$*\mathcal{X}$: a Diagrammatic Calculus
with a Classical Fragrance

Pierre Lescanne and Dragiša Žunić

Ecole Normale Supérieure de Lyon
Université de Lyon
{Pierre.Lescanne, Dragisa.Zunic}@ens-lyon.fr

Abstract $*\mathcal{X}$ is a diagrammatic calculus. This means that it describes programs by 2-dimensional diagrams and computations are reductions of those diagrams. In addition it has a 1-dimensional syntax. Type system of $*\mathcal{X}$ interprets simply classical logic in a Curry-Howard correspondence. Since λ-calculus can be easily implemented, its untyped version is Turing complete.

1 Introduction

Several proposals have been made for diagrammatic description of programs, for instance for functional programs there exists two-dimensional syntax [Car82] and interaction nets [Laf90] to cite a few. In this paper, we propose a new paradigm for describing computations and continuations by ways of modifying diagrams, based on a Curry-Howard correspondence with classical logic. For us, a computation is specified by a description of connections among diagrams and the process of computation is itself described by modifications of those connections.

$*\mathcal{X}$ is a diagrammatic calculus, this means that programs are represented by 2-dimensional drawings of components. $*\mathcal{X}$ has one basic element to start with, plus six constructors and one operator which show how to make new components by joining together more elementary ones. Diagrams are connected by wires and computations are described by diagram reductions giving untyped $*\mathcal{X}$ the power of Turing machines. Perhaps one of the main features of $*\mathcal{X}$ is its connection with classical logic. Usually languages giving such a connection come with abstract and complex concepts like continuations, evaluation contexts, etc. With typed $*\mathcal{X}$, the so-called Curry-Howard correspondence comes naturally in a very simple setting.

The motivations behind $*\mathcal{X}$ are the following. Most of the diagrammatic languages are imperative or event driven, ours is purely functional. The functional diagrammatic languages we cited (namely [Car82] and [Laf90]) are declarative whereas ours is purely functional, subsumes lambda-calculus and has a strong semantics. Moreover most of the languages have either a loose semantics or an ad hoc one (we mean built for that language), when our *semantics* relies on the old *classical logic* [AriBC] with its recent ¡¡¡¡¡¡¡ ESOP-starX.tex developments [Gir91,CH00,UB01].

The main contribution of this paper is connection between languages that give a computational content to classical logic like ======= developments [Gir91,CH00,UB01]. ¿¿¿¿¿¿¿ 1.59

The paper is organized as follows. First we introduce the diagrammatic calculus and the reductions that are the basis of computation. Then we give its 1-dimensional syntax and its rules. After that we show how to implement λ-calculus in 1-dimensional syntax. The following section presents the type system associated with $*\mathcal{X}$, revealing a Curry-Howard correspondence with classical logic. The last section is devoted to show related works.

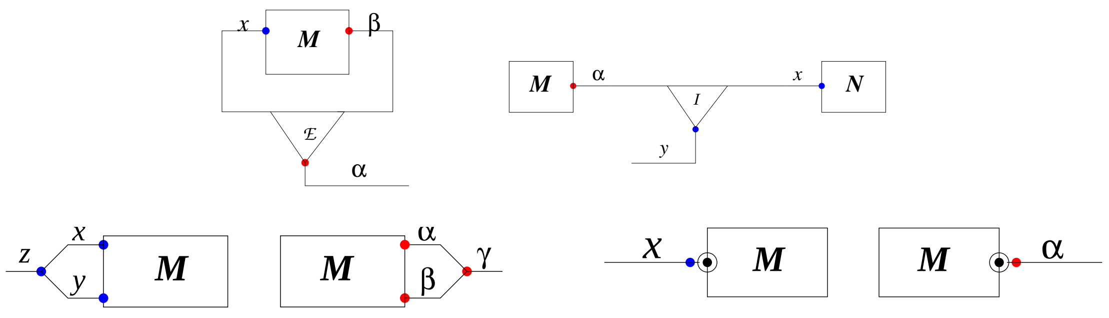


Figure 1. Basic constructors

2 Diagrammatic Programming

It is agreed that programming is a way to describe computations. In this view, $*\mathcal{X}$ can be seen as a programming language and even as a diagrammatic programming language, this means that its computations are described in a 2-dimensional syntax. More precisely, in $*\mathcal{X}$, programs are given by sets of diagrams connected by wires. Computations are reductions of those sets of diagrams. Wires are plugged on an *in-port* or on an *out-port* of a diagram[1]. By convention we decide that in-ports are located on the left of diagrams and that out-ports are located on the right. Thus the wires are oriented. In what follows, ports are named by putting an identifier above the wires[2]. We adopt a convention that in-ports names are taken in x, y, z, ... and out-ports names are taken in α, β, γ, ...

2.1 Basic Constructors

Suppose that we have well constructed diagrams with in-ports and out-ports. The $*\mathcal{X}$ diagrammatic calculus proposes six basic constructors (Figure 1) to make new diagrams from old ones.

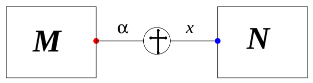
An $\mathcal{E}$-fan takes a diagram M which has an in-port, say x, and an out-port, say β, it connects them and creates an new out port, say α. We will be able to provide more meaning to this construction, and to the others as well, first when we will speak about 1-dimensional syntax (Section 3), second when we will speak about type (Section 5).

An $\mathcal{I}$-fan takes two diagrams M and N and builds a new diagram. Suppose that M has an out-port α and N has an in port x, this junction creates an in-port, say y.

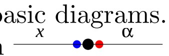
A fork takes a diagram that has two in-ports (x and y in Figure 1) and makes a diagram that has one in-port (z in Figure 1). There is a symmetric fork for out-ports.

A back-hole takes a diagram M and creates a new in-port ex nihilo. This new port, which did not exist in M, is therefore connected to a specific location denoted $\odot$, that is a black-hole where the wire ends. Also, there is a symmetric black-hole for out-ports.

Dagger as another operation The previous operators show how to combine diagrams to make new ones. There is another operation that we call *dagger*. It also connects two diagrams into one by linking an out-port α or one with the in-port x of the other. It is the only operation that creates no new port.



[1] A reader who uses a color device may see that we use a blue dot to denote an in-port and a red dot to denote a out-port.
[2] Later, we will consider that the reader is used to diagrams and we will drop names and colored dots.

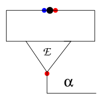
In-out as a ground diagram We have shown how to glue together diagrams, but we did not say how to build a diagram form scratch, i.e., we did not say what are the basic diagrams. There is only one ground diagram, it is called *in-out* and associated to the diagram 

It just connects an in-port (here x) to an out-port (here α).

Lemma 1 *Each diagram has at least an out-port.*

Proof. By case and induction. For $\mathcal{E}$*-fan*, *right fork*, *right black-hole* and, *in-out*, this is clear, for the others it comes by induction.

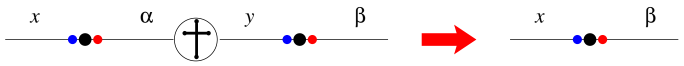
The diagrams with no in-port and only one out-port are of interest, we call them *closed diagrams*. The simplest example of such a closed diagram is represented here. Other examples are at the end of Section 6 and in Appendix B.



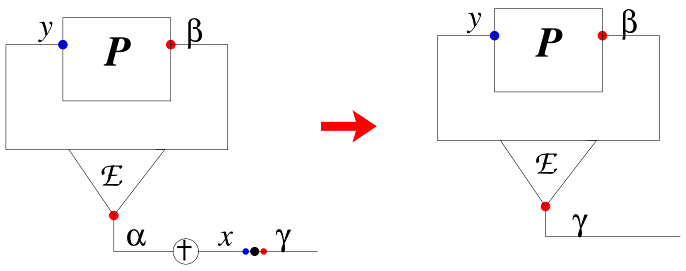
2.2 Reductions Among Diagrams

The goal of the reductions in $*\mathcal{X}$ is to eliminate the †'s (daggers). In $*\mathcal{X}$ those reductions are described by many rules. Here we are going to give only some of them. The reduction rules will be given when we will have a 1-dimensional syntax at hand (see Section 3).

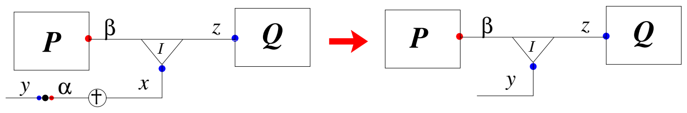
Logical Reductions The simplest reduction explains how two in-outs, connected with a dagger, are merged into one.



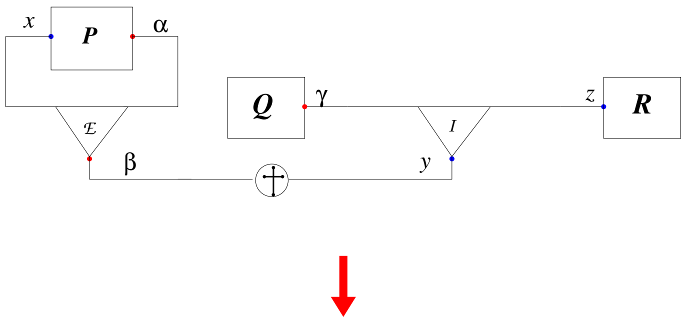
The next reduction rule specifies merging of $in-out$ with an $\mathcal{E}-fan$

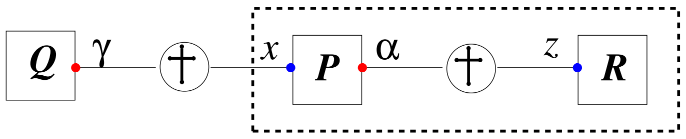


and the following specifies merging of *in-out* with an $\mathcal{I}-fan$

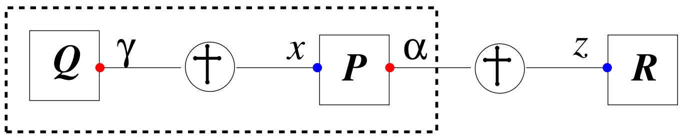


The basic computational step is given by the reduction $\mathcal{E}$-$\mathcal{I}$, which says how to reduce a diagram that connects an $\mathcal{E}$-fan with an $\mathcal{I}$-fan through a dagger.





