Concurrent program as proofs: compacting message passing logic

Aurore Alcolei
(directed by Robin Cockett)

October 10, 2014

Abstract

In 2009, Cockett and Pastro proposed a term logic, a proof theory and a categorical semantic for concurrency. The proof theory was essentially linear logic with the addition of message passing primitives while the categorical semantics used a linear actegory. The term logic associated to the proof theory can be viewed as a minimalist programming language for message-passing. The resulting language has the desirable property of producing programs that always terminate without deadlock. However, this property is secured at a price: the type discipline of the language does not accommodate the common semantics of concurrency. In particular, it does not allowed for multiple interconnections between processes and is a completely deterministic semantics. In this report I propose a “compact” version of this setting, where the "tensor" and the "par" operators of the linear linear logic (and, thus, of the linear actegory) are made equal. I will show how this allows for arbitrary topologies in channel connections and how this introduces non-determinism into the setting.

1 Introduction

In [3], Cockett and Pastro described a term calculus for multiplicative-additive linear logic which also might be viewed as a basic concurrent programming language for channel based communication. In [4], they further elaborated the setting, allowing messages to be passed along the channels, To do so, they provided a categorical semantic, a proof theory and a term logic, based on the notion of linear actegory. As before, the term logic could be viewed as a basic concurrent language for message passing, describing how a sequential programming semantics could interact with the linear world which manipulates communication channels.

The resulting programming language provides strong guarantees to programmers: programs always terminate without deadlock and are deterministic. Recall, a concurrent program terminates with a deadlock if the computation stops because all processes involved are waiting for each other. This should not be confused with a livelock in which processes "actively wait" by passing messages back and forth to each other resulting in a never ended computation.

In the basic setting described in [3] and [4], the cut elimination procedure in the proof logic is proved to terminate and be confluent. As a result, every program in the corresponding language must terminate and be deterministic: the cut elimination procedure corresponds to program evaluation. However, the net conditions described in [6] show that the interconnections via communication channels in this setting must always be acyclic (in a certain sense). It is this strong property that guarantees the absence of deadlock. More generally, this property also guarantees the absence of livelock when the setting is enriched with protocols, which are inductive and co-inductive data-types that enable infinite computations/communications. (cf.[11])

However, the acyclic property put severe constraints on the network topologies one can built since, for example, one cannot connect processes in a circle. From the perspective of distributed computing and of networks, this seems unrealistic, as in these settings there are definite advantages to having more than one communication channel between nodes.

A reasonable question to ask is whether there are any consequences of allowing more general topologies. We present here a system that enables more general topologies while trying to preserve the remaining logical structure. This amounts semantically to compacting the above setting by forcing the “tensor” and “par” of the linear systems to be equal. The question then is how does the whole setting behave when modified in this manner?

To answer this question, we have looked at the effect of the compactification from two different perspectives: the categorical perspective and the proof theoretical perspective. From the categorical perspective, the compacting of a linear actegory produces a “monoidal actegory”. A somewhat surprising consequence is that a monoidal actegory with products and coproducts then has biproducts, or equivalently is additively enriched. This observation means that semantically computation in such a setting becomes necessarily non-deterministic. The introduction of non-determinism in the setting, of course, takes it closer to the usual way of describing distributed computation.

From a proof theoretical perspective, the compacting of the multiplicative-additive linear logic is done so as to preserve the categorical semantic (as much as possible). This amounts to making both the multiplicatives (“tensor” and “par”) and the additives (“times” and “plus”) equal. This results in a smaller logic for a compact linearly distributive category from which the cut can still be eliminated. To complete the semantics it remains to add message passing to this compact logic to get a logic for compact linear actegory. In this fragment, cut elimination is not always possible. We show that this “failure” is induced by the possibility of creating deadlock in the corresponding programming language.

2 The original categorical setting: Linear actegories with product and coproduct

The term calculus described in [3] and [4] is based on a logic for linear actegory with product and coproduct. In this section, we briefly recall the notion of linearly distributive category and linear actegory as described in [5] and [4]: for more details we refer the reader to these articles. We also assume that the reader is familiar with the basics of category theory (monoidal categories, limits, functors, adjunctions, enrichment, ...). More information for this material can be found online [1] or in reference textbooks such as [10] and [2].

2.1 Linearly distributive categories

A *linearly distributive* category $\mathbf{H}$ is a category equipped with two tensor products $(\otimes, \top, a_\otimes, l_\otimes, r_\otimes)$, $(\oplus, \bot, a_\oplus, l_\oplus, r_\oplus)$, respectively called *tensor* and *par*, whose relative behaviour is described by two linear distributive laws:

$$d^\otimes_\oplus : A \otimes (B \oplus C) \longrightarrow (A \otimes B) \oplus C$$
$$d^\oplus_\otimes : (A \oplus B) \otimes C \longrightarrow A \oplus (B \otimes C)$$

The associative, the left and right units of each tensor products must also satisfy a list of coherence axioms that make every diagram involving monoidal structure, unit and distributivity behave properly.

A linearly distributive category is *symmetric* when its two tensors are symmetric, with symmetry $c_\otimes$, $c_\oplus$ respectively, and the two distributive laws coincide via:

$$\begin{tikzcd}
(A \oplus B) \otimes C \arrow[rrr, "d^\oplus_\otimes"] \arrow[d, "c_\otimes"] & & & A \oplus (B \otimes C) \\
C \otimes (A \oplus B) \arrow[r, "C \otimes c_\oplus"] & C \otimes (B \oplus A) \arrow[r, "d^\oplus_\otimes"] & (C \otimes B) \oplus A \arrow[r, "c_\otimes \oplus A"] & (B \otimes C) \oplus A \arrow[u, "c_\oplus"']
\end{tikzcd}$$

In that case, only one distribution law is required: $d : A \otimes (B \oplus C) \longrightarrow (A \otimes B) \oplus C$. The symmetries, $c_\oplus, c_\otimes$, must also verify some other coherence diagrams involving the monoidal ismorphisms, the unit and the linear distributions.

A linearly distributive category has *products* $(\times)$ and *coproducts* $(+)$ if the underlying category has products and coproducts such that par distributes over the product and tensor distributes over the coproduct via two natural isomorphisms:

$$d^\otimes_\times : A \oplus (B \times C) \cong A \times B \oplus A \times C$$
$$d^\oplus_+ A \otimes (B + C) :\cong A + B \otimes A + C$$

2.2 Linear actegories

A *(symmetric) linear actegory* is a (symmetric) linearly distributive category $(\mathbf{H}, \otimes, \oplus)$ with a symmetric monoidal category $(\mathbf{A}, *)$ acting on it via two actions

$$\circ : \mathbf{A} \times \mathbf{H} \longrightarrow \mathbf{H} \qquad\qquad \bullet : \mathbf{A}^{op} \times \mathbf{H} \longrightarrow \mathbf{H}$$

The actions $\circ$ and $\bullet$ are parametrized adjoints, that is $A \circ \_ \vdash A \bullet \_$ for all $A \in \mathbf{A}$. The left adjoint $\circ$ acts linearly on $(\mathbf{H}, \otimes)$ and preserves $\oplus$, while the right adjoint $\bullet$ is linear on $(\mathbf{H}, \oplus)$ and preserves $\otimes$. This translates into the following natural transformations:

$$\begin{array}{lll}
u_{\circ} : I \circ X \cong X, & u_{\bullet} : X \cong I \bullet X & \text{(unit)} \\
a^{*}_{\circ} : (A * B) \circ X \cong A \circ (B \circ X), & a^{*}_{\bullet} : A \bullet (B \circ X) \cong (A * B) \bullet X & \text{(linearity on } \mathbf{A}) \\
a^{\circ}_{\otimes} : A \circ (X \otimes Y) \cong (A \circ X) \otimes Y, & a^{\bullet}_{\oplus} : (A \bullet X) \oplus Y \cong A \bullet (X \oplus Y) & \text{(linearity on } \mathbf{H}) \\
d^{\circ}_{\oplus} : A \circ (X \oplus Y) \longrightarrow (A \circ X) \oplus Y, & d^{\bullet}_{\otimes} : (A \bullet X) \otimes Y \longrightarrow A \bullet (X \otimes Y) & \text{(to preserve the} \\
d^{\circ}_{\bullet} : A \circ (B \bullet X) \longrightarrow B \bullet (A \circ X) & & \text{linear structure)}
\end{array}$$

where u's and a's are natural isomorphisms.

Additionally, the symmetries on $*$, $\otimes$ and $\oplus$, induce the following natural transformation (or isomorphisms):

$$\begin{array}{ll}
a^{*'}_{\circ} : (A * B) \circ X \cong B \circ (A \circ X), & a^{*'}_{\bullet} : B \bullet (A \circ X) \cong (A * B) \bullet X \\
a^{\circ}_{\otimes'} : A \circ (X \otimes Y) \cong X \otimes (A \circ Y), & a^{\bullet}_{\oplus'} : X \oplus (A \bullet Y) \cong A \bullet (X \oplus Y) \\
d^{\circ}_{\oplus'} : A \circ (X \oplus Y) \longrightarrow X \oplus (A \circ Y), & d^{\bullet}_{\otimes'} : X \otimes (A \bullet Y) \longrightarrow A \bullet (X \otimes Y)
\end{array}$$

These data must satisfy several coherence axioms making units, associative laws, distributive laws and actions behave well together. Below are examples of how distributive, associative and symmetric laws must agree:

$$\begin{tikzcd}
& A \bullet (B \bullet (X \otimes Y) \arrow[dr, "A \bullet a^{\bullet}_{\otimes}"] & \\
(A * B) \bullet (X \otimes Y) \arrow[ur, "a^{*}_{\bullet}"] \arrow[dr, "a^{\bullet}_{\otimes}"'] & & A \bullet ((B \bullet X) \otimes Y) \arrow[d, "a^{\bullet}_{\otimes}"] \\
& ((A * B) \bullet X) \otimes Y \arrow[r, "a^{\bullet}_{\otimes} \otimes Y"] & (A \bullet (B \bullet X)) \otimes Y
\end{tikzcd}$$

$$\begin{tikzcd}
A \circ (X \otimes Y) \arrow[r, "A \circ c_{\otimes}"] \arrow[d, "a^{\circ}_{\otimes}"] & A \circ (Y \otimes X) \arrow[d, "a^{\circ}_{\otimes'}"] \\
(A \circ X) \otimes Y \arrow[r, "c\otimes"] & Y \otimes (A \circ X)
\end{tikzcd}$$

$$\begin{tikzcd}
A \circ (X \oplus Y) \arrow[r, "A \circ c_{\oplus}"] \arrow[d, "d^{\circ}_{\oplus}"] & A \circ (Y \oplus X) \arrow[d, "d^{\circ}_{\oplus'}"] \\
(A \circ X) \oplus Y \arrow[r, "c\oplus"] & Y \oplus (A \circ X)
\end{tikzcd}$$

$$\begin{tikzcd}
& A \circ ((X \otimes Y) \oplus Z) \arrow[dr, "d^{\circ}_{\oplus}"] & \\
A \circ (X \otimes (Y \oplus Z) \arrow[ur, "A \circ d^{\otimes}_{\oplus}"] \arrow[dr, "a^{\circ}_{\otimes}"'] & & (A \circ (X \otimes Y) \oplus Z \arrow[d, "a^{\circ}_{\otimes}"] \\
& (A \circ X) \otimes (Y \oplus Z) \arrow[r, "d^{\otimes}_{\oplus}"] & (A \circ X) \otimes Y) \oplus Z
\end{tikzcd}$$

We say that a linearly distributive category is $\boldsymbol{A}$*-additive* if the acting monoidal category $\mathbf{A}$ has coproduct over which $*$ distributes and if the action of $\circ$ preserves these coproducts while $\bullet$ turns them into products.

Lemma 1. *If H is a linearly distributive category with products then $\bullet$ preserves these products. Similarly, if H has coproducts then $\circ$ preserves these coproducts.*

Proof. This come directly from the fact that $A \circ \_$ and $A \bullet \_$ are adjoints and that right adjoints preserve limits while left adjoints preserve colimits. □

From this lemma, we can say that a linear actegory has products and coproducts if its underlying linearly distributive category has products and coproducts.

3 The compact categorical setting: Monoidal actegories with biproduct

As mentioned in the introduction, the concurrent logic developed by Cockett and Pastro is too strict to express some common features of distributed computation such as non-determinism and multiple connections between processes. The later restriction holds because a type distinction is made to differentiate when two wires of the same process are tensored together ($\otimes$ on input wires, $\oplus$ on output wires) from when two wires of distinct processes are tensored together ($\oplus$ on input wires, $\otimes$ on output wires). Thus one way to modify the setting so as to allow multiple connections is to make the tensor products, $\otimes$ and $\oplus$, equal.

