On the Lambek Calculus with an Exchange Modality

Jiaming Jiang
Computer Science
North Carolina State University
Raleigh, North Carolina, USA
jjiang13@ncsu.edu

Harley Eades III
Computer Science
Augusta University
Augusta, Georgia, USA
harley.eades@gmail.com

Valeria de Paiva
Nuance Communications
Sunnyvale, California, USA
valeria.depaiva@gmail.com

In this paper we introduce Commutative/Non-Commutative Logic (CNC logic) and two categorical models for CNC logic. This work abstracts Benton's Linear/Non-Linear Logic [4] by removing the existence of the exchange structural rule. One should view this logic as composed of two logics; one sitting to the left of the other. On the left, there is intuitionistic linear logic, and on the right is a mixed commutative/non-commutative formalization of the Lambek calculus. Then both of these logics are connected via a pair of monoidal adjoint functors. An exchange modality is then derivable within the logic using the adjunction between both sides. Thus, the adjoint functors allow one to pull the exchange structural rule from the left side to the right side. We then give a categorical model in terms of a monoidal adjunction, and then a concrete model in terms of dialectica Lambek spaces.

1 Introduction

Joachim Lambek first introduced the Syntactic Calculus, now known as the Lambek Calculus, in 1958 [13]. Since then the Lambek Calculus has largely been motivated by providing an explanation of the mathematics of sentence structure, and can be found at the core of Categorial Grammar; a term first used in the title of Bar-Hillel, Gaifman and Shamir (1960), but categorical grammar began with Ajdukiewicz (1935) quite a few years earlier. At the end of the eighties the Lambek calculus and other systems of categorial grammars were taken up by computational linguists as exemplified by [16, 15, 2, 10].

It was computational linguists who posed the question of whether it is possible to isolate exchange using a modality in the same way that the of-course modality of linear logic, $!A$, isolates weakening and contraction. de Paiva and Eades [8] propose one solution to this problem by extending the Lambek calculus with the modality characterized by the following sequent calculus inference rules:

$$\frac{\kappa\Gamma \vdash B}{\kappa\Gamma \vdash \kappa B}\,\text{E}_\text{R} \qquad \frac{\Gamma_1, A, \Gamma_2 \vdash B}{\Gamma_1, \kappa A, \Gamma_2 \vdash B}\,\text{E}_\text{L} \qquad \frac{\Gamma_1, \kappa A, B, \Gamma_2 \vdash C}{\Gamma_1, B, \kappa A, \Gamma_2 \vdash C}\,\text{E1} \qquad \frac{\Gamma_1, A, \kappa B, \Gamma_2 \vdash C}{\Gamma_1, \kappa B, A, \Gamma_2 \vdash C}\,\text{E2}$$

The thing to note is that the modality κA appears on only one of the operands being exchanged. That is, these rules along with those for the tensor product allow one to prove that $\kappa A \otimes B \multimap B \otimes \kappa A$ holds. This is somewhat at odds with algebraic intuition, and it is unclear how this modality could be decomposed into adjoint functors in a linear/non-linear (LNL) formalization of the Lambek calculus.

In this paper we show how to add an exchange modality, eA, where the modality now occurs on both operands being exchanged. That is, one can show that $eA \otimes eB \multimap eB \otimes eA$ holds. We give a LNL natural deduction formalization for the Lambek calculus with this new modality, and two categorical models: a LNL model and a concrete model in dialectica spaces. Thus giving a second solution to the problem proposed above.

The Lambek Calculus also has the potential for many applications in other areas of computer science, such as, modeling processes. Linear Logic has been at the forefront of the study of process calculi for

Submitted to:
LINEARITY 2018

many years [11, 17, 1]. We can think of the commutative tensor product of linear logic as a parallel operator. For example, given a process A and a process B, then we can form the process $A \otimes B$ which runs both processes in parallel. If we remove commutativity from the tensor product we obtain a sequential composition instead of parallel composition. That is, the process $A \triangleright B$ first runs process A and then process B in that order. Vaughan Pratt has stated that , "sequential composition has no evident counterpart in type theory" see page 11 of [17]. We believe that the Lambek Calculus will lead to filling this hole.

Acknowledgments. The first two authors were supported by NSF award #1565557. We thank the anonymous reviewers for their helpful feedback that made this a better paper.

2 An Adjoint Formalization of the Lambek Calculus

We now introduce Commutative/Non-commutative (CNC) logic in the form of a term assignment. One should view this logic as composed of two logics; one sitting to the left of the other. On the left, there is intuitionistic linear logic, denoted by $\mathcal{C}$, and on the right is the Lambek calculus denoted by $\mathcal{L}$. Then we connect these two systems by a pair of monoidal adjoint functors $\mathcal{C} : \mathsf{F} \dashv \mathsf{G} : \mathcal{L}$. Keeping this intuition in mind we now define the syntax for CNC logic.

Definition 1. *The following grammar describes the syntax of CNC logic:*

($\mathcal{C}$-Types)	$W, X, Y, Z ::= \mathsf{Unit} \mid X \otimes Y \mid X \multimap Y \mid \mathsf{G}A$
($\mathcal{L}$-Types)	$A, B, C, D ::= \mathsf{Unit} \mid A \triangleright B \mid A \rightharpoonup B \mid B \leftharpoondown A \mid \mathsf{F}X$
($\mathcal{C}$-Terms)	$t ::= x \mid \mathsf{triv} \mid t_1 \otimes t_2 \mid \mathsf{let}\, t_1 : X \,\mathsf{be}\, q \,\mathsf{in}\, t_2 \mid \lambda x : X.t \mid t_1 t_2 \mid \mathsf{ex}\, t_1, t_2 \,\mathsf{with}\, x_1, x_2 \,\mathsf{in}\, t_3 \mid \mathsf{G}\, s$
($\mathcal{L}$-Terms)	$s ::= x \mid \mathsf{triv} \mid s_1 \triangleright s_2 \mid \mathsf{let}\, s_1 : A \,\mathsf{be}\, p \,\mathsf{in}\, s_2 \mid \mathsf{let}\, t : X \,\mathsf{be}\, q \,\mathsf{in}\, s \mid \lambda_l x : A.s \mid \lambda_r x : A.s$ $\mid \mathsf{app}_l\, s_1\, s_2 \mid \mathsf{app}_r\, s_1\, s_2 \mid \mathsf{F}t$
($\mathcal{C}$-Patterns)	$q ::= \mathsf{triv} \mid x \mid q_1 \otimes q_2 \mid \mathsf{G}p$
($\mathcal{L}$-Patterns)	$p ::= \mathsf{triv} \mid x \mid p_1 \triangleright p_2 \mid \mathsf{F}\, q$
($\mathcal{C}$-Contexts)	$\Phi, \Psi ::= \cdot \mid x : X \mid \Phi, \Psi$
($\mathcal{L}$-Contexts)	$\Gamma, \Delta ::= \cdot \mid x : A \mid x : X \mid \Gamma; \Delta$
($\mathcal{C}$-Typing Judgment)	$\Phi \vdash_{\mathcal{C}} t : X$
($\mathcal{L}$-Typing Judgment)	$\Gamma \vdash_{\mathcal{L}} s : A$