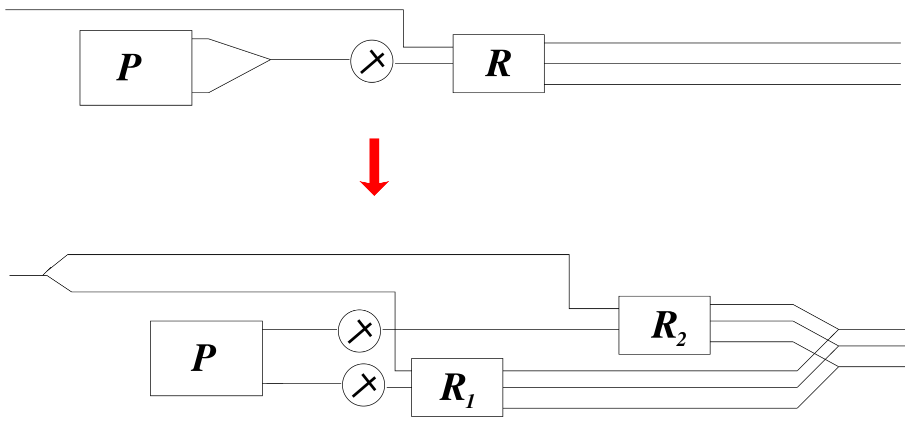
or



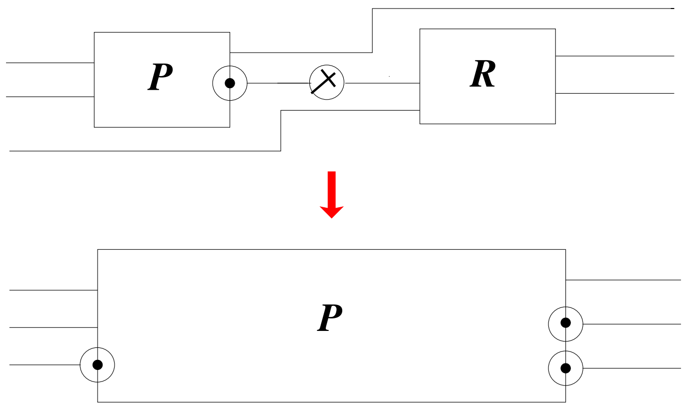
The two diagrams are the same. The dotted boxes are here to give a hint on how to view (to parse) them to get a term in the 1-dimensional syntax.

Activation For the other actions we have to chose a direction in which we perform those actions. This direction is expressed by bending the †. If the *dagger* is bended (or activated) on the left, giving ⟋, this means that the actions are performed toward the left, the other sense of activation, namely ⟍, goes the other way around, that is on the right. This feature makes $*\mathcal{X}$ *non-confluent.*

Duplication This reduction tells us what a left activated dagger (respectively a right activated dagger) does when it meets a fork. Actually it duplicates the diagrams it pulls.



Erasure Erasing says what happens when a left activated dagger of a diagram R meets a black-hole. R is erased by this action, but at the same time free names of R are meant to be preserved. Therefore the free names of R are connected to black holes.



3 A 1-dimensional Syntax

We propose a 1-dimensional syntax to describe the diagrams. (See table in Appendix A for a correspondence between concept of the 2-dimensional and the 1-dimensional syntax).

$$
\begin{array}{rll}
M, N ::= & \langle x.\alpha \rangle & \textit{capsule} \\
\mid & \widehat{x}\, M\, \widehat{\beta} \cdot \alpha & \textit{exporter} \\
\mid & M\, \widehat{\alpha}\ [y]\ \widehat{x}\, N & \textit{importer} \\
\mid & M\widehat{\alpha} \dagger \widehat{x}N & \textit{cut} \\
\mid & x \odot M & \textit{left-eraser} \\
\mid & M \odot \alpha & \textit{right-eraser} \\
\mid & z{<}^{\widehat{x}}_{\widehat{y}}\langle M] & \textit{left-duplicator} \\
\mid & [M\rangle^{\widehat{\alpha}}_{\widehat{\beta}}{>}\gamma & \textit{right-duplicator}
\end{array}
$$

Figure 2. Syntax

The coding of diagrams uses *names*[3], more precisely *innames* are taken in x, y, z, ... and *outnames* are taken in α, β, γ, ...[4]. As a matter of fact, innames (in 1-dimensional syntax) code in-ports (in 2-dimensional syntax) and outnames code out-ports.

About Bound Names The reader may have noticed the presence of hats on some names. This notation has been borrowed from *Principia Mathematica* [WR25] and is used to bind names. An interesting feature of $*\mathcal{X}$ is that a same construction, e.g., an exporter, a importer, a cut and a duplicator, binds two names. A duplicator binds either two innames or two outnames, whereas the other operators bind an inname and an outname.

Moreover the names that are bound can belong to the same term, or to two different terms. For example, *importer* and *cut*, which are dyadic operations, bind an outname in a term on the left and an inname in a term on the right.

3.1 Linearity

In $*\mathcal{X}$ we consider only *linear* terms[5]. This means that names occur only once in the terms. This is required to represent faithfully diagrams and wires. The predicate *linear* is formally described in Figure 3 and uses the concept of free name described in Figure 4. Duplicators and erasers allow to mimic non linear terms.

We remark that the syntax generates terms which, in general, are not linear.

The Bestiary In this section we describe the constituents of $*\mathcal{X}$.

$\langle x.\alpha \rangle$ is called a *capsule* and codes an *in-out*. It creates one free inname x and one free outname α.

$(\widehat{x}\, M\, \widehat{\beta} \cdot \alpha)$ is an *exporter* and codes an $\mathcal{E}$-fan. It binds two names, an inname x and an outname β of the subterm M and creates a new free outname. Because of linearity, the new outname α does not occur free in M.

$(M\, \widehat{\beta}\ [y]\ \widehat{x}\, N)$ is called a *importer* and codes an $\mathcal{I}$-fan. It is made of two subterms M and N and binds an outname β of the first term, namely M, and an inname x of the second, namely N. It creates an inname.

[3] Notice the difference between name and variable. Names that we use here cannot be replaced by terms. They can only be *renamed*.

[4] We assume that we start with non-indexed names (see Section 3.3).

[5] In that it departs from $\mathcal{X}$ described in [vBLL05].

$$\frac{}{\langle x.\alpha\rangle \ \ linear}$$

$$\frac{M \ \ linear, \ \ x \in f_{in}(M), \beta \in f_{on}(M), \ \ \alpha \notin f_{on}(M)}{\widehat{x}\, M\, \widehat{\beta} \cdot \alpha \ \ linear}$$

$$\frac{M, N \ \ linear, \ \ \alpha \in f_{on}(M), \ \ x \in f_{in}(N), \ \ y \notin f_{in}(M, N), \ \ f_n(M) \cap f_n(N) = \emptyset}{M\, \widehat{\alpha}\ [y]\ \widehat{x}\, N \ \ linear}$$

$$\frac{M, N \ \ linear, \ \ \alpha \in f_{on}(M), \ \ x \in f_{in}(N), \ \ f_n(M) \cap f_n(N) = \emptyset}{M\widehat{\alpha} \dagger \widehat{x}N \ \ linear}$$

$$\frac{M \ \ linear, \ \ x \notin f_{in}(M)}{x \odot M \ \ linear} \qquad \frac{M \ \ linear, \ \ \alpha \notin f_{on}(M)}{M \odot \alpha \ \ linear}$$

$$\frac{M \ \ linear, \ \ x, y \in f_{in}(M), \ \ z \notin f_{in}(M)}{z{<}^{\widehat{x}}_{\widehat{y}}\langle M] \ \ linear} \; \frac{M \ \ linear, \ \ \alpha, \beta \in f_{on}(M), \ \ \gamma \notin f_{on}(M)}{[M\rangle^{\widehat{\alpha}}_{\widehat{\beta}}{>}\gamma \ \ linear}$$

Figure 3. Linear Terms

The $(M\widehat{\alpha} \dagger \widehat{x}M)$ is called a *cut* and codes the dagger operation on diagrams. It should be noticed that it is the only operation that creates no free name.

$x \odot M$ is called a *left-eraser* and $M \odot \alpha$ is called a *right-eraser*. The first one adds explicitly the inname x (which does not occur in M) as a new free inname in the expression $x \odot M$, whereas $M \odot \alpha$ adds the free outname α to the term M.

$z{<}^{\widehat{x}}_{\widehat{y}}\langle M]$ is called a *left-duplicator* and $[M\rangle^{\widehat{\alpha}}_{\widehat{\beta}}{>}\gamma$ is called a *right-duplicator*. They code *forks*. As terms are linear, this is a way to simulate multiple occurrences of names in a term, more precisely $z{<}^{\widehat{x}}_{\widehat{y}}\langle M]$ allows z to behave like it would occur twice, although M is linear. One can also see z as being duplicated into x and y.

