CLASSICAL SEQUENTS AND COMPUTATION : AN OVERVIEW

To Henk Barendregt, in honour of his 60th birthday

STEFFEN VAN BAKEL

Department of Computing, Imperial College London,
180 Queen's Gate, London SW7 2BZ, UK
e-mail address: svb@doc.ic.ac.uk

ABSTRACT. This paper presents a short overview of some of the results achieved for the calculus $\mathcal{X}$, which is based on Gentzen's LK. It presents the calculus, its suitability for encoding the λ-calculus and the $\lambda\mu$-calculus, as well as a type-preserving encoding of $\mathcal{X}$ into the π-calculus.

INTRODUCTION

We discuss the calculus $\mathcal{X}$ (a first version of this calculus was proposed in [24, 23, 22]; the implicative fragment of $\mathcal{X}$ was studied in [5]), which has the Curry-Howard property for proofs of Gentzen's sequent calculus LK [11], and briefly discuss interpretations of various calculi as well as an encoding into the π-calculus [17], that all respect *cut*-elimination as well as assignable types.

LK is a logical system in which the rules only introduce connectives (but on either side of a sequent), in contrast to *natural deduction* (also introduced in [11]) which uses rules that introduce or eliminate connectives in the logical formulae. Natural deduction derives statements with a single conclusion, whereas LK allows for multiple conclusions, deriving sequents of the form $A_1, \ldots, A_n \vdash B_1, \ldots, B_m$. $\mathcal{X}$ achieves the Curry-Howard isomorphism for proofs in LK by constructing witnesses, called *nets*, for derivable sequents; this is achieved without using application.

Nets in $\mathcal{X}$ have multiple named inputs and multiple named outputs, that are collectively called *connectors*. Names on the left can be seen as inputs to the net, and names to the right as outputs. Similar to calculi like $\lambda\mu$ [18] and $\overline{\lambda}\mu\tilde{\mu}$ [10], there are two kinds of names in $\mathcal{X}$: *sockets* (inputs, with Roman names) that are attached to formulae in the left context, and *plugs* (outputs, with Greek names) for those in the right context. Plugs and sockets correspond, respectively, to *variables* and *co-variables* in [25], or, alternatively, to Parigot's λ- and μ-variables [18] (see also [10]).

In the construction of the witness, when in applying a rule a premise or conclusion disappears from the sequent, the corresponding name gets bound in the net that is constructed, and when a premise or conclusion gets created, a different free (often new) name is associated to it. For example, in the creation of the net for right-introduction of the arrow via rule (exp):

$$\frac{P \;\vdots\; \Gamma, x{:}A \vdash \alpha{:}B, \Delta}{\widehat{x}P\widehat{\alpha}\cdot\beta \;\vdots\; \Gamma \vdash \beta{:}A{\rightarrow}B\Delta}$$

the input x and the output α are bound, and β is free. This case is interesting in that it highlights a special feature of $\mathcal{X}$, not found in other calculi. In (applicative) calculi related to natural deduction, like the λ-calculus [8], only inputs are named, and the linking to a term that will be inserted is done via λ-abstraction and application. The output (i.e. result) on the other hand is anonymous; where a term 'moves to' carries a name via a variable that acts as a pointer to the positions where the term is to be inserted, but where it comes from is not mentioned, since it is implicit. Since a term P can have many inputs and outputs, it is unsound to consider P a function; however, fixing *one* input x and *one* output α, we can see P as a function 'from x to α'. We make this limited view of P available via the output β, thereby *exporting* 'P is a function from x to α'; notice that the types given to the connectors conform to this point of view.

