$ there is an object $A+B \in \mathscr{A}$ together with isomorphisms

$$1 \cong \mathscr{A}(0,C), \quad \mathscr{A}(A,C) \times \mathscr{A}(B,C) \cong \mathscr{A}(A+B,C),$$

$\mathbf{V}$-natural in C.

Definition 4.1. 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: $\begin{tikzcd} \mathscr{V} \arrow[r, bend left, "- \odot I"] \arrow[r, phantom, "\perp"] & \mathscr{C} \arrow[l, bend left, "\mathscr{C}(I,-)"] \end{tikzcd}$, together with a CLNL model on the underlying adjunction.

We also adopt the following notation: F and G are the underlying functors of $(- \odot I)$ and $\mathscr{C}(I,-)$ respectively and we use the same notation for the underlying CLNL model as in Definition 3.1.

By definition, every enriched CLNL model is a CLNL model with some additional (enriched) structure. But as the next theorem shows, every CLNL model induces the additional enriched structure as well. The CCC $\mathbf{V}$ can be equipped with its self-enrichment $\mathscr{V}$ in a canonical way. The symmetric monoidal structure of the adjunction then allows us to equip the SMCC $\mathbf{C}$ with a $\mathbf{V}$-enrichment by making use of the induced change-of-base functors which stem from the adjunction. Then one can show that the now constructed $\mathbf{V}$-enriched category $\mathscr{C}$ has $\mathbf{V}$-copowers and the original adjunction enriches to a $\mathbf{V}$-enriched one. We conclude:

Theorem 4.2. *Every CLNL model induces an enriched CLNL model.*

Proof. Combine [16, Proposition 6.7] and [31, Theorem 11.2]. □

The following proposition will be useful when defining the semantics of our language.

Proposition 4.3. *In every enriched CLNL model:*

1. *There is a* $\mathbf{V}$*-natural isomorphism* $G(A \multimap B) \cong \mathscr{C}(A,B)$;
2. $!(A \multimap B) \cong F(\mathscr{C}(A,B))$.

3. *There is a natural isomorphism* $\Psi : \mathbf{C}(A,B) \cong \mathbf{V}(1, \mathscr{C}(A,B))$.

Proof.

$$\begin{array}{ll} (1.) & G(A \multimap B) = \mathscr{C}(I, A \multimap B) \cong \mathscr{C}(A,B); \\ (2.) & \text{Apply } F \text{ to } (1.); \\ (3.) & \mathbf{C}(A,B) \cong \mathbf{C}(I, A \multimap B) \cong \mathbf{C}(F1, A \multimap B) \cong \mathbf{V}(1, G(A \multimap B)) \cong \mathbf{V}(1, \mathscr{C}(A,B)). \end{array}$$ □

4.1 The String Diagram 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 [39][1]. 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 [15], is setting $\mathbf{M}$ to be a suitable category of ZX-calculus diagrams. If $\mathbf{M} = \mathbf{PNB}$, the category of Petri Nets with Boundaries [43], then our calculus may be used to generate such Petri nets.

As with CLNL, our discussion of ECLNL begins with its categorical model.

Definition 4.4. An *ECLNL model* is given by the following data:

- An enriched CLNL model (Definition 4.1);
- A symmetric monoidal category $(\mathbf{M}, \boxtimes, J)$ and a strong symmetric monoidal functor $E : \mathbf{M} \to \mathbf{C}$.

For the remainder of the section, we consider an arbitrary, but fixed, ECLNL model.

4.2 Syntax and Semantics

We first introduce new types in our syntax that correspond to the objects of $\mathbf{M}$. Using terminology introduced in [38], where string diagrams are referred to as *circuits*, we let W be a fixed set of *wire types*, and we assume there is an interpretation $[\![-]\!]_{\mathbf{M}} : W \to \mathrm{Ob}(\mathbf{M})$. We use $\alpha, \beta, \ldots$ to range over the elements of W. For a wire type α, we define the interpretation of α in $\mathbf{C}$ to be $[\![\alpha]\!] = E([\![\alpha]\!]_M)$. The grammar for $\mathbf{M}$-types is given in Figure 1, and we extend $[\![-]\!]_{\mathbf{M}}$ to $\mathbf{M}$-types in the obvious way.

To build more complicated string diagrams from simpler components, we need to refer to certain wires of the component diagrams, to specify how to compose them. This is accomplished by assigning *labels* to the wires of our string diagrams, as demonstrated in the following construction.

Let L be a countably infinite set of labels. We use letters ℓ, k to range over the elements of L. A *label context* is a function from a finite subset of L to W, which we write as $\ell_1 : \alpha_1, \ldots, \ell_n : \alpha_n$. We use $Q_1, Q_2, \ldots$ to refer to label contexts. To each label context $Q = \ell_1 : \alpha_1, \ldots, \ell_n : \alpha_n$, we assign an object of $\mathbf{M}$ given by $[\![Q]\!]_{\mathbf{M}} := [\![\alpha_1]\!]_{\mathbf{M}} \boxtimes \cdots \boxtimes [\![\alpha_n]\!]_{\mathbf{M}}$. If $Q = \emptyset$, then $[\![Q]\!]_{\mathbf{M}} = J$. We denote *label tuples* by $\vec{\ell}$ and $\vec{k}$; these are simply tuples of label terms built up using the (pair) rule.

We now define the category $\mathbf{M}_L$ of *labelled string diagrams*:

- The objects of $\mathbf{M}_L$ are label contexts Q.
- The morphisms of $\mathbf{M}_L(Q_1, Q_2)$ are exactly the morphisms of $\mathbf{M}([\![Q_1]\!]_{\mathbf{M}}, [\![Q_2]\!]_{\mathbf{M}})$.

[1]The interested reader can consult [39] for more information on string diagrammatic representations of morphisms.

$$\frac{}{(S,x) \Downarrow \text{Error}} \qquad \frac{}{(S,\ell) \Downarrow (S,\ell)} \qquad \frac{}{(S,c) \Downarrow (S,c)} \qquad \frac{(S,m) \Downarrow (S',v) \quad (S',n[v/x]) \Downarrow (S'',v')}{(S,\text{let } x = m \text{ in } n) \Downarrow (S'',v')}$$

$$\frac{(S,m) \Downarrow (S',v)}{(S,\Box m) \Downarrow \text{Error}} \qquad \frac{(S,m) \Downarrow (S',v)}{(S,\text{left } m) \Downarrow (S',\text{left } v)} \qquad \frac{(S,m) \Downarrow (S',v)}{(S,\text{right } m) \Downarrow (S',\text{right } v)}$$

$$\frac{(S,m) \Downarrow (S',\text{left } v) \quad (S',n[v/x]) \Downarrow (S'',v')}{(S,\text{case } m \text{ of } \{\text{left } x \to n | \text{right } y \to p\}) \Downarrow (S'',v')} \qquad \frac{(S,m) \Downarrow (S',\text{right } v) \quad (S',n[v/y]) \Downarrow (S'',v')}{(S,\text{case } m \text{ of } \{\text{left } x \to n | \text{right } y \to p\}) \Downarrow (S'',v')}$$

$$\frac{(S,m) \Downarrow \text{otherwise}}{(S,\text{case } m \text{ of } \{\text{left } x \to n | \text{right } y \to p\}) \Downarrow \text{Error}}$$

$$\frac{}{(S,*) \Downarrow (S,*)} \qquad \frac{(S,m) \Downarrow (S',*) \quad (S',n) \Downarrow (S'',v')}{(S,m;n) \Downarrow (S'',v')} \qquad \frac{(S,m) \Downarrow \text{otherwise}}{(S,m;n) \Downarrow \text{Error}}$$

$$\frac{(S,m) \Downarrow (S',v) \quad (S',n) \Downarrow (S'',v')}{(S,\langle m,n\rangle) \Downarrow (S'',\langle v,v'\rangle)}$$