Lemma 2 *Each term has at least one (free) outname.*

Some Abbreviations To ease the reading, abbreviations are welcome.

instead of	we write
$x_1 \odot (... \odot (x_n \odot M)...)$	$x_1 \odot ... \odot x_n \odot M$
$(...(M \odot \alpha_1) \odot ...) \odot \alpha_n$	$M \odot \alpha_1 \odot ... \odot \alpha_n$
$x_1{<}^{\widehat{y_1}}_{\widehat{z_1}}\langle ... x_n{<}^{\widehat{y_n}}_{\widehat{z_n}}\langle M]...]$	$(x_1, ..., \ x_n){<}^{(\widehat{y_1}, ..., \widehat{y_n})}_{(\widehat{z_1}, ..., \widehat{z_n})}\langle M]$
$[...[M\rangle^{\widehat{\beta_1}}_{\widehat{\gamma_1}}{>}\alpha_1 ...\rangle^{\widehat{\beta_n}}_{\widehat{\gamma_n}}{>}\alpha_n$	$[M\rangle^{(\widehat{\beta_1}, ..., \widehat{\beta_n})}_{(\widehat{\gamma_1}, ..., \widehat{\gamma_n})}{>}(\alpha_1, ..., \alpha_n)$

And more generally, when Φ_{in}, $\Phi_{in,1}$ and $\Phi_{in,2}$ are lists of innames and Φ_{on}, $\Phi_{on,1}$ and $\Phi_{on,2}$ are lists of outnames we write:

$$\Phi_{in}{<}^{\widehat{\Phi_{in,1}}}_{\widehat{\Phi_{in,2}}}\langle M] \ \text{ and } \ [M\rangle^{\widehat{\Phi_{on,1}}}_{\widehat{\Phi_{on,2}}}{>}\Phi_{on}$$

with: $\quad \Phi_{in}{<}^{\widehat{\Phi_{in,1}}}_{\widehat{\Phi_{in,2}}}\langle M] = M$, when $\Phi_{in} = ()$ and $[M\rangle^{\widehat{\Phi_{on,1}}}_{\widehat{\Phi_{on,2}}}{>}\Phi_{on} = M$ when $\Phi_{on} = ()$.

3.2 Congruences

Sometimes more that on term correspond to the same diagram and congruences between terms come naturally like

$$\boxed{\begin{array}{c} x{<}\genfrac{}{}{0pt}{}{\widehat{x_1}}{\widehat{x_2}}\langle M] \equiv x{<}\genfrac{}{}{0pt}{}{\widehat{x_2}}{\widehat{x_1}}\langle M] \\ [M\rangle\genfrac{}{}{0pt}{}{\widehat{\alpha_1}}{\widehat{\alpha_2}}{>}\alpha \equiv [M\rangle\genfrac{}{}{0pt}{}{\widehat{\alpha_2}}{\widehat{\alpha_1}}{>}\alpha \end{array}}$$

which shows that names can be freely commuted in a duplicator, or

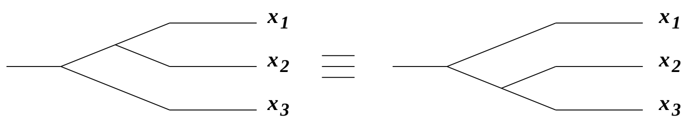
$$\boxed{\begin{array}{c} x{<}\genfrac{}{}{0pt}{}{\widehat{x_1}}{\widehat{x_2}}\langle y{<}\genfrac{}{}{0pt}{}{\widehat{y_1}}{\widehat{y_2}}\langle M]] \equiv y{<}\genfrac{}{}{0pt}{}{\widehat{y_1}}{\widehat{y_2}}\langle x{<}\genfrac{}{}{0pt}{}{\widehat{x_1}}{\widehat{x_2}}\langle M]] \\ [[M\rangle\genfrac{}{}{0pt}{}{\widehat{\alpha_1}}{\widehat{\alpha_2}}{>}\alpha\rangle\genfrac{}{}{0pt}{}{\widehat{\beta_1}}{\widehat{\beta_2}}{>}\beta \equiv [[M\rangle\genfrac{}{}{0pt}{}{\widehat{\beta_1}}{\widehat{\beta_2}}{>}\beta\rangle\genfrac{}{}{0pt}{}{\widehat{\alpha_1}}{\widehat{\alpha_2}}{>}\alpha \\ [x{<}\genfrac{}{}{0pt}{}{\widehat{x_1}}{\widehat{x_2}}\langle M]\rangle\genfrac{}{}{0pt}{}{\widehat{\alpha_1}}{\widehat{\alpha_2}}{>}\alpha \equiv x{<}\genfrac{}{}{0pt}{}{\widehat{x_1}}{\widehat{x_2}}\langle [M\rangle\genfrac{}{}{0pt}{}{\widehat{\alpha_1}}{\widehat{\alpha_2}}{>}\alpha] \end{array}}$$

where in the first congruence we have $y \notin \{x_1, x_2\}$ and $x \notin \{y_1, y_2\}$ and in the second $\beta \notin \{\alpha_1, \alpha_2\}$ and $\alpha \notin \{\beta_1, \beta_2\}$. Those congruences say that provided they are non related, duplicators can be freely commuted. The last congruence allows us to use a simplified notation $x < \genfrac{}{}{0pt}{}{\widehat{x_1}}{\widehat{x_2}}\langle M\rangle\genfrac{}{}{0pt}{}{\widehat{\alpha_1}}{\widehat{\alpha_2}} > \alpha$ and more generally $\Phi_{in} < \genfrac{}{}{0pt}{}{\widehat{\Phi_{in,1}}}{\widehat{\Phi_{in,2}}}\langle M\rangle\genfrac{}{}{0pt}{}{\widehat{\Phi_{on,1}}}{\widehat{\Phi_{on,2}}} > \Phi_{on}$. In the simpler case when $\Phi_{in} = ()$, we will write $[M\rangle\genfrac{}{}{0pt}{}{\widehat{\Phi_{on,1}}}{\widehat{\Phi_{on,2}}}{>}\Phi_{on}$, and analogously for $\Phi_{on} = ()$ we write only $\Phi_{in}{<}\genfrac{}{}{0pt}{}{\widehat{\Phi_{in,1}}}{\widehat{\Phi_{in,2}}}\langle M]$.

The wires can be braided as said by the following congruences:

$$\boxed{\begin{array}{c} z{<}\genfrac{}{}{0pt}{}{\widehat{y}}{\widehat{x_3}}\langle y{<}\genfrac{}{}{0pt}{}{\widehat{x_1}}{\widehat{x_2}}\langle M]] \equiv z{<}\genfrac{}{}{0pt}{}{\widehat{x_1}}{\widehat{y}}\langle y{<}\genfrac{}{}{0pt}{}{\widehat{x_2}}{\widehat{x_3}}\langle M]] \\ [[M\rangle\genfrac{}{}{0pt}{}{\widehat{\alpha_1}}{\widehat{\alpha_2}}{>}\beta\rangle\genfrac{}{}{0pt}{}{\widehat{\beta}}{\widehat{\alpha_3}}{>}\gamma \equiv [[M\rangle\genfrac{}{}{0pt}{}{\widehat{\alpha_2}}{\widehat{\alpha_3}}{>}\beta\rangle\genfrac{}{}{0pt}{}{\widehat{\alpha_1}}{\widehat{\beta}}{>}\gamma \end{array}}$$

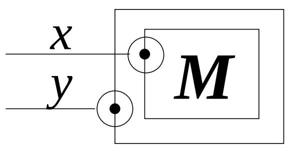
illustrated by the following diagram:

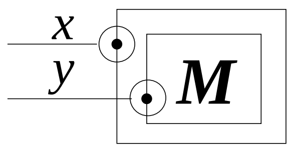


We also have congruences for erasers:

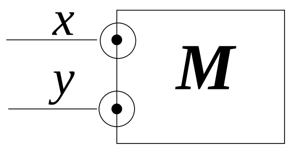
$$\boxed{\begin{array}{c} y \odot x \odot M \equiv x \odot y \odot M \\ M \odot \alpha \odot \beta \equiv M \odot \beta \odot \alpha \\ (x \odot M) \odot \alpha \equiv x \odot (M \odot \alpha) \end{array}}$$

The last congruence suggests the notation $x \odot M \odot \alpha$, and more generally $\Phi_{in} \odot M \odot \Phi_{on}$. The terms $y \odot x \odot M$ and $x \odot y \odot M$ correspond respectively to the following two diagrams:





The congruence $x \odot y \odot M \equiv y \odot x \odot M$ suggests that, instead of previous two, we can use the simpler diagram:



There are other congruences to be considered for the 1-dimensional syntax in order to reflect diagrams, but due to lack of space we are not going to see them here.

3.3 Free and Bound Names

Definition 1 (Free Names) *The set of* free names *, i.e.* free innames *and* free outnames *are defined in Figure 4.*

Q	$f_{in}(Q)$	$f_{on}(Q)$
$\langle x.\alpha\rangle$	x	α
$\widehat{x}\, M\, \widehat{\beta}\cdot\alpha$	$f_{in}(M)\backslash\{x\}$	$(f_{on}(M)\backslash\{\beta\})\cup\{\alpha\}$
$M\,\widehat{\alpha}\,[x]\,\widehat{y}\,N$	$f_{in}(M)\cup(f_{in}(N)\backslash\{y\})\cup\{x\}$	$(f_{on}(M)\backslash\{\alpha\})\cup f_{on}(N)$
$M\widehat{\alpha}\dagger\widehat{x}N$	$f_{in}(M)\cup(f_{in}(N)\backslash\{x\})$	$(f_{on}(M)\backslash\{\alpha\})\cup f_{on}(N)$
$x\odot M$	$f_{in}(M)\cup\{x\}$	$f_{on}(M)$
$M\odot\alpha$	$f_{in}(M)$	$f_{on}(M)\cup\{\alpha\}$
$x{<}^{\widehat{x_1}}_{\widehat{x_2}}\langle M]$	$(f_{in}(M)\backslash\{x_1,\ x_2\})\cup\{x\}$	$f_{on}(M)$
$[M\rangle^{\widehat{\alpha_1}}_{\widehat{\alpha_2}}{>}\alpha$	$f_{in}(M)$	$(f_{on}(M)\backslash\{\alpha_1,\ \alpha_2\})\cup\{\alpha\}$

Figure 4. Free and Bound Names

We write $f_n(M)$ the set *of free names of M, $f_{in}(M)$ the* sets *of free innames of M and, $f_{on}(M)$ the set of free outnames of M. Thus we have $f_n(M) = f_{in}(M)\cup f_{on}(M)$.*

If we wish to think of sets $f_n(M)$, $f_{in}(M)$ and $f_{on}(M)$ as lists, we will use the notation $\Phi_n(M)$ (or simply Φ), $\Phi_{in}(M)$ and $\Phi_{on}(M)$, respectively. Simpler way would be to write Φ_n^M, Φ_{in}^M and Φ_{on}^M. If we wish to exclude the name, say outname z, from the list, we write $\Phi_{on}^M\backslash z$

We adopt Barendregt's convention which says that free and bound names are always distinct.

