Enriching a Linear/Non-linear Lambda Calculus:

A Programming Language for String Diagrams

Bert Lindenhovius
Tulane University

Michael Mislove
Tulane University
{alindenh|mislove|vzamdzhi}@tulane.edu

Vladimir Zamdzhiev
Tulane University

Abstract

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.

CCS Concepts • **Theory of computation → Linear logic**; **Type theory**; **Denotational semantics**; **Operational semantics**; **Categorical semantics**;

Keywords string diagrams, programming languages, quantum computing, categorical semantics, enriched category theory

ACM Reference Format:
Bert Lindenhovius, Michael Mislove, and Vladimir Zamdzhiev. 2018. Enriching a Linear/Non-linear Lambda Calculus:: A Programming Language for String Diagrams. In *Proceedings of ACM/IEEE Symposium on Logic in Computer Science (LICS'18).* ACM, New York, NY, USA, 10 pages. https://doi.org/

1 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 [13]; in systems theory, where they are used in a calculus of signal flow diagrams [3]; and in quantum computing [6, 11] where they represent quantum circuits and have been used to completely axiomatize the Clifford+T segment of quantum mechanics [7].

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 [22] and VHDL [24] are popular hardware description languages that are used to generate very large digital circuits. More recently, the PNBml language [20] was developed to generate Petri nets, and Quipper [10] and QWIRE [14] 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 2, can be seen as a modified version of Benton's LNL calculus, first defined in [1]. 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 3, 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 [18], 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 [18], 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 4 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 intuitionistic types.

In Section 5, we conclude the paper and discuss further possible developments, such as adding inductive and recursive types, as well as a treatment of dependent types.

LICS'18, 9–12 July 2018, Oxford, United Kingdom
2018.

Related Work. Categorical models are fundamental for our results, and the ones we present rely on the LNL models first described by Benton in [1]. Our work also is inspired by the language Proto-Quipper-M [18] 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. $\overline{\mathbf{M}} := [\mathbf{M}^{\mathrm{op}}, \mathbf{Set}]$), 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" (intuitionistic) 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 [2], and present an adequacy result for the diagram-free fragment of the language.

QWIRE [14] also is a language for reasoning about quantum circuits. QWIRE is really two languages, an intuitionistic host language and a quantum circuits language. QWIRE led Rennela and Staton to consider a more general language Ewire [16, 17], 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 intuitionistic 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.

2 An alternative LNL calculus

LNL models were introduced by Benton [1] 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 intuitionistic), 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 [1]. The syntax and operational semantics for this language are derived as a special case of the language of Rios and Selinger [18]. 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 2.1 ([1]). A *model of the CLNL calculus* (CLNL model) is given by the following data: a cartesian closed category with finite coproducts $(\mathbf{V}, \times, \to, 1, \amalg, \varnothing)$; a symmetric monoidal closed category 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 $\eta : \mathrm{Id} \overset{.}{\longrightarrow} G \circ F$.
- The counit of the adjunction $F \dashv G$ is $\epsilon : ! \overset{.}{\longrightarrow} \mathrm{Id}$.

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 intuitionistic 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 intuitionistic 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 intuitionistic. 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:

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

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

because F is strong (symmetric) monoidal and also a left adjoint. Then a simple induction argument shows:

Proposition 2.2. *For every intuitionistic type P, there is a canonical isomorphism $[\![P]\!] \cong F(X)$.*

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 intuitionistic (linear) if it is assigned an intuitionistic (linear) type. A context that contains only intuitionistic variables is called an *intuitionistic context*. Note, that we do not define linear contexts, because our typing rules refer only to contexts that either are intuitionistic 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 intuitionistic variables may be part of an arbitrary context Γ. For example, if P_1 and P_2 are intuitionistic types, then:

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

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

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 (cf. Section 3), and it allows us to easily add general recursion (cf. Section 4). Moreover, the interpretation of any two derivations of the same judgement are equal (cf. Theorem 3.5).

Definition 2.3. A morphism $f : [\![P_1]\!] \to [\![P_2]\!]$ is called *intuitionistic*, if $f = [\![P_1]\!] \xrightarrow{\cong} F(X) \xrightarrow{F(f')} F(Y) \xrightarrow{\cong} [\![P_2]\!]$, for some $f' \in \mathbf{V}(X, Y)$.

