Final Report for Formalizing Non-Determinism in the Categorical Message Passing Language

Alexanna Little
University of Calgary

Robin Cockett [1]
Department of Computer Science, University of Calgary

Abstract

The semantics for non-determinism is defined in Categorical Message Passing Language (CaMPL) using sup-lattice enrichment, and in this project, we investigate how existing categorical structures that give important programming features to CaMPL are changed or preserved after the sup-lattice enrichment. Categorical structures are defined by the existence of certain relationships between objects, maps, and/or functors which must adhere to specific coherence conditions. The effect of sup-lattice enrichment can be understood by creating coherence diagrams of the defining relationships for a categorical structure and its sup-lattice enriched version. We have shown that the linearly distributive category is preserved after sup-lattice enrichment, and we have shown how the products and coproducts, inductive and coinductive datatypes, and protocols and coprotocols are changed but still maintain meaningful properties.

2012 ACM Subject Classification Theory of computation → Distributed computing models

Keywords and phrases concurrency, message passing, category theory, non-determinism, progress

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Introduction

Category theory defines mathematical structures in terms of relationships which are often called maps. This makes it well suited for the application of functional programming because functional programs can be thought of as a map from their input type to their output type. Categorical Message Passing Language (CaMPL) is a functional concurrent programming language where the semantics of the programming features of the language are defined by mathematical structures in category theory [1].

Non-determinism was added to CaMPL by changing the categorical structure that defined the semantics for its concurrent processes so that non-deterministic processes could be modelled. The method this was achieved by is called enrichment. Non-deterministic processes are modelled using sup-lattices, so the semantics for non-determinism was added by sup-lattice enrichment [4]. In this research project, I investigate how adding the ability for CaMPL programs to be non-deterministic changes the categorical semantics for the other features of the language.

CaMPL uses message passing as its form of concurrency, so programs are composed of multiple concurrent processes which can communicate with each other while they are running by passing messages along communication channels. The categorical semantics for how concurrent processes can be connected by communication channels is a linearly distributive category [1]. Deterministic CaMPL programs have been shown to have an important property called progress which guarantees that they will never result in a deadlock or livelock [5]. This guarantee is given by the categorical semantics of a linearly distributive category. In this

[1] Supervisor

project, we show the categorical semantics for how processes can be connected by channels in the non-deterministic semantics is still defined by a linearly distributive category.

The concurrent processes independently run their own sequential functional programs, and sending or receiving a message is an operation they can perform to send an output to or receive an input from another concurrent process. The semantics for passing a message along a channel is given by one of the two action functors in the linear actegory which represents the entire deterministic semantics of CaMPL [1]. The functor that is used depends on the direction the message is being passed on a channel, so if one functor is used for receiving a message, the other functor is used for sending a message.

A process can also initiate a session of message passing with another process according to a protocol or coprotocol on the communication channel between them. The difference between a protocol and a coprotocol is the direction it is initiated because, as mentioned previously, the semantics make a distinction between the directions messages are passed. Protocols and coprotocols are also called session types, and they are useful features because they allow for typed interactions of message passing operations which have the ability to be repeated an unspecified number of times. The deterministic semantics for protocols and coprotocols are given by inductive and coinductive datatypes [9]. The semantics for protocols and coprotocols are not the same in a non-deterministic semantics. However, they do exist in a non-deterministic way. In this project, we explore the non-deterministic categorical structures that can define the semantics for protocols and coprotocols.

2 Related Work

While working on CaMPL during this project, we considered many papers from authors working on projects with varying amounts of direct relevance. In this section, I have decided to mainly discuss the papers which document directly relevant contributions to CaMPL by authors who have previously worked with Dr. Cockett.[2] This is to clearly establish the context of my contributions to CaMPL which will be discussed in the Results section 4.

In this project, I have been exploring how the categorical structures that define the semantics of important features to CaMPL change after sup-lattice enrichment. Specifically, I look at the structures that define the semantics for guaranteeing no deadlocks and livelocks and for protocols and coprotocols. Previous work on CaMPL showed how the deterministic semantics for these features were defined and why these features are important.

The semantics of CaMPL were originally established in Cockett and Pastro's paper which specifically showed how a linear actegory gives the semantics for a concurrent programming language that uses message passing concurrency [1]. Cockett and Pastro's paper also took inspiration from Girard's paper on linear logic which suggested that the semantics of a concurrent programming language could be defined to guarantee no deadlocks or livelocks [3]. Then Lybbert's paper showed that the guarantee of deadlock and livelock freedom from Girard's paper was true for deterministic CaMPL programs [5]. Lybbert's paper is important to this project because it established that CaMPL has the useful properties it was intended to have which gives us the motivation now to show that CaMPL still has these properties after we add non-determinism.