The syntax for $\mathcal{C}$-types are the standard types for intuitionistic linear logic. We have a constant Unit, tensor product $X \otimes Y$, and linear implication $X \multimap Y$, but just as in LNL logic we also have a type $\mathsf{G}A$ where A is an $\mathcal{L}$-type; that is, a type from the non-commutative side corresponding to the right-adjoint functor between $\mathcal{L}$ and $\mathcal{C}$. This functor can be used to import types and terms from the non-commutative side into the commutative side. Now $\mathcal{C}$-typing judgments are denoted by $\Psi \vdash_{\mathcal{C}} t : X$ where Ψ is a sequence of pairs of variables and their types, denoted by $x : X$, t is a $\mathcal{C}$-term, and X is a $\mathcal{C}$-type. The $\mathcal{C}$-terms are all standard, but $\mathsf{G}\, s$ corresponds to the morphism part of the right-adjoint of the adjunction between both logics, and $\mathsf{ex}\, t_1, t_2 \,\mathsf{with}\, x_1, x_2 \,\mathsf{in}\, t_3$ is the introduction form for the structural rule exchange.

The non-commutative side is a bit more interesting than the commutative side just introduced. The $\mathcal{L}$-typing judgment has the form $\Gamma \vdash_{\mathcal{L}} s : A$ where Γ is now a $\mathcal{L}$-context, denoted by Γ or Δ. These contexts are ordered sequences of pairs of free variables with their types from *both* sides denoted by $x : B$ and $x : X$ respectively. Finally, the term s is a $\mathcal{L}$-term, and A is a $\mathcal{L}$-type. Given two typing contexts Γ and Δ we denote their concatenation by $\Gamma; \Delta$; we use a semicolon here to emphasize the fact that the contexts are ordered.

The context consisting of hypotheses from both sides goes back to Benton [4] and is a property unique to adjoint logics such as Benton's LNL logic and CNC logic. This is also a very useful property because it allows one to make use of both sides within the Lambek calculus without the need to annotate every formula with a modality.

$$\frac{}{x : X \vdash_C x : X}\, C\text{-id} \qquad \frac{}{\cdot \vdash_C \mathsf{triv} : \mathsf{Unit}}\, C\text{-Unit}_I \qquad \frac{\Phi \vdash_C t_1 : \mathsf{Unit} \quad \Psi \vdash_C t_2 : Y}{\Phi, \Psi \vdash_C \mathsf{let}\, t_1 : \mathsf{Unit}\, \mathsf{be}\, \mathsf{triv}\, \mathsf{in}\, t_2 : Y}\, C\text{-Unit}_E$$

$$\frac{\Phi \vdash_C t_1 : X \quad \Psi \vdash_C t_2 : Y}{\Phi, \Psi \vdash_C t_1 \otimes t_2 : X \otimes Y}\, C\text{-}\otimes_I \qquad \frac{\Phi \vdash_C t_1 : X \otimes Y \quad \Psi_1, x : X, y : Y, \Psi_2 \vdash_C t_2 : Z}{\Psi_1, \Phi, \Psi_2 \vdash_C \mathsf{let}\, t_1 : X \otimes Y\, \mathsf{be}\, x \otimes y\, \mathsf{in}\, t_2 : Z}\, C\text{-}\otimes_E \qquad \frac{\Phi, x : X \vdash_C t : Y}{\Phi \vdash_C \lambda x : X.t : X \multimap Y}\, C\text{-}\multimap_I$$

$$\frac{\Phi \vdash_C t_1 : X \multimap Y \quad \Psi \vdash_C t_2 : X}{\Phi, \Psi \vdash_C t_1 t_2 : Y}\, C\text{-}\multimap_E \qquad \frac{\Phi \vdash_{\mathcal{L}} s : A}{\Phi \vdash_C \mathsf{G}\, s : \mathsf{G}A}\, C\text{-G}_I \qquad \frac{\Phi, x : X, y : Y, \Psi \vdash_C t : Z}{\Phi, z : Y, w : X, \Psi \vdash_C \mathsf{ex}\, w, z\, \mathsf{with}\, x, y\, \mathsf{in}\, t : Z}\, C\text{-ex}$$

$$\frac{\Phi \vdash_C t_1 : X \quad \Psi_1, x : X, \Psi_2 \vdash_C t_2 : Y}{\Psi_1, \Phi, \Psi_2 \vdash_C [t_1/x]t_2 : Y}\, C\text{-Cut}$$

$$\frac{}{x : A \vdash_{\mathcal{L}} x : A}\, \mathcal{L}\text{-id} \qquad \frac{}{\cdot \vdash_{\mathcal{L}} \mathsf{triv} : \mathsf{Unit}}\, \mathcal{L}\text{-Unit}_I \qquad \frac{\Gamma \vdash_{\mathcal{L}} s_1 : \mathsf{Unit} \quad \Delta \vdash_{\mathcal{L}} s_2 : A}{\Gamma; \Delta \vdash_{\mathcal{L}} \mathsf{let}\, s_1 : \mathsf{Unit}\, \mathsf{be}\, \mathsf{triv}\, \mathsf{in}\, s_2 : A}\, \mathcal{L}\text{-Unit}_E$$

$$\frac{\Phi \vdash_C t : \mathsf{Unit} \quad \Gamma \vdash_{\mathcal{L}} s : A}{\Phi; \Gamma \vdash_{\mathcal{L}} \mathsf{let}\, t : \mathsf{Unit}\, \mathsf{be}\, \mathsf{triv}\, \mathsf{in}\, s : A}\, \mathcal{L}C\text{-Unit}_E \qquad \frac{\Gamma \vdash_{\mathcal{L}} s_1 : A \quad \Delta \vdash_{\mathcal{L}} s_2 : B}{\Gamma; \Delta \vdash_{\mathcal{L}} s_1 \triangleright s_2 : A \triangleright B}\, \mathcal{L}\text{-}\otimes_I$$

$$\frac{\Gamma \vdash_{\mathcal{L}} s_1 : A \triangleright B \quad \Delta_1; x : A; y : B; \Delta_2 \vdash_{\mathcal{L}} s_2 : C}{\Delta_1; \Gamma; \Delta_2 \vdash_{\mathcal{L}} \mathsf{let}\, s_1 : A \triangleright B\, \mathsf{be}\, x \triangleright y\, \mathsf{in}\, s_2 : C}\, \mathcal{L}\text{-}\otimes_E \qquad \frac{\Phi \vdash_C t : X \otimes Y \quad \Gamma_1; x : X; y : Y; \Gamma_2 \vdash_{\mathcal{L}} s : A}{\Gamma_1; \Phi; \Gamma_2 \vdash_{\mathcal{L}} \mathsf{let}\, t : X \otimes Y\, \mathsf{be}\, x \otimes y\, \mathsf{in}\, s : A}\, \mathcal{L}C\text{-}\otimes_E$$