Definition 2.4. We define maps on intuitionistic types as follows:

Discard: $\diamond_P := [\![P]\!] \xrightarrow{\cong} F(X) \xrightarrow{F(1_X)} F(1) \xrightarrow{\cong} I$;

Copy: $\Delta_P := [\![P]\!] \xrightarrow{\cong} F(X) \xrightarrow{F(\langle \mathrm{id}, \mathrm{id} \rangle)} F(X \times X) \xrightarrow{\cong} [\![P]\!] \otimes [\![P]\!]$;

Lift: $\mathbf{lift}_P := [\![P]\!] \xrightarrow{\cong} F(X) \xrightarrow{F(\eta_X)} !F(X) \xrightarrow{\cong} ![\![P]\!]$.

Proposition 2.5. *If $f : [\![P_1]\!] \to [\![P_2]\!]$ is intuitionistic, then:*

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

Because of space limitations, we are unable to provide a complete list of the operational and denotational semantics for the languages we discuss, so we confine ourselves to excerpts that present some "interesting" rules 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 2.6. *Theorems 3.5 – 3.9 also hold true when restricted to the CLNL calculus in the obvious way.*

3 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 [18]. 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 (cf. Subsection 3.3) 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* $F \colon \mathscr{A} \to \mathscr{B}$ 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 [4, 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 $G_* : \mathbf{C}$-$\mathbf{Cat} \to \mathbf{V}$-$\mathbf{Cat}$ 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. 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 $\mathbf{V}$ [12, §4]. It follows from [12, 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 3.1. 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 2.1.

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.

Theorem 3.2. *Every CLNL model induces an enriched CLNL model.*

Proof. Combine [8, Proposition 6.7] and [12, Theorem 11.2]. □

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

Proposition 3.3. *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.

(1.) $G(A \multimap B) = \mathscr{C}(I, A \multimap B) \cong \mathscr{C}(A, B)$;

(2.) Apply F to (1.);

(3.) $\mathbf{C}(A, B) \cong \mathbf{C}(I, A \multimap B) \cong \mathbf{C}(F1, A \multimap B) \cong \mathbf{V}(1, G(A \multimap B)) \cong \mathbf{V}(1, \mathscr{C}(A, B))$. □

3.1 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 [19][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 [7], 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 [21], then our calculus may be used to generate such Petri nets.

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

Definition 3.4. An *ECLNL model* is given by the following data:

- An enriched CLNL model (Definition 3.1);
- 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.

3.2 Syntax and Semantics

We first introduce new types in our syntax that correspond to the objects of $\mathbf{M}$. Using terminology introduced in [18], 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 $\mathbf{M}([\![Q_1]\!]_{\mathbf{M}}, [\![Q_2]\!]_{\mathbf{M}})$.

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.

[1] The interested reader can consult [19] for more information on string diagrammatic representations of morphisms.

The CLNL Calculus

Variables	x, y, z		
Types	A, B, C	$::=$	$0 \mid A+B \mid I \mid A \otimes B \mid A \multimap B \mid !A$
Intuitionistic types	P, R	$::=$	$0 \mid P+R \mid I \mid P \otimes R \mid !A$
Variable contexts	Γ	$::=$	$x_1 : A_1, x_2 : A_2, \ldots, x_n : A_n$
Intuitionistic variable contexts	Φ	$::=$	$x_1 : P_1, x_2 : P_2, \ldots, x_n : P_n$
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	$::=$	$\cdots \mid \alpha \mid \text{Diag}(T, U)$
Intuitionistic types	P, R	$::=$	$\cdots \mid \text{Diag}(T, U)$
M-types	T, U	$::=$	$\alpha \mid I \mid T \otimes U$
Label contexts	Q	$::=$	$\ell_1 : \alpha_1, \ell_2 : \alpha_2, \ldots, \ell_n : \alpha_n$
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	$Q \vdash (S, m) : A; Q'$		(cf. Definition 3.6)

The Typing Rules

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

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

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

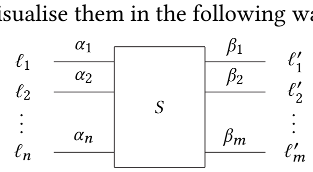
$$\frac{\Phi, \Gamma_1; Q_1 \vdash m : A \qquad \Phi, \Gamma_2; Q_2 \vdash n : B}{\Phi, \Gamma_1, \Gamma_2; Q_1, Q_2 \vdash \langle m, n\rangle : A \otimes B}\text{(pair)} \qquad \frac{\Phi, \Gamma_1; Q_1 \vdash m : A \otimes B \qquad \Phi, \Gamma_2, x : A, y : B; Q_2 \vdash n : C}{\Phi, \Gamma_1, \Gamma_2;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

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

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

Figure 1. Syntax of the CLNL and ECLNL calculi.