Implicative LK has four rules: *axiom*, *left introduction* of the arrow, *right introduction*, and *cut*.

$$(Ax): \frac{}{\Gamma, A \vdash A, \Delta} \quad (\Rightarrow L): \frac{\Gamma \vdash A, \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A \Rightarrow B \vdash \Delta} \quad (\Rightarrow R): \frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \Rightarrow B, \Delta} \quad (cut): \frac{\Gamma \vdash A, \Delta \quad \Gamma, A \vdash \Delta}{\Gamma \vdash \Delta}$$

Since LK has only introduction rules, the only way to eliminate a connective is to eliminate the whole formula in which it appears via an application of the (cut)-rule. Gentzen defined a *cut-elimination procedure* that eliminates all applications of the (cut)-rule from a proof of a sequent (via an innermost reduction strategy), generating a proof in *normal form* for the same sequent.

$\mathcal{X}$ is a true term rewriting language that distinguishes itself from other, more common programming paradigms in that it is a language of *connecting nets*, rather than a language based on application and substitution, with variables that can be replaced by entire terms. Reduction in $\mathcal{X}$ is expressed via a set of rewrite rules that represent *cut*-elimination, eventually leading to *renaming* of connectors, and gives computational meaning to classical (sequent) proof reduction. It is well known that *cut*-elimination in LK is not confluent, and consequently, neither is reduction in $\mathcal{X}$. The intuition behind reduction is: the cut $P\widehat{\alpha} \dagger \widehat{x}Q$ expresses the intention to connect all αs in P and xs in Q, which reduction will realise by either connecting all αs to all xs (if x does not exist in Q, P will disappear), or all xs to all αs (if α does not exist in P, Q will disappear). So, when P does not contain α and Q does not contain x, reducing $P\widehat{\alpha} \dagger \widehat{x}Q$ leads to both P and Q, two different nets.

This paper presents the calculus $\mathcal{X}$, together with some of its main results; these first appeared in various papers, of which we mention [4, 5, 6, 21, 7, 2]. In [4, 5] $\mathcal{X}$ is presented and, respectively, the embedding of the λ-calculus, $\lambda\mathbf{x}$ [9], the $\lambda\mu$-calculus and the $\overline{\lambda}\mu\tilde{\mu}$-calculus is studied. [2] studies mapping $\mathcal{X}$ into $\lambda\mu$. [21] studies the relation between $\mathcal{X}$ enriched with quantification and System F and ML. [6, 7] discuss implementation issues. [3] studies the encoding of $\mathcal{X}$ into the π-calculus. The intention of this paper is to be an overview, as a showcase of the expressive power of $\mathcal{X}$; for details of proofs, I would like to refer the reader to the papers mentioned above.

1. The calculus $\mathcal{X}$

In this section we will give the definition of the $\mathcal{X}$-calculus. $\mathcal{X}$ features two separate categories of 'connectors', *plugs* and *sockets*, that act as input and output channels. A consequence of the fact that the origin of $\mathcal{X}$ is a sequent calculus is that has no notion of substitution or application.

Definition 1.1 (Syntax)**.** The nets of the $\mathcal{X}$-calculus are defined by the following grammar:

$$\underset{}{P, Q} ::= \underset{\textit{capsule}}{\langle x \cdot \alpha \rangle} \mid \underset{\textit{export}}{\widehat{y} P \widehat{\beta} \cdot \alpha} \mid \underset{\textit{import}}{P\widehat{\beta}\,[y]\,\widehat{x}Q} \mid \underset{\textit{cut}}{P\widehat{\alpha} \dagger \widehat{x}Q}$$

where x, y range over *sockets*, α, β over *plugs* (together called *connectors*). The symbol '$\widehat{\ }$' is a *binder*; the definition of *free* (or *bound*) or connector is as usual; we write $\mathit{fs}(P)$, $\mathit{fp}(P)$ or $\mathit{fc}(P)$.

The origin of $\mathcal{X}$ can be found in Urban's PhD thesis; the calculus defined there is the same in spirit, but very different in presentation. Urban uses the first letters of the Latin alphabet for plugs, and the last for sockets; he expresses input and output behaviour by using a π-calculus-like notation, putting sockets between parentheses and plugs between angles.