[2] Most of these papers are unpublished project reports similar to this final report, and I am able to access them because Dr. Cockett has dutifully maintained an archive of the work that has been done on CaMPL over many years.

Yeasin's paper showed how inductive and coinductive datatypes provide the categorical semantics for protocols and coprotocols in CaMPL [9]. Later, Pon's paper described an implementation of CaMPL which includes protocols and coprotocols and races, which are the feature of CaMPL that allows programs to be non-deterministic [8]. The contribution of races to CaMPL by Pon created inspiration for the research problem "how do we define the categorical semantics of non-deterministic processes?" This question was responded to in my own, also unpublished, paper from my research project last summer where I discussed how a sup-lattice is an appropriate structure to represent a non-deterministic process, how including a non-deterministic process with a race in a CaMPL program makes that entire program non-deterministic, and how we can introduce the semantics for non-determinism to CaMPL by sup-lattice enrichment [4].

3 Methods

In category theory, coherence diagrams are used to show consistency of relationships that define categorical structures. These are diagrams of an abstract case that can be applied to all cases by showing that the necessary coherence maps exist and therefore the necessary equivalences exist which make the diagrams commute.

To understand the linearly distributive structure in the non-deterministic sup-lattice enriched semantics, we want to show that sup-lattice enriching a linearly distributive category produces a linearly distributive category. So to show that the linearly distributive structure was preserved, we must define the necessary maps and create coherence diagrams for the linearly distributive category and the sup-lattice enriched category. If the coherence diagrams can be drawn and shown to commute, then sup-lattice enriching a linearly distributive category gives a linearly distributive category.

Similarly, to understand inductive and coinductive datatypes in the non-deterministic sup-lattice enriched semantics, we want to show their universal properties exist in the non-deterministic semantics in a meaningful way. We also explore an example of non-determinizing an inductive datatype and show how the diagram commutes.

4 Results

4.1 Linearly distributive categories

First we will summarize how sup-lattice enriching a linearly distributive category gives a linearly distributive sup-lattice enriched category. To do so, we show how arbitrary natural transformations are lifted from a category to a sup-lattice enriched category, and we will use this to show how arbitrary monoidal structures are lifted from a category to a sup-lattice enriched category. We define a linearly distributive category in terms of its monoidal structures and natural transformations, and it is sufficient to show that the arbitrary cases are true to conclude that a linearly distributive category gives a linearly distributive sup-lattice enriched category.

4.1.1 Power set functors

Let $\mathscr{P} : \mathbf{Set} \to \mathbf{Set}$ be a functor that denotes taking the power set of a set. We can write $\mathscr{P} = \mathcal{PU}$ if we let $\mathcal{P} : \mathbf{Set} \to \mathbf{SupLat}$ be the functor that lifts a set to a sup-lattice by taking its power set and $\mathcal{U} : \mathbf{SupLat} \to \mathbf{Set}$ be the underlying functor that maps a sup-lattice back to the set that is its carrier. These functors form an adjunction

$(\eta, \epsilon) : \mathcal{P} \dashv \mathcal{U} : \mathbf{Set} \to \mathbf{SupLat}$ where $\eta : 1_{\mathbf{Set}} \Rightarrow \mathcal{PU}$ sends elements of a set to singleton sets. We can see this better if we use the equivalence from above and consider an abstract component map $\eta_X : X \to \mathscr{P}(X)\ ;\ x \mapsto \{x\}$ for $X \in \mathbf{Set}$.

4.1.2 Natural transformations

A natural transformation $\alpha : F \Rightarrow G$ is a map between functors $F : \mathbb{X} \to \mathbb{Y}$, and $G : \mathbb{X} \to \mathbb{Y}$:

$$\mathbb{X} \underset{G}{\overset{F}{\Downarrow \alpha}} \mathbb{Y}$$

We define α as a family of component maps $\alpha_X, \forall X \in \mathbb{X}$, such that there is an equivalence of paths $\alpha_X\ G(x) = F(x)\ \alpha_Y$ for all $X, Y, x : X \to Y \in \mathbb{X}$ in the following naturality square diagram:

$$\begin{array}{ccc} F(X) & \xrightarrow{\alpha_X} & G(X) \\ {\scriptstyle F(x)}\downarrow & & \downarrow{\scriptstyle G(x)} \\ F(Y) & \xrightarrow[\alpha_Y]{} & G(Y) \end{array}$$