The logic described in [3] generates a free linearly distributive category with product and coproduct. With the additional rules given in [4], that specify the interaction between the message world and the channel world, the whole logic is turned into a logic for linear actegory (with product and coproduct). We sketch here the effect of the compactification in the categorical setting. This is relevant in order to get an "external" point of view on what the compacted logic is going to be.

3.1 Compact linearly distributive categories

We define a compact linearly distributive category to be a linearly distributive category where tensor and par are equal, $(\otimes, \top, a_\otimes, l_\otimes, r_\otimes) = (\oplus, \bot, a_\oplus, l_\oplus, r_\oplus)$. In this context, we also identify the distributive laws, $d^\otimes_\oplus, d^\oplus_\otimes : A{\otimes}(B{\oplus}C) \longrightarrow (A{\otimes}B){\oplus}C$ to the associative law $a_\otimes = a_\oplus : A{\otimes}(B{\oplus}C) \longrightarrow (A{\otimes}B){\oplus}C$.

Proposition 1. *A compact linearly distributive category is exactly a monoidal category.*

Proof. By definition the monoidal structure of a compact linearly distributive category implies that it is a monoidal category, so there is nothing more to be done here.
Conversely, a monoidal category $(M, \otimes)$ can be regarded as a linearly distributive category $(M, \otimes, \otimes)$ where tensor and par are equal and where the distributive laws equal the associative law. The coherence diagrams for linearly distributive category holds by direct use of the coherence diagrams for monoidal category (pentagon, triangle, hexagon and symmetric laws) or by application of MacLane coherence theorem. We skip here the details of the proof that the diagrams are well-formed in the sense of Mac Lane.
From this two points we can say that a compact linearly distributive category is exactly a monoidal category. □

According to the definition of a linearly distributive category with products and coproducts, a compact linearly distributive category has products, $\times$, and coproducts, $+$, if for every object A, $A \otimes \_$ distributes over $\times$ and $\_ \otimes B$ distributes over $+$. In his paper [9], Houston shows that such a category is in fact a category with biproduct:

Proposition 2. *Let $\mathbf{C}$ be a monoidal category with finite products and coproducts and suppose that for all $A \in \mathbf{C}$, $A \otimes \_$ preserves products and $\_ \otimes A$ preserves coproducts, then $\mathbf{C}$ has finite biproducts. In particular its products/coproducts are biproducts.*

The above result shows that a compact linearly distributive category with products and coproducts is in fact a monoidal category with biproducts. This implies another interesting property for the compact setting: it is additively enriched.

Lemma 2. *Any category with biproducts is additively enriched.*

Proof. One can define the addition of two maps of same domain and codomain, $f, g : A \longrightarrow B$, to be $f + g : A \longrightarrow B := A \xrightarrow{\Delta} A * A \xrightarrow{f*g} B * B \xrightarrow{\nabla} B$. □

In the end, a compact linearly distributive category with products and coproducts is thus an additively enriched monoidal category with biproducts. According to this result, we can expect to get an additive enrichment when compacting the logic for linearly distributive categories (*i.e.* the multiplicative-additive fragment of linear logic). Semantically, having sums of processes of the same type expresses nondeterminism, and thus indicates that this is a nondeterministic setting.

3.2 Compact linear actegories

We now define a compact linear actegory to be a linear actegory over a compact linearly distributive category. Alternatively, we will call it a monoidal actegory. In this setting we identify the distributive laws, $d^\circ_\oplus$, $d^\bullet_\otimes$ with the associative laws, $a^\circ_\otimes$, $a^\bullet_\oplus$ respectively. More precisely:

Definition 1. *Let $\mathbf{A} = (\mathbf{A}, *, I, a_*, l_*, r_*, c_*)$ be a symmetric monoidal category, then a (symmetric) monoidal $\mathbf{A}$-actegory consists of the following data:*

- *A (symmetric) monoidal category $\mathbf{M} = (\mathbf{M}, \top, a_\otimes, r_\otimes, l_\otimes)$*
- *Two actions, $\circ : \mathbf{A} \times \mathbf{M} \longrightarrow \mathbf{M}$, and $\bullet : \mathbf{A}^{op} \times \mathbf{M} \longrightarrow \mathbf{M}$, such that $\circ$ is the left parametrized adjoint of $\bullet$. For all $A \in \mathbf{A}$, the adjunction $A \circ \_ \vdash A \bullet \_$ is equipped with unit, $\eta_{A,X} : X \longrightarrow A \bullet (A \circ X)$, and counit, $e_{A,X} : A \circ (A \bullet X) \longrightarrow X$, both natural in A and X.*

– *For all $A, B \in \mathbf{A}$, $X, Y \in \mathbf{M}$:*

$u_\circ : I \circ X \cong X$ $\quad$ $u_\bullet : X \cong I \bullet X$

$a^*_\circ : (A * B) \circ X \cong A \circ (B \circ X)$ $\quad$ $a^*_\bullet : A \bullet (B \bullet X) \cong (A * B) \bullet X$

$a^\circ_\otimes : A \circ (X \otimes Y) \cong (A \circ X) \otimes Y$ $\quad$ $a^\bullet_\otimes : (A \bullet X) \otimes Y \cong A \bullet (X \otimes Y)$

$d^\circ_\bullet : A \circ (B \bullet X) \longrightarrow B \bullet (A \circ X)$

which induce the following data, according to the symmetry of $*$, $\otimes$ and $\oplus$

$a^{*'}_\circ : (A * B) \circ X \cong B \circ (A \circ X)$ $\quad$ $a^{*'}_\bullet : A \bullet (B \bullet X) \cong (B * A) \bullet X$

$a^\circ_{\otimes'} : A \circ (X \otimes Y) \cong (A \circ Y) \otimes X$ $\quad$ $a^\bullet_{\otimes'} : (A \bullet X) \otimes Y \cong A \bullet (Y \otimes X)$

This data must satisfy the same coherence axioms as the one given for linear actegory, making units, associative laws, distributive laws, symmetric laws, and adjunction behave well together.

In section 3.2 we have seen that in a linearly distributive $\mathbf{A}$-actegory with products and coproducts, $A \circ \_$ preserves coproducts and $A \bullet \_$ preserves products. We now show that in the context of a monoidal $\mathbf{A}$-actegory with biproduct, $A \circ \_$ and $A \bullet \_$ both preserve biproducts and additive enrichment.

Proposition 3. *Let $\mathbf{C}$ and $\mathbf{D}$ be two categories with biproducts $*$ and let $\mathcal{F} : \mathbf{C} \longrightarrow \mathbf{D}$ be a functor that preserves products (or coproducts) then $\mathcal{F}$ preserves biproducts.*

Proof. A proof sketch is as follows:

1. We first show that $\mathcal{F}$ preserves the canonical additive enrichment of $\mathbf{C}$ described section 3.1, *i.e.* for every objects $A, B \in \mathbf{C}$ and maps $f, g \in \mathbf{C}(A, B)$, then $\mathcal{F}(0_{AB}) = 0_{\mathcal{F}(A)\mathcal{F}(B)}$ and $\mathcal{F}(f+g) = \mathcal{F}(f) + \mathcal{F}(g)$.
2. We then prove the following lemma: Let $\mathbf{C}$ be an additively enriched category with biproducts, and let $\begin{tikzcd} A_1 \arrow[r, bend left, "i_1"] & X \arrow[l, bend left, "p_1"] \arrow[r, bend right, "p_2"'] & A_2 \arrow[l, bend right, "i_2"'] \end{tikzcd}$ be such that $i_i p_i = A_i$, $i_i p_j = 0_{A_i A_j}$ and $p_1 i_1 + p_2 i_2 = id_X$ then X is a biproduct for A_1 and A_2.
3. Using the previous lemma, the proposition follows by showing that given a biproduct $\begin{tikzcd} A \arrow[r, bend left, "\sigma_0"] & A * B \arrow[l, bend left, "\pi_0"] \arrow[r, bend right, "\pi_1"'] & B \arrow[l, bend right, "\sigma_1"'] \end{tikzcd}$ in $\mathbf{C}$, then $\begin{tikzcd} \mathcal{F}(A) \arrow[r, bend left, "\mathcal{F}(\sigma_0)"] & \mathcal{F}(A * B) \arrow[l, bend left, "\mathcal{F}(\pi_0)"] \arrow[r, bend right, "\mathcal{F}(\pi_1)"'] & \mathcal{F}(B) \arrow[l, bend right, "\mathcal{F}(\sigma_1)"'] \end{tikzcd}$ is a biproduct in $\mathbf{D}$.

Remark 1. In the case of a symmetric monoidal actegory, the associative laws induce an isomorphism between $A \circ (B \bullet X)$ and $B \bullet (A \circ X)$:

$$\begin{array}{rcll} A \circ (B \bullet X) & \cong & A \circ (B \bullet (X \otimes \top) & [l_\otimes^{-1}] \\ & \cong & A \circ ((B \bullet X) \otimes \top) & [(a^\bullet_\otimes)^{-1}] \\ & \cong & (B \bullet X) \otimes (A \circ \top) & [a^\circ_\otimes] \\ & \cong & B \bullet (X \otimes (A \circ \top)) & [a^\bullet_\otimes] \\ & \cong & B \bullet (A \circ (X \otimes \top)) & [(a^\circ_\otimes)^{-1}] \\ & \cong & B \bullet (A \circ X) & [l_\otimes] \end{array}$$

Thus in the above definition, we require $d^\circ_\bullet : A \circ (B \bullet X) \longrightarrow B \bullet (A \circ X)$ to be an isomorphism. This property is counter-intuitive because in the corresponding programming language, $A \bullet X$ represents the action of getting a message of type A on a output polarity channel of type X (or putting a message of type A on an input polarity channel of type X), while $A \circ X$ represents the action of getting a message of type A on a input polarity channel of type X (or putting a message of type A on an output polarity channel of type X). Having an isomorphism between $A \circ (B \bullet X)$ and $B \bullet (A \circ X)$ for all A and B then means that the logic allows one to swap the order of a “get” and a “put”, no matter what their content is, *i.e.* it allows one to flip the read of a value x with the writing of a message depending on x! This seems undesirable but it is a consequence of the fact that concurrent programs in this setting can deadlock. We shall see the possibility that programs can deadlock corresponds to the failure of cut elimination.

Remark 2. Part of my research was dedicated to describing a way to generate a free monoidal actegory with biproduct from any monoidal category, $(M, \otimes)$. The idea was to use bag and matrix constructions to first generate a monoidal category with biproduct, $(H, \otimes)$, and then use the subcategory of objects with dual, $\mathbf{M}^D$, to act on $\mathbf{H}$ via $\otimes$. This work is not presented here since it is not directly related to the main story of this report. An unfinished version can be find online [1]. Other example of monoidal actegories are given by compact models for linear logic (with negation) or the compact logic for message passing described below!

$$\frac{}{A \vdash A}\ \text{(identity), } A \text{ atomic}$$

$$\frac{}{\vdash \top}\ (\text{unit intro}_R) \qquad \frac{}{\top \vdash}\ (\text{unit intro}_L)$$

$$\frac{\Gamma, \Gamma' \vdash \Delta}{\Gamma, \top, \Gamma' \vdash \Delta}\ (\text{unit}_L) \qquad \frac{\Gamma \vdash \Delta, \Delta'}{\Gamma \vdash \Delta, \top, \Delta'}\ (\text{unit}_R)$$

$$\frac{\Gamma, A, B, \Gamma' \vdash \Delta}{\Gamma, A \otimes B, \Gamma' \vdash \Delta}\ (\text{split}_L) \qquad \frac{\Gamma \vdash \Delta, A, B, \Delta'}{\Gamma \vdash \Delta, A \otimes B, \Delta'}\ (\text{split}_R)$$

$$\frac{\Gamma_1, A \vdash \Delta_1 \qquad B, \Gamma_2 \vdash \Delta_2}{\Gamma_1, A \otimes B, \Gamma_2 \vdash \Delta_1, \Delta_2}\ (\text{fork}_L)$$

$$\frac{\Gamma_1 \vdash \Delta_1, A \qquad \Gamma_2, \vdash B, \Delta_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, A \otimes B, \Delta_2}\ (\text{fork}_R)$$

$$\frac{\Gamma \vdash \Delta, A \qquad A, \Gamma' \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\ (\text{cut})$$

Table 1: Compacted multiplicative linear logic

4 Logic for compact linearly distributive category

In the following two sections we describe the result of compacting the term logic described in [3] and [4]. The aim is to provide a term calculus for concurrency based on a logic with a categorical semantic, namely a compact linearly distibutive category with biproducts (or monoidal category with biproduct) for the multiplicative-additive fragment, and a compact linear actegory ((or monoidal actegory) for the logic with message passing.