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

$$\frac{}{(S,\lambda x.m) \Downarrow (S,\lambda x.m)}$$

$$\frac{(S,m) \Downarrow (S',\lambda x.m') \quad (S',n) \Downarrow (S'',v) \quad (S'',m'[v/x]) \Downarrow (S''',v')}{(S,mn) \Downarrow (S''',v')} \qquad \frac{(S,m) \Downarrow \text{otherwise}}{(S,mn) \Downarrow \text{Error}}$$

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

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

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

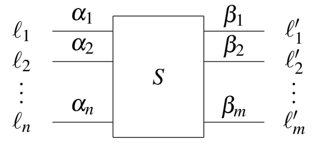
$$\frac{(S,m) \Downarrow (S',(\vec{\ell},D,\vec{\ell}')) \quad (S',n) \Downarrow (S'',\vec{k}) \quad \text{append}(S'',\vec{k},\vec{\ell},D,\vec{\ell}') = (S''',\vec{k}')}{(S,\text{apply}(m,n)) \Downarrow (S''',\vec{k}')}$$

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

Figure 3: Operational semantics of the ECLNL calculus

So, by construction, $[\![-]\!]_{\mathbf{M}} : \mathbf{M}_L \to \mathbf{M}$ is a full and faithful functor. Observe that if Q and Q' are label contexts that differ only by a renaming of labels, then $Q \cong Q'$. Moreover, for any two label contexts Q_1 and Q_2, by renaming labels we can construct $Q_1' \cong Q_1$ such that Q_1' and Q_2 are disjoint.

We equip the category $\mathbf{M}_L$ with the unique (up to natural isomorphism) symmetric monoidal structure that makes $[\![-]\!]_{\mathbf{M}}$ a symmetric monoidal functor. We then have $Q \otimes Q' \cong Q \cup Q'$ for any pair of disjoint label contexts. We use S, D to range over the morphisms of $\mathbf{M}_L$ and we visualise them in the following way:



where $S : \{\ell_1 : \alpha_1, \ldots, \ell_n : \alpha_n\} \to \{\ell_1' : \beta_1, \ldots, \ell_m' : \beta_m\} \in \mathbf{M}_L$ and $[\![S]\!]_{\mathbf{M}} : [\![\alpha_1]\!]_{\mathbf{M}} \boxtimes \cdots \boxtimes [\![\alpha_n]\!]_{\mathbf{M}} \to [\![\beta_1]\!]_{\mathbf{M}} \boxtimes \cdots \boxtimes [\![\beta_m]\!]_{\mathbf{M}} \in \mathbf{M}$.

A label context $Q = \ell_1 : \alpha_1, \ldots, \ell_n : \alpha_n$ is interpreted in $\mathbf{C}$ as $[\![Q]\!] = [\![\alpha_1]\!] \otimes \cdots \otimes [\![\alpha_n]\!]$ or by $[\![Q]\!] = I$ if $Q = \emptyset$. A labelled string diagram $S : Q \to Q'$ is interpreted in $\mathbf{C}$ as the composition:

$$[\![S]\!] := [\![Q]\!] \xrightarrow{\cong} E([\![Q]\!]_{\mathbf{M}}) \xrightarrow{E([\![S]\!]_{\mathbf{M}})} E([\![Q']\!]_{\mathbf{M}}) \xrightarrow{\cong} [\![Q']\!].$$

We also add the type $\mathrm{Diag}(T,U)$ to the language (see Figure 1); $\mathrm{Diag}(T,U)$ should be thought of as the type of string diagrams with inputs T and outputs U, where T and U are $\mathbf{M}$-types.

The term language is extended by adding the labels and label tuples just discussed, and the terms $\mathrm{box}_T m$, $\mathrm{apply}(m,n)$ and $(\vec{\ell}, S, \vec{\ell'})$. The term $\mathrm{box}_T m$ should be thought of as "boxing up" an already completed diagram m; $\mathrm{apply}(m,n)$ represents the application of the boxed diagram m to the state n; and the term $(\vec{\ell}, S, \vec{\ell'})$ is a value which represents a boxed diagram. Users of the ECLNL programming language are not expected to write labelled string diagrams S or terms such as $(\vec{\ell}, S, \vec{\ell'})$. Instead, these terms are computed by the programming language itself. Depending on the diagram model, the language should be extended with constants that are exposed to the user, for example, for quantum computing, a constant $H : \mathrm{Diag}(\mathbf{qubit}, \mathbf{qubit})$ could be utilised by the user to build quantum circuits.

The term typing judgements from the previous section are now extended to include a label context as well, which is separated from the variable context using a semicolon; the new format of a term typing judgement is $\Gamma; Q \vdash m : A$. Its interpretation is a morphism $[\![\Gamma]\!] \otimes [\![Q]\!] \to [\![A]\!]$ in $\mathbf{C}$ that is defined by induction on the derivation as shown in Figure 2.

In the definition of the (diag) rule in the denotational semantics, we use a function ϕ, which we now explain. From the premises of the rule, it follows that $[\![\vec{\ell}]\!] : [\![Q]\!] \to [\![T]\!]$ and $[\![\vec{\ell'}]\!] : [\![Q']\!] \to [\![U]\!]$ are isomorphisms. Then, $\phi(\vec{\ell}, S, \vec{\ell'})$ is defined to be the morphism:

$$\phi(\vec{\ell}, S, \vec{\ell'}) = [\![T]\!] \xrightarrow{[\![\vec{\ell}]\!]^{-1}} [\![Q]\!] \xrightarrow{[\![S]\!]} [\![Q']\!] \xrightarrow{[\![\vec{\ell'}]\!]} [\![U]\!].$$

Theorem 4.5. *Let D_1 and D_2 be derivations of a judgement $\Gamma; Q \vdash m : A$. Then $[\![D_1]\!] = [\![D_2]\!]$.*

Because of this theorem, we write $[\![\Gamma; Q \vdash m : A]\!]$ instead of $[\![D]\!]$.