We want to show that we can use $\mathcal{P} : \mathbf{Set} \to \mathbf{SupLat}$ to lift natural transformations in $\mathbf{Set}$ to natural transformations in $\mathbf{SupLat}$. That is, if $\alpha : F \Rightarrow G : \mathbb{X} \to \mathbb{Y}$ is a natural transformation, then $\eta(\alpha) : \mathcal{P}(F) \Rightarrow \mathcal{P}(G) : \mathcal{P}(\mathbb{X}) \to \mathcal{P}(\mathbb{Y})$ is also a natural transformation. We define $\eta(\alpha)$ with a naturality square for $X, Y, f : X \to Y \in \mathcal{P}(\mathbb{X})$:

$$\begin{array}{ccc} \mathcal{P}(F)(X) & \xrightarrow{\eta(\alpha)_X} & \mathcal{P}(G)(X) \\ {\scriptstyle \mathcal{P}(F)(f)}\downarrow & & \downarrow{\scriptstyle \mathcal{P}(G)(f)} \\ \mathcal{P}(F)(Y) & \xrightarrow[\eta(\alpha)_Y]{} & \mathcal{P}(G)(Y) \end{array}$$

And check that the naturality square commutes by showing $\eta(\alpha)_X\ \mathcal{P}(G)(f) = \mathcal{P}(F)(f)\ \eta(\alpha)_Y$:

$$\begin{aligned} & \eta(\alpha)_X\ \mathcal{P}(G)(f) \\ =& \{\alpha_X\}\ \{G(x) \mid x \in f\} \\ =& \{\alpha_X\ G(x) \mid x \in f\} \\ =& \{F(x)\ \alpha_Y \mid x \in f\} \\ =& \{F(x) \mid x \in f\}\ \{\alpha_Y\} \\ =& \mathcal{P}(F)(f)\ \eta(\alpha)_Y \end{aligned}$$

We have shown the naturality square commutes which is the coherence condition for natural transformations, so we have shown that $\mathcal{P} : \mathbf{Set} \to \mathbf{SupLat}$ lifts natural transformations in $\mathbf{Set}$ to natural transformations in $\mathbf{SupLat}$.

4.1.3 Monoidal structures

A monoidal structure $(\otimes, \top, a, u^L, u^R)$ on a category $\mathbb{X}$ has a functor $\otimes$, called the tensor, with unit object $\top$, and natural isomorphisms a, called the associator, u^L called the left unitor, and u^R called the right unitor [7]. These data have the following types with $X, Y, Z \in \mathbb{X}$:

$$\begin{aligned} \otimes &: \mathbb{X} \times \mathbb{X} \to \mathbb{X} \\ \top &\in \mathbb{X} \\ a_{X,Y,Z} &: (X \otimes Y) \otimes Z \to X \otimes (Y \otimes Z) \\ u^L_X &: \top \otimes X \to X \\ u^R_X &: X \otimes \top \to X \end{aligned}$$

And the natural isomorphisms must satisfy the Maclane pentagon, (1), and Kelly triangle, (2), coherence diagrams with $W, X, Y, Z \in \mathbb{X}$ [7]:

$$\begin{array}{ccc} ((W \otimes X) \otimes Y) \otimes Z & \xrightarrow{a_{W,X,Y} \otimes 1_Z} & (W \otimes (X \otimes Y)) \otimes Z \\ \downarrow{\scriptstyle a_{W \otimes X, Y, Z}} & & \downarrow{\scriptstyle a_{W, X \otimes Y, Z}} \\ (W \otimes X) \otimes (Y \otimes Z) & & W \otimes ((X \otimes Y) \otimes Z) \\ & {\scriptstyle a_{W,X,Y \otimes Z}} \searrow \quad \swarrow {\scriptstyle 1_W \otimes a_{X,Y,Z}} & \\ & W \otimes (X \otimes (Y \otimes Z)) & \end{array} \tag{1}$$

$$\begin{array}{ccc} (X \otimes \top) \otimes Y & \xrightarrow{a_{X,\top,Y}} & X \otimes (\top \otimes Y) \\ & {\scriptstyle u^R_X \otimes 1_Y} \searrow \quad \swarrow {\scriptstyle 1_X \otimes u^L_Y} & \\ & X \otimes Y & \end{array} \tag{2}$$

We can summarize these conditions by saying that the tensor $\otimes$ is associative and follows unit laws. If a category $\mathbb{X}$ has a tensor that satisfies these coherence conditions, we say $\mathbb{X}$ is a monoidal category.