$$\frac{\Gamma; x : A \vdash_{\mathcal{L}} s : B}{\Gamma \vdash_{\mathcal{L}} \lambda_r x : A.s : A \rightharpoonup B}\, \mathcal{L}\text{-}\rightharpoonup_I \qquad \frac{\Gamma \vdash_{\mathcal{L}} s_1 : A \rightharpoonup B \quad \Delta \vdash_{\mathcal{L}} s_2 : A}{\Gamma; \Delta \vdash_{\mathcal{L}} \mathsf{app}_r\, s_1\, s_2 : B}\, \mathcal{L}\text{-}\rightharpoonup_E \qquad \frac{x : A; \Gamma \vdash_{\mathcal{L}} s : B}{\Gamma \vdash_{\mathcal{L}} \lambda_l x : A.s : B \leftharpoonup A}\, \mathcal{L}\text{-}\leftharpoonup_I$$

$$\frac{\Gamma \vdash_{\mathcal{L}} s_1 : B \leftharpoonup A \quad \Delta \vdash_{\mathcal{L}} s_2 : A}{\Delta; \Gamma \vdash_{\mathcal{L}} \mathsf{app}_l\, s_1\, s_2 : B}\, \mathcal{L}\text{-}\leftharpoonup_E \qquad \frac{\Phi \vdash_C t : X}{\Phi \vdash_{\mathcal{L}} \mathsf{F}t : \mathsf{F}X}\, \mathcal{L}\text{-F}_I \qquad \frac{\Gamma \vdash_{\mathcal{L}} s_1 : \mathsf{F}X \quad \Delta_1; x : X; \Delta_2 \vdash_{\mathcal{L}} s_2 : A}{\Delta_1; \Gamma; \Delta_2 \vdash_{\mathcal{L}} \mathsf{let}\, s_1 : \mathsf{F}X\, \mathsf{be}\, \mathsf{F}\, x\, \mathsf{in}\, s_2 : A}\, \mathcal{L}\text{-F}_E$$

$$\frac{\Phi \vdash_C t : \mathsf{G}A}{\Phi \vdash_{\mathcal{L}} \mathsf{derelict}\, t : A}\, \mathcal{L}\text{-G}_E \qquad \frac{\Gamma; x : X; y : Y; \Delta \vdash_{\mathcal{L}} s : A}{\Gamma; z : Y; w : X; \Delta \vdash_{\mathcal{L}} \mathsf{ex}\, w, z\, \mathsf{with}\, x, y\, \mathsf{in}\, s : A}\, \mathcal{L}\text{-ex}$$

$$\frac{\Gamma \vdash_{\mathcal{L}} s_1 : A \quad \Delta_1; x : A; \Delta_2 \vdash_{\mathcal{L}} s_2 : B}{\Delta_1; \Gamma; \Delta_2 \vdash_{\mathcal{L}} [s_1/x]s_2 : B}\, \mathcal{L}\text{-Cut} \qquad \frac{\Phi \vdash_C t : X \quad \Gamma_1; x : X; \Gamma_2 \vdash_{\mathcal{L}} s : A}{\Gamma_1; \Phi; \Gamma_1 \vdash_{\mathcal{L}} [t/x]s : A}\, \mathcal{L}C\text{-Cut}$$

Figure 1: Typing Rules for CNC Logic

The reader familiar with LNL logic will notice that our typing judgment, $\Gamma \vdash_{\mathcal{L}} s : A$, differs from Benton's. His is of the form $\Gamma; \Delta \vdash t : A$, where Γ contains non-linear formulas, and Δ contains linear formulas. Just as Benton remarks, the splitting of his contexts was a presentational device. One should view his contexts as merged, and hence, linear formulas were fully mixed with non-linear formulas. Now why did we not use this presentational device? Because, when contexts from LNL logic become out of order Benton could use the exchange rule to put them back in order again, but we no longer have general exchange. Thus, we are not able to keep the context organized in this way.

The syntax for $\mathcal{L}$-types are of the typical form for the Lambek Calculus. We have an unit type Unit, a non-commutative tensor product $A \triangleright B$, right implication $A \rightharpoonup B$, and left implication $B \leftharpoonup A$. $\mathcal{L}$-terms correspond to introduction and elimination forms for each of the previous types. For example, $s_1 \triangleright s_2$ introduces a tensor, and $\mathsf{let}\, s_1 : A \triangleright B\, \mathsf{be}\, x \triangleright y\, \mathsf{in}\, s_2$ eliminates a tensor.

The typing rules for CNC logic can be found in Figure 1. We split the figure in two: the top of the figure are the rules of intuitionistic linear logic whose judgment is the C-typing judgment denoted by

$\Psi \vdash_C t : X$, and the bottom of the figure are the rules for the mixed commutative/non-commutative Lambek calculus whose judgment is the $\mathcal{L}$-judgment denoted by $\Gamma \vdash_{\mathcal{L}} s : A$, but the two halves are connected via the functor rules C-G_I, $\mathcal{L}$-G_E, $\mathcal{L}$-F_I, and $\mathcal{L}$-F_E. Just as in LNL logic, the fact that the context Γ on the $\mathcal{L}$ side of the logic is mixed results in this side having additional elimination rules, because the type being eliminated is constructed on the C side of the logic; for example, the rules $\mathcal{LC}$-Unit_E, $\mathcal{LC}$-$\otimes_E$, and $\mathcal{LC}$-Cut are additional mixed rules.

The one step β-reduction rules are listed in Figure 2. Similarly to the typing rules, the figure is split in two: the top lists the rules of the intuitionistic linear logic, and the bottom are those of the mixed commutative/non-commutative Lambek calculus.

$$\overline{\mathsf{let}\,\mathsf{triv} : \mathsf{Unit}\,\mathsf{be}\,\mathsf{triv}\,\mathsf{in}\,t \rightsquigarrow_\beta t} \qquad \overline{\mathsf{let}\,t_1 \otimes t_2 : X \otimes Y\,\mathsf{be}\,x \otimes y\,\mathsf{in}\,t_3 \rightsquigarrow_\beta [t_1/x][t_2/y]t_3} \qquad \overline{(\lambda x : X.t_1)t_2 \rightsquigarrow_\beta [t_2/x]t_1}$$

$$\overline{\mathsf{let}\,\mathsf{triv} : \mathsf{Unit}\,\mathsf{be}\,\mathsf{triv}\,\mathsf{in}\,s \rightsquigarrow_\beta s} \qquad \overline{\mathsf{let}\,t_1 \otimes t_2 : X \otimes Y\,\mathsf{be}\,x \triangleright y\,\mathsf{in}\,s \rightsquigarrow_\beta [t_1/x][t_2/y]s}$$

$$\overline{\mathsf{let}\,s_1 \triangleright s_2 : A \triangleright B\,\mathsf{be}\,x \triangleright y\,\mathsf{in}\,s_3 \rightsquigarrow_\beta [s_1/x][s_2/y]s_3} \qquad \overline{\mathsf{let}\,\mathsf{F}t : \mathsf{F}X\,\mathsf{be}\,\mathsf{F}x\,\mathsf{in}\,s \rightsquigarrow_\beta [t/x]s} \qquad \overline{\mathsf{app}_l\,(\lambda_l x : A.s_1)\,s_2 \rightsquigarrow_\beta [s_2/x]s_1}$$

$$\overline{\mathsf{app}_r\,(\lambda_r x : A.s_1)\,s_2 \rightsquigarrow_\beta [s_2/x]s_1} \qquad \overline{\mathsf{derelict}\,(\mathsf{G}\,s) \rightsquigarrow_\beta s}$$