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 which remains to be executed.

Definition 4.6. 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 $\emptyset; Q'' \vdash m : A$.

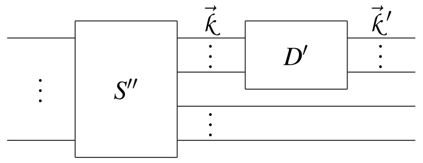
Thus, in a well-typed configuration, the term m has no free variables and its labels correspond to a subset of the outputs of S. We interpret a well-typed configuration $Q \vdash (S,m) : A; Q'$, by:

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

The big-step semantics is defined on configurations; because of space reasons, we only show an excerpt of the rules in Figure 3. The rest of the rules are standard. A *configuration value* is a configuration (S,v), where v is a value. The evaluation relation $(S,m) \Downarrow (S',v)$ then relates configurations to configuration values. Intuitively, this can be interpreted in the following way: assuming a constructed diagram S, then evaluating term m results in a diagram S' (obtained from S by appending other subdiagrams described by m) and value v. There's also an error relation $(S,m) \Downarrow$ Error which indicates that a run-time error occurs when we execute term m from configuration S. There are many such Error rules, but they are uninteresting, so we omit some of them (also see Theorem 4.8).

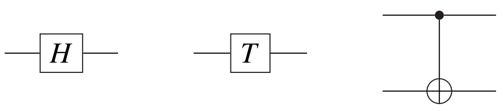
The operational semantics is presented in Figure 3. The evaluation rule for $\mathrm{box}_T m$ makes use of a function *freshlabels*. Given a **M**-type T, freshlabels(T) returns a pair $(Q,\vec{\ell})$ such that $\emptyset; Q \vdash \vec{\ell} : T$, where the labels in $\vec{\ell}$ are fresh in the sense that they do not occur anywhere else in the derivation. This can always be done, and the resulting Q and $\vec{\ell}$ are determined uniquely, up to a renaming of labels (which is inessential).

The evaluation rule for apply(m,n) makes use of a function *append*. Given a labelled string diagram S'' together with a label tuple $\vec{k}$ and term $(\vec{\ell},D,\vec{\ell}')$, it is defined as follows. Assuming that $\vec{\ell}$ and $\vec{k}$ correspond exactly to the inputs of D and that $\vec{\ell}'$ contains exactly the outputs of D, then we may construct a term $(\vec{k},D',\vec{k}')$ which is equivalent to $(\vec{\ell},D,\vec{\ell}')$ in the sense that they only differ by a renaming of labels. Moreover, we may do so by choosing D' and $\vec{k}'$ such that the labels in $\vec{k}'$ are fresh. Then, assuming the labels in $\vec{k}$ correspond to a subset of the outputs of S'', we may construct the labelled string diagram S''' given by the composition:



Finally, append$(S'',\vec{k},\vec{\ell},D,\vec{\ell}')$ returns the pair $(S''',\vec{k}')$ if the above assumptions are met, and is undefined otherwise (which would result in a run-time error).

Example 4.7. Let us assume that our string diagram model is given by quantum circuits. More precisely, assume that **M** contains at least the following diagram generators:

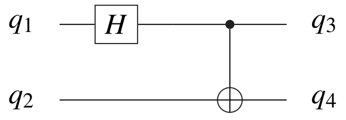


representing the diagrams that describe (from left to right): the Hadamard gate, the T gate and the CNOT gate of quantum computing. Assume further that the syntax of the language is extended with a built-in wire type **qubit** and term constants H : Diag(**qubit**, **qubit**), T : Diag(**qubit**, **qubit**) and $CNOT$: Diag(**qubit** $\otimes$ **qubit**, **qubit** $\otimes$ **qubit**) which are the user-exposed constants for constructing quantum circuits. Here we are overloading notation which hopefully should not lead to confusion.

Next, consider the term $m \equiv \mathrm{apply}(CNOT, \langle \mathrm{apply}(H,q_1),q_2 \rangle)$, where

$$\emptyset; q_1 : \mathbf{qubit}, q_2 : \mathbf{qubit} \vdash m : \mathbf{qubit} \otimes \mathbf{qubit}.$$

This represents a unitary operation on two qubits, where we first apply a Hadamard gate on the first qubit and then we apply a CNOT gate on the two qubits, with the first one being the control. If we add some (obvious) rules for dealing with constants in the operational semantics, then we may evaluate m when it appears as part of a configuration. For example, $(\mathrm{id}_{\mathbf{qubit}\otimes\mathbf{qubit}}, m) \Downarrow (S, \langle q_3, q_4 \rangle)$, where S is the labelled string diagram given by



where we have omitted annotating the types of the wires (which are all **qubit**).

Theorem 4.8 (Error freeness [38]). *If $Q \vdash (S,m) : A; Q'$ then $(S,m) \not\Downarrow$ Error.*

Theorem 4.9 (Subject reduction [38]). *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 may now show our abstract model is sound. We remark that our abstract model is strictly more general than the one of Rios and Selinger (see Section 2, ***Related Work***).

Theorem 4.10. *(Soundness) If $Q \vdash (S,m) : A; Q'$ and $(S,m) \Downarrow (S',v)$, then $[\![(S,m)]\!] = [\![(S',v)]\!]$.*

4.3 A constructive property

If we assume, in addition, that $E : \mathbf{M} \to \mathbf{C}$ is fully faithful, then setting $\mathcal{M}(T,U) := \mathscr{C}(ET, EU)$ for $T, U \in \mathbf{M}$ defines a $\mathbf{V}$-enriched category $\mathcal{M}$ 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 $\underline{E} : \mathcal{M} \to \mathscr{C}$. As a consequence, our abstract model enjoys the following constructive property:

$$\begin{aligned}
&\mathbf{C}([\![\Phi]\!], [\![T]\!] \multimap [\![U]\!]) \cong \mathbf{C}(F(X), [\![T]\!] \multimap [\![U]\!]) \cong \\
&\mathbf{V}(X, G([\![T]\!] \multimap [\![U]\!])) \cong \mathbf{V}(X, \mathscr{C}([\![T]\!], [\![U]\!])) \cong \\
&\mathbf{V}(X, \mathscr{C}(\underline{E}[\![T]\!]_{\mathbf{M}}, \underline{E}[\![U]\!]_{\mathbf{M}})) = \mathbf{V}(X, \mathcal{M}([\![T]\!]_{\mathbf{M}}, [\![U]\!]_{\mathbf{M}}))
\end{aligned}$$

where we use the additional structure only in the last step. 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 (Scott-continuous) function from X to $\mathcal{M}([\![T]\!]_{\mathbf{M}}, [\![U]\!]_{\mathbf{M}})$ or in other words, a (Scott-continuous) family of string diagrams from $\mathbf{M}$.