The original logic for message passing corresponds to a logic for channels (the multiplicative-additive fragment of linear logic) augmented with rules specifying the interactions with the message world (represented by a logic for monoidal category with coproduct). At each step, the process of compactification follows roughly the same pattern: we first look at the original logic under the compactification conditions ($\otimes = \oplus$, $+ = \times$), then, if necessary, we introduce new primitives that will enable cut to be eliminated. Finally, we derive a cut elimination procedure from the original setting to allow evaluation of concurrent programs built in the new compact logic.

In the following, capital letters such as A, B represent basic channel types while Greek capital letters such as Γ, Δ represent (non ordered) list of basic types.

4.1 Proof logic for compact multiplicative linear logic

Table 1 presents the logic rules for multiplicative linear logic as they look after compactification, *i.e.* when one uses the same operator, $\otimes$, to denote tensor and par and the same symbol, $\top$, to represent their units.

The merge between tensor and par (and their units) increases the number of possible cuts, and some of these cannot be eliminate using the original rules. In particular, the three following cuts do not have any reduction:

1. $$\frac{\frac{}{\vdash \top}\,\text{u}_L \qquad \frac{}{\top \vdash}\,\text{u}_R}{\vdash}\,\text{cut}$$

2. $$\frac{\dfrac{\dfrac{\pi}{\Gamma \vdash \Delta, A, B}}{\Gamma \vdash \Delta, A \otimes B}\,\text{s}_R \qquad \dfrac{\dfrac{\pi'}{A, B, \Gamma' \vdash \Delta'}}{A \otimes B, \Gamma' \vdash \Delta'}\,\text{s}_L}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\,\text{cut}$$

$$3.\quad \dfrac{\dfrac{\dfrac{\pi_1}{\Gamma_1 \vdash \Delta, A} \qquad \dfrac{\pi_2}{\Gamma_2 \vdash B}}{\Gamma_1, \Gamma_2 \vdash \Delta, A \otimes B}\,\mathrm{f}_R \qquad \dfrac{\dfrac{\pi'_1}{A \vdash \Delta'_1} \qquad \dfrac{\pi'_2}{B, \Gamma' \vdash \Delta'_2}}{A \otimes B, \Gamma' \vdash \Delta'_1, \Delta'_2}\,\mathrm{f}_L}{\Gamma_1, \Gamma_2, \Gamma' \vdash \Delta, \Delta'_1, \Delta'_2}\,\text{cut}$$

In order to fix this issue one needs to introduce three new rules: the empty rule, multicut and the parallel rules.

$$\dfrac{}{\vdash}\,\text{(empty)} \qquad \dfrac{\Gamma \vdash \Delta, \Lambda \qquad \Lambda, \Gamma' \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\,\text{(multicut)} \qquad \dfrac{\Gamma \vdash \Delta \qquad \Gamma' \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\,\text{(parallel)}$$

The parallel rule is in fact a special case of the multicut rule when Λ is empty. However, handling the empty multicut is sometimes subtle and, in such a case, we will refer to the parallel rule to make the distinction.

These rules allow one to reduce the previous proofs as follow:

$$1.\quad \dfrac{\dfrac{}{\vdash \top}\,\mathrm{u}_L \qquad \dfrac{}{\top \vdash}\,\mathrm{u}_R}{\vdash}\,\text{cut} \quad \Longrightarrow \quad \dfrac{}{\vdash}\,\text{emp}$$

$$2.\quad \dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash \Delta, A, B}}{\Gamma \vdash \Delta, A \otimes B}\,\mathrm{s}_R \qquad \dfrac{\dfrac{\pi'}{A, B, \Gamma' \vdash \Delta'}}{A \otimes B, \Gamma' \vdash \Delta'}\,\mathrm{s}_L}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\,\text{cut} \quad \Longrightarrow \quad \dfrac{\dfrac{\pi}{\Gamma \vdash \Delta, A, B} \qquad \dfrac{\pi'}{A, B, \Gamma' \vdash \Delta'}}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\,\text{mc}$$

3.

$$\dfrac{\dfrac{\dfrac{\pi_1}{\Gamma_1 \vdash \Delta, A} \qquad \dfrac{\pi_2}{\Gamma_2 \vdash B}}{\Gamma_1, \Gamma_2 \vdash \Delta, A \otimes B}\,\mathrm{f}_R \qquad \dfrac{\dfrac{\pi'_1}{A \vdash \Delta'_1} \qquad \dfrac{\pi'_2}{B, \Gamma' \vdash \Delta'_2}}{A \otimes B, \Gamma' \vdash \Delta'_1, \Delta'_2}\,\mathrm{f}_L}{\Gamma_1, \Gamma_2, \Gamma' \vdash \Delta, \Delta'_1, \Delta'_2}\,\text{cut}$$

$$\Longrightarrow \quad \dfrac{\dfrac{\dfrac{\pi_1}{\Gamma_1 \vdash \Delta, A} \qquad \dfrac{\pi'_1}{A \vdash \Delta'_1}}{\Gamma_1 \vdash \Delta, \Delta'_1}\,\text{cut} \qquad \dfrac{\dfrac{\pi_2}{\Gamma_2 \vdash B} \qquad \dfrac{\pi'_2}{B, \Gamma' \vdash \Delta'_2}}{\Gamma_2, \Gamma' \vdash \Delta'_2}\,\text{cut}}{\Gamma_1, \Gamma_2, \Gamma' \vdash \Delta, \Delta'_1, \Delta'_2}\,\text{para}$$

The multicut rule introduces new proofs that cannot be cut eliminate: proofs that use axiom rules in parallel. In our case, this corresponds to the use of identities in parallel:

$$\dfrac{\dfrac{}{A \vdash A}\,\text{id} \qquad \dfrac{\pi}{\Gamma \vdash \Delta}}{A, \Gamma \vdash A, \Delta}\,\text{para}$$

One thus need to introduce a generalization of the identity rule:

$$\dfrac{\Gamma, \Gamma' \vdash \Delta, \Delta'}{\Gamma, A, \Gamma' \vdash \Delta, A, \Delta'}\; A \text{ atomic (identity)}$$

where either Γ, Δ or Γ', Δ' are empty in the non commutative case. The rewrite is then:

$$\dfrac{\dfrac{}{A \vdash A}\,\text{id} \qquad \dfrac{\pi}{\Gamma \vdash \Delta}}{A, \Gamma \vdash A, \Delta}\,\text{para} \quad \Longrightarrow \quad \dfrac{\dfrac{\pi}{\Gamma \vdash \Delta}}{A, \Gamma \vdash A, \Delta}\,\text{id}$$

Augmented with the empty rule, the multicut rule and the generalization of the identity rule, the set of rules in Table 1 has dependencies.

$$\frac{\Gamma, \Gamma' \vdash \Delta, \Delta' \qquad A \text{ atomic}}{\Gamma, A, \Gamma' \vdash \Delta, A, \Delta'}\ (\text{identity}) \qquad \frac{}{\vdash}\ (\text{empty})$$

$$\frac{\Gamma, \Gamma' \vdash \Delta}{\Gamma, \top, \Gamma' \vdash \Delta}\ (\text{unit}_L) \qquad \frac{\Gamma \vdash \Delta, \Delta'}{\Gamma \vdash \Delta, \top, \Delta'}\ (\text{unit}_R)$$

$$\frac{\Gamma, A, B, \Gamma' \vdash \Delta}{\Gamma, A \otimes B, \Gamma' \vdash \Delta}\ (\text{split}_L) \qquad \frac{\Gamma \vdash \Delta, A, B, \Delta'}{\Gamma \vdash \Delta, A \otimes B, \Delta'}\ (\text{split}_R)$$

$$\frac{\Gamma \vdash \Delta, \Lambda \qquad \Lambda, \Gamma' \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\ (\text{multicut})$$

Table 2: The compact multiplicative linear

First of all, the generalization of the identity can be used to derive the identity rule:

$$\frac{}{A \vdash A}\ \text{id} \quad = \quad \frac{\vdash\ \text{emp}}{A \vdash A}\ \text{id}$$

Similarly the unit introduction rules can be obtained from the unit rules and the empty rule via:

$$\frac{}{\top \vdash}\ \text{ui}_L \quad = \quad \frac{\vdash\ \text{emp}}{\top \vdash}\ \text{u}_L \qquad \text{and} \qquad \frac{}{\vdash \top}\ \text{ui}_R \quad = \quad \frac{\vdash\ \text{emp}}{\vdash \top}\ \text{u}_R$$

Finally, the fork rules can be derived from the split rules and the parallel rule via:

$$\frac{\dfrac{\Gamma_1, A \vdash \Delta_1 \qquad B, \Gamma_2 \vdash \Delta_2}{\Gamma_1, A, B, \Gamma_2 \vdash \Delta_1, \Delta_2}\ \text{par}}{\Gamma_1, A \otimes B, \Gamma_2 \vdash \Delta_1, \Delta_2}\ \text{s}_L \qquad \text{and} \qquad \frac{\dfrac{\Gamma_1 \vdash \Delta_1, A \qquad \Gamma_2 \vdash B, \Delta_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, A, B, \Delta_2}\ \text{par}}{\Gamma_1, \Gamma_2 \vdash \Delta_1, A \otimes B, \Delta_2}\ \text{s}_R$$

In the end, the compact multiplicative linear logic can be described in seven rules, displayed in Table 2. The rules are given for a non-commutative setting. However, because we are considering having a programming language corresponding to the logic, we will allow an implicit exchange rule while constructing proofs.

4.2 Term calculus for compact multiplicative linear logic

Table 4 describes a term calculus for the compact multiplicative linear logic presented above. The annotations have been chosen so as to integrate the terms from the non compact setting into the compact setting. Thus our annotations match the the ones described in [4] (see Table 3), where applicable. In order to fix the gap between the original setting and the compact setting, due to the removal of some rules, we simply redefine the terms associated to rules that have disappeared as macros using primitives from the compact setting. These macros are obtained by following the proofs associated to each discarded rule and they are gathered in Table 5.

The annotations splitting or closing a channel, forking or ending a process suggest the programming semantic behind the logic rules. Tables [1],[2] and [3] in the appendix online ([1]) present the same tables as above but with the programming language annotations. They might help the reader to get a feel for the intended semantic of the term calculus.

4.3 Tensor rewriting rules and equations

In [3] and [4], the cut elimination procedure on $P_\alpha;_\beta Q$ is performed with respect to the leading component of each term P and Q, which corresponds to the last rule applied in the corresponding proof. This gives rise to two types of reduction rules:

$$\dfrac{}{\alpha =_A \beta :: \alpha : A \vdash \beta : A}\;\text{id} \qquad \dfrac{}{\beta[] :: \vdash \beta : \top}\;(\text{ui}_R) \qquad \dfrac{}{\alpha[] :: \alpha : \top \vdash}\;(\text{ui}_L)$$

$$\dfrac{P :: \Gamma, \Gamma' \vdash \Delta}{\alpha\langle\rangle \cdot P :: \Gamma, \alpha : \top, \Gamma' \vdash \Delta}\;\text{u}_R \qquad \dfrac{P :: \Gamma \vdash \Delta, \Delta'}{\beta\langle\rangle \cdot P :: \Gamma \vdash \Delta, \beta : \top, \Delta'}\;\text{u}_L$$

$$\dfrac{P :: \Gamma, \alpha_1 : A, \alpha_2 : B, \Gamma' \vdash \Delta}{\alpha\langle\alpha_1, \alpha_2\rangle \cdot P :: \Gamma, \alpha : A \otimes B, \Gamma' \vdash \Delta}\;\text{s}_L \qquad \dfrac{P :: \Gamma \vdash \Delta, \beta_1 : A, \beta_2 : B, \Delta'}{\beta\langle\beta_1, \beta_2\rangle \cdot P :: \Gamma \vdash \Delta, \beta : A \otimes B, \Delta'}\;\text{s}_R$$

$$\dfrac{P :: \Gamma_1, \alpha_1 : A \vdash \Delta_1 \qquad Q :: \alpha_2 B, \Gamma_2 \vdash \Delta_2}{\alpha\left[\begin{array}{l}\alpha_1 \longmapsto P\\ \alpha_2 \longmapsto Q\end{array}\right] :: \Gamma_1, \alpha : A \otimes B, \Gamma_2 \vdash \Delta_1, \Delta_2}\;\text{f}_L \qquad \dfrac{P :: \Gamma_1 \vdash \Delta_1, \beta_1 : A \qquad Q :: \Gamma_2, \vdash \beta_2 : B, \Delta_2}{\beta\left[\begin{array}{l}\beta_1 \longmapsto P\\ \beta_2 \longmapsto Q\end{array}\right] :: \Gamma_1, \Gamma_2 \vdash \Delta_1, \beta : A \otimes B, \Delta_2}\;\text{f}_R$$

$$\dfrac{P :: \Gamma \vdash \Delta, \alpha : A \qquad Q :: \beta : A, \Gamma' \vdash \Delta'}{P_{\alpha} ;_{\beta} Q :: \Gamma, \Gamma' \vdash \Delta, \Delta'}\;\text{cut}$$