Figure 2: β-reductions for CNC Logic

The commuting conversions can be found in Figures 3-5. We divide the rules into three parts due to the length. The first part, Figure 3, includes the rules for the intuitionistic linear logic. The second, Figure 4, includes the rules for the commutative/non-commutative Lambek calculus. The third, Figure 5, includes the mixed rules $\mathcal{LC}$-Unit_E and $\mathcal{LC}$-$\otimes_E$.

$$\overline{\mathsf{let}\,(\mathsf{let}\,t_2 : \mathsf{Unit}\,\mathsf{be}\,\mathsf{triv}\,\mathsf{in}\,t_1) : \mathsf{Unit}\,\mathsf{be}\,\mathsf{triv}\,\mathsf{in}\,t_3 \rightsquigarrow_{\mathsf{c}} \mathsf{let}\,t_2 : \mathsf{Unit}\,\mathsf{be}\,\mathsf{triv}\,\mathsf{in}\,(\mathsf{let}\,t_1 : \mathsf{Unit}\,\mathsf{be}\,\mathsf{triv}\,\mathsf{in}\,t_3)}$$

$$\overline{\mathsf{let}\,(\mathsf{let}\,t_2 : \mathsf{Unit}\,\mathsf{be}\,\mathsf{triv}\,\mathsf{in}\,t_1) : X \otimes Y\,\mathsf{be}\,x \otimes y\,\mathsf{in}\,t_3 \rightsquigarrow_{\mathsf{c}} \mathsf{let}\,t_2 : \mathsf{Unit}\,\mathsf{be}\,\mathsf{triv}\,\mathsf{in}\,(\mathsf{let}\,t_1 : X \otimes Y\,\mathsf{be}\,x \otimes y\,\mathsf{in}\,t_3)}$$

$$\overline{(\mathsf{let}\,t_2 : \mathsf{Unit}\,\mathsf{be}\,\mathsf{triv}\,\mathsf{in}\,t_1)t_3 \rightsquigarrow_{\mathsf{c}} \mathsf{let}\,t_2 : \mathsf{Unit}\,\mathsf{be}\,\mathsf{triv}\,\mathsf{in}\,(t_1 t_3)}$$

$$\overline{\mathsf{let}\,(\mathsf{let}\,t_2 : X \otimes Y\,\mathsf{be}\,x \otimes y\,\mathsf{in}\,t_1) : \mathsf{Unit}\,\mathsf{be}\,\mathsf{triv}\,\mathsf{in}\,t_3 \rightsquigarrow_{\mathsf{c}} \mathsf{let}\,t_2 : X \otimes Y\,\mathsf{be}\,x \otimes y\,\mathsf{in}\,(\mathsf{let}\,t_1 : \mathsf{Unit}\,\mathsf{be}\,\mathsf{triv}\,\mathsf{in}\,t_3)}$$

$$\overline{\mathsf{let}\,(\mathsf{let}\,t_2 : X_2 \otimes Y_2\,\mathsf{be}\,x \otimes y\,\mathsf{in}\,t_1) : X_1 \otimes Y_1\,\mathsf{be}\,w \otimes z\,\mathsf{in}\,t_3 \rightsquigarrow_{\mathsf{c}} \mathsf{let}\,t_2 : X_2 \otimes Y_2\,\mathsf{be}\,x \otimes y\,\mathsf{in}\,(\mathsf{let}\,t_1 : X_1 \otimes Y_1\,\mathsf{be}\,w \otimes z\,\mathsf{in}\,t_3)}$$

$$\overline{(\mathsf{let}\,t_2 : X_2 \otimes Y_2\,\mathsf{be}\,x \otimes y\,\mathsf{in}\,t_1)t_3 \rightsquigarrow_{\mathsf{c}} \mathsf{let}\,t_2 : X_2 \otimes Y_2\,\mathsf{be}\,x \otimes y\,\mathsf{in}\,(t_1 t_3)} \qquad \overline{\mathsf{let}\,(t_1 t_2) : \mathsf{Unit}\,\mathsf{be}\,\mathsf{triv}\,\mathsf{in}\,t_3 \rightsquigarrow_{\mathsf{c}} t_1(\mathsf{let}\,t_2 : \mathsf{Unit}\,\mathsf{be}\,\mathsf{triv}\,\mathsf{in}\,t_3)}$$

Figure 3: Commuting Conversions: Intuitionistic Linear Logic

Additional Results. The following subsection shows the reduction rules for CNC Logic. In addition to the results given in this paper we also have defined a sequent calculus for CNC logic, proved cut elimination, and proved the the sequent calculus formalization is equivalent to the natural deduction

$$\overline{\mathsf{let}(\mathsf{let}\, s_2 : \mathsf{Unit}\, \mathsf{be}\, \mathsf{triv}\, \mathsf{in}\, s_1) : \mathsf{Unit}\, \mathsf{be}\, \mathsf{triv}\, \mathsf{in}\, s_3 \rightsquigarrow_{\mathsf{c}} \mathsf{let}\, s_2 : \mathsf{Unit}\, \mathsf{be}\, \mathsf{triv}\, \mathsf{in}\, (\mathsf{let}\, s_1 : \mathsf{Unit}\, \mathsf{be}\, \mathsf{triv}\, \mathsf{in}\, s_3)}$$

$$\overline{\mathsf{app}_r\, (\mathsf{let}\, s_2 : \mathsf{Unit}\, \mathsf{be}\, \mathsf{triv}\, \mathsf{in}\, s_1)\, s_3 \rightsquigarrow_{\mathsf{c}} \mathsf{let}\, s_2 : \mathsf{Unit}\, \mathsf{be}\, \mathsf{triv}\, \mathsf{in}\, (\mathsf{app}_r\, s_1\, s_3)}$$

$$\overline{\mathsf{let}(\mathsf{let}\, s_2 : \mathsf{Unit}\, \mathsf{be}\, \mathsf{triv}\, \mathsf{in}\, s_1) : \mathsf{F}X\, \mathsf{be}\, \mathsf{F}x\, \mathsf{in}\, s_3 \rightsquigarrow_{\mathsf{c}} \mathsf{let}\, s_2 : \mathsf{Unit}\, \mathsf{be}\, \mathsf{triv}\, \mathsf{in}\, (\mathsf{let}\, s_1 : \mathsf{F}X\, \mathsf{be}\, \mathsf{F}x\, \mathsf{in}\, s_3)}$$

$$\overline{\mathsf{let}(\mathsf{let}\, s_2 : A \triangleright B\, \mathsf{be}\, x \triangleright y\, \mathsf{in}\, s_1) : \mathsf{Unit}\, \mathsf{be}\, \mathsf{triv}\, \mathsf{in}\, s_3 \rightsquigarrow_{\mathsf{c}} \mathsf{let}\, s_2 : A \triangleright B\, \mathsf{be}\, x \triangleright y\, \mathsf{in}\, (\mathsf{let}\, s_1 : \mathsf{Unit}\, \mathsf{be}\, \mathsf{triv}\, \mathsf{in}\, s_3)}$$