4.4 Concrete Models

The original concrete model of Rios and Selinger is now easily recovered as an instance of our abstract model:

$$\begin{tikzcd}
\mathbf{Set} \arrow[r, bend left, "- \odot I"] \arrow[r, phantom, "\perp"] & \mathbf{Fam}([\mathbf{M}^{\mathrm{op}}, \mathbf{Set}]) \arrow[l, bend left, "\mathbf{Fam}([\mathbf{M}^{\mathrm{op}}, \mathbf{Set}])(I, -)"] & {[\mathbf{M}^{\mathrm{op}}, \mathbf{Set}]} \arrow[l, hook'] & \mathbf{M} \arrow[l, hook', "Y"']
\end{tikzcd}$$

where $\mathbf{Fam}(-)$ is the well-known *families construction*. However, our abstract treatment of the language allows us to present a simpler sound model:

$$\begin{tikzcd}
\mathbf{Set} \arrow[r, bend left, "- \odot I"] \arrow[r, phantom, "\perp"] & {[\mathbf{M}^{\mathrm{op}}, \mathbf{Set}]} \arrow[l, bend left, "{[\mathbf{M}^{\mathrm{op}}, \mathbf{Set}](I, -)}"] & \mathbf{M} \arrow[l, hook', "Y"']
\end{tikzcd}$$

And, an order-enriched model is given by:

$$\begin{tikzcd} \mathcal{CPO} \arrow[r, bend left, "- \odot I"] \arrow[r, phantom, "\perp"] & {[\mathcal{M}^{\mathrm{op}}, \mathcal{CPO}]} \arrow[l, bend left, "{[\mathcal{M}^{\mathrm{op}}, \mathcal{CPO}](I,-)}"] & \mathcal{M} \arrow[l, hook', "Y"'] \end{tikzcd}$$

where $\mathcal{M}$ is the free **CPO**-enrichment of **M** (obtained by discretely ordering its homsets) and $\mathcal{CPO}$ is the self-enrichment of **CPO**.

5 The ECLNL calculus with recursion

Additional structure for Benton's LNL models needed to support recursion was discussed by Benton and Wadler in [7]. This structure allows them to model recursion in related lambda calculi, and in the LNL calculus (renamed the "adjoint calculus") as well. However, they present no syntax or operational semantics for recursion in their LNL calculus and instead they "... *omit the rather messy details*". Here we extend both the CLNL and ECLNL calculi with recursion in a simple way by using similar additional semantic structure. We conjecture the simplicity of our extension is due to our use of a single type of judgement that employs mixed contexts; this is the main distinguishing feature of our CLNL calculus compared to the LNL calculus of Benton and Wadler. Furthermore, we also include a computational adequacy result for the CLNL calculus with recursion.

5.1 Extension with recursion

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

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

Notice that in the typing rule, the label contexts are empty and all free variables in m are non-linear. As a special case, the CLNL calculus also can be extended with recursion:

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

In both cases, (parametrised) algebraic compactness of the !-endofunctor is what is needed to soundly model the extension; Benton and Wadler make the same assumption.

Definition 5.1. An endofunctor $T : \mathbf{C} \to \mathbf{C}$ is *algebraically compact* if T has an initial T-algebra $T(\Omega) \xrightarrow{\omega} \Omega$ for which $\Omega \xrightarrow{\omega^{-1}} T(\Omega)$ is a final T-coalgebra. If the category $\mathbf{C}$ is monoidal, then an endofunctor $T : \mathbf{C} \to \mathbf{C}$ is *parametrically algebraically compact* if the endofunctor $A \otimes T(-)$ is algebraically compact for every $A \in \mathbf{C}$.

We note that this notion of parametrised algebraic compactness is weaker than Fiore's corresponding notion [17], but it suffices for our purposes. This allows us to extend both ECLNL and CLNL models with recursion in the same way.

Definition 5.2. A model of the (E)CLNL calculus with recursion is given by a model of the (E)CLNL calculus for which the !-endofunctor is parametrically algebraically compact.

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

Figure 4: Definition of σ_m and γ_Φ.

If $\Phi \in \mathbf{C}$ is a non-linear object, then the endofunctor $\Phi\otimes!(-)$ is algebraically compact. Let $\Phi\otimes!\Omega_\Phi \xrightarrow{\omega_\Phi} \Omega_\Phi$ be its initial algebra and let $m : \Phi\otimes!A \to A$ be an arbitrary morphism. We define γ_Φ and σ_m to be the unique anamorphism and catamorphism, respectively, such that the diagram in Figure 4 commutes. Using this notation, we extend the denotational semantics to interpret recursion by adding the rule:

$$[\![\Phi;\emptyset \vdash \mathrm{rec}\ x^{!A}.m : A]\!] := \sigma_{[\![m]\!]} \circ \gamma_{[\![\Phi]\!]}.$$

Observe that when $\Phi = \emptyset$, we get:

$$[\![\mathrm{rec}\ x^{!A}.m]\!] = [\![m]\!] \circ ![\![\mathrm{rec}\ x^{!A}.m]\!] \circ \mathbf{lift} = [\![m]\!] \circ [\![\mathrm{lift}\ \mathrm{rec}\ x^{!A}.m]\!]$$

which is precisely a *linear fixpoint* in the sense of Braüner [12].

Theorem 5.3. *Theorems 4.5 – 4.10 from the previous section remain true for the (E)CLNL calculus extended with recursion.*

5.2 Concrete Models

Let **CPO** be the category of cpo's (possibly without bottom) and Scott-continuous functions, and let $\mathbf{CPO}_{\perp!}$ be the category of *pointed* cpo's and *strict* Scott-continuous functions.

We present a concrete model for an arbitrary symmetric monoidal **M**. Let $\mathcal{M}$ be the free **CPO**-enrichment of **M**. Then $\mathcal{M}$ has the same objects as **M** and hom-cpo's $\mathcal{M}(A,B)$ given by the hom-sets $\mathbf{M}(A,B)$ equipped with the discrete order. $\mathcal{M}$ is then a **CPO**-symmetric monoidal category with the same monoidal structure as **M**.

Let $\mathcal{M}_\perp$ be the free $\mathbf{CPO}_{\perp!}$-enrichment of **M**. Then, $\mathcal{M}_\perp$ has the same objects as **M** and hom-cpo's $\mathcal{M}_\perp(A,B) = \mathcal{M}(A,B)_\perp$, where $(-)_\perp : \mathcal{CPO} \to \mathcal{CPO}_{\perp!}$ is the domain-theoretic lifting functor. $\mathcal{M}_\perp$ is then a $\mathbf{CPO}_{\perp!}$-symmetric monoidal category with the same monoidal structure as that of $\mathcal{M}$ where, in addition, $\perp_{A,B}$ satisfies the conditions of Proposition 5.7 (see Section 5.3 below).

By using the enriched Yoneda lemma together with the Day convolution monoidal structure, we see that the enriched functor category $[\mathcal{M}_\perp^{\mathrm{op}}, \mathcal{CPO}_{\perp!}]$ is $\mathbf{CPO}_{\perp!}$-symmetric monoidal closed.