We want to show that $\mathcal{P} : \mathbf{Set} \to \mathbf{SupLat}$ is a monoidal functor; that is, the monoidal structure in **Set** is lifted to the monoidal structure in **SupLat**. Let the monoidal structure in **Set** be $(\times, \emptyset, a_\times, u^L_\times, u^R_\times)$ where the functor $\times$ is the cartesian product, which is associative, and $\emptyset$ is the empty set, which satisfies the unit laws. And let the monoidal structure in **SupLat** be $(\otimes, I, a_\otimes, u^L_\otimes, u^R_\otimes)$ [4].

To show that $\mathcal{P}$ is a monoidal functor, we must define coherence maps $m_1 : \emptyset \to \mathcal{P}(\emptyset)$ and $m_\otimes : \mathcal{P}(\mathbf{Set}) \otimes \mathcal{P}(\mathbf{Set}) \to \mathcal{P}(\mathbf{Set} \times \mathbf{Set})$ so that associativity and unit conditions are satisfied in **SupLat**. The map m_1 is a map in **SupLat**. The power set of the empty set is $\mathcal{P}(\emptyset) = \{\emptyset\}$, and we define $m_1 = I \to \mathcal{P}(\emptyset); \top \mapsto \{\emptyset\}, \bot \mapsto \emptyset$. The map $m_\otimes$ is a natural transformation, and we can define $m_\otimes = (X, Y) \to (X, Y)$ on objects $X, Y \in \mathbf{Set}$ and $m_\otimes = (\{f_i \mid f_i \in f\}, \{g_j \in g\}) \to \{(f_i, g_j) \mid f_i \in f, g_j \in g\}$ on maps $f, g \in \mathbf{Set}$.

The left and right unitor coherence diagrams with $X \in \mathbf{Set}$ are as follows:

$$\begin{array}{ccc} I \otimes \mathcal{P}(X) & \xrightarrow{m_1 \otimes 1_{\mathcal{P}(X)}} & \mathcal{P}(\emptyset) \otimes \mathcal{P}(X) \\ {\scriptstyle u^L_\otimes}\downarrow & & \downarrow{\scriptstyle m_\otimes} \\ \mathcal{P}(X) & \xleftarrow[\mathcal{P}(u^L_\times)]{} & \mathcal{P}(\emptyset \times X) \end{array} \qquad \begin{array}{ccc} \mathcal{P}(X) \otimes I & \xrightarrow{1_{\mathcal{P}(X)} \otimes m_1} & \mathcal{P}(X) \otimes \mathcal{P}(\emptyset) \\ {\scriptstyle u^R_\otimes}\downarrow & & \downarrow{\scriptstyle m_\otimes} \\ \mathcal{P}(X) & \xleftarrow[\mathcal{P}(u^R_\times)]{} & \mathcal{P}(X \times \emptyset) \end{array}$$

And the coherence diagram for associativity with $X, Y, Z \in \mathbf{Set}$ is as follows:

$$\begin{array}{ccc} (\mathcal{P}(X) \otimes \mathcal{P}(Y)) \otimes \mathcal{P}(Z) & \xrightarrow{a_\otimes} & \mathcal{P}(X) \otimes (\mathcal{P}(Y) \otimes \mathcal{P}(Z)) \\ {\scriptstyle m_\otimes \otimes 1_{\mathcal{P}(Z)}}\downarrow & & \downarrow{\scriptstyle 1_{\mathcal{P}(X)} \otimes m_\otimes} \\ \mathcal{P}(X \times Y) \otimes \mathcal{P}(Z) & & \mathcal{P}(X) \otimes \mathcal{P}(Y \times Z) \\ {\scriptstyle m_\otimes}\downarrow & & \downarrow{\scriptstyle m_\otimes} \\ \mathcal{P}((X \times Y) \times Z) & \xrightarrow[\mathcal{P}(a_\times)]{} & \mathcal{P}(X \times (Y \times Z)) \end{array}$$

So we have shown that $\mathcal{P} : \mathbf{Set} \to \mathbf{SupLat}$ is a monoidal functor which lifts the monoidal structure from **Set** to the monoidal structure in **SupLat** by defining the coherence maps and showing that the coherence conditions are satisfied.