Table 3: Terms for compacted multiplicative linear logic

$$\dfrac{}{\alpha =_A \beta :: \alpha : A \vdash \beta : A}\;\text{id} \qquad \dfrac{}{\varnothing :: \;\vdash}\;\text{emp}$$

$$\dfrac{P :: \Gamma, \Gamma' \vdash \Delta}{\alpha\langle\rangle \cdot P :: \Gamma, \alpha : \top, \Gamma' \vdash \Delta}\;\text{u}_R \qquad \dfrac{P :: \Gamma \vdash \Delta, \Delta'}{\beta\langle\rangle \cdot P :: \Gamma \vdash \Delta, \beta : \top, \Delta'}\;\text{u}_L$$

$$\dfrac{P :: \Gamma, \alpha_1 : A, \alpha_2 : B, \Gamma' \vdash \Delta}{\alpha\langle\alpha_1, \alpha_2\rangle \cdot P :: \Gamma, \alpha : A \otimes B, \Gamma' \vdash \Delta}\;\text{s}_L \qquad \dfrac{P :: \Gamma \vdash \Delta, \beta_1 : A, \beta_2 : B, \Delta'}{\beta\langle\beta_1, \beta_2\rangle \cdot P :: \Gamma \vdash \Delta, \beta : A \otimes B, \Delta'}\;\text{s}_R$$

$$\dfrac{P :: \Gamma \vdash \Delta \qquad Q :: \Gamma' \vdash \Delta'}{P \parallel Q :: \Gamma, \Gamma' \vdash \Delta, \Delta'}\;\text{para}$$

$$\dfrac{P :: \Gamma \vdash \Delta, [\alpha_1, \ldots, \alpha_n] : \Lambda \qquad Q :: [\beta_1, \ldots, \beta_n] : \Lambda, \Gamma' \vdash \Delta'}{P_{\alpha_1} ;_{\beta_1} \cdots {}_{\alpha_n} ;_{\beta_n} Q :: \Gamma, \Gamma' \vdash \Delta, \Delta'}\;\text{mc}$$

Table 4: Terms for compact multiplicative linear logic

$$\alpha[] := \alpha\langle\rangle \cdot \varnothing \qquad\qquad \alpha\left[\begin{array}{l}\alpha_1 \longmapsto P\\ \alpha_2 \longmapsto Q\end{array}\right] := \alpha\langle\alpha_1, \alpha_2\rangle \cdot (P \parallel Q)$$

Table 5: Term macros for compact multiplicative linear logic

– The "no interaction" rules, which apply when one of the process is not active on the channel involved in the cut, and have the sole effect of lifting up the cut in the corresponding proof. For example:

$$\dfrac{\dfrac{\dfrac{\pi}{P :: \Gamma \vdash \Delta, \gamma_1 : B_1, \gamma_2 : B_2, \alpha : A}}{\gamma\langle\gamma_1,\gamma_2\rangle \cdot P :: \Gamma \vdash \Delta, \gamma : B_1 \otimes B_2, \alpha : A}\,\mathrm{s}_R \qquad \dfrac{\pi'}{Q :: \beta : A, \Gamma' \vdash \Delta'}}{\gamma\langle\gamma_1,\gamma_2\rangle \cdot P\,{}_{\alpha}{;}_{\beta}\,Q :: \Gamma, \Gamma' \vdash \Delta, B_1 \otimes B_2, \Delta'}\,\mathrm{cut} \quad \Longrightarrow$$

$$\dfrac{\dfrac{\dfrac{\pi}{P :: \Gamma \vdash \Delta, \gamma_1 : B_1, \gamma_2 : B_2, \alpha : A} \qquad \dfrac{\pi'}{Q :: \beta : A, \Gamma' \vdash \Delta'}}{P\,{}_{\alpha}{;}_{\beta}\,Q :: \Gamma, \Gamma' \vdash \Delta, B_1, B_2, \Delta'}\,\mathrm{cut}}{\gamma\langle\gamma_1,\gamma_2\rangle \cdot (P\,{}_{\alpha}{;}_{\beta}\,Q) :: \Gamma, \Gamma' \vdash \Delta, B_1 \otimes B_2, \Delta'}\,\mathrm{s}_R$$

– The "interaction" rules, that apply when the two processes are simultaneously active on the channel of the cut, and that rearranges the internal structure of the proof. For example:

$$\dfrac{\dfrac{\dfrac{\pi}{P :: \Gamma \vdash \Delta, \alpha_1 : A_1, \alpha_2 : A_2}}{\alpha\langle\alpha_1,\alpha_2\rangle \cdot P :: \Gamma \vdash \Delta, \alpha : A_1 \otimes A_2}\,\mathrm{s}_R \qquad \dfrac{\dfrac{\pi'_1}{Q_1 :: \beta_1 : A_1, \Gamma_1 \vdash \Delta_1} \qquad \dfrac{\pi'_2}{Q_2 :: \beta_2 : A_2, \Gamma_2 \vdash \Delta_2}}{\beta\begin{bmatrix} \beta_1 \mapsto Q_1 \\ \beta_2 \mapsto Q_2 \end{bmatrix} :: \beta : A_1 \otimes A_2, \Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\,\mathrm{f}_L}{\alpha\langle\alpha_1,\alpha_2\rangle \cdot P\,{}_{\alpha}{;}_{\beta}\,\beta\begin{bmatrix} \beta_1 \mapsto Q_1 \\ \beta_2 \mapsto Q_2 \end{bmatrix} :: \Gamma, \Gamma_1, \Gamma_2 \vdash \Delta, \Delta_1, \Delta_2}\,\mathrm{cut}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{\pi}{P :: \Gamma \vdash \Delta, \alpha_1 : A_1, \alpha_2 : A_2} \qquad \dfrac{\pi'_1}{Q_1 :: \beta_1 : A_1, \Gamma_1 \vdash \Delta_1}}{P\,{}_{\alpha_1}{;}_{\beta_1}\,Q_1 :: \Gamma, \Gamma_1 \vdash \Delta, \Delta_1, \alpha_2 : A_2}\,\mathrm{cut} \qquad \dfrac{\pi'_2}{Q_2 :: \beta_2 : A_2, \Gamma_2 \vdash \Delta_2}}{(P\,{}_{\alpha_1}{;}_{\beta_1}\,Q_1)\,{}_{\alpha_2}{;}_{\beta_2}\,Q_2 :: \Gamma, \Gamma_1, \Gamma_2 \vdash \Delta, \Delta_1, \Delta_2}\,\mathrm{cut}$$

From the reduction procedure described in [3] and [4], one can derive the interaction and no-interaction rules for the compact setting as follows:
Let ${}_{\tilde\alpha}{;}_{\tilde\beta}$ be a shortcut for the multicut ${}_{\alpha_1}{;}_{\beta_1}\cdots{}_{\alpha_n}{;}_{\beta_n}$ (n possibly null), then,

– For $\gamma \notin \{\alpha_1 \ldots, \alpha_n, \beta_1, \ldots, \beta_n\}$, we have the following no-interaction rules:

[empty-seq] $\varnothing \parallel P \implies P$

[seq-empty] $P \parallel \varnothing \implies P$

[id-seq] $\gamma' =_X \gamma \cdot P\,{}_{\tilde\alpha}{;}_{\tilde\beta}\,Q \implies \gamma' =_X \gamma(P\,{}_{\tilde\alpha}{;}_{\tilde\beta}\,Q)$

[seq-id] $P\,{}_{\tilde\alpha}{;}_{\tilde\beta}\,\gamma =_X \gamma' \cdot Q \implies \gamma =_X \gamma'(P\,{}_{\tilde\alpha}{;}_{\tilde\beta}\,Q)$

[u$_L$-seq] $\gamma\langle\rangle \cdot P\,{}_{\tilde\alpha}{;}_{\tilde\beta}\,Q \implies \gamma\langle\rangle \cdot (P\,{}_{\tilde\alpha}{;}_{\tilde\beta}\,Q)$

[seq-u$_R$] $P\,{}_{\tilde\alpha}{;}_{\tilde\beta}\,\gamma\langle\rangle \cdot Q \implies \gamma\langle\rangle \cdot (P\,{}_{\tilde\alpha}{;}_{\tilde\beta}\,Q)$

[s$_L$-seq] $\gamma\langle\alpha_1,\alpha_2\rangle \cdot P\,{}_{\tilde\alpha}{;}_{\tilde\beta}\,Q \implies \gamma\langle\alpha_1,\alpha_2\rangle \cdot (P\,{}_{\tilde\alpha}{;}_{\tilde\beta}\,Q)$

[seq-s$_R$] $P\,{}_{\tilde\alpha}{;}_{\tilde\beta}\,\gamma\langle\alpha_1,\alpha_2\rangle \cdot Q \implies \gamma\langle\alpha_1,\alpha_2\rangle \cdot (P\,{}_{\tilde\alpha}{;}_{\tilde\beta}\,Q)$

– For $i \in \{1, \ldots, n\}$, we have the following interaction rules:

[id$_L$] $\gamma = \alpha_i \cdot P\,{}_{\tilde\alpha}{;}_{\tilde\beta}\,Q \implies P\,{}_{\alpha_1}{;}_{\beta_1}\cdots{}_{\alpha_{i-1}}{;}_{\beta_{i-1}}\,{}_{\alpha_{i+1}}{;}_{\beta_{i+1}}\cdots{}_{\alpha_n}{;}_{\beta_n}\,Q[\gamma/\beta_i]$

[id$_R$] $P\,{}_{\tilde\alpha}{;}_{\tilde\beta}\,\beta_i = \gamma \cdot Q \implies P[\gamma/\alpha_i]\,{}_{\alpha_1}{;}_{\beta_1}\cdots{}_{\alpha_{i-1}}{;}_{\beta_{i-1}}\,{}_{\alpha_{i+1}}{;}_{\beta_{i+1}}\cdots{}_{\alpha_n}{;}_{\beta_n}\,Q$

[u$_R$-u$_L$] $\alpha_i\langle\rangle \cdot P\,{}_{\tilde\alpha}{;}_{\tilde\beta}\,\beta_i\langle\rangle \cdot Q \implies P\,{}_{\alpha_1}{;}_{\beta_1}\cdots{}_{\alpha_{i-1}}{;}_{\beta_{i-1}}\,{}_{\alpha_{i+1}}{;}_{\beta_{i+1}}\cdots{}_{\alpha_n}{;}_{\beta_n}\,Q$

[s$_R$-s$_L$] $\alpha_i\langle\alpha_{i,1},\alpha_{i,2}\rangle \cdot P\,{}_{\tilde\alpha}{;}_{\tilde\beta}\,\beta_i\langle\beta_{i,1},\beta_{i,2}\rangle \cdot Q \implies P\,{}_{\alpha_1}{;}_{\beta_1}\cdots{}_{\alpha_{i,1}}{;}_{\beta_{i,1}}\,{}_{\alpha_{i,2}}{;}_{\beta_{i,2}}\cdots{}_{\alpha_n}{;}_{\beta_n}\,Q$

Note that in the no-interaction rules, n can be null and so, in this semantic, the evaluation of $P \parallel Q$ amounts to choose an order over the components of P and Q that preserves the order in P and Q.