Theorem 5.4. *The following data:*

$$\begin{tikzcd} \mathcal{CPO} \arrow[r, bend left, "- \odot I"] \arrow[r, phantom, "\perp"] & {[\mathcal{M}_\perp^{op}, \mathcal{CPO}_{\perp !}]} \arrow[l, bend left, "{[\mathcal{M}_\perp^{op}, \mathcal{CPO}_{\perp !}](I,-)}"] & \mathcal{M}_\perp \arrow[l, hook', "Y"'] & \mathcal{M} \arrow[l, hook'] \end{tikzcd}$$

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

Proof. The subcategory inclusion $\mathcal{M} \hookrightarrow \mathcal{M}_\perp$ is **CPO**-enriched, faithful and strong symmetric monoidal, as is the enriched Yoneda embedding Y. The **CPO**-copower $(- \odot I)$ is given by:

$$(- \odot I) = (- \bullet I) \circ (-)_\perp,$$

where $(- \bullet I) : \mathcal{CPO}_{\perp !} \to [\mathcal{M}_\perp^{\mathrm{op}}, \mathcal{CPO}_{\perp !}]$ is the $\mathbf{CPO}_{\perp !}$-copower with the tensor unit (see [11]). This follows because the right adjoint and the adjunction factor through $\mathcal{CPO}_{\perp !}$. Parametrised algebraic compactness of the !-endofunctor follows from [17, pp. 161-162]. □

Moreover, the concrete model enjoys a constructive property similar to the one in Subsection 4.3. Using the same argument, if $\Phi; \emptyset \vdash m : T \multimap U$, then we obtain:

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

Therefore, the interpretation of m corresponds to a Scott-continuous function from X to $\mathcal{M}_\perp(\llbracket T \rrbracket_{\mathbf{M}}, \llbracket U \rrbracket_{\mathbf{M}})$. 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 5.5. *The CLNL model* $\begin{tikzcd} \mathbf{CPO} \arrow[r, bend left, "(-)_\perp"] \arrow[r, phantom, "\perp"] & \mathbf{CPO}_{\perp !} \arrow[l, bend left, "U"] \end{tikzcd}$ *, where U is the forgetful functor, is a sound model for the CLNL calculus with recursion.*

Proof. Again, parametrised algebraic compactness of the !-endofunctor follows from [17, pp. 161-162]. □

5.3 Computational adequacy

In this subsection we show that computational adequacy holds at non-linear types for the concrete CLNL model given in the previous subsection.

We begin by showing that in any (E)CLNL model with recursion, the category **C** is pointed, which allows us to introduce a notion of undefinedness. Towards that end, we first introduce a slightly weaker notion, following Braüner [12].

Definition 5.6. A symmetric monoidal closed category is *weakly pointed* if it is equipped with a morphism $\perp_A : I \to A$ for each object A, such that for every morphism $h : A \to B$, we have $h \circ \perp_A = \perp_B$. In this case, for each pair of objects A and B, there is a morphism $\perp_{A,B} = A \xrightarrow{\lambda_A^{-1}} I \otimes A \xrightarrow{\mathbf{uncurry}(\perp_{A \multimap B})} B$.

Proposition 5.7 ([12])**.** *Let* **A** *be a weakly pointed category. Then:*

1. *$f \circ \perp_{A,B} = \perp_{A,C}$ for each morphism $f : B \to C$;*
2. *$\perp_{B,C} \circ f = \perp_{A,C}$ for each morphism $f : A \to B$;*
3. *$\perp_{A,B} \otimes f = \perp_{A \otimes C, B \otimes D}$ for each morphism $f : C \to D$.*

4. *$f \otimes \perp_{A,B} = \perp_{C\otimes A, D\otimes B}$ for each morphism $f : C \to D$.*

Lemma 5.8. *Any weakly pointed category with an initial object* 0 *is pointed. Moreover, $\perp_A = \perp_{I,A}$ and $\perp_{A,B}$ are zero morphisms.*

Theorem 5.9. *For every model of the (E)CLNL calculus with recursion,* $\mathbf{C}$ *is a pointed category with*

$$\perp_A = I \xrightarrow{\gamma_I} \Omega_I \xrightarrow{\sigma_{\varepsilon_A}} A,$$

where Ω_I is the carrier of the initial algebra for the !-endofunctor.

Proof. It suffices to show for any $h : A \to B$ that $h \circ \perp_A = \perp_B$ which follows from the naturality of ε and initiality of σ_ε. □

In particular, we have: $[\![\emptyset;\emptyset \vdash \text{rec } x^{!A}.\text{force } x : A]\!] = \perp_{[\![A]\!]}$. Thus, the interpretation of the simplest non-terminating program (of any type) is a zero morphism, as one would expect. Naturally, we use the zero morphisms of $\mathbf{C}$ to denote undefinedness in our adequacy result.

Assume that $\mathscr{C}$ is $\mathbf{CPO}$-enriched and that $\perp_{A,B}$ is least in $\mathscr{C}(A,B)$. We shall use $\bigvee_i a_i$ to denote the supremum of the increasing chain $(a_i)_{i\in\mathbb{N}}$. For any Scott-continuous function $K : \mathscr{C}(A,B) \to \mathscr{C}(A,B)$, let $K^0 = \perp_{A,B}$ and $K^{i+1} = K(K^i)$, for $i \in \mathbb{N}$. Then $\bigvee_i K^i$ is the least fixpoint of K. Note that K isn't strict in general.

Lemma 5.10. *Consider an (E)CLNL model with recursion, where $\mathbf{V} = \mathbf{CPO}$ and where $\perp_{A,B}$ is least in $\mathscr{C}(A,B)$, for all objects A and B (or equivalently $\mathscr{C}$ is $\mathbf{CPO}_{\perp!}$-enriched). Let $m : \Phi \otimes !A \to A$ be a morphism in $\mathbf{C}$. Let K_m be the Scott-continuous function $K_m : \mathscr{C}(\Phi,A) \to \mathscr{C}(\Phi,A)$ given by $K_m(f) = m \circ (id \otimes !f) \circ (id \otimes \textbf{lift}) \circ \Delta$. Then:*

$$\sigma_m \circ \gamma_\Phi = \bigvee_i K_m^i.$$

The significance of this lemma is that it provides an equivalent semantic definition for the (rec) rule in terms of least fixpoints, provided we assume order-enrichment for our (E)CLNL models.

For the remainder of the section, we consider only the CLNL calculus which we interpret in the CLNL model of Theorem 5.5. Therefore, in what follows $\mathbf{C} = \mathbf{CPO}_{\perp!}$.

Lemma 5.11. *Let $\emptyset \vdash v : P$ be a well-typed value, where P is a non-linear type. Then $[\![\emptyset \vdash v : P]\!] \neq \perp$.*

Next, we prove adequacy using the standard method based on *formal approximation relations*, a notion first devised by Plotkin [35].