4.1.4 Linearly distributive categories

A linearly distributive category $\mathbb{X}$ has two monoidal structures; one with a functor $\otimes$, called tensor: $(\otimes, \top, a_\otimes, u^L_\otimes, u^R_\otimes)$, and another with a functor $\oplus$, called par: $(\oplus, \bot, a_\oplus, u^L_\oplus, u^R_\oplus)$. Additionally, it has natural transformations δ^L and δ^R, called the left and right distributors with the following types [6]:

$$\delta^L_{X,Y,Z} : X \otimes (Y \oplus Z) \to (X \otimes Y) \oplus Z \qquad \delta^R_{X,Y,Z} : (X \oplus Y) \otimes Z \to X \oplus (Y \otimes Z)$$

There are symmetries in the interactions of these components which simplify the coherence diagrams for the distributors significantly. The [op'] symmetry, (3), reverses the arrows, swaps the $\otimes$ and $\oplus$, and swaps the units $\top$ and $\bot$. The [$\odot$'] symmetry, (4), reverses the $\otimes$ and $\oplus$. These give the following assignments [2]:

$$\delta^L \leftrightarrow \delta^L \qquad a_\otimes \mapsto a_\oplus^{-1} \qquad a_\oplus \mapsto a_\otimes^{-1} \tag{3}$$

$$\delta^L \leftrightarrow \delta^L \qquad a_\otimes \mapsto a_\otimes^{-1} \qquad a_\oplus \mapsto a_\oplus^{-1} \tag{4}$$

The distributors must satisfy the coherence diagrams and all variations given by the symmetries which are included in the Appendix A [2]. We can summarize these conditions by saying that the natural transformations δ^L and δ^R distribute the tensor $\otimes$ linearly over the par $\oplus$. If a category $\mathbb{X}$ has two monoidal structures that satisfy these coherence conditions, we say $\mathbb{X}$ is a linearly distributive category.

4.1.5 Lifting linearly distributive structures to a sup-lattice enriched category

We have shown that arbitrary natural transformations and monoidal structures in **Set** are lifted by the functor $\mathcal{P}$. Therefore the natural transformations in the linearly distributive

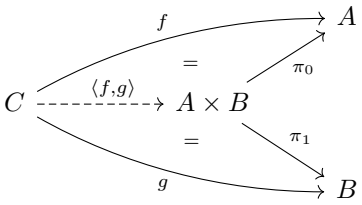
category $\mathbb{X}$, the associator, unitors, and linear distributors, will be lifted as singletons to the sup-lattice enriched category $\mathcal{P}(\mathbb{X})$. And the monoidal structures tensor and par in the linearly distributive category $\mathbb{X}$ will be lifted to the sup-lattice enriched category $\mathcal{P}(\mathbb{X})$. So we can conclude that, since all of these coherence conditions have been satisfied, applying the functor $\mathcal{P}$ to a linearly distributive category gives a linearly distributive sup-lattice enriched category.

4.2 Inductive and coinductive datatypes

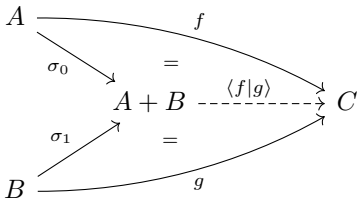
Now, we will summarize how the products and coproducts and inductive and coinductive datatypes change after sup-lattice enrichment. This is important for understanding how protocols and coprotocols behave in non-deterministic CaMPL programs. We want to establish that the universal properties for products and coproducts and inductive and coinductive datatypes exist in a meaningful way in the non-deterministic semantics for CaMPL. To demonstrate how useful concepts in computer science can be encoded with inductive and coinductive datatypes, we show an example of how a deterministic initial datatype can be lifted to a non-deterministic inital datatype.

4.2.1 Products and coproducts

Deterministic products have the universal property that there exists a unique map $\langle f, g\rangle$ such that $\langle f, g\rangle\pi_0 = f$ and $\langle f, g\rangle\pi_1 = g$ so the following diagram commutes:

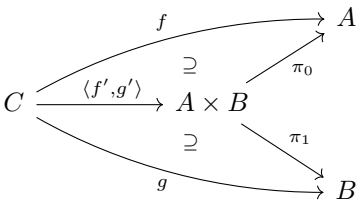


And deterministic coproducts have the universal property that there exists a unique map $\langle f \mid g\rangle$ such that $\sigma_0\langle f \mid g\rangle = f$ and $\sigma_1\langle f \mid g\rangle = g$ so the following diagram commutes:

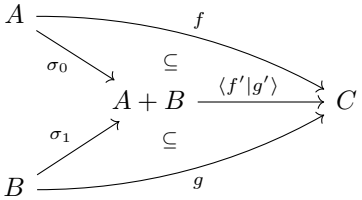


In a sup-lattice enriched category, the f and g maps are non-deterministic maps, so the unique maps will become $\langle f, g\rangle = \{\langle f_i, g_j\rangle \mid f_i \in f, g_j \in g\}$ for products and $\langle f \mid g\rangle = \{\langle f_i \mid g_j\rangle \mid f_i \in f, g_j \in g\}$ for coproducts. Since f and g are non-deterministic, we can make the diagrams commute with subset inclusion equalities where $\langle f', g'\rangle \subseteq \langle f, g\rangle$ and $\langle f' \mid g'\rangle \subseteq \langle f \mid g\rangle$ rather than strict equalities. However, $\langle f', g'\rangle$ and $\langle f' \mid g'\rangle$ are no longer unique as many subsets of $\langle f, g\rangle$ or $\langle f \mid g\rangle$ could make the diagram commute. So rather than

a universal property, we have a lax universal property for non-deterministic products:



And a lax universal property for non-deterministic coproducts:



It is important to note that when f and g are singleton sets $\langle\{f\},\{g\}\rangle = \{\langle f,g\rangle\}$ and $\langle\{f\} \mid \{g\}\rangle = \{\langle f \mid g\rangle\}$. This means $\langle f',g'\rangle = \langle f,g\rangle$ and $\langle f' \mid g'\rangle = \langle f \mid g\rangle$, and the product and coproduct are recaptured.

4.2.2 Inductive and coinductive datatypes

In the deterministic semantics, inductive and coinductive datatypes are datatypes that can be constructed inductively or coinductively, respectfully [9]. Examples of inductive datatypes include natural numbers, lists, and trees, and examples of coinductive datatypes include conatural numbers and infinite lists. More generally, inductive and coinductive datatypes are initial algebras and final coalgebras for a functor $F : \mathbb{X} \to \mathbb{X}$ [9].

An F-algebra for a functor $F : \mathbb{X} \to \mathbb{X}$ is a pair $(A, f : F(A) \to A)$ of an object and a map in $\mathbb{X}$. An initial algebra for F is a pair $(F^{@}, cons : F(F^{@}) \to F^{@})$ such that for every F-algebra there exists a unique map $f' = fold(f)$ which makes the following diagram commute:

$$\begin{array}{ccc} F(F^{@}) & \xrightarrow{cons} & F^{@} \\ {\scriptstyle F(f')}\downarrow & = & \downarrow{\scriptstyle f'=fold(f)} \\ F(A) & \xrightarrow[f]{} & A \end{array}$$

An F-coalgebra for a functor $F : \mathbb{X} \to \mathbb{X}$ is a pair $(A, g : A \to F(A))$ of an object and a map in $\mathbb{X}$. A final coalgebra for F is a pair $(F_{@}, dest : F_{@} \to F(F_{@}))$ such that for every F-coalgebra there exists a unique map $g' = unfold(g)$ which makes the following diagram commute:

$$\begin{array}{ccc} A & \xrightarrow{g} & F(A) \\ {\scriptstyle unfold(g)=g'}\downarrow & = & \downarrow{\scriptstyle F(g')} \\ F_{@} & \xrightarrow[dest]{} & F(F_{@}) \end{array}$$

We will consider an example of an initial algebra which uses coproducts to demonstrate how products and coproducts are required for inductive and coinductive datatypes. For the

functor $F(X) = X + 1$, the initial algebra is the pair of the natural numbers object and its constructor which is the successor or zero function: $(\mathbb{N}, \langle succ \mid zero \rangle)$. The diagram for this initial algebra must commute $\langle succ \mid zero \rangle \ fold(\langle f \mid a_0 \rangle) = (fold(\langle f \mid a_0 \rangle) + 1) \ \langle f \mid a_0 \rangle$ as follows:

$$\begin{array}{ccc} \mathbb{N}+1 & \xrightarrow{\langle succ|zero\rangle} & \mathbb{N} \\ {\scriptstyle fold(\langle f|a_0\rangle)+1}\downarrow & = & \downarrow{\scriptstyle f'=fold(\langle f|a_0\rangle)} \\ A+1 & \xrightarrow[f=\langle f|a_0\rangle]{} & A \end{array}$$