The rules above follow the most natural semantic for evaluating terms in the compact setting. However, this set of reductions rules is not sufficient to cover all of the possible cases that can occur in this logic. Indeed, the multicut rule can create interleaving cuts that cannot be reduced using the rules above. Here are two examples of this situation:

1.
$$\dfrac{\dfrac{\dfrac{\varnothing :: \vdash \ \mathrm{emp}}{\alpha\langle\rangle \cdot \varnothing :: \vdash \alpha : \top}\ \mathrm{u}_R}{\beta\langle\rangle \cdot \alpha\langle\rangle \cdot \varnothing :: \vdash \alpha : \top, \beta : \top}\ \mathrm{u}_R \qquad \dfrac{\dfrac{\varnothing :: \vdash \ \mathrm{emp}}{\beta'\langle\rangle \cdot \varnothing :: \beta' : \top \vdash}\ \mathrm{u}_L}{\alpha'\langle\rangle \cdot \beta'\langle\rangle \cdot \varnothing :: \alpha' : \top, \beta' : \top \vdash}\ \mathrm{u}_L}{\beta\langle\rangle \cdot \alpha\langle\rangle \cdot \varnothing_{\alpha};_{\alpha'\ \beta};_{\beta'} \alpha'\langle\rangle \cdot \beta'\langle\rangle \cdot \varnothing :: \Gamma, \Gamma' \vdash \Delta, \Delta'}\ \mathrm{mc}$$

2.

$$\dfrac{\dfrac{\dfrac{\dfrac{\pi_1}{P :: \Gamma \vdash \Delta, \alpha_1 : A_1, \alpha_2 : A_2, \beta_1 : B_1, \beta_2 : B_2}}{\alpha\langle\alpha_1, \alpha_2\rangle \cdot P :: \Gamma \vdash \Delta, \alpha : A_1 \otimes A_2, \beta_1, \beta_2}\ \mathrm{s}_R}{\beta\langle\beta_1, \beta_2\rangle \cdot \alpha\langle\alpha_1, \alpha_2\rangle \cdot P :: \Gamma \vdash \Delta, \alpha, \beta : B_1 \otimes B_2}\ \mathrm{s}_R \qquad \dfrac{\dfrac{\dfrac{\pi_2}{Q :: \alpha'_1 : A_1, \alpha'_2 : A_2, \beta'_1 : B_1, \beta'_2 : B_2, \Gamma' \vdash \Delta'}}{\beta'\langle\beta'_1, \beta'_2\rangle \cdot Q :: \alpha'_1, \alpha'_2, \beta' : B_1 \otimes B_2, \Gamma' \vdash \Delta'}\ \mathrm{s}_L}{\alpha'\langle\alpha'_1, \alpha'_2\rangle \cdot \beta'\langle\beta'_1, \beta'_2\rangle \cdot Q :: \alpha' : A_1 \otimes A_2, \beta', \Gamma' \vdash \Delta'}\ \mathrm{s}_L}{\beta\langle\beta_1, \beta_2\rangle \cdot \alpha\langle\alpha_1, \alpha_2\rangle \cdot P\ {}_{\alpha};_{\alpha'\ \beta};_{\beta'}\ \alpha'\langle\alpha'_1, \alpha'_2\rangle \cdot \beta'\langle\beta'_1, \beta'_2\rangle \cdot Q :: \Gamma, \Gamma' \vdash \Delta, \Delta'}\ \mathrm{mc}$$

One possible way to deal with these cases would be to enable rewriting up to permutation equivalence. Indeed, the following equivalences are intrinsically present in the logic:

[id-id] $\alpha =_X \alpha' \cdot \beta =_Y \beta' \cdot P \ \mapsto\!\!\!\dashv \ \beta =_Y \beta' \cdot \alpha =_X \alpha' \cdot P$

[id-split] $\alpha =_X \alpha' \cdot \beta\langle\beta_1, \beta_2\rangle \cdot P \ \mapsto\!\!\!\dashv \ \beta\langle\beta_1, \beta_2\rangle \cdot \alpha =_X \alpha' \cdot P$

[id-unit] $\alpha =_X \alpha' \cdot \beta\langle\rangle \cdot P \ \mapsto\!\!\!\dashv \ \beta\langle\rangle \cdot \alpha =_X \alpha' \cdot P$

[split-split] $\alpha\langle\alpha_1, \alpha_2\rangle \cdot \beta\langle\beta_1, \beta_2\rangle \cdot P \ \mapsto\!\!\!\dashv \ \beta\langle\beta_1, \beta_2\rangle \cdot \alpha\langle\alpha_1, \alpha_2\rangle \cdot P$

[split-unit] $\alpha\langle\alpha_1, \alpha_2\rangle \cdot \beta\langle\rangle \cdot P \ \mapsto\!\!\!\dashv \ \beta\langle\rangle \cdot \alpha\langle\alpha_1, \alpha_2\rangle \cdot P$

[unit-unit] $\alpha\langle\rangle \cdot \beta\langle\rangle \cdot P \ \mapsto\!\!\!\dashv \ \beta\langle\rangle \cdot \alpha\langle\rangle \cdot P$

[multicut-multicut] $P\ {}_{\tilde{\alpha}_Q};_{\tilde{\beta}_Q}\ {}_{\tilde{\alpha}_R};_{\tilde{\beta}_R} \left(Q\ {}_{\tilde{\alpha}'};_{\tilde{\beta}'} R\right) \ \mapsto\!\!\!\dashv \ \left(P\ {}_{\tilde{\alpha}_Q};_{\tilde{\beta}_Q} Q\right)\ {}_{\tilde{\alpha}_R};_{\tilde{\beta}_R}\ {}_{\tilde{\alpha}'};_{\tilde{\beta}'} R$

However, this solution is not very satisfactory since in the worst case, one need to scan the whole terms of a cut at each step of the reduction. We would prefer to have an evaluation procedure that can be performed by looking only at the leading component of the terms involved in the cut.

To achieve this objective we take the following approach: since the operations on channels are independent, a process does not have to wait for the other part to be ready to complete an operation, it can just send a notification to the other process that this will read later when it is ready. To allow these partial evaluations, we introduce new notations:
A multicut, M, is now an unordered list of patterns, where a *pattern* is recursively defined by:

$$p := \alpha \mid () \mid (p, p) \qquad \text{with } \alpha \text{ a channel.}$$

The no-interaction rules as well as the [id$_L$] and [id$_R$] rules remain unchanged but we modify the other interaction rules by splitting them in two steps.

[u$_R$-1] $\alpha\langle\rangle \cdot P\ ;_{(\alpha,\beta)M}\ Q \implies P\ ;_{((),\beta)M}\ Q$

[u$_R$-2] $\alpha\langle\rangle \cdot P\ ;_{(\alpha,())M}\ Q \implies P\ ;_M\ Q$

[1-u$_L$] $P\ ;_{(\alpha,\beta)M} \beta\langle\rangle \cdot Q \implies P\ ;_{(\alpha,())M}\ Q$

[2-u$_L$] $P\ ;_{((),\beta)M} \beta\langle\rangle \cdot Q \implies P\ ;_M\ Q$

[s$_R$-1] $\alpha\langle\alpha_1, \alpha_2\rangle \cdot P\ ;_{(\alpha,\beta)M}\ Q \implies P\ ;_{((\alpha_1,\alpha_2),\beta)M}\ Q$

[s$_R$-2] $\alpha\langle\alpha_1, \alpha_2\rangle \cdot P\ ;_{(\alpha,(\beta_1,\beta_2))M}\ Q \implies P\ ;_{(\alpha_1,\beta_1)(\alpha_2,\beta_2)M}\ Q$

[1-s$_L$] $P\ ;_{(\alpha,\beta)} \beta\langle\beta_1, \beta_2\rangle \cdot Q \implies P\ ;_{((\alpha_1,\alpha_2),\beta)M}\ Q$

[2-s$_L$] $P\ ;_{((\alpha_1,\alpha_2),\beta)M} \beta\langle\beta_1, \beta_2\rangle \cdot Q \implies P\ ;_{(\alpha_1,\beta_1)(\alpha_2,\beta_2)}\ Q$

Proposition 4. *The term rewriting system given by the no-interaction rules, the [id$_R$] and [id$_L$] rules and the split interaction rules terminates.*

Proof. We show that the above cut elimination procedure terminates by defining a multiset of cut height and showing that the multiset is strictly reduced on each of the cut elimination rewrites. We define the *height* of a term as:

– hgt[] = 1

– hgt[$\alpha =_X \beta \cdot P$] = hgt[$\alpha\langle\rangle \cdot P$] = hgt[$\alpha\langle\alpha_1, \alpha_2\rangle \cdot P$] = 1 + hgt[$P$]

– hgt[$P;_M Q$] = hgt[P] + hgt[Q]

$$\frac{}{\Gamma \vdash \Delta, 0}\ \text{(terminal)} \qquad \frac{}{0, \Gamma \vdash \Delta}\ \text{(initial)}$$

$$\frac{\Gamma \vdash \Delta, A_k}{\Gamma \vdash \Delta, \Sigma_I A_i}\ \text{(injection)} \qquad \frac{A_k, \Gamma \vdash \Delta}{\Sigma_I A_i, \Gamma \vdash \Delta}\ \text{(projection)}$$

$$\frac{\{\Gamma \vdash \Delta, A_i\}_I}{\Gamma \vdash \Delta, \Sigma_I A_i}\ \text{(tuple)} \qquad \frac{\{A_i, \Gamma \vdash \Delta\}_I}{\Sigma_I A_i, \Gamma \vdash \Delta}\ \text{(cotuple)}$$

Table 6: Additive for compacted linear logic

We then define a function $\Lambda : T \longrightarrow \mathcal{M}(\mathbb{N})$ which takes a term to its multiset of cuts' height.
As a measure on $\mathcal{M}(\mathbb{N})$, we take the multiset ordering of Dershowitz and Manna [7]:
For $M, N \in \mathcal{M}(\mathbb{N})$, $M > N$ if there are multisets $X, Y \in \mathcal{M}(\mathbb{N})$ such that $\emptyset \neq X \subseteq M$, $N = (M \backslash N) \cup Y$ and $(\forall y \in Y)(\exists x \in X) x > y$.

We now show that if $t_1 \implies t_2$ then $\Lambda(t_1) > \Lambda(t_2)$. It is enough to show that if $t_1 \implies t_2$ then, $\text{hgt}[t_1] \geq \text{hgt}[t_2]$ and the height of any cut produced from the principal is strictly less than the height of the principal cut. This is true by the following observations:

- [empty-seq] $\text{hgt}[\varnothing \parallel P] = 1 + \text{hgt}[P] > \text{hgt}[P]$. Dually [seq-empty].
- [id-seq] $\text{hgt}[\gamma' =_X \gamma \cdot P\ {}_{\tilde{\alpha}};_{\tilde{\beta}}\ Q] = 1 + \text{hgt}[P] + \text{hgt}[Q] = \text{hgt}[\gamma' =_X \gamma(P\ {}_{\tilde{\alpha}};_{\tilde{\beta}}\ Q)]$ and
  $\text{hgt}[\gamma' =_X \gamma \cdot P\ {}_{\tilde{\alpha}};_{\tilde{\beta}}\ Q] = 1 + \text{hgt}[P] + \text{hgt}[Q] > \text{hgt}[(P\ {}_{\tilde{\alpha}};_{\tilde{\beta}}\ Q)] = \text{hgt}[P] + \text{hgt}[Q]$
  Similarly [seq-id], [u$_L$-seq], [seq-u$_R$], [s$_L$-seq], [seq-s$_R$].
- [id$_L$] $\text{hgt}[\gamma = \alpha_i {\cdot} P\ {}_{\tilde{\alpha}};_{\tilde{\beta}}\ Q] = 1 + \text{hgt}[P] + \text{hgt}[Q] > \text{hgt}[P_{\alpha_1};_{\beta_1} \cdots {}_{\alpha_{i-1}};_{\beta_{i-1}}\ {}_{\alpha_{i+1}};_{\beta_{i+1}} \cdots {}_{\alpha_n};_{\beta_n} Q[\gamma/\beta_i]]$
  $= \text{hgt}[P] + \text{hgt}[Q]$. Dually [id$_R$]
- [u$_R$-u$_L$] $\text{hgt}[\alpha_i\langle\rangle {\cdot} P\ {}_{\tilde{\alpha}};_{\tilde{\beta}}\ \beta_i\langle\rangle {\cdot} Q] = 1 + \text{hgt}[P] + 1 + \text{hgt}[Q] > \text{hgt}[P_{\alpha_1};_{\beta_1} \cdots {}_{\alpha_{i-1}};_{\beta_{i-1}}\ {}_{\alpha_{i+1}};_{\beta_{i+1}} \cdots {}_{\alpha_n};_{\beta_n} Q]$
  $= \text{hgt}[P] + \text{hgt}[Q]$. Dually [s$_R$-s$_L$].

Thus the rewrite system terminates. □

As defined, the rewriting system terminates. It is possible to make it confluent (although we have not proved it yet...) if we consider the results of the rewritings modulo the first six permutation equivalences mentioned above. In this case the logic can be view as generating a free monoidal (poly)category, where types are objects, terms are (poly)maps between types and where the cut corresponds to the explicit composition of two maps and its elimination giving the evaluation of the resulting map.