Definition 5.12. For any type A, let:

$$V_A := \{v \mid v \text{ is a value and } \emptyset \vdash v : A\};$$
$$T_A := \{m \mid \emptyset \vdash m : A\}.$$

We define two families of *formal approximation relations:*

$$\trianglelefteq_A \subseteq (\mathbf{C}(I, [\![A]\!]) - \{\perp\}) \times V_A$$
$$\sqsubseteq_A \subseteq \mathbf{C}(I, [\![A]\!]) \times T_A$$

by induction on the structure of A:

(A1) $f \trianglelefteq_I *$ iff $f = \text{id}_I$;

(A2.1) $f \trianglelefteq_{A+B} \text{left } v$ iff $\exists f'.\ f = \text{left} \circ f'$ and $f' \trianglelefteq_A v$;

(A2.2) $f \trianglelefteq_{A+B}$ right v iff $\exists f'.\ f = \text{right} \circ f'$ and $f' \trianglelefteq_B v$;

(A3) $f \trianglelefteq_{A\otimes B} \langle v, w\rangle$ iff $\exists f', f''$, such that:
$f = f' \otimes f'' \circ \lambda_I^{-1}$ and $f' \trianglelefteq_A v$ and $f'' \trianglelefteq_B w$;

(A4) $f \trianglelefteq_{A\multimap B} \lambda x.\ m$ iff $\forall f' \in \mathbf{C}(I, [\![A]\!]), \forall v \in V_A$:

$$f' \trianglelefteq_A v \Rightarrow \text{eval} \circ (f \otimes f') \circ \lambda_I^{-1} \sqsubseteq_B m[v/x];$$

(A5) $f \trianglelefteq_{!A}$ lift m iff f is a non-linear morphism and
$\varepsilon_A \circ f \sqsubseteq_A m$;

(B) $f \sqsubseteq_A m$ iff $f \neq \perp\ \Rightarrow\ \exists v \in V_A.\ m \Downarrow v$ and $f \trianglelefteq_A v$.

So, the relation $\trianglelefteq$ relates morphisms to values and $\sqsubseteq$ relates morphisms to terms.

Lemma 5.13. *If $f \trianglelefteq_P v$, where P is a non-linear type, then f is a non-linear morphism.*

Lemma 5.14. *For any $m \in T_A$, the property $(- \sqsubseteq_A m)$ is admissible for the (pointed) cpo $\mathscr{C}(I, [\![A]\!])$ in the sense that Scott fixpoint induction is sound.*

Proof. One has to show $\perp \sqsubseteq_A m$, which is trivial, and also that $(- \sqsubseteq_A m)$ is closed under suprema of increasing chains of morphisms, which is easily proven by induction on A. □

Proposition 5.15. *Let $\Gamma \vdash m : A$, where $\Gamma = x_1 : A_1, \ldots, x_n : A_n$. Let $v_i \in V_{A_i}$ such that $f_i \trianglelefteq_{A_i} v_i$. If f is the composition:*

$$f := I \xrightarrow{\cong} I \otimes \cdots \otimes I \xrightarrow{f_1 \otimes \cdots \otimes f_n} [\![\Gamma]\!] \xrightarrow{[\![\Gamma \vdash m:A]\!]} [\![A]\!],$$

then $f \sqsubseteq_A m[\overline{v}\ /\ \overline{x}]$.

Proof. By induction on the derivation of m. For the (rec) case, one should use Lemma 5.14 and Lemma 5.10. □

Definition 5.16. We shall say that a well-typed term m *terminates*, in symbols $m \Downarrow$, iff there exists a value v, such that $m \Downarrow v$.

The next theorem establishes sufficient conditions for termination at *any* type.

Theorem 5.17 (Termination)**.** *Let $\emptyset \vdash m : A$ be a well-typed term. If $[\![\emptyset \vdash m : A]\!] \neq \perp$, then $m \Downarrow$.*

Proof. This is a special case of the previous proposition when $\Gamma = \emptyset$. We get $[\![\emptyset \vdash m : A]\!] \sqsubseteq_A m$, and thus $m \Downarrow$ by definition of $\sqsubseteq_A$. □

We can now finally state our adequacy result.

Theorem 5.18 (Adequacy)**.** *Let $\emptyset \vdash m : P$ be a well-typed term, where P is a non-linear type. Then:*

$$m \Downarrow \ \textit{iff} \ \ [\![\emptyset \vdash m : P]\!] \neq \perp .$$

Proof. The right-to-left direction follows from Theorem 5.17. The other direction follows from soundness and Lemma 5.11. □

The model of Theorem 5.5 was presented as an example by Benton and Wadler [7] for their LNL calculus extended with recursion, however without stating an adequacy result. We have now shown that it is computationally adequate at non-linear types for our CLNL calculus.

6 Conclusion

We begin by briefly reviewing our results. Starting from Benton's LNL calculus [8], we defined the CLNL calculus, which adopts a single combined typing judgement, as opposed to the more traditional approach with separate non-linear and linear typing judgements. Our approach is inspired by the language Proto-Quipper-M and its initial model [38], in which the combined typing judgement is meant to be a convenience for the programmer. We showed that these calculi — Benton's LNL and our CLNL – have the same categorical models, and we showed the CLNL calculus can be extended with recursion in the same categorical model, bringing rigor to the discussion in [7]. Next, we showed that the CLNL calculus can be extended with language features that turn it into a lambda calculus for string diagrams, which we named the ECLNL calculus (this is essentially Proto-Quipper-M [38]). We then identified the abstract models of ECLNL by considering the categorical enrichment of LNL models. Our abstract approach allowed us to identify concrete models that are simpler than those previously considered, and it also allowed us to extend the language with general recursion, thereby solving an open problem posed by Rios and Selinger. The enrichment structure also made it possible to easily establish the constructivity properties that are expected to hold of a string diagram description language. Finally, we proved an adequacy result for the CLNL calculus, which is the diagram-free fragment of the ECLNL calculus.

There are a number of ingredients in the development of these results that deserve further discussion:

- Categorical quantum mechanics;
- Linear logic and linear/non-linear models;
- Domain theory and support for recursion;
- String diagrammatic languages and calculi.

Our aim is to document Samson's influence on these developments, and on the direction of future research along these lines.

The first item on the list has yet to be mentioned. The work of Abramsky and Coecke [6] established the beginning of the program on *categorical quantum mechanics* which has played a crucial role in the development of logical methods for quantum information processing and has led to the development of quantum diagrammatic calculi, most notably, the ZX-calculus [14], which provide an alternative to the usual linear algebraic language for quantum computing. In this paper, we have also focused on a diagrammatic approach to quantum information processing where our language is used to generate (quantum) circuit diagrams. However, we only take into account the symmetric monoidal structure of the diagrams, whereas a large (and very important) part of categorical quantum mechanics focuses further on the compact closed structure of the diagrams which is a key component in the design of many quantum diagrammatic calculi (e.g. ZX-calculus). It will be interesting to see whether our work can be extended to also take into account the compact closed structure and we return to this point below when we discuss future work.