$$\overline{\mathsf{let}(\mathsf{let}\, s_2 : A_2 \triangleright B_2\, \mathsf{be}\, x \triangleright y\, \mathsf{in}\, s_1) : A_1 \triangleright B_1\, \mathsf{be}\, w \triangleright z\, \mathsf{in}\, s_3 \rightsquigarrow_{\mathsf{c}} \mathsf{let}\, s_2 : A_2 \triangleright B_2\, \mathsf{be}\, x \triangleright y\, \mathsf{in}\, (\mathsf{let}\, s_1 : A_1 \triangleright B_1\, \mathsf{be}\, w \triangleright z\, \mathsf{in}\, s_3)}$$

$$\overline{\mathsf{app}_r\, (\mathsf{let}\, s_2 : A_2 \triangleright B_2\, \mathsf{be}\, x \triangleright y\, \mathsf{in}\, s_1)\, s_3 \rightsquigarrow_{\mathsf{c}} \mathsf{let}\, s_2 : A_2 \triangleright B_2\, \mathsf{be}\, x \triangleright y\, \mathsf{in}\, (\mathsf{app}_r\, s_1\, s_3)}$$

$$\overline{\mathsf{app}_l\, (\mathsf{let}\, s_2 : A_2 \triangleright B_2\, \mathsf{be}\, x \triangleright y\, \mathsf{in}\, s_1)\, s_3 \rightsquigarrow_{\mathsf{c}} \mathsf{let}\, s_2 : A_2 \triangleright B_2\, \mathsf{be}\, x \triangleright y\, \mathsf{in}\, (\mathsf{app}_l\, s_1\, s_3)}$$

$$\overline{\mathsf{let}(\mathsf{let}\, s_2 : A \triangleright B\, \mathsf{be}\, x \triangleright y\, \mathsf{in}\, s_1) : \mathsf{F}X\, \mathsf{be}\, \mathsf{F}z\, \mathsf{in}\, s_3 \rightsquigarrow_{\mathsf{c}} \mathsf{let}\, s_2 : A \triangleright B\, \mathsf{be}\, x \triangleright y\, \mathsf{in}\, (\mathsf{let}\, s_1 : \mathsf{F}X\, \mathsf{be}\, \mathsf{F}z\, \mathsf{in}\, s_3)}$$

$$\overline{\mathsf{let}(\mathsf{let}\, s_2 : \mathsf{F}X\, \mathsf{be}\, \mathsf{F}x\, \mathsf{in}\, s_1) : \mathsf{Unit}\, \mathsf{be}\, \mathsf{triv}\, \mathsf{in}\, s_3 \rightsquigarrow_{\mathsf{c}} \mathsf{let}\, s_2 : \mathsf{F}X\, \mathsf{be}\, \mathsf{F}x\, \mathsf{in}\, (\mathsf{let}\, s_1 : \mathsf{Unit}\, \mathsf{be}\, \mathsf{triv}\, \mathsf{in}\, s_2)}$$

$$\overline{\mathsf{let}(\mathsf{let}\, s_2 : \mathsf{F}X\, \mathsf{be}\, \mathsf{F}x\, \mathsf{in}\, s_1) : A \triangleright B\, \mathsf{be}\, x \triangleright y\, \mathsf{in}\, s_3 \rightsquigarrow_{\mathsf{c}} \mathsf{let}\, s_2 : \mathsf{F}X\, \mathsf{be}\, \mathsf{F}x\, \mathsf{in}\, (\mathsf{let}\, s_1 : A \triangleright B\, \mathsf{be}\, x \triangleright y\, \mathsf{in}\, s_3)}$$

$$\overline{\mathsf{app}_r\, (\mathsf{let}\, s_2 : \mathsf{F}X\, \mathsf{be}\, \mathsf{F}x\, \mathsf{in}\, s_1)\, s_3 \rightsquigarrow_{\mathsf{c}} \mathsf{let}\, s_2 : \mathsf{F}X\, \mathsf{be}\, \mathsf{F}x\, \mathsf{in}\, (\mathsf{app}_r\, s_1\, s_3)}$$

$$\overline{\mathsf{app}_l\, (\mathsf{let}\, s_2 : \mathsf{F}X\, \mathsf{be}\, \mathsf{F}x\, \mathsf{in}\, s_1)\, s_3 \rightsquigarrow_{\mathsf{c}} \mathsf{let}\, s_2 : \mathsf{F}X\, \mathsf{be}\, \mathsf{F}x\, \mathsf{in}\, (\mathsf{app}_l\, s_1\, s_3)}$$

$$\overline{\mathsf{let}(\mathsf{let}\, s_2 : \mathsf{F}X\, \mathsf{be}\, \mathsf{F}x\, \mathsf{in}\, s_1) : \mathsf{F}Y\, \mathsf{be}\, \mathsf{F}y\, \mathsf{in}\, s_3 \rightsquigarrow_{\mathsf{c}} \mathsf{let}\, s_2 : \mathsf{F}X\, \mathsf{be}\, \mathsf{F}x\, \mathsf{in}\, (\mathsf{let}\, s_1 : \mathsf{F}Y\, \mathsf{be}\, \mathsf{F}y\, \mathsf{in}\, s_3)}$$

Figure 4: Commuting Conversions: Commutative/Non-commutative Lambek Calculus

formalization given here. Furthermore, we proved strong normalization of CNC logic via a translation to LNL logic. We omit these results due to space.

3 Adjoint Model

In this section we introduce Lambek Adjoint Models (LAMs). Benton's LNL model consists of a symmetric monoidal adjunction $F : C \dashv \mathcal{L} : G$ between a cartesian closed category C and a symmetric monoidal closed category $\mathcal{L}$. LAM consists of a monoidal adjunction between a symmetric monoidal closed category and a Lambek category.

Definition 2. *A **Lambek category** is a monoidal category $(\mathcal{L}, \triangleright, I', \alpha', \lambda', \rho')$ with two functors $- \rightharpoonup - : \mathcal{L}^{\mathsf{op}} \times \mathcal{L} \longrightarrow \mathcal{L}$ and $- \leftharpoonup - : \mathcal{L} \times \mathcal{L}^{\mathsf{op}} \longrightarrow \mathcal{L}$ such that the following two natural bijections hold:*

$$\mathsf{Hom}_{\mathcal{L}}(A \triangleright B, C) \cong \mathsf{Hom}_{\mathcal{L}}(A, B \rightharpoonup C) \qquad \mathsf{Hom}_{\mathcal{L}}(A \triangleright B, C) \cong \mathsf{Hom}_{\mathcal{L}}(B, C \leftharpoonup A)$$

Definition 3. *A **Lambek Adjoint Model (LAM)**, $(C, \mathcal{L}, F, G, \eta, \varepsilon)$, consists of*