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 $\text{Diag}(T, U)$ to the language (see Figure 1); $\text{Diag}(T, U)$ 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 $\text{box}_T m$, $\text{apply}(m, n)$ and $(\vec{\ell}, S, \vec{\ell}')$. The term $\text{box}_T m$ should be thought of as "boxing up" an already completed diagram m; $\text{apply}(m, n)$ represents the application of the

$$[\![\alpha]\!] = E([\![\alpha]\!]_{\mathbf{M}}); \quad [\![0]\!] = 0; \quad [\![A+B]\!] = [\![A]\!] + [\![B]\!]; \quad [\![I]\!] = I; \quad [\![A \otimes B]\!] = [\![A]\!] \otimes [\![B]\!]; \quad [\![A \multimap B]\!] = [\![A]\!] \multimap [\![B]\!];$$
$$[\![!A]\!] = ![\![A]\!]; \quad [\![\mathrm{Diag}(T,U)]\!] = F(\mathscr{C}([\![T]\!], [\![U]\!]));$$

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

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

$$[\![\Phi; \emptyset \vdash \mathrm{lift}\ m : !A]\!] := [\![\Phi]\!] \xrightarrow{\mathbf{lift}} ![\![\Phi]\!] \xrightarrow{![\![m]\!]} ![\![A]\!]$$

$$[\![\Gamma; Q \vdash \mathrm{force}\ m : A]\!] := [\![\Gamma]\!] \otimes [\![Q]\!] \xrightarrow{[\![m]\!]} ![\![A]\!] \xrightarrow{\epsilon} [\![A]\!]$$

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

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

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

Figure 2. Denotational semantics of the ECLNL calculus (excerpt)

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 $h : (\mathbf{qubit} \multimap \mathbf{qubit})$ could be utilised by the user to build quantum circuits. Then the term $\mathrm{box}_{\mathbf{qubit}}$ lift h would reduce to a term (ℓ, H, k) where H is a labelled string diagram representing the Hadamard gate (where technically each term should be part of a configuration, see below).

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 3.5. *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 3.6. 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:

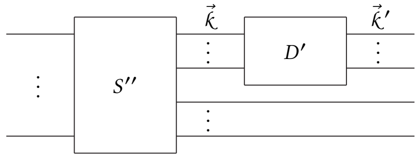
$$[\![(S, m)]\!] := [\![Q]\!] \xrightarrow{[\![S]\!]} [\![Q'']\!] \otimes [\![Q']\!] \xrightarrow{[\![\emptyset; Q'' \vdash m : A]\!] \otimes \mathrm{id}} [\![A]\!] \otimes [\![Q']\!]$$

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 all but one of them (also see Theorem 3.7).

An excerpt of the operational semantics is presented in Figure 3. The evaluation rule for $\mathrm{box}_T m$ makes use of a function *freshlabels*. Given a $\mathbf{M}$-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 $\mathrm{apply}(m, n)$ 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).

Theorem 3.7 (Error freeness [18]). *If $Q \vdash (S, m) : A; Q'$ then $(S, m) \not\Downarrow$ Error.*

Theorem 3.8 (Subject reduction [18]). *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.

Theorem 3.9. *(Soundness) If $Q \vdash (S, m) : A; Q'$ and $(S, m) \Downarrow (S', v)$, then $[\![(S, m)]\!] = [\![(S', v)]\!]$.*

3.3 A constructive property

If we assume, in addition, that $E : \mathbf{M} \to \mathbf{C}$ is fully faithful, then setting $\mathcal{M}(T, U) = \mathscr{C}(T, U)$ 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:

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

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 $\mathbf{M}$.

3.4 Concrete Models

The original 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 $\mathbf{CPO}$-enrichment of $\mathbf{M}$ (obtained by discretely ordering its homsets) and $\mathcal{CPO}$ is the self-enrichment of $\mathbf{CPO}$.

4 The ECLNL calculus with recursion

Additional structure for Benton's LNL models needed to support recursion was discussed by Benton and Wadler in [2]. 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 exactly the same additional semantic structure they use. 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.