Soon after its introduction, Girard's linear logic [19] was hailed as the “logic behind logic" by logicians, and as a means for “controlling the use of resources" by theoretical computer scientists. Samson's seminal article [1] showed how to interpret linear logic from a computational perspective. In those early days, the focus was as much on the classical fragment, where the application was in the concurrent setting, as on the intuitionistic setting, where the, “...refinement of the lambda calculus" led to the linear lambda calculus. Samson showed the classical fragment yields a concurrent process paradigm with an operational semantics in the style of Berry and Boudol's chemical abstract machine [9]. His interpretation of the intuitionistic fragment allowed, “finer control over the order of evaluation and storage allocation, while maintaining the logical content of programs as proofs, interpreting computation as cut-elimination". Samson predicted that the result would, "...open up a promising new approach to the parallel implementation of functional programming languages; and offers the prospect of typed concurrent programming in which correctness is guaranteed by the typing," predictions that have largely succeeded. Moreover, our work in this paper on our linear/non-linear lambda calculus has also been heavily influenced by his work on the linear lambda calculus [1].

Samson contrasts the simple, concrete computational interpretation driven by syntax presented in [1] with Girard's geometric view of computation [21, 20]. He then expressed the hope that the results in [1] would lead to connections with Girard's approach. His later work realized this hope: the Geometry of Interaction was employed in [3] to give a game semantics solution to the longstanding full abstraction problem for PCF. In fact, this line of research had a profound effect on our understanding of computation as taking place between two equal partners: a process run by a user on some device, and "the environment", understood as what was happening outside the control of the individual user. While we have avoided the problem of full abstraction in this paper, this is still a very important property that should be considered as part of future work.

Early work on linear logic focused equally on its classical and its intuitionistic fragments, with each fragment having a non-linear analog. Over time, the computer science community focused more narrowly on the intuitionistic fragment, for obvious reasons. Benton and Wadler [7] were the first to consider LNL models from a programming language perspective. Several authors followed this with models of languages featuring both linear and non-linear types within the same language, but it was Selinger and Valiron [40] who first used LNL models in the context of quantum computing, albeit as a model for a call-by-value computational linear lambda calculus. That line of research includes the work on the quantum lambda calculus [41] and it runs up to to [38], which is a direct inspiration for our work.

One result of both logicians and computer scientists working in the same arena was an inevitable clash of terminology. Among computer scientists working on quantum computing, it is customary to use "classical" to refer to a computer relying on current technology, and "quantum" to refer to a device utilizing quantum systems to perform computations. In the logic community, "classical" is reserved for classical logic, as opposed to, e.g., intuitionistic logic. Likewise, when discussing linear/non-linear models and the associated lambda calculi, one sometimes hears "intuitionistic" used to refer to notions that are not linear. The resulting overloading of terminology can easily cause confusion. Our solution is to reserve "classical" to refer to physical notions from classical physics, and we use "linear" and "non-linear" to distinguish forms of lambda calculi and, likewise, primitives of programming languages. This works, of course, because we have virtually no need to refer to classical logic.

The use of domain theory for giving semantics to lambda calculi that feature both linear and non-linear fragments, and by extension, to programming languages that include both linear and non-linear features has been mainly in the form of category theory, and in particular, in categories enriched over directed complete partial orders. Indeed, little use has been made of continuous domains in this setting, and we doubt that any will emerge because of the eventual need to support monoidal closed categories that also are invariant under the valuations monad. The major impetus for semantic models has been type theory, where support for term recursion and recursive types depends heavily on the work of Freyd on algebraic completeness [18]. We have chosen to focus on (ω-) CPOs – partial orders in which increasing sequences have least upper bounds – because these allow simpler arguments (as typified, e.g., in [27]). Nevertheless, Samson has been a leader in the adoption of domain theory and category theory and their techniques in modeling logics and programming languages, and his Handbook chapter with Jung [4] on domain theory is a standard reference for the area. Indeed, we used it many times in the preparation of this paper.

7 Future Work

To discuss future work, we first need to mention extensions of this work that already have appeared. First, the papers [29, 30] consider the question of adding type recursion to Proto-Quipper-M (PQM). The approach is based on [17] on recursive types in FPC, and it presents an extension of FPC, the fixed point calculus originally due to Plotkin, in which recursive types are introduced to a linear/non-linear lambda calculus. The new metalanguage is called LNL-FPC, to indicate it is an extension of FPC to include linear primitives. The results include soundness and computational adequacy at non-linear types. The metalanguage LNL-FPC represents an extension of the circuit-free fragment of PQM.

A primary issue along this line is to extend the computational adequacy result to also include circuits. The problem has been the basic structure of models for PQM [38]. It starts with a symmetric monoidal category $\mathbf{M}$ whose morphisms are meant to represent quantum circuits (and perhaps additional morphisms). The next step is to form a certain categorical completion containing $\mathbf{M}$, which is accomplished via an enriched presheaf category such as $[\mathbf{M}^{\mathrm{op}}, \mathbf{CPO}_{\perp!}]$. But this presheaf category uses the Day convolution to define the tensor product, and the resulting tensor product is not order reflecting, a key hypothesis for the proof of computational adequacy in [29]. Recent work offers some promise for overcoming this problem. As reported at a recent POPL workshop [26], the first two authors have been collaborating with a colleague on a new model whose linear category has a tensor product that is order reflecting. We believe this may lead to a model in which computational adequacy can be shown to hold for an extension of PQM that includes circuits, and that also supports general recursion.

Going further, recent work by the third author and his colleagues [34] shows that computational adequacy at all types can be achieved in a model consisting of W*-algebras. Since the model in [26] has many similar features to the former one, we are hopeful this result can transfer to that setting.

Returning to the work of Abramsky and Coecke on categorical quantum mechanics [2], it is very important to consider an extension of our language which can accommodate compact closed structure. This paves the way for new applications of the language with quantum diagrammatic calculi, such as the ZX-calculus [14]. The work in [26] seems like a good foundation for building up such models.

Finally, the work in [26] has succeeded in many aspects, and we hope that the model can be extended to also support the execution of quantum circuits, and not just their generation (which is the case for the present paper). To accomplish that goal, we first need to incorporate state preparation in the model, and this depends in turn on adding a monad of probability measures / valuations as a computational effect. Our efforts to accomplish that have led to an interesting new result in the realm of “classical" domain theory. In [24] the second author and a colleague have devised a new monad of valuations on DCPOs for which the Fubini Theorem holds. The construction uses Keimel and Lawson’s D-completion of the simple valuations [25], and relies on showing this family admits a barycenter map for the algebras of the monad. Our hope is that this monad will show the way for adding state preparation to the model [26].