$\mathcal{X}$:	$\langle x{\cdot}\alpha\rangle$	$\widehat{x}P\widehat{\beta}{\cdot}\alpha$	$P\widehat{\alpha}\,[x]\,\widehat{y}Q$	$P\widehat{\alpha} \dagger \widehat{x}Q$
Urban :	$\mathit{Ax}(x,a)$	$\mathit{ImpR}((x)\langle b\rangle P, a)$	$\mathit{ImpL}(\langle a\rangle P, (y)Q, x)$	$\mathit{Cut}(\langle a\rangle P, (x)Q)$

Notice that Urban's notation is pre-fix, which distorts the notion of 'flow' $\mathcal{X}$ expresses; also, in $\mathit{ImpL}(\langle a\rangle P, (y)Q, x)$, it is not clear that x will interface between P and Q.

Reduction strongly depends on the notion of an *introduced connector*.

Definition 1.2 (Introduction)**.** $(P \textit{ introduces } x)$: $P = \langle x{\cdot}\alpha\rangle$ or $P = Q\widehat{\beta}\,[x]\,\widehat{y}R$ with $x \notin \mathit{fs}(Q,R)$.
$(P \textit{ introduces } \alpha)$: $P = \langle x{\cdot}\alpha\rangle$ or $P = \widehat{x}Q\widehat{\beta}{\cdot}\alpha$ and $\alpha \notin \mathit{fp}(Q)$.

The principal reduction rules are:

Definition 1.3 (Logical rules)**.** Let α and x be introduced in, respectively, the left- and right-hand side of the main cuts below.

$$
\begin{aligned}
(\mathit{cap}) &: \quad \langle y{\cdot}\alpha\rangle\widehat{\alpha} \dagger \widehat{x}\langle x{\cdot}\beta\rangle \;\to\; \langle y{\cdot}\beta\rangle \\
(\mathit{exp}) &: \quad (\widehat{y}P\widehat{\beta}{\cdot}\alpha)\widehat{\alpha} \dagger \widehat{x}\langle x{\cdot}\gamma\rangle \;\to\; \widehat{y}P\widehat{\beta}{\cdot}\gamma \\
(\mathit{imp}) &: \quad \langle y{\cdot}\alpha\rangle\widehat{\alpha} \dagger \widehat{x}(Q\widehat{\beta}\,[x]\,\widehat{z}R) \;\to\; Q\widehat{\beta}\,[y]\,\widehat{z}R \\
(\mathit{exp\text{-}imp}) &: \quad (\widehat{y}P\widehat{\beta}{\cdot}\alpha)\widehat{\alpha} \dagger \widehat{x}(Q\widehat{\gamma}\,[x]\,\widehat{z}R) \;\to\; \begin{cases} Q\widehat{\gamma} \dagger \widehat{y}(P\widehat{\beta} \dagger \widehat{z}R) \\ (Q\widehat{\gamma} \dagger \widehat{y}P)\widehat{\beta} \dagger \widehat{z}R \end{cases}
\end{aligned}
$$

The first three logical rules above specify a renaming procedure, whereas the last rule specifies the basic computational step: it links the export of a function, available on the plug α, to an adjacent import via the socket x. The effect of the reduction will be that the exported function is placed in-between the two sub-terms of the import, acting as interface.

In $\mathcal{X}$ there are in fact two kinds of reduction, the one above, and the one which defines how to reduce a cut when one of its sub-nets *does not* introduce a connector mentioned in the cut. This will involve moving the cut inwards, towards a position where the connector *is* introduced. In case both connectors are not introduced, this search can start in either direction, (either to the left or to the right), indicated by the tilting of the dagger.