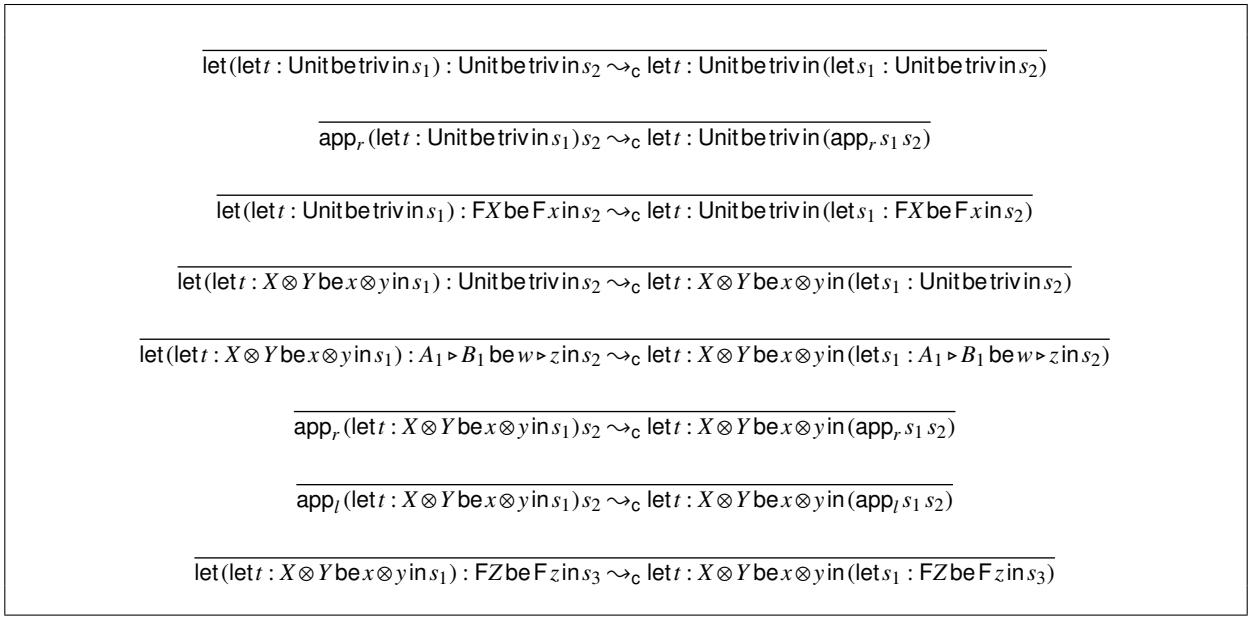

Figure 5: Commuting Conversions: Mixed Rules

- *a symmetric monoidal closed category* $(C, \otimes, I, \alpha, \lambda, \rho)$*;*
- *a Lambek category* $(\mathcal{L}, \triangleright, I', \alpha', \lambda', \rho')$*;*
- *a monoidal adjunction* $F : C \dashv \mathcal{L} : G$ *with unit* $\eta : \mathsf{Id}_C \to GF$ *and counit* $\varepsilon : FG \to \mathsf{Id}_{\mathcal{L}}$*, where* $(F : C \to \mathcal{L}, m)$ *and* $(G : \mathcal{L} \to C, n)$ *are monoidal functors.*

Following the tradition, we use letters X, Y, Z for objects in C and A, B, C for objects in $\mathcal{L}$. The rest of this section proves essential properties of any LAM.

An isomorphism. Let $(C, \mathcal{L}, F, G, \eta, \varepsilon)$ be a LAM, where (F, m) and (G, n) are monoidal functors. Similarly as in Benton's LNL model, $\mathsf{m}_{X,Y} : FX \triangleright FY \longrightarrow F(X \otimes Y)$ are components of a natural isomorphism, and $\mathsf{m}_I : I' \longrightarrow FI$ is an isomorphism. This is essential for modeling certain rules of CNC logic, such as tensor elimination in natural deduction. We define the inverses of $\mathsf{m}_{X,Y} : FX \triangleright FY \to F(X \otimes Y)$ and $\mathsf{m}_I : I' \to FI$ as:

$$\mathsf{p}_{X,Y} : F(X \otimes Y) \xrightarrow{F(\eta_X \otimes \eta_Y)} F(GFX \otimes GFY) \xrightarrow{F\mathsf{n}_{FX,FY}} FG(FX \triangleright FY) \xrightarrow{\varepsilon_{FX \triangleright FX}} FX \triangleright FY$$

$$\mathsf{p}_I : FI \xrightarrow{F\mathsf{n}_{I'}} FGI' \xrightarrow{\varepsilon_{I'}} I'$$

We can now see that this straightforwardly follows from Benton's proof.

Theorem 4. $\mathsf{m}_{X,Y}$ *are components of a natural isomorphism and their inverses are* $\mathsf{p}_{X,Y}$*.*

Proof. This proof follows by a diagram chase using the adjunction in the definition of a LAM and naturality. □

Strong non-commutative monad. Next we show that the monad on C in LAM is strong but non-commutative. In Benton's LNL model, the monad on the cartesian closed category is commutative, but later Benton and Wadler [3] wonder, is it possible to model non-commutative monads using adjoint models similar to LNL models? The following answers their question in the positive.

Lemma 5. *The monad induced by any LAM, $GF : C \longrightarrow C$, is monoidal.*

However, the monad is not symmetric because the following diagram does not commute.

$$\begin{array}{ccccc} GFX \otimes GFY & \xrightarrow{\mathsf{ex}_{GFX,GFY}} & GFY \otimes GFX & \xrightarrow{\mathsf{n}_{FY,FX}} & G(FY \triangleright FX) \\ \downarrow{\scriptstyle \mathsf{n}_{FX,FY}} & & & & \downarrow{\scriptstyle G\mathsf{m}_{Y,X}} \\ G(FX \triangleright FY) & \xrightarrow[G\mathsf{m}_{X,Y}]{} & GF(X \otimes Y) & \xrightarrow[GF\mathsf{ex}_{X,Y}]{} & GF(Y \otimes X) \end{array}$$

Commutativity fails, because the functors defining the monad are not symmetric monoidal, but only monoidal. This means that the diagram

$$\begin{array}{ccc} FA \otimes' FB & \xrightarrow{\mathsf{ex}_{FA,FB}} & FB \otimes' FA \\ \downarrow{\scriptstyle \mathsf{m}_{A,B}} & & \downarrow{\scriptstyle \mathsf{m}_{B,A}} \\ F(A \otimes B) & \xrightarrow[F\mathsf{ex}_{A,B}]{} & F(B \otimes A) \end{array}$$

does not hold for G nor F. However, we can prove the monad is strong.

Lemma 6. *The monad, $GF : C \longrightarrow C$, on the symmetric monoidal closed category in LAM is strong.*

Finally, we obtain the non-communativity of the monad induced by any LAM as follows.

Lemma 7 (Due to Kock [12])**.** *Let $\mathcal{M}$ be a symmetric monoidal category and T be a strong monad on $\mathcal{M}$. Then T is commutative iff it is symmetric monoidal.*

Theorem 8. *The monad, $GF : C \longrightarrow C$, on the SMCC in LAM is strong but non-commutative.*

Proof. This proof follows from Lemma 6 and Lemma 7. □

Comonad for exchange. We conclude this section by showing that the comonad induced by any LAM is monoidal and extends $\mathcal{L}$ with exchange. The latter is shown by proving that its corresponding co-Eilenberg-Moore category is symmetric monoidal closed.