Acknowledgements. We thank Francisco Rios and Peter Selinger for many conversations about their work on Proto-Quipper-M, as well as on our work. We also thank Sam Staton for raising the question of why the families construction is needed in a model of Proto-Quipper-M (it isn’t). We also thank Samson Abramsky and Mathys Rennela for valuable conversations about this work during their recent visits to Tulane. We also thank the Simons Institute for the Theory of Computing where much of the initial portion of this work took place. This work was partially funded by the AFOSR under the MURI grant number FA9550-16-1-0082 entitled, "Semantics, Formal Reasoning, and Tool Support for Quantum Programming".

References

[1] S. Abramsky (1993): *Computational interpretations of linear logic*. Theoretical Computer Science 111, pp. 3–57.

[2] S. Abramsky & B. Coecke (2009): *Handbook of Quantum Logic and Quantum Structures, Quantum Logic*, chapter Categorical Quantum Mechanics, pp. 261–324. Elsevier.

[3] S. Abramsky, R. Jagadeesan & P. Malacaria (2000): *Full abstraction for PCF*. Information and Computation 163, pp. 409–470.

[4] S. Abramsky & A. Jung (1994): chapter Domain Theory. 3, Clarendon Press, Oxford.

[5] S. Abramsky & M. Mislove, editors (2012): *Mathematical Foundations of Information Flow*. Proceedings of Symposia in Applied Mathematics 71, AMS, Providence, RI.

[6] Samson Abramsky & Bob Coecke (2004): *A Categorical Semantics of Quantum Protocols*. In: 19th IEEE Symposium on Logic in Computer Science (LICS 2004), 14-17 July 2004, Turku, Finland, Proceedings, IEEE Computer Society, pp. 415–425, doi:10.1109/LICS.2004.1319636. Available at `https://doi.org/10.1109/LICS.2004.1319636`.

[7] P. N. Benton & P. Wadler (1996): *Linear Logic, Monads and the Lambda Calculus*. In: LICS 1996.

[8] P.N. Benton (1995): *A mixed linear and non-linear logic: Proofs, terms and models*. In: Computer Science Logic: 8th Workshop, CSL '94, Selected Papaers.

[9] G. Berry & G. Boudol (1990): *The chemical abstract machine*. In: 17th ACM Symposium on Principles of Programming Semantics, 111, ACM Press, pp. 81–94.

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

[11] F. Borceux (1994): *Handbook of Categorical Algebra 2: Categories and Structures*. Cambridge University Press.

[12] T. Braüner (1997): *A general adequacy result for a linear funcitonal language*. Theoretical Computer Science 177, pp. 27–58.

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

[14] Bob Coecke & Ross Duncan (2008): *Interacting Quantum Observables*. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir & Igor Walukiewicz, editors: Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations, Lecture Notes in Computer Science 5126, Springer, pp. 298–310, doi:10.1007/978-3-540-70583-3_25. Available at `https://doi.org/10.1007/978-3-540-70583-3_25`.

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

[16] J. Egger, R. E. Møgelberg & A. Simpson (2014): *The enriched effect calculus: syntax and semantics*. Journal of Logic and Computation 24(3), pp. 615–654.

[17] M. P. Fiore (1994): *Axiomatic domain theory in categories of partial maps*. Ph.D. thesis, University of Edinburgh, UK.

[18] Peter Freyd (1991): *Algebraically complete categories*. In: Category Theory, Springer, pp. 95–104.

[19] J.-Y. Girard (1987): *Linear Logic*. Theoretical Computer Science 50, pp. 1–102.

[20] J.-Y. Girard (1995): *On Geometry of Interaction*. In H. Schwichtenberg, editor: Proof and Computation, NATO ASI Series F: Computer and Systems Sciences 139, Springer, pp. 145–191.

[21] J-Y. Girard (2011): *Geometry of Interaction V: logic in the hyperfinite factor*. Theoretical Computer Science 412(20), pp. 1860–1883.

[22] 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.

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

[24] X. Jia & M. Mislove (2020): *Completing Simple Valuations in K-categories*. Available at `https://arxiv.org/abs/2002.01865`.

[25] K. Keimel & J. D. Lawson (2020): *D-completions and the d-topology*. *Annals of Pure and Applied Logic* 159, pp. 292–306. Available at `https://doi.org/10.1016/j.apal.2008.06.019`.

[26] A. Kornell, B. Lindenhovius & M. Mislove (2020): *Quantum CPOs*. Available at `https://popl20.sigplan.org/home/planqc-2020`.

[27] D. J Lehmann & M. B Smyth (1981): *Algebraic specification of data types: A synthetic approach*. *Mathematical Systems Theory*.

[28] B. Lindenhovius, M. Mislove & V. Zamdzhiev (2018): *Enriching a linear/non-linear lambda calculus: A programming language for string diagrams*. Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, pp. 659–668. Available at `https://doi.org/10.1145/3209108.3209196`.

[29] B. Lindenhovius, M. Mislove & V. Zamdzhiev (2019): *Mixed linear and non-linear recursive types*. *Proceedings of the ACM on Programming Languages* 3 Issue ICFP, ACM New York, NY, USA.

[30] Bert Lindenhovius, Michael W. Mislove & Vladimir Zamdzhiev (2020): *LNL-FPC: The Linear/Non-linear Fixpoint Calculus*. Accepted subject to minor revisions for the journal LMCS (Logical Methods in Computer Science). Available at `http://arxiv.org/abs/1906.09503`.

[31] R.B.B. Lucyshyn-Wright (2016): *Relative Symmetric Monoidal Closed Categories I: Autoenrichment and Change of Base*. *Theory and Applications of Categories*.

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

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

[34] R. Péchoux, S. Perdrix, M. Rennela & V. Zamdzhiev (2020): *Quantum Programming with Inductive Datatypes: Causality and Affine Type Theory*. *Proceedings of FOSSACS 2020 to appear*.

[35] G. D. Plotkin (1985): *Lectures on predomains and partial functions. Notes for a course given at CSLI Stanford University*.

[36] M. Rennela & S. Staton (2017): *Classical control and quantum circuits in enriched category theory*. To appear in MFPS XXXIII.

[37] M. Rennela & S. Staton (2017): *Classical Control, Quantum Circuits and Linear Logic in Enriched Category Theory*.

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

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

[40] P. Selinger & B. Valiron (2008): *A linear-non-linear model for a computational call-by-value lambda calculus*. In: *Proceedings of the Eleventh International Conference on Foundations of Software Science and Computation Structures (FOSSACS 2008)*, *Lecture Notes in Computer Science* 4962, Springer, pp. 81–96.

[41] P. Selinger & B. Valiron (2009): *Quantum Lambda Calculus*. In: *Semantic Techniques in Quantum Computation*, Cambridge University Press, pp. 135–172.

[42] P. Sobocinski & O. Stephens (2014): *A Programming Language for Spatial Distribution of Net Systems*. In: *Petri Nets*.

[43] Owen Stephens (2015): *Compositional specification and reachability checking of net systems*. Ph.D. thesis, University of Southampton, UK.

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

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