Definition 1.4 (Active cuts)**.** The syntax is extended with two *flagged* or *active* cuts:

$$P \;::=\; \ldots \mid P_1\widehat{\alpha} \not\leftarrow \widehat{x}P_2 \mid P_1\widehat{\alpha} \not\rightarrow \widehat{x}P_2$$

We define two *cut-activation* rules.

$$
\begin{aligned}
(a\not\leftarrow) &: \; P\widehat{\alpha} \dagger \widehat{x}Q \;\to\; P\widehat{\alpha} \not\leftarrow \widehat{x}Q \;\; \textit{if } P \textit{ does not introduce } \alpha \\
(\not\rightarrow a) &: \; P\widehat{\alpha} \dagger \widehat{x}Q \;\to\; P\widehat{\alpha} \not\rightarrow \widehat{x}Q \;\; \textit{if } Q \textit{ does not introduce } x
\end{aligned}
$$

The next rules define how to move an activated dagger inwards.

Definition 1.5 (Propagation rules)**.** Left propagation:

$$\begin{array}{rrcll}
(d\not\leftarrow): & \langle y\cdot\alpha\rangle\widehat{\alpha} \not\leftarrow \widehat{x}P & \to & \langle y\cdot\alpha\rangle\widehat{\alpha} \dagger \widehat{x}P & \\
(cap\not\leftarrow): & \langle y\cdot\beta\rangle\widehat{\alpha} \not\leftarrow \widehat{x}P & \to & \langle y\cdot\beta\rangle, & \beta \neq \alpha \\
(\textit{exp-outs}\not\leftarrow): & (\widehat{y}Q\widehat{\beta}\cdot\alpha)\widehat{\alpha} \not\leftarrow \widehat{x}P & \to & (\widehat{y}(Q\widehat{\alpha} \not\leftarrow \widehat{x}P)\widehat{\beta}\cdot\gamma)\widehat{\gamma} \dagger \widehat{x}P, & \gamma \textit{ fresh} \\
(\textit{exp-ins}\not\leftarrow): & (\widehat{y}Q\widehat{\beta}\cdot\gamma)\widehat{\alpha} \not\leftarrow \widehat{x}P & \to & \widehat{y}(Q\widehat{\alpha} \not\leftarrow \widehat{x}P)\widehat{\beta}\cdot\gamma, & \gamma \neq \alpha \\
(imp\not\leftarrow): & (Q\widehat{\beta}\,[z]\,\widehat{y}R)\widehat{\alpha} \not\leftarrow \widehat{x}P & \to & (Q\widehat{\alpha} \not\leftarrow \widehat{x}P)\widehat{\beta}\,[z]\,\widehat{y}(R\widehat{\alpha} \not\leftarrow \widehat{x}P) & \\
(cut\not\leftarrow): & (Q\widehat{\beta} \dagger \widehat{y}R)\widehat{\alpha} \not\leftarrow \widehat{x}P & \to & (Q\widehat{\alpha} \not\leftarrow \widehat{x}P)\widehat{\beta} \dagger \widehat{y}(R\widehat{\alpha} \not\leftarrow \widehat{x}P) &
\end{array}$$

Right propagation:

$$\begin{array}{rrcll}
(\not\rightarrow d): & P\widehat{\alpha} \not\rightarrow \widehat{x}\langle x\cdot\beta\rangle & \to & P\widehat{\alpha} \dagger \widehat{x}\langle x\cdot\beta\rangle & \\
(\not\rightarrow cap): & P\widehat{\alpha} \not\rightarrow \widehat{x}\langle y\cdot\beta\rangle & \to & \langle y\cdot\beta\rangle, & y \neq x \\
(\not\rightarrow exp): & P\widehat{\alpha} \not\rightarrow \widehat{x}(\widehat{y}Q\widehat{\beta}\cdot\gamma) & \to & \widehat{y}(P\widehat{\alpha} \not\rightarrow \widehat{x}Q)\widehat{\beta}\cdot\gamma & \\
(\not\rightarrow \textit{imp-outs}): & P\widehat{\alpha} \not\rightarrow \widehat{x}(Q\widehat{\beta}\,[x]\,\widehat{y}R) & \to & P\widehat{\alpha} \dagger \widehat{z}((P\widehat{\alpha} \not\rightarrow \widehat{x}Q)\widehat{\beta}\,[z]\,\widehat{y}(P\widehat{\alpha} \not\rightarrow \widehat{x}R)), & z \textit{ fresh} \\
(\not\rightarrow \textit{imp-ins}): & P\widehat{\alpha} \not\rightarrow \widehat{x}(Q\widehat{\beta}\,[z]\,\widehat{y}R) & \to & (P\widehat{\alpha} \not\rightarrow \widehat{x}Q)\widehat{\beta}\,[z]\,\widehat{y}(P\widehat{\alpha} \not\rightarrow \widehat{x}R), & z \neq x \\
(\not\rightarrow cut): & P\widehat{\alpha} \not\rightarrow \widehat{x}(Q\widehat{\beta} \dagger \widehat{y}R) & \to & (P\widehat{\alpha} \not\rightarrow \widehat{x}Q)\widehat{\beta} \dagger \widehat{y}(P\widehat{\alpha} \not\rightarrow \widehat{x}R) &
\end{array}$$

We write $\to$ for the (reflexive, transitive, compatible) reduction relation generated by the logical, propagation and activation rules, and write $P \downarrow Q$ to express that P and Q share a reduct, i.e. when there exists an R such that $P \to R$ and $Q \to R$.

As mentioned above, the reduction relation $\to$ is not confluent; this comes from the critical pair that activates a cut $P\widehat{\alpha} \dagger \widehat{x}Q$ in two ways, and the critical pair that is the rule $(\textit{exp-imp})$.

In short, reduction brings all cuts down via propagation to logical cuts or to elimination cuts that are cutting towards a capsule that does not contain the relevant connector, as in $P\widehat{\alpha} \not\rightarrow \widehat{x}\langle z\cdot\beta\rangle$ or $\langle z\cdot\beta\rangle\widehat{\alpha} \not\leftarrow \widehat{x}P$; performing the elimination cuts, via $(\not\rightarrow cap)$ or $(cap\not\leftarrow)$, will remove the term P.

Two sub-reduction systems are introduced which explicitly favour one kind of activation whenever the above critical pair occurs:

Definition 1.6. We define Call-By-Name and Call-By-Value reduction by:

- If a cut can be activated in two ways, the CBV strategy only allows to activate it via $(a\not\leftarrow)$; we write $P \to_{\mathrm{V}} Q$ in that case. This can be obtained by replacing rule $(\not\rightarrow a)$ by:

  $(\not\rightarrow a)$: $P\widehat{\alpha} \dagger \widehat{x}Q \to P\widehat{\alpha} \not\rightarrow \widehat{x}Q$, *if P introduces α and Q does not introduce x.*

- The CBN strategy can only activate such a cut via $(\not\rightarrow a)$, and we write $P \to_{\mathrm{N}} Q$. Likewise, we can obtain this by replacing rule $(a\not\leftarrow)$ by:

  $(a\not\leftarrow)$: $P\widehat{\alpha} \dagger \widehat{x}Q \to P\widehat{\alpha} \not\leftarrow \widehat{x}Q$, *if P does not introduce α and Q introduces x.*

- We split the two variants of $(\textit{exp-imp})$ over the two notions of reduction: for CBV we take:

  $$(\widehat{y}P\widehat{\beta}\cdot\alpha)\widehat{\alpha} \dagger \widehat{x}(Q\widehat{\gamma}\,[x]\,\widehat{z}R) \to Q\widehat{\gamma} \dagger \widehat{y}(P\widehat{\beta} \dagger \widehat{z}R)$$

  and for CBN:

  $$(\widehat{y}P\widehat{\beta}\cdot\alpha)\widehat{\alpha} \dagger \widehat{x}(Q\widehat{\gamma}\,[x]\,\widehat{z}R) \to (Q\widehat{\gamma} \dagger \widehat{y}P)\widehat{\beta} \dagger \widehat{z}R$$

We so obtain two notions of reduction that are confluent: all rules are left-linear and non-overlapping.