Remark 1 (Convention on names). A name is never both, bound and free, in the same term.

Remark 2 (Indexing). In what follows, free names will be renamed by indexing. If i is an index (a natural), M a term and Φ a set of free names occurring in M, $ind(M,\Phi,i)$ means, for example, that a name x belonging to Φ is replaced in M by x_i and a name α belonging to Φ is replaced in M by α_i. To avoid losing linearity, we assume that we start with non-indexed names. As we use it, indexing preserves linearity.

When Φ_{in} is a list of innames, we write $\Phi_{in,i}$ for the same list of names indexed by i and similarly with Φ_{on} for outnames.

We assume that terms are defined up to α-conversion, that is that the renaming of bound innames or outnames does not change them.

4 Reduction Rules

In this chapter we define a reduction relation, denoted by $\xrightarrow{*\mathcal{X}}$. The rules are divided into four categories. The first ones are called *activation* and have an effect on the direction in which further rules will be performed. The second ones are called *logical* and deal with cuts which can not be activated; they perform changes on terms. The third ones are called *actions* and they also perform changes on the terms. The fourth ones are called *propagations* as they only propagate active cuts through terms. Actions and propagations are themselves split into two symmetric sub-categories "left" and "right".

Activation Rules Cuts will be eliminated in terms. Actually cut-elimination is the basis of computation in $^*\mathcal{X}$ when seen as a programming language. In the cut elimination procedure there are situations where one has to *choose* to which side the cuts will be propagated to, in order to be eliminated. It appears that this choice is the source of non-confluence in $^*\mathcal{X}$.

The direction of activation is denoted by bending the cut, like it was done with daggers in 2-dimensional syntax. A bended cut is called an *active cut*. The syntax shows the activation together with its direction. See Figure 5.

$$\boxed{\begin{array}{lcll} (act-L) & : & P\widehat{\alpha} \dagger \widehat{x}Q \ \to P\widehat{\alpha} \nearrow \widehat{x}Q, & \text{with } P \neq \widehat{y}\,P'\,\widehat{\beta}\cdot\alpha \text{ and } P \neq \langle y.\alpha\rangle \\ (act-R) & : & P\widehat{\alpha} \dagger \widehat{x}Q \ \to P\widehat{\alpha} \nwarrow \widehat{x}Q, & \text{with } Q \neq Q'\,\widehat{\beta}\,[x]\,\widehat{y}\,Q'' \text{ and } Q \neq \langle x.\beta\rangle \end{array}}$$

Figure 5. Activation Rules

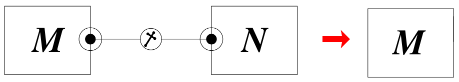
Definition 2 (Active Cuts) *The syntax is extended with the two* activated cuts*:*

$$M, N ::= \ldots \mid M\widehat{\alpha} \nearrow \widehat{x}N \mid M\widehat{\alpha} \nwarrow \widehat{x}N$$

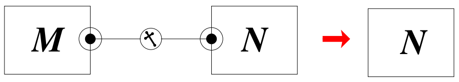
The terms $P\widehat{\alpha} \nearrow \widehat{x}Q$ and $P\widehat{\alpha} \nwarrow \widehat{x}Q$ are essentially different. Perhaps this is best illustrated by Lafont's example [GLT89]. Take $P = M \odot \alpha$ and $Q = x \odot N$, where M and N are arbitrary terms. Then we have:

$$\begin{array}{l} (M \odot \alpha)\widehat{\alpha} \nearrow \widehat{x}(x \odot N) \to \Phi^N_{in} \odot M \odot \Phi^N_{on} \\ (M \odot \alpha)\widehat{\alpha} \nwarrow \widehat{x}(x \odot N) \to \Phi^M_{in} \odot N \odot \Phi^M_{on} \end{array}$$

corresponding to the diagrams



and



when M and N are chosen to be closed diagrams for simplicity.

Notice that once activated cuts could be "deactivated" after been propagated. This will happen when the conditions assigned to $(act-L)$ and $(act-R)$ are no longer fulfilled. These actions are specified by the first two rules in Figures 7 and 9.

Notice also that due to side conditions, interplay between the reductions specifying the activation and the ones specifying the deactivation can not cause infinite reduction sequences like *act*, *deact*, *act*, *deact* ...

Priority Sometimes a cut can be activated in two ways. We define a *to-the-left* strategy, written $\to_{\nearrow}$, as a strategy of cut activation that always activates the cut to the left when side conditions are enabled in both activation rules in Figure 5. The *to-the-right* strategy, $\to_{\nwarrow}$, is defined likewise. The two reductions $\to_{\nearrow}$ and $\to_{\nwarrow}$ are confluent.

Logical Rules Logical rules describe the cut-elimination between two terms which both introduce names involved in a cut, i.e. they refer to cuts which can not be activated. See Figure 6

The first three logical rules specify a renaming. The third logical rule represents the basic computational step. It describes the direct interaction between an exporter and a importer, which results in inserting the subterm of an exporter between the two subterms of a importer.

Inname and Outname Actions Each of these groups has four rules. The first two rules specify deactivation of cuts. The other two specify erasure and duplication by referring to the situation when that cut which is being propagated meets a weakening or a contraction. See Figures 7 and 9.

$$\begin{array}{llll}
(cap-ren) & : & \langle y.\alpha\rangle\widehat{\alpha} \dagger \widehat{x}\langle x.\beta\rangle & \rightarrow \langle y.\beta\rangle \\
(exp-ren) & : & (\widehat{y}\,P\,\widehat{\beta}\cdot\alpha)\widehat{\alpha} \dagger \widehat{x}\langle x.\gamma\rangle & \rightarrow \widehat{y}\,P\,\widehat{\beta}\cdot\gamma \\
(med-ren) & : & \langle y.\alpha\rangle\widehat{\alpha} \dagger \widehat{x}(P\,\widehat{\beta}\,[x]\,\widehat{z}\,Q) & \rightarrow P\,\widehat{\beta}\,[y]\,\widehat{z}\,Q \\
(exp-med) & : & (\widehat{y}\,P\,\widehat{\beta}\cdot\alpha)\widehat{\alpha} \dagger \widehat{x}(Q\,\widehat{\gamma}\,[x]\,\widehat{z}\,R) & \rightarrow \mathit{either}\begin{cases}(Q\widehat{\gamma}\dagger\widehat{y}P)\widehat{\beta}\dagger\widehat{z}R\\ Q\widehat{\gamma}\dagger\widehat{y}(P\widehat{\beta}\dagger\widehat{z}R)\end{cases}
\end{array}$$

Figure 6. Logical Rules

$$\begin{array}{lll}
(\nwarrow cap-deactivation) & : P\widehat{\alpha}\nwarrow\widehat{x}\langle x.\beta\rangle & \rightarrow P\widehat{\alpha}\dagger\widehat{x}\langle x.\beta\rangle \\
(\nwarrow imp-deactivation) & : P\widehat{\alpha}\nwarrow\widehat{x}(Q\,\widehat{\beta}\,[x]\,\widehat{y}\,R) & \rightarrow P\widehat{\alpha}\dagger\widehat{x}(Q\,\widehat{\beta}\,[x]\,\widehat{y}\,R) \\
(\nwarrow erasure) & : P\widehat{\alpha}\nwarrow\widehat{x}(x\odot Q) & \rightarrow \Phi_{in}\odot Q\odot\Phi_{on} \\
 & & \quad \Phi_{in}=f_{in}(P),\ \Phi_{on}=f_{on}(P)\backslash\{\alpha\} \\
(\nwarrow duplication) & : P\widehat{\alpha}\nwarrow\widehat{x}(x{<}^{\widehat{x_1}}_{\widehat{x_2}}\langle Q]) & \rightarrow \Phi_{in}<^{\widehat{\Phi_{in,1}}}_{\widehat{\Phi_{in,2}}}\langle P_2\widehat{\alpha_2}\nwarrow\widehat{x_2}(P_1\widehat{\alpha_1}\nwarrow\widehat{x_1}Q\rangle^{\widehat{\Phi_{on,1}}}_{\widehat{\Phi_{on,2}}})>\Phi_{on} \\
 & & \quad \Phi_{in}=f_{in}(P),\ \Phi_{on}=f_{on}(P)\setminus\{\alpha\} \\
 & & \quad P_1=ind(P,\Phi_{in}\cup\Phi_{on},1) \qquad P_2=ind(P,\Phi_{in}\cup\Phi_{on},2)
\end{array}$$

Figure 7. Inname Actions

Left and Right-Propagation Rules Propagation rules describe how cut is propagated through the structure of terms (Figure 8 and Figure 10). Even the propagation of cuts over other inactive cuts is enabled, therefore it is possible to represent β-reduction. The propagation rules are relevant only in 1-dimensional syntax, i.e. they do not correspond to operations on diagrams, they only change the view (the parsing) one has on diagrams.

Reduction system enjoys some nice properties as expressed by the following lemma.

Lemma 3 (Basic properties of $\xrightarrow{*\mathcal{X}}$)

1. *The reductions preserve the set of free names: If $P \xrightarrow{*\mathcal{X}} Q$ then $f_n(P) = f_n(Q)$ (Lafont's interface preservation [Laf95]).*
2. *The reductions preserve linearity: If P is linear and $P \xrightarrow{*\mathcal{X}} Q$ then Q is linear.*

Proof. The proof proceeds by induction on the length of the reduction rules.

5 A Type System

In this framework, types are arrow types. Given a set T of basic types, a type is given by

$$A, B ::= T \mid A \rightarrow B.$$

The type assignment of a term is given by a sequent $\Gamma \vdash \Delta$, where Γ is a set[6] of type declarations for innames, e.g., $\Gamma \equiv x : A, y : B$ and Δ is a set of declarations for outnames, e.g., $\Delta \equiv \alpha : A, \beta : A \rightarrow B, \gamma : C$. The type system for $*\mathcal{X}$ is given in Figure 11.