Lemma 9. *The comonad, $FG : \mathcal{L} \longrightarrow \mathcal{L}$, on the Lambek category in any LAM is monoidal.*

Proof. This proof follows by several diagram chases, but really does not depart much from Benton's proof [4]. □

Theorem 10. *Given a LAM $(C, \mathcal{L}, F, G, \eta, \varepsilon)$ and the comonad $FG : \mathcal{L} \longrightarrow \mathcal{L}$, the co-Eilenberg-Moore category $\mathcal{L}^{FG}$ has an exchange natural transformation $\mathsf{ex}^{FG}_{A,B} : A \triangleright B \to B \triangleright A$.*

Proof. The natural transformation $\mathsf{ex}^{FG}_{A,B} : A \triangleright B \to B \triangleright A$ is defied as follows:

$$A \triangleright B \xrightarrow{h_A \triangleright h_B} FGA \triangleright FGB \xrightarrow{\mathsf{m}_{GA,GB}} F(GA \otimes GB) \xrightarrow{F\mathsf{ex}_{GA,GB}} F(GB \otimes GA) \xrightarrow{F\mathsf{n}_{B,A}} FG(B \triangleright A) \xrightarrow{\varepsilon_{B \triangleright A}} B \triangleright A$$

in which ex is the exchange for C. The remainder of the proof is fairly straightforward, and shows that the typical diagrams for any symmetric monoidal category hold. □

Corollary 11. *The subcategory, $\mathsf{Exp}(\mathcal{L}^{FG})$, of the co-Eilenberg-Moore category $\mathcal{L}^{FG}$ consisting of the free exponential coalgebras is symmetric monoidal closed.*

Proof. The proof that $\mathsf{Exp}(\mathcal{L}^{FG})$ is monoidal closed follows similarly to Benton [4]. □

4 A Model in Dialectica Spaces

In this section we give a different categorical model in terms of dialectica categories; which are a sound and complete categorical model of the Lambek Calculus as was shown by de Paiva and Eades [8]. This section is largely the same as the corresponding section de Paiva and Eades give, but with some modifications to their definition of biclosed posets with exchange (see Definition 16). However, we try to make this section as self contained as possible.

Dialectica categories were first introduced by de Paiva as a categorification of Gödel's Dialectica interpretation [7]. Dialectica categories were one of the first sound categorical models of intuitionistic linear logic with linear modalities. We show in this section that they can be adapted to become a sound and complete model for CNC logic, with both the exchange and of-course modalities. Due to the complexities of working with dialectica categories we have formally verified[1] this section in the proof assistant Agda [6].

First, we define the notion of a biclosed poset. These are used to control the definition of morphisms in the dialectica model.

Definition 12. *Suppose $(M,\leq,\circ,e)$ is an ordered non-commutative monoid. If there exists a largest $x \in M$ such that $a \circ x \leq b$ for any $a,b \in M$, then we denote x by $a \rightharpoonup b$ and called it the **left-pseudocomplement** of a w.r.t b. Additionally, if there exists a largest $x \in M$ such that $x \circ a \leq b$ for any $a,b \in M$, then we denote x by $b \leftharpoonup a$ and called it the **right-pseudocomplement** of a w.r.t b.*

*A **biclosed poset**, $(M,\leq,\circ,e,\rightharpoonup,\leftharpoonup)$, is an ordered non-commutative monoid, $(M,\leq,\circ,e)$, such that $a \rightharpoonup b$ and $b \leftharpoonup a$ exist for any $a,b \in M$.*

Now using the previous definition we define dialectica Lambek spaces.

Definition 13. *Suppose $(M,\leq,\circ,e,\rightharpoonup,\leftharpoonup)$ is a biclosed poset. Then we define the category of **dialectica Lambek spaces**, $\mathsf{Dial}_M(\mathsf{Set})$, as follows:*

- *objects, or dialectica Lambek spaces, are triples (U,X,α) where U and X are sets, and $\alpha : U \times X \longrightarrow M$ is a generalized relation over M, and*
- *maps that are pairs $(f,F) : (U,X,\alpha) \longrightarrow (V,Y,\beta)$ where $f : U \longrightarrow V$, and $F : Y \longrightarrow X$ are functions such that the weak adjointness condition $\forall u \in U. \forall y \in Y. \alpha(u,F(y)) \leq \beta(f(u),y)$ holds.*

Notice that the biclosed poset is used here as the target of the relations in objects, but also as providing the order relation in the weak adjoint condition on morphisms. This will allow the structure of the biclosed poset to lift up into $\mathsf{Dial}_M(\mathsf{Set})$.

We will show that $\mathsf{Dial}_M(\mathsf{Set})$ is a model of the Lambek Calculus with modalities. First, we must show that $\mathsf{Dial}_M(\mathsf{Set})$ is monoidal biclosed.

Definition 14. *Suppose (U,X,α) and (V,Y,β) are two objects of $\mathsf{Dial}_M(\mathsf{Set})$. Then their tensor product is defined as follows:*

$$(U,X,\alpha) \triangleright (V,Y,\beta) = (U \times V, (V \to X) \times (U \to Y), \alpha \triangleright \beta)$$

where $- \to -$ is the function space from Set, and $(\alpha \triangleright \beta)((u,v),(f,g)) = \alpha(u,f(v)) \circ \beta(g(u),v)$.

It follows from de Paiva and Eades [8] that this does indeed define a monoidal tensor product, but take note of the fact that this tensor product is indeed non-commutative, because the non-commutative multiplication of the biclosed poset is used to define the relation of the tensor product.

The tensor product has two right adjoints making $\mathsf{Dial}_M(\mathsf{Set})$ biclosed.

[1] The complete formalization can be found online at `https://github.com/MonoidalAttackTrees/non-comm-monads-adjoint-models/tree/master/dialectica-formalization`.

Definition 15. *Suppose* (U,X,α) *and* (V,Y,β) *are two objects of* $\mathsf{Dial}_M(\mathsf{Set})$*. Then two internal-homs can be defined as follows:*

$$(U,X,\alpha) \rightharpoonup (V,Y,\beta) = ((U \to V)\times(Y \to X), U\times Y, \alpha \rightharpoonup \beta)$$
$$(V,Y,\beta) \leftharpoonup (U,X,\alpha) = ((U \to V)\times(Y \to X), U\times Y, \alpha \leftharpoonup \beta)$$

It is straightforward to show that the typical bijections defining the corresponding adjunctions hold; see de Paiva and Eades for the details [8].

We now extend $\mathsf{Dial}_M(\mathsf{Set})$ with two modalities: the usual modality, of-course, denoted $!A$, and the exchange modality denoted ξA. However, we must first extended biclosed posets to include an exchange operation.

Definition 16. *A* ***biclosed poset with exchange*** *is a biclosed poset* $(M,\leq,\circ,e,\rightharpoonup,\leftharpoonup)$ *equipped with an unary operation* $\xi : M \to M$ *satisfying the following:*

(Compatibility)	$a \leq b$ *implies* $\xi a \leq \xi b$ *for all* $a,b,c \in M$	*(Duplication)*	$\xi a \leq \xi\xi a$ *for all* $a \in M$
(Minimality)	$\xi a \leq a$ *for all* $a \in M$	*(Exchange)*	$(\xi a \circ \xi b) \leq (\xi b \circ \xi a)$ *for all* $a,b \in M$

This definition is where the construction given here departs from the definition of biclosed posets with exchange given by de Paiva and Eades [8].