In this context, it is also worth noting that the rewriting rules for the compact setting match the rewriting rule of the non compact setting since the reduction of macros gives equivalent results:

1. $\alpha[]_{\alpha};_{\beta}\beta\langle\rangle \cdot P = \alpha\langle\rangle \cdot \varnothing_{\alpha};_{\beta}\beta\langle\rangle \cdot P \implies \varnothing; P \implies P$
2. $$\begin{aligned} \alpha\langle\alpha_1,\alpha_2\rangle \cdot P_{\alpha};_{\beta}\beta\left[\begin{array}{l} \beta_1 \longmapsto Q_1 \\ \beta_2 \longmapsto Q_2 \end{array}\right] &:= \alpha\langle\alpha_1,\alpha_2\rangle \cdot P_{\alpha};_{\beta}\beta\langle\beta_1,\beta_2\rangle(Q_1 \parallel Q_2) \\ &\implies P_{(\alpha_1,\alpha_2)};_{\beta}\beta\langle\beta_1,\beta_2\rangle(Q_1 \parallel Q_2) \\ &\implies P_{\alpha_1};_{\beta_1}\ {}_{\alpha_2};_{\beta_2}(Q_1 \parallel Q_2) \\ &\vdash\!\dashv (P_{\alpha_1};_{\beta_1} Q_1)\ {}_{\alpha_2};_{\beta_2} Q_2 \end{aligned}$$

4.4 The proof theory of biproduct

In Section 3.1 we have shown that a compact linearly distributive category with products and coproducts is in fact an additively enriched monoidal category with biproducts. In particular we have seen that coproducts are biproducts using the isomorphism between products and coproducts. The compactification of the rules for additives in the multiplicative-additive linear logic thus amounts to declare product and coproduct equal ($+ = \times$).

Table 6 gives the compacted version of the rules for additives. In this table, the letter I denotes a finite set of index. I can be empty for the tuple and cotuple rules, these cases respectively correspond

$$\frac{\Gamma \vdash \Delta, A_k}{\Gamma \vdash \Delta, \Sigma_I A_i}\ \text{(injection)} \qquad \frac{A_k, \Gamma \vdash \Delta}{\Sigma_I A_i, \Gamma \vdash \Delta}\ \text{(projection)}$$

$$\frac{\{\Gamma \vdash \Delta\}_I}{\Gamma \vdash \Delta}\ \text{(sum)} \qquad \frac{}{\Gamma \vdash \Delta}\ \text{(zero)}$$

Table 7: Compact logic for biproducts

to the terminal rule and initial rule. We make use of the symbol Σ to denote biproducts and we represent the initial and terminal object by 0. A biproduct, $\Sigma_{i \in I} X_i$, is characterized by its indexed set $X : I \longrightarrow \{\text{Basic Type}\}$. We will sometimes denote the biproduct of two types X and Y by $X + Y$, a shorthand for $\Sigma_{i \in \{1,2\}}(X + Y)_i$ where $(X + Y)$ is the indexed set $(X + Y) := \begin{cases} 1 \mapsto X \\ 2 \mapsto Y \end{cases}$.

Similarly to the tensor case, the compacting of product and coproduct generates modified rules and allows proofs which cannot be cut eliminated unless further rules are introduced. This for example happens for cuts between tuple and cotuple (and the special initial-terminal case) for which we introduce the sum rule and its special case, zero, when I is empty:

$$\frac{\{\Gamma \vdash \Delta\}_I}{\Gamma \vdash \Delta}\ \text{(sum)} \qquad \frac{}{\Gamma \vdash \Delta}\ \text{(zero)}$$

We get the following reductions:

1. $$\frac{\dfrac{\left\{\dfrac{\pi_i}{\Gamma \vdash \Delta_i, A_i}\right\}_I}{\Gamma \vdash \Delta, \Sigma A_i}\ \text{tup} \qquad \dfrac{\left\{\dfrac{\pi'_i}{A_i, \Gamma' \vdash \Delta'}\right\}_I}{\Sigma A_i, \Gamma' \vdash \Delta'}\ \text{cotup}}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\ \text{cut} \implies \frac{\left\{\dfrac{\dfrac{\pi_i}{\Gamma \vdash \Delta, A_i} \qquad \dfrac{\pi'_i}{A_i, \Gamma' \vdash \Delta'}}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\ \text{cut}\right\}_I}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\ \text{sum}$$

2. $$\frac{\dfrac{}{\Gamma \vdash \Delta, 0}\ \text{term} \qquad \dfrac{}{0, \Gamma \vdash \Delta}\ \text{init}}{\Gamma \vdash \Delta}\ \text{cut} \implies \frac{}{\Gamma \vdash \Delta}\ \text{zero}$$

One does not need to add any other rules to be able to eliminate cuts between projection and injection though the case is slightly more subtle:

$$\frac{\dfrac{\dfrac{\pi}{\Gamma \vdash \Delta, A_k}}{\Gamma \vdash \Delta, \Sigma A_i}\ \text{inj} \qquad \dfrac{\dfrac{\pi'}{A_l, \Gamma' \vdash \Delta'}}{\Sigma A_i, \Gamma' \vdash \Delta'}\ \text{proj}}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\ \text{cut} \implies \begin{cases} \dfrac{\dfrac{\pi}{\Gamma \vdash \Delta, A_k} \qquad \dfrac{\pi'}{A_l, \Gamma' \vdash \Delta'}}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\ \text{cut}, & \text{if } k = l \\[2ex] \dfrac{}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\ \text{zero}, & \text{if } k \neq l \end{cases}$$

Again, the introduction of the sum rule creates redundancy in the compact setting since the tuple and cotuple rules (as well as the initial and terminal rules) can be derived from the injection, projection and sum rules:

$$\frac{\{\Gamma \vdash \Delta, A_i\}_I}{\Gamma \vdash \Delta, \Sigma_I A_i}\ \text{tup} = \frac{\left\{\dfrac{\Gamma \vdash \Delta, A_i}{\Gamma \vdash \Delta, \Sigma_I A_i}\ \text{inj}\right\}_I}{\Gamma \vdash \Delta, \Sigma_I A_i}\ \text{sum} \qquad \frac{}{\Gamma \vdash \Delta, 0}\ \text{term} = \frac{}{\Gamma \vdash \Delta, 0}\ \text{zero}$$

$$\frac{\{A_i, \Gamma \vdash \Delta\}_I}{\Sigma_I A_i, \Gamma \vdash \Delta}\ \text{cotup} = \frac{\left\{\dfrac{A_i, \Gamma \vdash \Delta}{\Sigma_I A_i, \Gamma \vdash \Delta}\ \text{proj}\right\}_I}{\Sigma_I A_i, \Gamma \vdash \Delta}\ \text{sum} \qquad \frac{}{0, \Gamma \vdash \Delta}\ \text{init} = \frac{}{0, \Gamma \vdash \Delta}\ \text{zero}$$

The final rules for biproduct are summarized in Table 7 and their annotated versions are provided Table 9. Whenever possible, the annotations have been chosen so as to match the term logic in [3]

$\dfrac{P :: \Gamma \vdash \Delta, \alpha : A_k}{\alpha(k) \cdot P :: \Gamma \vdash \Delta, \alpha : \Sigma_I A_i}\ \text{inj}$	$\dfrac{P :: \alpha : A_k, \Gamma \vdash \Delta}{\alpha(k) \cdot P :: \alpha : \Pi_I A_i, \Gamma \vdash \Delta}\ \text{proj}$
$\dfrac{\{P_i :: \Gamma \vdash \Delta, \alpha : A_i\}_I}{\alpha\{i \mapsto P_i\}_I :: \Gamma \vdash \Delta, \Pi_I A_i}\ \text{tup}$	$\dfrac{\{P_i :: \alpha : A_i, \Gamma \vdash \Delta\}_I}{\alpha\{i \mapsto P_i\}_I :: \alpha : \Sigma_I A_i, \Gamma \vdash \Delta}\ \text{cotup}$
$\dfrac{}{\alpha\{\}_{\Gamma \vdash \Delta} :: \Gamma \vdash \Delta, \alpha : 1}\ \text{term}$	$\dfrac{}{\alpha\{\}_{\Gamma \vdash \Delta} :: \alpha : 0, \Gamma \vdash \Delta}\ \text{init}$

Table 8: Additives for the linear logic term calculus

$\dfrac{P :: \Gamma \vdash \Delta, \alpha : A_k}{\alpha(k) \cdot P :: \Gamma \vdash \Delta, \Sigma_I A_i}\ \text{inj}$	$\dfrac{P :: \alpha : A_k, \Gamma \vdash \Delta}{\alpha(k) \cdot P :: \Sigma_I A_i, \Gamma \vdash \Delta}\ \text{proj}$
$\dfrac{\{P_i :: \Gamma \vdash \Delta\}_{i \in I}}{\Sigma_I P_i :: \Gamma \vdash \Delta}\ \text{sum}$	$\dfrac{}{0_{\Gamma \vdash \Delta} :: \Gamma \vdash \Delta}\ \text{zero}$

Table 9: Term calculus for biproducts

and [4] - recalled Table 8. A list of macros is given in Table 10 to complete the translation from the non-compact setting to the compact setting. Semantically, projection and injection correspond to the choice of a process while sum express nondeterminism. A programming version of the previous three tables are given in the appendix online ([1]) Tables [4][5][6].

4.5 Biproduct rewriting rules and equations

Rewriting rules for biproducts are built on the same lines as the rules described for compacting multiplicative linear logic.

First are the no-interaction rules which allow one to look deeper into a process' code if the leading component of this process is not active on any channels involved in the multicut. For $\gamma \notin M$:

[inj/proj-seq] $\gamma(i) \cdot P;_M Q \implies \gamma(i) \cdot (P;_M\ )$

[seq-inj/proj] $P;_M \gamma(i) \cdot Q \implies \gamma(i) \cdot (P;_M\ )$

[sum-seq] $\Sigma_I P_i;_M Q \implies \Sigma_I(P_i;_M Q)$, in particular [zero-seq] $0_{\Gamma \vdash \Delta}\ ;_M P \implies 0_{\Gamma \vdash \Delta}$

[seq-sum] $P;_M \Sigma_I Q_i \implies \Sigma_I(P;_M Q_i)$, in particular [zero-seq] $P;_M 0_{\Gamma \vdash \Delta} \implies 0_{\Gamma \vdash \Delta}$

Then come the interaction rules which really express the programming semantic of the calculus. For $P :: \Gamma \vdash \Delta$ and $Q :: \Gamma' \vdash \Delta'$ we have:

[inj-proj] $\alpha(i) \cdot P;_{(\alpha,\beta)M} \beta(j) \cdot Q \implies P;_{(\alpha,\beta)M} Q,\quad$ when $i = j$.

[inj-proj] $\alpha(i) \cdot P;_{(\alpha,\beta)M} \beta(j) \cdot Q \implies 0_{\Gamma,\Gamma' \vdash \Delta,\Delta'},\quad$ when $i \neq j$

As previously the interaction rules need to be split in two steps if we want to be able to drive the reduction by the leading components of the terms involved in a cut. To this purpose we complete the set of patterns by:

$$p := \alpha \mid () \mid (p, p) \mid \sigma_i(p)$$

$\alpha\{i \mapsto P_i\}_{i \in I} := \Sigma_I(\alpha(i) \cdot P_i)$	$\alpha\{\}_{\Gamma \vdash \Delta} := 0_{\Gamma \vdash \Delta}$

Table 10: Term macros for biproducts

We then rewrite the interaction rules as follows:

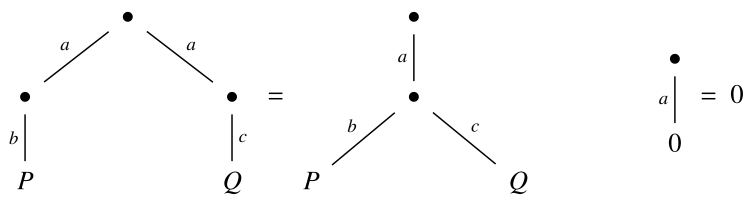
$$[\text{inj-1}]\ \alpha(i)\cdot P;_{(p(\alpha),\beta)M}\,Q \implies P;_{(p(\sigma_i(\alpha),\beta)}\,Q$$
$$[\text{proj-1}]\ P;_{(\alpha,p(\beta))M}\,\beta(i)\cdot Q \implies P;_{(\alpha,p(\sigma_i(\beta))}\,Q$$
$$[\text{inj-2}]\ \text{when } i=j,\ \alpha(j)\cdot P;_{(\alpha,\sigma_i(p))M}\,Q \implies P;_{(\alpha,p)M}\,Q$$
$$[\text{proj-2}]\ \text{when } i=j,\ P;_{(\sigma_i(p),\beta)M}\,\beta(j)\cdot Q \implies P;_{(p,\beta)M}\,Q$$
$$[\text{inj-2}]\ \text{when } i\neq j,\ \alpha(j)\cdot P;_{(\alpha,\sigma_i(p))M}\,Q \implies 0_{\Gamma,\Gamma'\vdash\Delta,\Delta'}$$
$$[\text{proj-2}]\ \text{when } i\neq j,\ P;_{(\sigma_i(p),\beta)M}\,\beta(j)\cdot Q \implies 0_{\Gamma,\Gamma'\vdash\Delta,\Delta'}$$

Because we want the compact logic with biproduct to work as a compact linearly distributive category with product and coproduct, that is a monoidal category enriched in commutative monoid, one need to add some equivalences to the setting. As previously, these equivalences are permutation equivalences that make the rewriting system confluent, but also "structural" equivalences that provides the distribution of the sum over the tensor.