Implicational Sequent Calculus If we remove the term-decoration in the type system of Figure 11, we get the system in Figure 12.

We notice that this system is just the ***context splitting classical implicational sequent calculus***.

[6] In fact Γ and Δ are multisets, but since all names are distinct we can treat them as sets

$$\begin{array}{lll}
(\nwarrow exp-prop) & : P\widehat{\alpha}\nwarrow\widehat{x}(\widehat{y}\,Q\,\widehat{\beta}\cdot\gamma) & \to \widehat{y}\,(P\widehat{\alpha}\nwarrow\widehat{x}Q)\,\widehat{\beta}\cdot\gamma \\
(\nwarrow imp-prop_1) & : P\widehat{\alpha}\nwarrow\widehat{x}(Q\,\widehat{\beta}\,[v]\,\widehat{z}\,R) & \to (P\widehat{\alpha}\nwarrow\widehat{x}Q)\,\widehat{\beta}\,[y]\,\widehat{z}\,R, \quad x\in f_{in}(Q) \\
(\nwarrow imp-prop_2) & : P\widehat{\alpha}\nwarrow\widehat{x}(Q\,\widehat{\beta}\,[v]\,\widehat{z}\,R) & \to Q\,\widehat{\beta}\,[y]\,\widehat{z}\,(P\widehat{\alpha}\nwarrow\widehat{x}R), \quad x\in f_{in}(R) \\
& & \\
(\nwarrow cut(caps)-prop) & : P\widehat{\alpha}\nwarrow\widehat{x}(\langle x.\beta\rangle\widehat{\beta}\dagger\widehat{y}R) & \to P\widehat{\alpha}\dagger\widehat{y}R \\
(\nwarrow cut-prop_1) & : P\widehat{\alpha}\nwarrow\widehat{x}(Q\widehat{\beta}\dagger\widehat{y}R) & \to (P\widehat{\alpha}\nwarrow\widehat{x}Q)\widehat{\beta}\dagger\widehat{y}R, \quad x\in f_{in}(Q),\ Q\neq\langle x.\beta\rangle \\
(\nwarrow cut-prop_2) & : P\widehat{\alpha}\nwarrow\widehat{x}(Q\widehat{\beta}\dagger\widehat{y}R) & \to Q\widehat{\beta}\dagger\widehat{y}(P\widehat{\alpha}\nwarrow\widehat{x}R), \quad x\in f_{in}(R),\ Q\neq\langle x.\beta\rangle \\
& & \\
(\nwarrow L\text{-}eras-prop) & : P\widehat{\alpha}\nwarrow\widehat{x}(y\odot Q) & \to y\odot(P\widehat{\alpha}\nwarrow\widehat{x}Q), \quad x\neq y \\
(\nwarrow R\text{-}eras-prop) & : P\widehat{\alpha}\nwarrow\widehat{x}(Q\odot\beta) & \to (P\widehat{\alpha}\nwarrow\widehat{x}Q)\odot\beta \\
(\nwarrow L\text{-}dupl-prop) & : P\widehat{\alpha}\nwarrow\widehat{x}(y{<}\tfrac{\widehat{y_1}}{\widehat{y_2}}\langle Q]) & \to y{<}\tfrac{\widehat{y_1}}{\widehat{y_2}}\langle P\widehat{\alpha}\nwarrow\widehat{x}Q], \quad x\neq y \\
(\nwarrow R\text{-}dupl-prop) & : P\widehat{\alpha}\nwarrow\widehat{x}([Q\rangle\tfrac{\widehat{\beta_1}}{\widehat{\beta_2}}{>}\beta) & \to [P\widehat{\alpha}\nwarrow\widehat{x}Q\rangle\tfrac{\widehat{\beta_1}}{\widehat{\beta_2}}{>}\beta
\end{array}$$

Figure 8. Right-propagation

$$\begin{array}{lll}
(cap\nearrow-deactivation) & : \langle x.\beta\rangle\widehat{\beta}\nearrow\widehat{y}R & \to \langle x.\beta\rangle\widehat{\beta}\dagger\widehat{y}R \\
(exp\nearrow-deactivation) & : (\widehat{x}\,P\,\widehat{\gamma}\cdot\beta)\widehat{\beta}\nearrow\widehat{y}R & \to (\widehat{x}\,P\,\widehat{\gamma}\cdot\beta)\widehat{\beta}\dagger\widehat{y}R \\
(\nearrow erasure) & : (P\odot\beta)\widehat{\beta}\nearrow\widehat{y}R & \to \Phi_{in}\odot P\odot\Phi_{on} \\
& & \quad \Phi_{in}=f_{in}(R)\backslash\{y\},\ \Phi_{on}=f_{on}(R) \\
(\nearrow duplication) & : ([P\rangle\tfrac{\widehat{\beta_1}}{\widehat{\beta_2}}{>}\beta)\widehat{\beta}\nearrow\widehat{y}R & \to \Phi_{in}{<}\tfrac{\widehat{\Phi_{in,1}}}{\widehat{\Phi_{in,2}}}\langle(P\widehat{\beta_1}\nearrow\widehat{y_1}R_1)\widehat{\beta_2}\nearrow\widehat{y_2}R_2\rangle\tfrac{\widehat{\Phi_{on,1}}}{\widehat{\Phi_{on,2}}}{>}\Phi_{on} \\
& & \quad \Phi_{in}=f_{in}(R)\setminus\{y\},\ \Phi_{on}=f_{on}(R) \\
& & \quad R_1=ind(R,\Phi_{in}\cup\Phi_{on},1) \qquad R_2=ind(R,\Phi_{in}\cup\Phi_{on},2)
\end{array}$$

Figure 9. Outname Actions

- *Implicational* means that only arrows (or implications) occur to form types (or propositions).
- *Classical* means that the types (or the propositions) are these of the implicational fragment of classical propositional logic.
- *Context splitting* means that in the rules with two premises, namely $(L\to)$ and (cut), contexts are split.

Lemma 4 (Subject Reduction)

- *If* $M :\cdot\ \Gamma\vdash\Delta$ *and* $M\equiv M'$, *then* $M' :\cdot\ \Gamma\vdash\Delta$
- *If* $S :\cdot\ \Gamma\vdash\Delta$ *and* $S\xrightarrow{*\mathcal{X}}S'$, *then* $S' :\cdot\ \Gamma\vdash\Delta$

6 Representing $\boldsymbol{\lambda}$-Calculus

$*\mathcal{X}$ is closely connected to explicit substitution. It can be seen as a calculus of explicit substitution in which entities propagate into two directions. In one direction, the substitution propagates through the term. In the other direction, the term propagates through the substitution. This is actually the duality exhibited by interpretations of classical logic [CH00].

In order to present the encoding elegantly we define the operation $\circledcirc$ which adds erasers where necessary:

$$x\circledcirc[\![M]\!]_\alpha=\begin{cases} x\odot[\![M]\!]_\alpha,\ x\notin fv(M)\\ \quad\ \ [\![M]\!]_\alpha,\ x\in fv(M)\end{cases}$$

$$\boxed{\begin{array}{lcll}
(\mathit{exp}\not\nearrow - \mathit{prop}) & : & (\widehat{x}\,P\,\widehat{\gamma}\cdot\alpha)\widehat{\beta}\not\nearrow\widehat{y}R & \rightarrow\ \widehat{x}\,(P\widehat{\beta}\not\nearrow\widehat{y}R)\,\widehat{\gamma}\cdot\alpha \\
(\mathit{imp}\not\nearrow - \mathit{prop}_1) & : & (P\,\widehat{\alpha}\ [x]\ \widehat{z}\,Q)\widehat{\beta}\not\nearrow\widehat{y}R & \rightarrow\ (P\widehat{\beta}\not\nearrow\widehat{y}R)\,\widehat{\alpha}\ [x]\ \widehat{z}\,Q, \quad \beta\in f_{on}(P) \\
(\mathit{imp}\not\nearrow - \mathit{prop}_2) & : & (P\,\widehat{\alpha}\ [x]\ \widehat{z}\,Q)\widehat{\beta}\not\nearrow\widehat{y}R & \rightarrow\ P\,\widehat{\alpha}\ [x]\ \widehat{z}\,(Q\widehat{\beta}\not\nearrow\widehat{y}R), \quad \beta\in f_{on}(Q) \\
 & & & \\
(\mathit{cut}(\mathit{caps})\not\nearrow - \mathit{prop}) & : & (P\widehat{\alpha}\dagger\widehat{x}\langle x.\beta\rangle)\widehat{\beta}\not\nearrow\widehat{y}R & \rightarrow\ P\widehat{\alpha}\dagger\widehat{y}R \\
(\mathit{cut}\not\nearrow - \mathit{prop}_1) & : & (P\widehat{\alpha}\dagger\widehat{x}Q)\widehat{\beta}\not\nearrow\widehat{y}R & \rightarrow\ (P\widehat{\beta}\not\nearrow\widehat{y}R)\widehat{\alpha}\dagger\widehat{x}Q, \quad \beta\in f_{on}(P),\ q\neq\langle x.\beta\rangle \\
(\mathit{cut}\not\nearrow - \mathit{prop}_2) & : & (P\widehat{\alpha}\dagger\widehat{x}Q)\widehat{\beta}\not\nearrow\widehat{y}R & \rightarrow\ P\widehat{\alpha}\dagger\widehat{x}(Q\widehat{\beta}\not\nearrow\widehat{y}R), \quad \beta\in f_{on}(Q),\ q\neq\langle x.\beta\rangle \\
 & & & \\
(L\text{-}\mathit{eras}\not\nearrow - \mathit{prop}) & : & (x\odot M)\widehat{\beta}\not\nearrow\widehat{y}R & \rightarrow\ x\odot(M\widehat{\beta}\not\nearrow\widehat{y}R) \\
(R\text{-}\mathit{eras}\not\nearrow - \mathit{prop}) & : & (M\odot\alpha)\widehat{\beta}\not\nearrow\widehat{y}R & \rightarrow\ (M\widehat{\beta}\not\nearrow\widehat{y}R)\odot\alpha, \quad \alpha\neq\beta \\
 & & & \\
(L\text{-}\mathit{dupl}\not\nearrow - \mathit{prop}) & : & (x{<}\genfrac{}{}{0pt}{}{\widehat{x_1}}{\widehat{x_2}}\langle M])\widehat{\beta}\not\nearrow\widehat{y}R & \rightarrow\ x{<}\genfrac{}{}{0pt}{}{\widehat{x_1}}{\widehat{x_2}}\langle M\widehat{\beta}\not\nearrow\widehat{y}R] \\
(R\text{-}\mathit{dupl}\not\nearrow - \mathit{prop}) & : & ([M\rangle\genfrac{}{}{0pt}{}{\widehat{\alpha_1}}{\widehat{\alpha_2}}{>}\alpha)\widehat{\beta}\not\nearrow\widehat{y}R & \rightarrow\ [M\widehat{\beta}\not\nearrow\widehat{y}R\rangle\genfrac{}{}{0pt}{}{\widehat{\alpha_1}}{\widehat{\alpha_2}}{>}\alpha, \quad \alpha\neq\beta
\end{array}}$$