We can now define the two modalities in $\mathsf{Dial}_M(\mathsf{Set})$ where M is a biclosed poset with exchange.

Definition 17. *Suppose* (U,X,α) *is an object of* $\mathsf{Dial}_M(\mathsf{Set})$ *where* M *is a biclosed poset with exchange. Then the* ***of-course*** *and* ***exchange*** *modalities can be defined as* $!(U,X,\alpha) = (U, U \to X^*, !\alpha)$ *and* $\xi(U,X,\alpha) = (U,X,\xi\alpha)$ *where* X^* *is the free commutative monoid on* X*,* $(!\alpha)(u,f) = \alpha(u,x_1)\circ\cdots\circ\alpha(u,x_i)$ *for* $f(u) = (x_1,\ldots,x_i)$*, and* $(\xi\alpha)(u,x) = \xi(\alpha(u,x))$*.*

This definition highlights a fundamental difference between the two modalities. The definition of the exchange modality relies on an extension of biclosed posets with essentially the exchange modality in the category of posets. However, the of-course modality is defined by the structure already present in $\mathsf{Dial}_M(\mathsf{Set})$, specifically, the structure of Set.

Both of the modalities have the structure of a comonad. That is, there are monoidal natural transformations $\varepsilon_! : !A \longrightarrow A$, $\varepsilon_\xi : \xi A \longrightarrow A$, $\delta_! : !A \longrightarrow !!A$, and $\delta_\xi : \xi A \longrightarrow \xi\xi A$ which satisfy the appropriate diagrams; see the formalization for the full proofs. Furthermore, these comonads come equipped with arrows $w : !A \longrightarrow I$, $d : !A \longrightarrow !A \otimes !A$, $\mathsf{ex}_{A,B} : \xi A \otimes \xi B \longrightarrow \xi B \otimes \xi A$.

Finally, using the fact that $\mathsf{Dial}_M(\mathsf{Set})$ for any biclosed poset is essentially a non-commutative formalization of Bierman's linear categories [5] we can use Benton's construction of an LNL model from a linear category to obtain a LAM model, and hence, obtain the following.

Theorem 18. *Suppose* M *is a biclosed poset with exchange. Then* $\mathsf{Dial}_M(\mathsf{Set})$ *is a sound and complete (w.r.t. derivability and the equational theory) model for CNC logic.*

5 Future Work

We introduce the idea above of having a modality for exchange, but what about individual modalities for weakening and contraction? Indeed it is possible to give modalities for these structural rules as well using adjoint models. Now that we have each structural rule isolated into their own modality is it possible to put them together to form new modalities that combine structural rules? The answer to this question has already been shown to be positive, at least for weakening and contraction, by Melliés [14], but we plan to extend this line of work to include exchange.

The monads induced by the adjunction in CNC logic is non-commutative, but Benton and Wadler show that the monads induced by the adjunction in LNL logic [3] are commutative. Using the extension of Melliés' work we mention above would allow us to combine both CNC logic with LNL logic, and then be able to support both commutative monads as well as non-commutative monads. We plan on exploring this in the future.

Hasegawa [9] studies the linear of-course modality, $!A$, as a comonad induced by an adjunction between a cartesian closed category a (non-symmetric) monoidal category. The results here generalizes his by generalizing the cartesian closed category to a symmetric monoidal closed category. However, his approach focuses on the comonad rather than the adjunctions. It would be interesting to do the same for LAM as well.

References

[1] Samson Abramsky. Proofs as processes. *Theoretical Computer Science*, 135(1):5 – 9, 1994. URL: http://www.sciencedirect.com/science/article/pii/0304397594001030, doi:http://dx.doi.org/10.1016/0304-3975(94)00103-0.

[2] Guy Barry, Mark Hepple, Neil Leslie, and Glyn Morrill. Proof figures and structural operators for categorial grammar. In *Proceedings of the Fifth Conference on European Chapter of the Association for Computational Linguistics*, EACL '91, pages 198–203, Stroudsburg, PA, USA, 1991. Association for Computational Linguistics. URL: https://doi.org/10.3115/977180.977215, doi:10.3115/977180.977215.

[3] Nick Benton and Philip Wadler. Linear logic, monads and the lambda calculus. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science*, LICS '96, pages 420–, Washington, DC, USA, 1996. IEEE Computer Society.

[4] P. N. Benton. A mixed linear and non-linear logic: Proofs, terms and models (preliminary report). Technical Report UCAM-CL-TR-352, University of Cambridge Computer Laboratory, 1994.

[5] G. M. Bierman. *On Intuitionistic Linear Logic*. PhD thesis, Wolfson College, Cambridge, December 1993.

[6] Ana Bove, Peter Dybjer, and Ulf Norell. A brief overview of Agda-a functional language with dependent types. *TPHOLs*, 5674:73–78, 2009.

[7] Valeria de Paiva. *The dialectica categories*. PhD thesis, Computer Laboratory, University of Cambridge, PhD Thesis, 1990. Computer Laboratory, University of Cambridge, PhD Thesis.

[8] Valeria de Paiva and Harley Eades. *Dialectica Categories for the Lambek Calculus*, pages 256–272. Springer International Publishing, Cham, 2018. URL: https://doi.org/10.1007/978-3-319-72056-2_16, doi:10.1007/978-3-319-72056-2_16.

[9] Masahito Hasegawa. Linear exponential comonads without symmetry. In Iliano Cervesato and Maribel Fernández, editors, Proceedings Fourth International Workshop on *Linearity,* Porto, Portugal, 25 June 2016, volume 238 of *Electronic Proceedings in Theoretical Computer Science*, pages 54–63. Open Publishing Association, 2017. doi:10.4204/EPTCS.238.6.

[10] Mark Hepple. *The grammar and processing of order and dependency: A categorial approach*. PhD thesis.

[11] Kohei Honda and Olivier Laurent. An exact correspondence between a typed pi-calculus and polarised proof-nets. *Theoretical Computer Science*, 411(22):2223 – 2238, 2010. URL: http://www.sciencedirect.com/science/article/pii/S0304397510000538, doi:http://dx.doi.org/10.1016/j.tcs.2010.01.028.

[12] Anders Kock. Strong functors and monoidal monads. *Archiv der Mathematik*, 23(1):113–120, 1972.

[13] Joachim Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, pages 154–170, 1958.

[14] Paul-André Melliés. Comparing hierarchies of types in models of linear logic. *Information and Computation*, 189(2):202 – 234, 2004.

[15] Michael Moortgat. Categorial investigations: Logical and linguistics aspects of the lambek calculus. 1988.

[16] Richard T Oehrle, Emmon Bach, and Deirdre Wheeler. *Categorial grammars and natural language structures*, volume 32. Springer Science & Business Media, 2012.

[17] Vaughan R. Pratt. Types as processes, via Chu spaces. *Electronic Notes in Theoretical Computer Science*, 7:227 – 247, 1997.