For $P :: \Gamma \vdash \Delta$, channels $\alpha,\beta,\beta_1,\beta_2$ distinct, sets I, J non empty and I' a permutation of I, we have:

$$[\text{inj/proj-inj/proj}]\ \alpha(i)\cdot\beta(j)\cdot P \dashv\vdash \beta(j)\cdot\alpha(i)\cdot P$$
$$[\text{inj/proj-id}]\ \alpha(i)\cdot\beta_1 =_X \beta_2\cdot P \dashv\vdash \beta_1 =_X \beta_2\cdot\alpha(i)\cdot P$$
$$[\text{inj/proj-split}]\ \alpha(i)\cdot\beta\langle\beta_1,\beta_2\rangle\cdot P \dashv\vdash \beta\langle\beta_1,\beta_2\rangle\cdot\alpha(i)\cdot P$$
$$[\text{inj/proj-unit}]\ \alpha(i)\cdot\beta\langle\rangle\cdot P \dashv\vdash \beta\langle\rangle\cdot\alpha(i)\cdot P$$
$$[\text{sum-id}]\ \Sigma_I(\beta_1 =_X \beta_2\cdot P_i) \dashv\vdash \beta_1 =_X \beta_2\cdot\Sigma_I P_i$$
$$[\text{sum-split}]\ \Sigma_I(\beta\langle\beta_1,\beta_2\rangle\cdot P) \dashv\vdash \beta\langle\beta_1,\beta_2\rangle\cdot\Sigma_I P_i$$
$$[\text{sum-unit}]\ \Sigma_I(\beta\langle\rangle\cdot P) \dashv\vdash \beta\langle\rangle\cdot\Sigma_I P_i$$
$$[\text{sum-inj/proj}]\ \Sigma_I(\beta(j)\cdot P) \dashv\vdash \beta(j)\cdot\Sigma_I P_i$$
$$[\text{sum-sum}]\ \Sigma_I\Sigma_J P_{ij} \dashv\vdash \Sigma_{I\times J} P_{ij}$$
$$[\text{sum}]\ \Sigma_I P_i \dashv\vdash \Sigma_{I'} P_i$$
$$[\text{sum-zero}]\ P + 0_{\Gamma\vdash\Delta} \dashv\vdash P$$
$$[\text{zero}]\ a\cdot 0_{\Gamma'\vdash\Delta'} :: \Gamma\vdash\Delta \dashv\vdash 0_{\Gamma\vdash\Delta}$$

Remark 3. If the permutation equivalences for injection and projection seem natural, the equivalences involving zero and sum are more questionable with respect to the operational semantic. Indeed, the equations above suggest for example that the following diagrams are equivalent:



This semantic might not be the one wanted to describe parallel computation, as using bisimulation as the basis for equivalence these are usually judged inequivalent [8] . Nevertheless, in the context of having a categorical semantic associated to the compact logic, the above equations are necessary. For example, they enable one to solve the following critical pairs:

$$\begin{tikzcd}
 & (P_1+P_2);_M f(\gamma)\cdot Q \arrow[dl, Rightarrow] \arrow[dr, Rightarrow] & \\
P_1;_M f(\gamma)\cdot Q + P_2;_M f(\gamma)\cdot Q \arrow[d, Rightarrow] & & f(\gamma)\cdot((P_1+P_2);_M Q) \arrow[d, Rightarrow] \\
f(\gamma)\cdot(P_1;_M Q) + f(\gamma)\cdot(P_2;_M Q) & & f(\gamma)\cdot(P_1;_M Q + P_2;_M Q)
\end{tikzcd}$$

and

$$\begin{tikzcd}
 & 0_{\Gamma'\vdash\Delta'};_M f(\gamma)\cdot 0_{\Gamma\vdash\Delta} \arrow[dl, Rightarrow] \arrow[dr, Rightarrow] & \\
f(\gamma)\cdot(0_{\Gamma'\vdash\Delta'};_M 0_{\Gamma\vdash\Delta} \arrow[d, Rightarrow] & & 0_{\Gamma,\Gamma'\vdash\gamma,\Delta,\Delta'} \\
f(\gamma)\cdot 0_{\Gamma,\Gamma'\vdash\Delta,\Delta'} & &
\end{tikzcd}$$

where $f(\gamma)$ represents an action on channel $\gamma \notin M$.

$$\dfrac{\text{axiom } f}{x_1 : A_1, \ldots, x_n : A_n \vdash f(x_1, \ldots, x_n) : A}\ (\text{axiom}) \qquad \dfrac{\Phi \vdash f : A \qquad \Psi_1, x : A, \Psi_2 \vdash g : B}{\Psi_1, \Phi, \Psi_2 \vdash (x \mapsto g)f : B}\ (\text{subs})$$

$$\dfrac{\Phi, x : A, y : B \vdash f : C}{\Phi, (x, y) : A * B \vdash f : C}\ (*_l) \qquad \dfrac{\Phi \vdash f : A \qquad \Psi \vdash g : B}{\Phi, \Psi \vdash (f, g) : A * B}\ (*_r) \qquad \dfrac{\Phi \vdash f : A}{\Phi, () : I \vdash f : A}\ (I_l) \qquad \dfrac{}{\vdash () : I}\ (I_r)$$

$$\dfrac{\Phi, x : A \vdash f : C \qquad \Phi, y : B \vdash g : C}{\Phi, z : A + B \vdash \left\{ \begin{array}{c} \sigma_1(x) \mapsto f \\ \sigma_2(y) \mapsto g \end{array} \right\} z : C}\ (\text{coprod}) \qquad \dfrac{}{\Phi, z : 0 \vdash \{\}z : A}\ 0$$

$$\dfrac{\Phi \vdash f : A}{\Phi \vdash \sigma_1(f) : A + B}\ (\text{inj}_l) \qquad \dfrac{\Phi \vdash f : B}{\Phi \vdash \sigma_2(f) : A + B}\ (\text{inj}_r)$$

Table 11: Term calculus for Messages

5 Logic for compact linear actegory

In this section we propose to describe interaction between the compact logic for channels presented Section 4. and a (very simple) message world, whose logic corresponds to a logic for monoidal (multi)category with coproduct. The term logic for messages is recalled in Table 11, its monoidal structure is denoted $(*, I)$ and its coproducts are written $(+, 0)$. Interaction rules of the message world onto the compact channel world are described Table 12. In this setting sequents are of the form $\Phi \mid \Gamma \Vdash \Delta$ where Φ is a set of message variables and Γ, Δ are sets of channels. By contrast, sequents from the message world are represented using a simple turnstile: $\Phi \vdash A$. Most of the rules only described the effect of the application of rules from the message world. They were introduced to allow multicut and substitution elimination. The proper interaction rules only correspond to the $\bullet$ and $\circ$ rules that allow to "get" ($\alpha\langle x\rangle$) and "put" ($\alpha[f]$) values on a channel. A programming version of the above tables can be found in the appendix online ([1]) Tables [7] and[8]. Finally, we refer the reader to the original setting [4] for a more detailed description of these rules since we have not changed them.

From a categorical point of view, the message world and its actions are not directly affected by the compactification ; one only need to ensure that the associative and distributive isomorphisms exist and behave as desired. However, some of these isomorphisms are counter intuitive in the context of a concurrent semantic. For example this is the case for the isomorphism $A \circ B \bullet X \cong B \bullet A \circ X$ that we have was discussed in Section 3.2. This told us that the logic resulting from the compact multiplicative-additive linear logic augmented with the rules from message passing allows one to build terms where a process can send a message depending on a value it has not yet received. In particular, it is possible to write programs with deadlock in this logic as in the following example which plugs together two processes P and Q which wait for each other to provide a value to pass:

$$P \ := \ \dfrac{\dfrac{\dfrac{\dfrac{}{x : A \vdash id(x) : A}\ \text{id} \qquad \dfrac{\varnothing :: \Vdash \text{emp}}{\alpha\langle\rangle \cdot \varnothing :: \Vdash \alpha : \top}\ \text{u}_r}{\alpha[id[x] \cdot \alpha\langle\rangle \cdot \varnothing :: x : A \mid \ \Vdash \alpha : A \circ \top}\ \circ_r}{\beta\langle\rangle \cdot \alpha[id[x] \cdot \alpha\langle\rangle \cdot \varnothing :: x : A \mid \ \Vdash \alpha : A \circ \top, \beta : \top}\ \text{u}_r}{\beta\langle x\rangle \cdot \beta\langle\rangle \cdot \alpha[id(x)] \cdot \alpha\langle\rangle \cdot \varnothing :: \Vdash \alpha : A \circ \top, \beta : A \bullet \top}\ \bullet_r$$

$$Q \ := \ \dfrac{\dfrac{\dfrac{\dfrac{}{y : A \vdash id(y) : A}\ \text{id} \qquad \dfrac{\varnothing :: \Vdash \text{emp}}{\beta'\langle\rangle \cdot \varnothing :: \beta' : \top \Vdash}\ \text{u}_l}{\beta'[id[y] \cdot \beta'\langle\rangle \cdot \varnothing :: y : A \mid \beta' : A \bullet \top \Vdash}\ \bullet_l}{\alpha'\langle\rangle \cdot \beta'[id[y] \cdot \beta'\langle\rangle \cdot \varnothing :: y : A \mid \alpha' : \top, \beta' : A \bullet \top \Vdash}\ \text{u}_l}{\alpha'\langle y\rangle \cdot \alpha'\langle\rangle \cdot \beta'[id(y)] \cdot \beta'\langle\rangle \cdot \varnothing :: \ \alpha' : A \circ \top, \beta' : A \bullet \top \Vdash}\ \circ_l$$

$$P \ ;_{(\alpha,\alpha')(\beta,\beta')} Q \ := \ \beta\langle x\rangle \cdot \beta\langle\rangle \cdot \alpha[id(x)] \cdot \alpha\langle\rangle \cdot \varnothing;_{(\alpha,\alpha')(\beta,\beta')} \alpha'\langle y\rangle \cdot \alpha'\langle\rangle \cdot \beta'[id(y)] \cdot \beta'\langle\rangle \cdot \varnothing :: \vdash$$

$$\frac{\Phi \vdash f : A \qquad P :: \Psi, x : A \mid \Gamma \Vdash \Delta}{(x \mapsto P)f :: \Phi, \Psi \mid \Gamma \Vdash \Delta}\ (\text{subs})$$

$$\frac{}{\{\}z :: \Phi, z : 0 \mid \Gamma \Vdash \Delta}\ 0 \qquad \frac{P :: \Phi, x : A \mid \Gamma \Vdash \Delta \qquad Q :: \Phi, y : B \mid \Gamma \Vdash \Delta}{\left\{\begin{array}{l}\sigma_1(x) \mapsto P\\ \sigma_2(y) \mapsto Q\end{array}\right\} z :: \Phi, z : A + B \mid \Gamma \Vdash \Delta}\ (\text{cop})$$

$$\frac{P :: \Phi, x : A, y : B \mid \Gamma \Vdash \Delta}{P :: \Phi, (x, y) : A * B \mid \Gamma \Vdash \Delta}\ (*) \qquad \frac{P :: \Phi \mid \Gamma \Vdash \Delta}{P :: \Phi, () : I \mid \Gamma \Vdash \Delta}\ (I)$$

$$\frac{\Phi \vdash f : A \qquad \Psi \mid \Gamma \Vdash \alpha : X, \Delta}{\alpha[f] \cdot P :: \Phi, \Psi \mid \Gamma \Vdash A \bullet X, \Delta}\ (\bullet_r) \qquad \frac{x : A, \Phi \mid \Gamma, \alpha : X \Vdash \Delta}{\alpha\langle x\rangle \cdot P :: \Phi \mid \Gamma, A \bullet X \Vdash \Delta}\ (\bullet_l)$$

$$\frac{P :: x : A, \Phi \mid \Gamma \Vdash \alpha : X, \Delta}{\alpha\langle x\rangle \cdot P :: \Phi \mid \Gamma \Vdash A \circ X, \Delta}\ (\circ_r) \qquad \frac{\Phi \vdash f : A \qquad P :: \Psi \mid \Gamma, \alpha X \Vdash \Delta}{\alpha[f] \cdot P :: \Phi, \Psi \mid \Gamma, A \circ X \Vdash \Delta}\ (\circ_l)$$

Table 12: Term calculus for Message-passing

This example cannot be cut eliminated: thus it is temping to to equate deadlock with failure of cut eliminations. However, the question of what can be cut eliminated is delicate and we still do not have a complete understanding of the situation. Nevertheless we present below some directions that have been considered.