Figure 10. Left propagation

Notice that M is a λ-calculus term, and that x is here a variable of the λ-calculus (hence the use fv for "free variables"). The same letters are used for variables as for innames in $^*\mathcal{X}$. Moreover, they correspond to each other in the sense that a free variable in λ-calculus becomes a free inname in $*\mathcal{X}$ after the encoding.

Definition 3 (Encoding λ-calculus) *The encoding of λ-calculus terms in $^*\mathcal{X}$-calculus is defined as follows:*

$$\begin{aligned}
\llbracket x\rrbracket_\alpha &:= \langle x.\alpha\rangle \\
\llbracket \lambda x.M\rrbracket_\alpha &:= \widehat{x}\,(x\circledcirc\llbracket M\rrbracket_\beta)\,\widehat{\beta}\cdot\alpha \\
\llbracket MN\rrbracket_\alpha &:= \Phi_{in}{<}\genfrac{}{}{0pt}{}{\widehat{\Phi_{in,1}}}{\widehat{\Phi_{in,2}}}\langle\underline{M}\widehat{\gamma}\dagger\widehat{x}\underline{N}],
\end{aligned}$$

$$\begin{aligned}
&\text{where} \\
&\Phi_{in} = fv(M)\cap fv(N)\setminus\{x\} \\
&\underline{M} = \mathit{ind}(\llbracket M\rrbracket_\gamma,\Phi_{in},1) \\
&\underline{N} = \mathit{ind}(\llbracket N\rrbracket_\beta,\Phi_{in},2)\,\widehat{\beta}\ [x]\ \widehat{y}\,\langle y.\alpha\rangle
\end{aligned}$$

Lemma 5 (Faithfulness) *If $M\xrightarrow{\beta}N$, then $\llbracket M\rrbracket_\alpha\xrightarrow{*\mathcal{X}}(f_{in}(\llbracket M\rrbracket_\alpha)\setminus f_{in}(\llbracket N\rrbracket_\alpha))\odot\llbracket N\rrbracket_\alpha$.*

For the simple case, when $f_{in}(\llbracket M\rrbracket_\alpha)\setminus f_{in}(\llbracket N\rrbracket_\alpha)=\emptyset$, the property becomes $\llbracket M\rrbracket_\alpha\xrightarrow{*\mathcal{X}}\llbracket N\rrbracket_\alpha$.

Representing λx To add the coding for explicit substitution, namely for λx-calculus is easy [BR95,LLD$^+$04]. We just have to add the definition:

$$\llbracket M\langle x=N\rangle\rrbracket_\alpha \ := \ \Phi_{in}{<}\genfrac{}{}{0pt}{}{\widehat{\Phi_{in,1}}}{\widehat{\Phi_{in,2}}}\langle\underline{N}\,\widehat{\beta}\not\nwarrow\widehat{x}\underline{M}\,] \quad \text{where} \quad \begin{aligned}\Phi_{in} &= fv(M)\cap fv(N)\setminus\{x\} \\ \underline{N} &= \mathit{ind}(\llbracket N\rrbracket_\beta,\Phi_{in},1) \\ \underline{M} &= \mathit{ind}(\llbracket M\rrbracket_\alpha,\Phi_{in},2)\end{aligned}$$

Untyped $^*\mathcal{X}$ is Turing Complete Since it implements λ-calculus, untyped $^*\mathcal{X}$ is clearly Turing complete.

Call-by-Name and Call-by-Value $^*\mathcal{X}$ can easily represent and clarify *call-by-name* and *call-by-value* in the λ-calculus following the approach presented in [vBLL05] to the new operators and which is based on strategies $\rightarrow_{\not\nearrow}$ and $\rightarrow_{\not\nwarrow}$.

$$\frac{}{\langle x.\alpha\rangle : x : A \vdash \alpha : A}\ \textit{(capsule)}$$

$$\frac{M : \Gamma \vdash \alpha : A, \Delta \qquad N : \Gamma', x : B \vdash \Delta'}{M\,\widehat{\alpha}\,[y]\,\widehat{x}\,N : \Gamma, \Gamma', A \to B \vdash \Delta, \Delta'}\ \textit{(importer)} \qquad \frac{M : \Gamma, x : A \vdash \alpha : B, \Delta}{\widehat{x}\,M\,\widehat{\alpha}\cdot\beta : \Gamma \vdash \beta : A \to B, \Delta}\ \textit{(exporter)}$$

$$\frac{P : \Gamma \vdash \alpha : A, \Delta \qquad Q : \Gamma', x : A \vdash \Delta'}{P\widehat{\alpha} \dagger \widehat{x}Q : \Gamma, \Gamma' \vdash \Delta, \Delta'}\ \textit{(cut)}$$

$$\frac{M : \Gamma \vdash \Delta}{x \odot M : \Gamma, x : A \vdash \Delta}\ \textit{(L-eraser)} \qquad \frac{M : \Gamma \vdash \Delta}{M \odot \alpha : \Gamma \vdash \alpha : A, \Delta}\ \textit{(R-eraser)}$$

$$\frac{M : \Gamma, x : A, y : A \vdash \Delta}{z{<}^{\widehat{x}}_{\widehat{y}}\langle M] : \Gamma, z : A \vdash \Delta}\ \textit{(L-duplicator)} \qquad \frac{M : \Gamma \vdash \alpha : A, \beta : A, \Delta}{[M\rangle^{\widehat{\alpha}}_{\widehat{\beta}}{>}\gamma : \Gamma \vdash \gamma : A, \Delta}\ \textit{(R-duplicator)}$$

Figure 11. Type System

$$\frac{}{A \vdash A}\ \textit{(axiom)}$$

$$\frac{\Gamma \vdash A, \Delta \qquad \Gamma', B \vdash \Delta'}{\Gamma, \Gamma', A \to B \vdash \Delta, \Delta'}\ \textit{(L}{\to}\textit{)} \qquad \frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \to B, \Delta}\ \textit{(R}{\to}\textit{)}$$

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

$$\frac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta}\ \textit{(L-weakening)} \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash A, \Delta}\ \textit{(R-weakening)}$$

$$\frac{\Gamma, A, A \vdash \Delta}{\Gamma, A \vdash \Delta}\ \textit{(L-contraction)} \qquad \frac{\Gamma \vdash A, A, \Delta}{\Gamma \vdash A, \Delta}\ \textit{(R-contraction)}$$

Figure 12. Classical Sequent Calculus

Using Diagrams To end, we give two diagrams. The diagram on the left corresponds to the combinator $S \equiv \lambda xyz.xz(yz)$ (encoded in $^*\mathcal{X}$) while the diagram on the right corresponds to the Peirce law (Peirce law can not be associated with a λ-term). They can be typed.

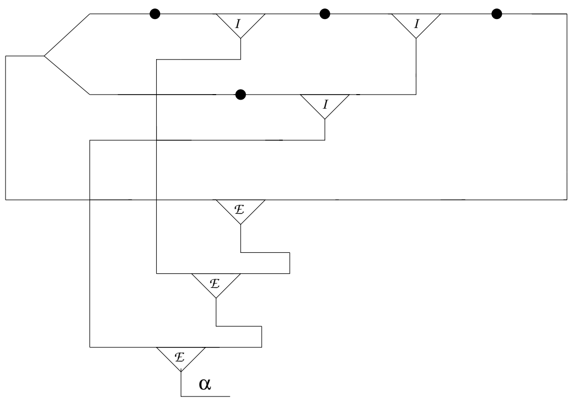
– The type $(A \to B \to C) \to (A \to B) \to A \to C$ can be assigned to α, for the first diagram (S-combinator).
– The type $((A \to B) \to A) \to A$ can be assigned to α, for the second diagram (Peirce law).

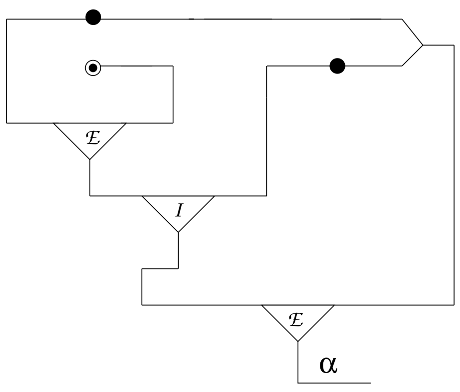
If one is interested to compare, we also give 1-dimensional $^*\mathcal{X}$-terms corresponding to these diagrams, thus we have:

$$\widehat{\omega}\,(\widehat{u}\,(\widehat{x}\,(x{<}^{\widehat{x_1}}_{\widehat{x_2}}\langle\langle x_2.\epsilon\rangle\,\widehat{\epsilon}\,[w]\,\widehat{v}\,(((\langle x_1.\delta\rangle\,\widehat{\delta}\,[u]\,\widehat{y}\,\langle y.\beta\rangle)\,\widehat{\beta}\,[v]\,\widehat{z}\,\langle z.\gamma\rangle))])\,\widehat{\gamma}\cdot\eta)\,\widehat{\eta}\cdot\theta)\,\widehat{\theta}\cdot\alpha$$

$$\widehat{z}\,([(\widehat{x}\,(\langle x.\delta_1\rangle \odot \beta)\,\widehat{\beta}\cdot\gamma)\,\widehat{\gamma}\,[z]\,\widehat{y}\,\langle y.\delta_2\rangle\rangle^{\widehat{\delta_1}}_{\widehat{\delta_2}}{>}\delta)\,\widehat{\delta}\cdot\alpha$$

for the first and second diagram, respectively.