4.1 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:

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

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

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

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 4.1. 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 [9], but it suffices for our purposes. This allows us to extend both ECLNL and CLNL models with recursion in the same way.

Definition 4.2. 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.

Benton and Wadler point out that if $\mathbf{C}$ is symmetric monoidal closed, then algebraic compactness of ! implies that it also is parametrically algebraically compact. Nevertheless, we include parametric algebraic compactness in our definition to emphasize that this is exactly what is needed to interpret recursion in our models.

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

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

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

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

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

Figure 3. Operational semantics of the ECLNL calculus (excerpt)

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

Figure 4. Definition of σ_m and γ_Φ.

If $\Phi \in \mathbf{C}$ is an intuitionistic 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 morphisms 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 [5].

Theorem 4.3. *Theorems 3.5 – 3.9 from the previous section remain true for the (E)CLNL calculus extended with recursion.*

4.2 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 4.7 (see Section 4.3 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 4.4. *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 [4]). This follows because the right adjoint and the adjunction factor through $\mathcal{CPO}_{\perp!}$. Parametrised algebraic compactness of the !-endofunctor follows from [9, pp. 161-162]. □

Moreover, the concrete model enjoys a constructive property similar to the one in Subsection 3.3. Using the same argument, if $\Phi;\emptyset \vdash m : T \multimap U$, then we obtain:

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

Therefore, the interpretation of m corresponds to a Scott-continuous function from X to $\mathcal{M}_\perp([\![T]\!]_{\mathbf{M}}, [\![U]\!]_{\mathbf{M}})$. 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 4.5. *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 [9, pp. 161-162]. □

4.3 Computational adequacy

In this subsection we show that computational adequacy holds at intuitionistic 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 $\mathbf{C}$ is pointed, which allows us to introduce a notion of undefinedness. Towards that end, we first introduce a slightly weaker notion, following Braüner [5].

Definition 4.6. 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 4.7 ([5]). *Let* $\mathbf{A}$ *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 4.8. *Any weakly pointed category with an initial object* 0 *is pointed. Moreover,* $\perp_A = \perp_{I,A}$ *and* $\perp_{A,B}$ *are zero morphisms.*

Proof. For any morphism $f : A \to 0$ and object B, let us define $f_B = A \xrightarrow{f} 0 \to B$. For any $h : B \to C$ we have $h \circ f_B = f_C$ and so:

$$f = f_0 = \perp_{0,0} \circ f_0 = \perp_{A,0}$$

which shows 0 is also terminal, hence a zero object. Then $\perp_{A,B}$ is a zero morphism because $\perp_{A,B} = \perp_{0,B} \circ \perp_{A,0}$. Finally, by definition we have $\perp_A = I \xrightarrow{\perp_0} 0 \to A = \perp_{I,A}$ which completes the proof. □

Theorem 4.9. *For every model of the (E)CLNL calculus with recursion,* $\mathbf{C}$ *is a pointed category with*

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

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 4.10. *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* $K_m(f) = m \circ (id \otimes !f) \circ (id \otimes \boldsymbol{lift}) \circ \Delta$*. Then:*

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

Proof. Define $T : \mathscr{C}(\Omega_\Phi, \Omega_\Phi) \to \mathscr{C}(\Omega_\Phi, \Omega_\Phi)$ by $T(f) = \omega_\Phi \circ (\mathrm{id} \otimes !f) \circ \omega_\Phi^{-1}$. Then $\mathrm{id}_{\Omega_\Phi} = \bigvee_i T^i$, which follows after recognising that $\bigvee_i T^i$ is a $\Phi \otimes !(-)$-algebra morphism and then using the initiality of Ω_Φ. Next, one can show by induction that $K_m^i = \sigma_m \circ T^i \circ \gamma_\Phi$. Finally:

$$\sigma_m \circ \gamma_\Phi = \sigma_m \circ \left(\bigvee_i T^i \right) \circ \gamma_\Phi = \bigvee_i K_m^i$$

where Scott continuity of composition implies the last equality. □

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 4.5. Therefore, in what follows $\mathbf{C} = \mathbf{CPO}_{\perp!}$.

Lemma 4.11. *Let* $\emptyset \vdash v : P$ *be a well-typed value, where* P *is an intuitionistic 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 [15].