The more obvious option is to take as cut elimination procedure, the inefficient rewrite system given by the original reduction rules up to permutation equivalences. This system is formed by the symmetric rewriting rules and the equivalences described in Section 4, and by the following rewriting and permutation rules for message passing:

(Multicut rewriting rules): For $\gamma \notin M$

[get-seq] $\gamma\langle x\rangle \cdot P;_M Q \implies \gamma\langle x\rangle \cdot (P;_M Q)$

[seq-get] $P;_M \gamma\langle x\rangle \cdot Q \implies \gamma\langle x\rangle \cdot (P;_M Q)$

[put-seq] $\gamma[f] \cdot P;_M Q \implies \gamma[f] \cdot (P;_M Q)$

[seq-put] $P;_M \gamma[f] \cdot Q \implies \gamma[f] \cdot (P;_M Q)$

[get-put] $\alpha\langle x\rangle \cdot P;_{(\alpha,\alpha)M} \alpha[f] \cdot Q \implies (x \mapsto P)f;_{(\alpha,\alpha)M} Q$

[put-get] $\alpha[f] \cdot P;_{(\alpha,\alpha)M} \alpha\langle x\rangle \cdot Q \implies P;_{(\alpha,\alpha)M} (x \mapsto Q)f$

(Substitution Rewrite):

[subs-split] $(x \mapsto \alpha\langle\alpha_1, \alpha_2\rangle \cdot P)f \implies \alpha\langle\alpha_1, \alpha_2\rangle \cdot (x \mapsto P)f$

[subs-unit] $(x \mapsto \alpha\langle\ \rangle \cdot P)f \implies \alpha\langle\ \rangle \cdot (x \mapsto P)f$

[subs-id] $(x \mapsto \alpha =_X \beta.P)f \implies \alpha =_X \beta \cdot (x \mapsto P)f$

[subs-inj/proj] $(x \mapsto \alpha(k) \cdot P)f \implies \alpha(k) \cdot (x \mapsto P)f$

[subs-sum] $(x \mapsto \Sigma_I P_i)f \implies \Sigma_I (x \mapsto P_i)f$

[subs-get] $(x \mapsto \alpha\langle\ y\rangle \cdot P)f \implies \alpha\langle\ y\rangle \cdot (x \mapsto P)f$

[subs-put] $(x \mapsto \alpha[g] \cdot P)f \implies \alpha[g] \cdot (x \mapsto P)f \quad$ if $x \notin Dom(g)$

[subs-put] $(x \mapsto \alpha[g] \cdot P)f \implies \alpha[g[f/x]] \cdot P) \quad$ if $x \in Dom(g)$

[sub-0] $(x \mapsto z)f \implies z$

[sub-$*$] $((x, y) \mapsto P)(f_1, f_2) \implies (x \mapsto (y \mapsto P)f_2)f_1$

[sub-cop] $(x \mapsto \left\{\begin{array}{l}\sigma_1(y_1) \mapsto P_1\\ \sigma_2(y_2) \mapsto P_2\end{array}\right\} z)f \implies \left\{\begin{array}{l}\sigma_1(y_1) \mapsto (x \mapsto P_1)f\\ \sigma_2(y_2) \mapsto (x \mapsto P_2)f\end{array}\right\} z \quad$ if $x \neq z$

[sub-cop] $(z \mapsto \left\{\begin{array}{l}\sigma_1(y_1) \mapsto P_1\\ \sigma_2(y_2) \mapsto P_2\end{array}\right\} z)\sigma_i(f) \implies (y_i \mapsto P_i)f$

(Permutation equivalences):

[subs-cut] $(x \mapsto P)f;_M Q \;\vdash\!\dashv\; (x \mapsto P;_M Q)f \;\vdash\!\dashv\; P;_M (x \mapsto Q)f$

[get-split] $\alpha\langle\alpha_1, \alpha_2\rangle \cdot \beta\langle x\rangle \cdot P \;\vdash\!\dashv\; \beta\langle x\rangle \cdot \alpha\langle\alpha_1, \alpha_2\rangle \cdot P$

[get-unit] $\alpha\langle\ \rangle \cdot \beta\langle x\rangle \cdot P \;\vdash\!\dashv\; \beta\langle x\rangle \cdot \alpha\langle\ \rangle \cdot P$

[get-id] $\alpha =_X \gamma \cdot \beta\langle x\rangle \cdot P \;\vdash\!\dashv\; \beta\langle x\rangle \cdot \alpha =_X \gamma \cdot P$

[get-put] $\alpha\langle\ x\rangle \cdot \beta[f] \cdot P \;\vdash\!\dashv\; \beta[f] \cdot \alpha\langle x\rangle \cdot P$ **if** $x \notin Dom(f)$

[get-get] $\alpha\langle\ x\rangle \cdot \beta\langle y\rangle \cdot P \;\vdash\!\dashv\; \beta\langle y\rangle \cdot \alpha\langle x\rangle \cdot P$

[get-sum] $\Sigma_I(\alpha\langle x\rangle \cdot P_i) \;\vdash\!\dashv\; \alpha\langle x\rangle \cdot (\Sigma_I P_i)$

[get-coprod] $\alpha\langle x\rangle \cdot \left\{ \begin{array}{l} \sigma_1(y_1) \mapsto P_1 \\ \sigma_2(y_2) \mapsto P_2 \end{array} \right\} z \;\vdash\!\dashv\; \left\{ \begin{array}{l} \sigma_1(y_1) \mapsto \alpha\langle x\rangle \cdot P_1 \\ \sigma_2(y_2) \mapsto \alpha\langle x\rangle \cdot P_2 \end{array} \right\} z$

[get-0] $\alpha\langle x\rangle \cdot z \;\vdash\!\dashv\; z$

[put-split] $\alpha\langle\alpha_1, \alpha_2\rangle \cdot \beta[f] \cdot P \;\vdash\!\dashv\; \beta[f] \cdot \alpha\langle\alpha_1, \alpha_2\rangle \cdot P$

[put-unit] $\alpha\langle\ \rangle \cdot \beta[f] \cdot P \;\vdash\!\dashv\; \beta[f] \cdot \alpha\langle\rangle \cdot P$

[put-id] $\alpha =_X \gamma \cdot \beta[f] \cdot P \;\vdash\!\dashv\; \beta[f] \cdot \alpha =_X \gamma \cdot P$

[put-put] $\alpha[g] \cdot \beta[f] \cdot P \;\vdash\!\dashv\; \beta[f] \cdot \alpha[g] \cdot P$

[put-sum] $\Sigma_I(\alpha[f] \cdot P_i) \;\vdash\!\dashv\; \alpha[f] \cdot (\Sigma_I P_i)$

[put-cop] $\alpha[f] \cdot \left\{ \begin{array}{l} \sigma_1(y_1) \mapsto P_1 \\ \sigma_2(y_2) \mapsto P_2 \end{array} \right\} z \;\vdash\!\dashv\; \left\{ \begin{array}{l} \sigma_1(y_1) \mapsto \alpha[f] \cdot P_1 \\ \sigma_2(y_2) \mapsto \alpha[f] \cdot P_2 \end{array} \right\} z$

[put-0] $\alpha[f] \cdot z \;\vdash\!\dashv\; z$

etc...

In this system the term $P\ ;_{(\alpha,\alpha')(\beta,\beta')}\ Q$ cannot be reduced since any of its equivalent term can be reduced too:

$$\begin{aligned}
&\beta\langle x\rangle \cdot \beta\langle\rangle \cdot \alpha[id(x)] \cdot \alpha\langle\rangle \cdot \emptyset;_{(\alpha,\alpha')(\beta,\beta')} \alpha'\langle y\rangle \cdot \alpha'\langle\rangle \cdot \beta'[id(y)] \cdot \beta'\langle\rangle \cdot \emptyset \\
\vdash\!\dashv\;& \beta\langle x\rangle \cdot \alpha[id(x)] \cdot \beta\langle\rangle \cdot \alpha\langle\rangle \cdot \emptyset;_{(\alpha,\alpha')(\beta,\beta')} \alpha'\langle y\rangle \cdot \beta'[id(y)] \cdot \alpha'\langle\rangle \cdot \beta'\langle\rangle \cdot \emptyset \\
\vdash\!\dashv\;& ... \\
\vdash\!\dashv\;& \beta\langle x\rangle \cdot \alpha[id(x)] \cdot \alpha\langle\rangle\beta\langle\rangle \cdot \cdot\emptyset;_{(\alpha,\alpha')(\beta,\beta')} \alpha'\langle y\rangle \cdot \beta'[id(y)] \cdot \alpha'\langle\rangle \cdot \beta'\langle\rangle \cdot \emptyset
\end{aligned}$$

More generally, it seems that the above rewriting system allows exactly the non deadlocking terms to be cut and substitution eliminated completely. However this has not been proved.

To simplify the proofs on the above system and make it more efficient, it is tempting to develop a rewrite system by leading component without permutation as was done in Section 4. However dependencies between messages received and sent from different channels make the trick of allowing partial evaluation almost impracticable.

The last possibility that was considered was to looked at modificating the logic so as to allow circular terms. This seems to be a fruitful idea as possibly it allows cut elimination to work and for deadlock to be expressed in the circularity of the terms but we did not have time to study it carefully.

6 Conclusion

I started to work on this project because I was interested to see if it was possible to modify the formal setting described in [3] and [4] to express nondeterminism. Indeed, I wanted to see if and how this categorical and logical setting could be related to the notion of correctness for distributed algorithm on which I had worked for an other project.

Beside the nondeterminism, we also realized that the logic described by Cockett and Pastro was too restrictive in the way of connecting processes. However, we knew that this could be "easily" achieved by compacting the setting, that ia by making the tensor and the par equal. Surprisingly the study of the effect of this compactification on the categorical side revealed that this was also providing nondeterminism "for free". The goal was then to study the meaning of this nondeterminism and, more generally, of the whole process of compactification on the logic side.

I thus started to developed the corresponding logic for compact linearly distributed category. However this logic was not rich enough to express concurrent computation in the common sense and thus we decided to go one step further and add message passing the next step would have been to add protocols. Adding message passing in the compact setting was more complicated than expected and it takes time to realized that this was due to the result of the compactification on the actions which allows one to write terms with deadlock. In this context we could not expect the cut elimination to work (it would not make sense to be able to reduce a deadlocking term!). Yet, in the end, the compact logic for message passing we defined, gives a logical way to described deadlock, since a deadlock seems to correspond exactly to a failure in the cut elimination procedure.

In addition to the failure of cut elimination and its ramifications in the final system, the expressiveness of the resulting language remains to be explored. In particular, we did not have time to study the effects of the nondeterminism on the setting, nor the relationship between this system and other formalisms for concurrent programming such as the pi-calculus. This is also partly due to the fact that we first needed to develop the logic for protocols so as to be able to express infinite processes. There are, thus, many possibilities for future work!

To conclude this report, I would like to thank my supervisor, Robin Cockett, for the time and the attention he spent on this project, for his precious advice and for his invitation to the FMCS conference. A General thanks too, to the team that took time to listen to my presentations and to Poon Lueng for his lecture on enriched category.

References

[1] A. Alcolei. Notes on category theory. `http://people.ucalgary.ca/~asgalcol/category_theory`.

[2] M. Barr and C. Wells. *Category Theory for Computing Science*. International Series in Computer Science. Prentice-Hall, 1990. Second edition, 1995.

[3] J.R.B. Cockett and C.A. Pastro. A language for multiplicative-additive linear logic. *Electronic Notes in Theoretical Computer Science*, 122(0):23 – 65, 2005. Proceedings of the 10th Conference on Category Theory in Computer Science (CTCS 2004) Category Theory in Computer Science 2004.

[4] J.R.B. Cockett and C.A. Pastro. The logic of message-passing. *Science of Computer Programming*, 74(8):498 – 533, 2009. Special Issue on Mathematics of Program Construction (MPC 2006).

[5] J.R.B. Cockett and R.A.G. Seely. Weakly distributive categories. *Journal of Pure and Applied Algebra*, 114(2):133 – 173, 1997.

[6] V. Danos and L. Regnier. The structure of multiplicatives. *Archive for Mathematical Logic*, 28(3):181–203, 1989.

[7] Nachum Dershowitz and Zohar Manna. Proving termination with multiset orderings. *Commun. ACM*, 22(8):465–476, August 1979.

[8] M. Hennessy and R. Milner. On observing nondeterminism and concurrency. 85:299–309, 1980.

[9] R. Houston. Finite Products are Biproducts in a Compact Closed Category. *ArXiv Mathematics e-prints*, April 2006.

[10] S.M. Lane. *Categories for the Working Mathematician*. Graduate Texts in Mathematics. Springer New York, 1998.

[11] M. Yeasin". Linear functors and their fixed points, December 2012.