In the non-deterministic semantics, we have lax products and lax coproducts, and similarly, in the case of initial algebras, we have $f' \subseteq Fold(f)$ where $f = \{f_i \mid f_i \in f\}$ and $Fold(f) = \{fold(f_i) \mid f_i \in f\}$ which makes the following diagram commute:

$$\begin{array}{ccc} F(F^{@}) & \xrightarrow{cons} & F^{@} \\ {\scriptstyle F(f')}\downarrow & \supseteq & \downarrow{\scriptstyle f'\subseteq Fold(f)} \\ F(A) & \xrightarrow[f=\{f_i|f_i\in f\}]{} & A \end{array}$$

We can consider the example of lifting the deterministic functor $F(X) = X + 1$ with the initial algebra of the natural numbers object with constructor $(\mathbb{N}, \langle succ \mid zero \rangle : \mathbb{N} + 1 \to \mathbb{N})$. We construct the non-deterministic functor $\mathcal{P}(F)(X) = \mathcal{P}(X + 1)$. The initial algebra we have is deterministic, but its map can be lifted to a singleton with η, so we obtain $(\mathbb{N}, \eta(\langle succ \mid zero \rangle) = \{\langle succ \mid zero \rangle\})$. Additionally, the F-algebra maps may be non-deterministic and of the form $f = \{\langle f_i \mid a_i \rangle \mid \langle f_i \mid a_i \rangle \in f\}$. We can express this in the following diagram:

$$\begin{array}{ccc} \mathbb{N}+1 & \xrightarrow{\eta(\langle succ|zero\rangle)} & \mathbb{N} \\ {\scriptstyle f'+1}\downarrow & \supseteq & \downarrow{\scriptstyle f'\subseteq Fold(\{\langle f_i|a_i\rangle|\langle f_i|a_i\rangle\in f\})} \\ A+1 & \xrightarrow[\{\langle f_i|a_i\rangle|\langle f_i|a_i\rangle\in f\}]{} & A \end{array}$$

We can check that the diagram commutes by showing first that $\eta(\langle succ \mid zero \rangle) \ Fold(f) = (Fold(f) + 1) \ f$ is true:

$$\begin{aligned} & \eta(\langle succ \mid zero \rangle) \ Fold(f) \\ =& \{\langle succ \mid zero \rangle\} \ \{fold(\langle f_i \mid a_i \rangle) \mid \langle f_i \mid a_i \rangle \in f\} \\ =& \{\langle succ \mid zero \rangle \ fold(\langle f_i \mid a_i \rangle) \mid \langle f_i \mid a_i \rangle \in f\} \\ =& \{(fold(\langle f_i \mid a_i \rangle) + 1) \langle f_i \mid a_i \rangle \mid \langle f_i \mid a_i \rangle \in f\} \\ =& \{(fold(\langle f_i \mid a_i \rangle) + 1) \mid \langle f_i \mid a_i \rangle \in f\} \ \{\langle f_i \mid a_i \rangle \mid \langle f_i \mid a_i \rangle \in f\} \\ =& (Fold(f) + 1) \ f \end{aligned}$$

Second, we can show that $\eta(\langle succ \mid zero \rangle) \ f' \subseteq (f' + 1) \ f$ is also true.

Let $f' = \{fold(\langle f_i \mid a_i \rangle) \mid \langle f_i \mid a_i \rangle \in f''\}$ where $f'' \subseteq f$:

$$
\begin{aligned}
&\eta(\langle succ \mid zero \rangle)\ f' \\
=&\{\langle succ \mid zero \rangle\}\ \{fold(\langle f_i \mid a_i \rangle) \mid \langle f_i \mid a_i \rangle \in f''\} \\
=&\{\langle succ \mid zero \rangle\ fold(\langle f_i \mid a_i \rangle) \mid \langle f_i \mid a_i \rangle \in f''\} \\
=&\{(fold(\langle f_i \mid a_i \rangle) + 1)\langle f_i \mid a_i \rangle \mid \langle f_i \mid a_i \rangle \in f''\} \\
=&\{(fold(\langle f_i \mid a_i \rangle) + 1) \mid \langle f_i \mid a_i \rangle \in f''\}\ \{\langle f_i \mid a_i \rangle \mid \langle f_i \mid a_i \rangle \in f''\} \\
=&(f' + 1)\ f'' \\
\subseteq&(f' + 1)\ f
\end{aligned}
$$

We have shown that the diagrams commute, therefore we have an example of how a deterministic initial algebra could be lifted to be a non-deterministic initial algebra.

5 Conclusion

We have shown that the linearly distributive structure that gives the categorical semantics for arrangements of concurrent processes and communication channels in CaMPL programs is preserved after sup-lattice enrichment. These semantics guaranteed deterministic CaMPL programs have the property of progress which is the useful property of deadlock and livelock freedom. Intuitively, this means that non-deterministic CaMPL programs should also have progress. At the beginning of this project, we had planned to write a careful proof that non-deterministic CaMPL programs have progress by referencing a previous paper that showed progress for deterministic CaMPL programs [5]. We did not get to this point in this project, so this is a future area of study.