7 Related Work and Inspiration

The paper lies in a long chain of research on classical logic initiated by Gentzen [Gen35] and Griffin [Gri90]. Here is a non exhaustive list of works which inspired us.

- *The several calculi for interpreting classical logic* were a source of inspiration among them the works on constructive classical logic of Girard [Gir91] and Danos, Joinet and Schellinx [DJS97], the works on $\lambda\mu$ by Parigot [Par92,Par97], Ong and Steward [OS97] and de Groote [dG94] and the work on the symmetric calculus by Barbanera and Berardi [BB96] and Wadler [Wad03,DGL05]. The strongest influence in this respect is this of Curien and Herbelin [CH00] with their dual calculus.
- Further in that direction the researches about strong normalization of cut elimination in classical logic which proposed languages, namely this of Urban [Urb00,UB01] and Lengrand [Len03] which deeply influenced us.
- *Proof nets in linear logic* are clearly behind this work [Gir87,Laf95,Lau03], but also *interaction nets* [Laf90,Laf95,AG99].
- Last but not least, the connection with the main stream of *diagram directed design of computer system* is important [Laf90,Mil99] and has been presented in the introduction.

We would like to give a special mention to two works that are the main sources of our inspiration, namely the work on $\mathcal{X}$ [vBLL05], which is the ancestor of our $^*\mathcal{X}$, and the work of Kesner and Lengrand [KL06] who introduced explicit duplicators and explicit erasers in a calculus of explicit substitution.

8 Conclusion

In this paper we have presented the calculus $^*\mathcal{X}$ which is first a diagrammatic calculus, but which possesses also a 1-dimensional syntax. Our approach can be summarized as *the power of logic with the clarity of diagrams in programming.*

References

[AG99] A. Asperti and S. Guerrini. *The optimal implementation of functional programming languages.* Cambridge Univ. Press, 1999.

[AriBC] Aristote. Organon, circa 350 BC.

[BB96] F. Barbanera and S. Berardi. A symmetric lambda calculus for classical program extraction. *Inform. and Comput.*, 125(2):103–117, 1996.

[BR95] R. Bloo and K.H. Rose. Preservation of strong normalisation in named lambda calculi with explicit substitution and garbage collection. In *CSN'95 – Comput. Sci. in the Netherlands*, pages 62–72, 1995.

[Car82] L. Cardelli. Two-dimensional syntax for functional languages. In *Proc. European Conf. on Integrated Interactive Computing Systems, ECICS 82*, pages 139–151. North Holland, 1982.

[CH00] Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In *Proc. 5 th ACM SIGPLAN Int. Conf. on Functional Programming (ICFP'00)*, pages 233–243. ACM, 2000.

[dG94] Ph. de Groote. On the relation between the $\lambda\mu$-calculus and the syntactic theory of sequential control. In *5th Int. Conf. LPAR*, volume 822 of *Lecture Notes in Comput. Sci.*, pages 31–43, 1994.

[DGL05] D. Dougherty, S. Ghilezan, and P. Lescanne. Strong normalization of the dual classical sequent calculus. In *12th Int. Conf. LPAR*, volume 3835 of *Lecture Notes in Comput. Sci.*, pages 169–183, 2005.

[DJS97] V. Danos, J.-B. Joinet, and H. Schellinx. A new deconstructive logic: Linear logic. *J. Symbolic Logic*, 62, 1997.

[Gen35] G. Gentzen. Untersuchungen über das logische Schließen. *Math. Z.*, 39:176–210, 405–431, 1935.

[Gir87] J.-Y. Girard. Linear logic. *Theoret. Comput. Sci.*, 50:1–102, 1987.

[Gir91] J.-Y. Girard. A new constructive logic: classical logic. *Math. Structures Comput. Sci.*, 1(3):255–296, 1991.

[GLT89] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoret Comput. Sci.* Cambridge Univ. Press, 1989.

[Gri90] T. Griffin. A formulae-as-types notion of control. In *Proc.17th Annual ACM Symp. on Principles Of Programming Languages*, pages 47–58, 1990.

[KL06] D. Kesner and S. Lengrand. Explicit operators for λ-calculus. *Inform. and Comput.*, 2006. extended version of a communication at RTA-05, to be published.

[Laf90] Y. Lafont. Interaction nets. In *POPL'90*, pages 95– 108, 1990.

[Laf95] Y. Lafont. From proof nets to interaction nets. In *Advances in Linear Logic*, pages 225–247. Cambridge Univ. Press, 1995.

[Lau03] O. Laurent. Polarized proof-nets and $\lambda\mu$-calculus. *Theoret. Comput. Sci.*, 290(1):161–188, 2003.

[Len03] S. Lengrand. Call-by-value, call-by-name, and strong normalization for the classical sequent calculus. In *Electron. Notes Theor. Comput. Sci.*, volume 86, 2003.

[LLD$^+$04] S. Lengrand, P.. Lescanne, D. Dougherty, M. Dezani-Ciancaglini, and S. van Bakel. Intersection types for explicit substitutions. *Inform. and Comput.*, 189(1):17–42, 2004.

[Mil99] R. Milner. *Communicating and Mobile Systems: theπ-Calculus.* Cambridge Univ. Press, 1999.

[OS97] C.-H. L. Ong and C. A. Stewart. A Curry-Howard foundation for functional computation with control. In *Proc.24th Annual ACM Symp. on Principles Of Programming Languages*, pages 215–227, 1997.

[Par92] M. Parigot. An algorithmic interpretation of classical natural deduction. In *Int. Conf. LPAR*, volume 624 of *Lecture Notes in Comput. Sci.*, pages 190–201, 1992.

[Par97] M. Parigot. Proofs of strong normalisation for second order classical natural deduction. *J. Symbolic Logic*, 62(4):1461–1479, December 1997.

[UB01] C. Urban and G. M. Bierman. Strong normalisation of cut-elimination in classical logic. *Fundam. Inf.*, 45(1,2):123–155, 2001.

[Urb00] C. Urban. *Classical Logic and Computation.* PhD thesis, Univ. of Cambridge, October 2000.

[vBLL05] S. van Bakel, S. Lengrand, and P. Lescanne. The language $\mathcal{X}$: circuits, computations and classical logic. In *Proc.9th Italian Conf. on Theoretical Computer Science (ICTCS'05)*, volume 3701 of *Lecture Notes in Comput. Sci.*, pages 81–96, 2005.

[Wad03] P. Wadler. Call-by-value is dual to call-by-name. In *Proc.8th Int. Conf. on Functional Programming*, 2003.

[WR25] A N. Whitehead and B. Russell. *Principia Mathematica.* Cambridge Univ. Press, 2nd edition, 1925.

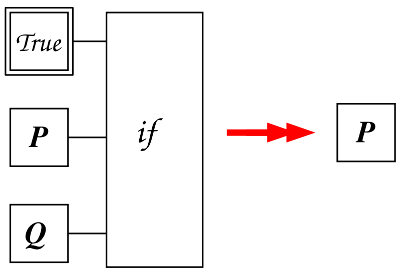
A The Terminology

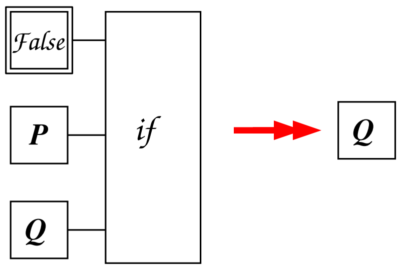
The two systems presented in this work exhibit a Curry-Howard correspondence for classical logic. The terminology is summarized in the following table.

diagrams	*1-dim syntax*	*types*
in-out	capsule	axiom
$\mathcal{E}$-fan	exporter	→R-intro
$\mathcal{I}$-fan	importer	→L-intro
fork	duplicator	contraction
black hole	eraser	weakening
dagger	cut	cut
port	name	named proposition

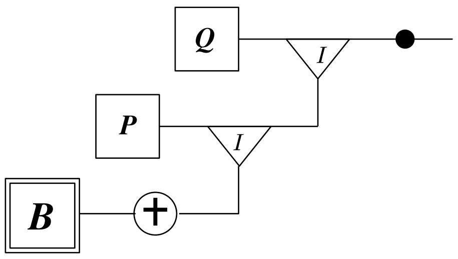
B An Example

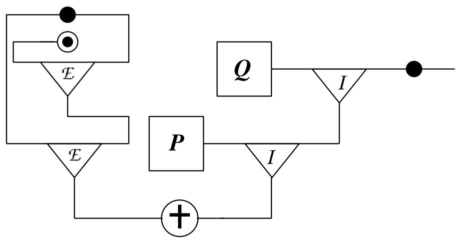
In a programming language, one of the most basic construction is a *conditional* (aka *if*). Let us say in terms of diagrams what we expect from an *if*. When the diagram $True$ and two arbitrary diagrams P and Q are plugged into *if* it reduces to P and, if instead of $True$ we use $False$ it yields Q.