Definition 4.12. 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 $f = \mathrm{id}_I$;
(A2.1) $f \trianglelefteq_{A+B}$ 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 $\forall f' \in \mathbf{C}(I, [\![A]\!]), \forall v \in V_A$:

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

(A5) $f \trianglelefteq_{!A}$ lift m iff f is an intuitionistic morphism and $\epsilon_A \circ f \sqsubseteq_A m$;
(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 4.13. *If* $f \trianglelefteq_P v$*, where* P *is an intuitionistic type, then* f *is an intuitionistic morphism.*

Lemma 4.14. *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 4.15. *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 4.14 and Lemma 4.10. □

Definition 4.16. 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 4.17 (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 4.18 (Adequacy)**.** *Let* $\emptyset \vdash m : P$ *be a well-typed term, where* P *is an intuitionistic type. Then:*

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

Proof. The right-to-left direction follows from Theorem 4.17. The other direction follows from soundness and Lemma 4.11. □

The model of Theorem 4.5 was presented as an example by Benton and Wadler [2] for their LNL calculus extended with recursion, however without stating an adequacy result. We have now shown that it is computationally adequate at intuitionistic types for our CLNL calculus. We also note that the simple proof is very similar to the classical proof of adequacy for PCF.

5 Conclusion and Future Work

We considered the CLNL calculus, which is a variant of Benton's LNL calculus [1], and showed that both calculi have the same categorical models. We then showed the CLNL calculus can be extended with recursion in a simple way while still using the same categorical model as described by Benton and Wadler [2]. Moreover, the CLNL calculus also can be extended with language features that turn it into a lambda calculus for string diagrams, which we named the ECLNL calculus (originally Proto-Quipper-M [18]). We next identified abstract models for 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, moreover, it 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 one would expect to hold for 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.

For future work, we will consider extending ECLNL with dynamic lifting. In quantum computing, this would allow the language to execute quantum circuits and then use a measurement outcome to parametrize subsequent circuit generation. Another line of future work is to consider the introduction of inductive/recursive datatypes. Our concrete models appear to have sufficient structure, so we believe this could be achieved in the usual way. We will also investigate alternative proof strategies for establishing computational adequacy (at intuitionistic types) for the ECLNL calculus. Finally, we are interested in extending the language with dependent types. The original model of Proto-Quipper-M was defined in terms of the **Fam**(−) construction and has the structure of a strict indexed symmetric monoidal category [23], which suggests a potential approach for adding type dependency.

Acknowledgments

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 entitled, "Semantics, Formal Reasoning, and Tool Support for Quantum Programming".

References

[1] 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.*

[2] P. N. Benton and P. Wadler. 1996. Linear Logic, Monads and the Lambda Calculus. In *LICS 1996.*

[3] F. Bonchi, P. Sobocinski, and F. Zanasi. 2015. Full Abstraction for Signal Flow Graphs. In *POPL*. ACM, 515–526.

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

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

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

[7] S. Perdrix E. Jeandel and R. Vilmart. 2017. A Complete Axiomatisation of the ZX-Calculus for Clifford+T Quantum Mechanics. (2017). arXiv:1705.11151

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

[9] M. P. Fiore. 1994. *Axiomatic domain theory in categories of partial maps.* Ph.D. Dissertation. University of Edinburgh, UK.

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

[11] A. Hadzihasanovic. 2015. A Diagrammatic Axiomatisation for Qubit Entanglement. In *LICS*. IEEE Computer Society, 573–584.

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

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

[14] J. Paykin, R. Rand, and S. Zdancewic. 2017. QWIRE: a core language for quantum circuits. In *POPL*. ACM, 846–858.

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

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

[17] M. Rennela and S. Staton. 2017. Classical Control, Quantum Circuits and Linear Logic in Enriched Category Theory. (2017). arXiv:1711.05159

[18] F. Rios and P. Selinger. 2017. A categorical model for a quantum circuit description language. (2017). arXiv:1706.02630 To appear in QPL 2017.

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

[20] P. Sobocinski and O. Stephens. 2014. A Programming Language for Spatial Distribution of Net Systems. In *Petri Nets.*

[21] Owen Stephens. 2015. *Compositional specification and reachability checking of net systems.* Ph.D. Dissertation. University of Southampton, UK.

[22] D. Thomas and P. Moorby. 2008. *The Verilog Hardware Description Language.* Springer Science & Business Media.

[23] M. Vákár. 2015. A Categorical Semantics for Linear Logical Frameworks. In *FoSSaCS (Lecture Notes in Computer Science)*, Vol. 9034. Springer, 102–116.

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