We have shown that, although products and coproducts are not preserved after sup-lattice enrichment, a non-deterministic version of lax products and coproducts exist. Using products and coproducts, we can define inductive and coinductive datatypes using initial algebras and final coalgebras. We looked at an example of the natural numbers, which are a deterministic inductive datatype, and we showed that this can be lifted to a non-deterministic inductive datatype. Inductive and coinductive datatypes are useful features of CaMPL as they can be used to represent protocols and coprotocols [9].

Future work in this area includes developing a better understanding of non-deterministic protocols and coprotocols by creating more examples of non-deterministic datatypes and examining what happens to them. We have not yet shown that all deterministic initial algebras (or final coalgebras) can become non-deterministic. Likely a generalization of what we showed above would suffice. Additionally, we can explore if it is possible to create an initial algebra with a non-deterministic constructor, and dually, a final coalgebra with a non-deterministic destructor.

I have accepted a PURE award to work with Dr. Cockett over the summer, and the research project we have planned includes creating more examples of CaMPL programs, translating them into categorical semantics, and describing these examples and explanations in a document for other computer scientists to learn about CaMPL. In this future PURE project, we will be building on our work from this project, and this will especially involve creating examples with protocols and coprotocols to answer the questions we still have.

References

1 J. R. B. Cockett and Craig Pastro. The Logic of Message Passing. *Science of Computer Programming*, 74(8):498–533, 2009.

2 J. R. B. Cockett and R. A. G. Seely. Weakly Distributive Categories. *Journal of Pure and Applied Algebra*, 114(2):133–173, 1997.

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

4 Alexanna Little. Semantics for Non-Determinism in the Categorical Message Passing Language. PURE Final Assignment: Research Findings and Synthesis, University of Calgary, 2022. Provided through correspondence with the author.

5 Reginald Lybbert. Progress for the Message Passing Logic. Undergraduate thesis, University of Calgary, 2018. Provided through correspondence with the author.

6 nLab authors. Linearly Distributive Category. `https://ncatlab.org/nlab/show/linearly+distributive+category`, December 2022. Revision 22.

7 nLab authors. Monoidal Category. `https://ncatlab.org/nlab/show/monoidal+category`, January 2023. Revision 147.

8 Jared Pon. Implementation Status of CMPL. Undergraduate thesis Interim Report, University of Calgary, 2021. Provided through correspondence with the author.

9 Masuka Yeasin. Linear Functors and their Fixed Points. Master's thesis, University of Calgary, Calgary, AB, December 2012. Provided through correspondence with the author.

A Linearly distributive category distributor diagrams

$$\begin{array}{ccc} \top \otimes (X \oplus Y) & & \\ {\scriptstyle \delta^L}\downarrow & \searrow^{u^L_\otimes} & \\ (\top \otimes X) \oplus Y & \xrightarrow[u^L_\otimes \oplus 1_Y]{} & X \oplus Y \end{array} \tag{5}$$

$$\begin{array}{ccc} (W \otimes X) \otimes (Y \oplus Z) & \xrightarrow{a_\otimes} & W \otimes (X \otimes (Y \oplus Z)) \\ & & {\scriptstyle 1_W \otimes d^L}\downarrow \\ {\scriptstyle \delta^L}\downarrow & = & W \otimes ((X \otimes Y) \oplus Z) \\ & & {\scriptstyle d^L}\downarrow \\ ((W \otimes X) \otimes Y) \oplus Z & \xrightarrow[a_\otimes \oplus 1_Z]{} & (W \otimes (X \otimes Y)) \oplus Z \end{array} \tag{6}$$

$$\begin{array}{ccccc} & & (W \oplus X) \otimes (Y \oplus Z) & & \\ & \swarrow^{\delta^L} & & \searrow^{\delta^R} & \\ ((W \oplus X) \otimes Y) \oplus Z & & = & & W \oplus (X \otimes (Y \oplus Z)) \\ {\scriptstyle \delta^R \oplus 1_Z}\downarrow & & & & {\scriptstyle 1_W \oplus \delta^L}\downarrow \\ (W \oplus (X \otimes Y)) \oplus Z & & \xrightarrow[a_\oplus]{} & & W \oplus ((X \otimes Y) \oplus Z) \end{array} \tag{7}$$