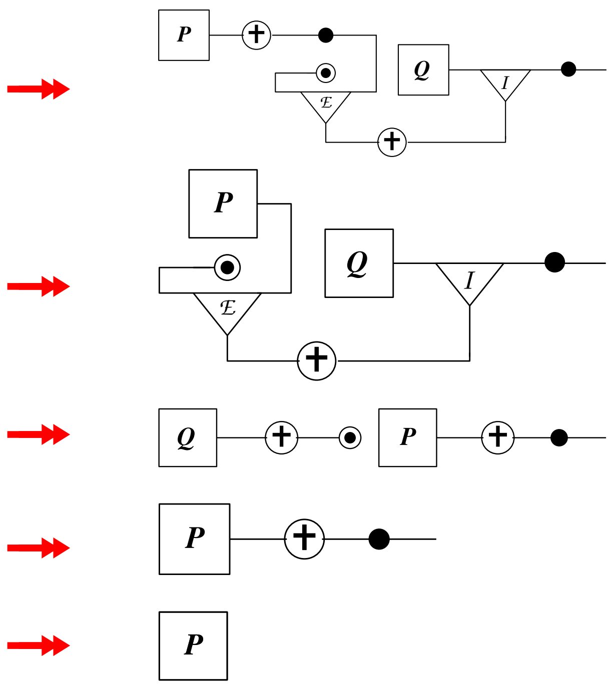


Actually $if(B, P, Q)$ corresponds to the construction represented by the first, while the second diagram gives a more detailed view (one see the diagram proposed for $B = True$).





Below are reduction steps leading to P. For simplicity we assume that M and N are closed diagrams diagrams.



Notice that starting from the first diagram in this sequence, we would have got the same result if we would have reduced the right dagger first. This means that, in this specific case, priority of daggers is irrelevant. Since $^*\mathcal{X}$ is not confluent, this cannot be generalized.

C The Example Revisited

Here we wish to revisit the example given in Section B, this time using one-dimensional syntax. The terms coding the diagrams for *True* and *False* are:

$$\begin{aligned} True &\triangleq \widehat{x}\,(\widehat{y}\,(y \odot \langle x.\alpha\rangle)\,\widehat{\alpha}\cdot\beta)\,\widehat{\beta}\cdot\gamma \\ False &\triangleq \widehat{y}\,(\widehat{x}\,(y \odot \langle x.\alpha\rangle)\,\widehat{\alpha}\cdot\beta)\,\widehat{\beta}\cdot\gamma \end{aligned}$$

The term representing the conditional is:

$$\mathbf{B}\widehat{\gamma} \dagger \widehat{z}(P\,\widehat{\delta}\;[z]\;\widehat{t}\,(Q\,\widehat{\epsilon}\;[t]\;\widehat{u}\,\langle u.\sigma\rangle))$$

Let P and Q be arbitrary terms (for simplicity we will assume that they are closed), then the above term when $\mathbf{B} = True$ reduces to P and for $\mathbf{B} = False$ reduces to Q, as shown bellow.

Reducing the conditional term when $\mathbf{B} = True$ yields:

$$\begin{aligned} &(\widehat{x}\,(\widehat{y}\,(y \odot \langle x.\alpha\rangle)\,\widehat{\alpha}\cdot\beta)\,\widehat{\beta}\cdot\gamma)\widehat{\gamma} \dagger \widehat{z}(P\,\widehat{\delta}\;[z]\;\widehat{t}\,(Q\,\widehat{\epsilon}\;[t]\;\widehat{u}\,\langle u.\sigma\rangle)) \\ &\to (P\widehat{\delta} \dagger \widehat{x}(\widehat{y}\,(y \odot \langle x.\{\rangle\alpha\})\,\widehat{\{}\cdot\alpha\}\beta))\widehat{\beta} \dagger \widehat{t}(Q\,\widehat{\epsilon}\;[t]\;\widehat{u}\,\langle u.\sigma\rangle)) \\ &\to (\widehat{y}\,(y \odot P\{\alpha/\delta\})\,\widehat{\alpha}\cdot\beta)\widehat{\beta} \dagger \widehat{t}(Q\,\widehat{\epsilon}\;[t]\;\widehat{u}\,\langle u.\sigma\rangle)) \\ &\to (Q\widehat{\epsilon} \dagger \widehat{y}(y \odot P\{\alpha/\delta\}))\widehat{\alpha} \dagger \widehat{u}\langle u.\sigma\rangle \\ &\to (y \odot P\{\alpha/\delta\})\widehat{\alpha} \dagger \widehat{u}\langle u.\sigma\rangle \\ &\to P\{\alpha/\delta\}\{\sigma/\alpha\} \end{aligned}$$

Reducing the conditional term when $\mathbf{B} = False$ yields:

$$\begin{aligned} &(\widehat{y}\,(\widehat{x}\,(y \odot \langle x.\alpha\rangle)\,\widehat{\alpha}\cdot\beta)\,\widehat{\beta}\cdot\gamma)\widehat{\gamma} \dagger \widehat{z}(P\,\widehat{\delta}\;[z]\;\widehat{t}\,(Q\,\widehat{\epsilon}\;[t]\;\widehat{u}\,\langle u.\sigma\rangle)) \\ &\to (P\widehat{\delta} \dagger \widehat{y}(\widehat{x}\,(y \odot \langle x.\alpha\rangle)\,\widehat{\alpha}\cdot\beta))\widehat{\beta} \dagger \widehat{t}(Q\,\widehat{\epsilon}\;[t]\;\widehat{u}\,\langle u.\sigma\rangle)) \\ &\to (\widehat{x}\,\langle x.\alpha\rangle\,\widehat{\alpha}\cdot\beta)\widehat{\beta} \dagger \widehat{t}(Q\,\widehat{\epsilon}\;[t]\;\widehat{u}\,\langle u.\sigma\rangle)) \\ &\to (Q\widehat{\epsilon} \dagger \widehat{x}\langle x.\alpha\rangle)\widehat{\alpha} \dagger \widehat{u}\langle u.\sigma\rangle \\ &\to (Q\{\alpha/\epsilon\})\widehat{\alpha} \dagger \widehat{u}\langle u.\sigma\rangle \\ &\to Q\{\alpha/\epsilon\}\{\sigma/\alpha\} \end{aligned}$$

D Encoding $\boldsymbol{\lambda}$-calculus; Illustrations and Examples

We give several instances of encoding, as an illustration of the general definition:

$$\begin{array}{ll}
[\![\lambda x.M]\!]_\alpha & = \widehat{x}\,[\![M]\!]_\beta\,\widehat{\beta}\cdot\alpha,\ \text{when } x\in fv(M)\\
[\![MN]\!]_\alpha & = [\![M]\!]_\gamma\widehat{\gamma}\ \dagger\ \widehat{x}([\![N]\!]_\beta\,\widehat{\beta}\ [x]\ \widehat{y}\,\langle y.\alpha\rangle),\ \text{when } fv(M)\cap fv(N)=\emptyset\\
[\![MN]\!]_\alpha & = z{<}^{\widehat{z_1}}_{\widehat{z_2}}\langle(\mathbb{R}^z_{z_1}([\![M]\!]_\gamma))\widehat{\gamma}\ \dagger\ \widehat{x}((\mathbb{R}^z_{z_2}([\![N]\!]_\beta))\,\widehat{\beta}\ [x]\ \widehat{y}\,\langle y.\alpha\rangle)],\ \text{when } fv(M)\cap fv(N)=z
\end{array}$$

Example 1. We give several examples of the encoding of non-linear λ-terms:

$$\begin{array}{lll}
[\![zz]\!]_\alpha & \triangleq & z{<}^{\widehat{z_1}}_{\widehat{z_2}}\langle[\![z]\!]_\gamma\{z_1/z\}\widehat{\gamma}\ \dagger\ \widehat{x}[\![z]\!]_\beta\{z_2/z\}\,\widehat{\beta}\ [x]\ \widehat{y}\,\langle y.\alpha\rangle]\\
 & \triangleq & z{<}^{\widehat{z_1}}_{\widehat{z_2}}\langle[\![z_1]\!]_\gamma\widehat{\gamma}\ \dagger\ \widehat{x}([\![z_2]\!]_\beta\,\widehat{\beta}\ [x]\ \widehat{y}\,\langle y.\alpha\rangle)]\\
 & \triangleq & z{<}^{\widehat{z_1}}_{\widehat{z_2}}\langle\langle z_1.\gamma\rangle\widehat{\gamma}\ \dagger\ \widehat{x}(\langle z_2.\beta\rangle\,\widehat{\beta}\ [x]\ \widehat{y}\,\langle y.\alpha\rangle)]\\
 & \xrightarrow{*\mathcal{X}} & z{<}^{\widehat{z_1}}_{\widehat{z_2}}\langle\langle z_2.\beta\rangle\,\widehat{\beta}\ [z_1]\ \widehat{y}\,\langle y.\alpha\rangle]\\
 & & \\
[\![\lambda t.u]\!]_\alpha & \triangleq & \widehat{t}\,(t\circledcirc[\![u]\!]_\beta)\,\widehat{\beta}\cdot\alpha\\
 & \triangleq & \widehat{t}\,(t\odot\langle u.\beta\rangle)\,\widehat{\beta}\cdot\alpha\\
 & & \\
[\![(\lambda t.u)v]\!]_\alpha & \triangleq & [\![\lambda t.u]\!]_\gamma\widehat{\gamma}\ \dagger\ \widehat{x}([\![v]\!]_\beta\,\widehat{\beta}\ [x]\ \widehat{y}\,\langle y.\alpha\rangle)\\
 & \triangleq & \widehat{t}\,(t\odot\langle u.\delta\rangle)\,\widehat{\delta}\cdot\gamma\widehat{\gamma}\ \dagger\ \widehat{x}(\langle v.\beta\rangle\,\widehat{\beta}\ [x]\ \widehat{y}\,\langle y.\alpha\rangle)\\
 & \xrightarrow{*\mathcal{X}} & (\langle v.\beta\rangle\widehat{\beta}\dagger\widehat{t}(t\odot\langle u.\delta\rangle))\widehat{\delta}\dagger\widehat{y}\langle y.\alpha\rangle\\
 & \xrightarrow{*\mathcal{X}} & v\odot\langle u.\alpha\rangle\\
 & \triangleq & v\odot[\![u]\!]_\alpha
\end{array}$$