2. EXPLICIT α-CONVERSION

Normally, renaming is an essential part of *α-conversion*, the process of renaming bound objects in a language to avoid clashes during computation. The most familiar context in which this occurs is the λ-calculus, where, when reducing $(\lambda xy.xy)(\lambda xy.xy)$, α-conversion is essential.

While building an efficient implementation of an interpreter for $\mathcal{X}$, it was noted that the α-conversion can in $\mathcal{X}$ be dealt with *at the level of the language itself*, unlike for the λ-calculus. There proved to be several ways to do this in $\mathcal{X}$: three solutions to the problem of α-conversion are proposed in [6, 7], that are compared in terms of efficiency. The first uses a *lazy-copying* strategy to avoid sharing of bound connectors; the second *enforces* Barendregt's convention, by renaming bound connectors when nesting is created; the third avoids *capture* of names, but allows breaches of Barendregt's convention. α-conversion is necessary, for example, in rule $(\textit{exp-imp})$

$$(\widehat{y}P\widehat{\beta}\cdot\alpha)\widehat{\alpha}\dagger\widehat{x}(Q\widehat{\gamma}\,[x]\,\widehat{z}R)\;\rightarrow\;\begin{cases}Q\widehat{\gamma}\dagger\widehat{y}(P\widehat{\beta}\dagger\widehat{z}R)\\(Q\widehat{\gamma}\dagger\widehat{y}P)\widehat{\beta}\dagger\widehat{z}R\end{cases}$$

A conflict with Barendregt's convention is generated in this rule by the fact that perhaps $\beta=\gamma$ or $y=z$. Or, when striving for capture avoidance, it might be that y occurs free in R, or β in Q. In either case, these connectors need to be renamed; one of the great plus points of $\mathcal{X}$ is that this can be done *within* the language itself. For example, to accurately deal with α-conversion for the case of capture-avoidance, the rule $(\textit{exp-imp})$ needs to be replaced by (α, x are introduced, v,δ are fresh):

$$(\widehat{y}P\widehat{\beta}\cdot\alpha)\widehat{\alpha}\dagger\widehat{x}(Q\widehat{\gamma}\,[x]\,\widehat{z}R)\;\rightarrow\;\begin{cases}Q\widehat{\gamma}\dagger\widehat{y}(P\widehat{\beta}\dagger\widehat{z}R), & y\notin fs(R)\\Q\widehat{\gamma}\dagger\widehat{v}((\langle v\cdot\delta\rangle\widehat{\delta}\nwarrow\widehat{y}P)\widehat{\beta}\dagger\widehat{z}R), & y\in fs(R)\\(Q\widehat{\gamma}\dagger\widehat{y}P)\widehat{\beta}\dagger\widehat{z}R, & \beta\notin fp(Q)\\(Q\widehat{\gamma}\dagger\widehat{y}(P\widehat{\beta}\nearrow\widehat{v}\langle v\cdot\delta\rangle))\widehat{\delta}\dagger\widehat{z}R, & \beta\in fp(Q)\end{cases}$$

Almost all propagation rules (but for $(d\nearrow)$, $(cap\nearrow)$, $(\nwarrow d)$, and $(\nwarrow cap)$) need dealing with as well.

3. TYPING FOR $\mathcal{X}$: FROM LK TO $\mathcal{X}$

The notion of type assignment on $\mathcal{X}$ that we present in this section is the basic implicative system for Classical Logic (Gentzen's system LK) as described above. When building witnesses for proofs, propositions receive names; those that appear in the left part of a sequent receive names like x,y,z, etc, and those that appear in the right part of a sequent receive names like α,β,γ, etc. When in applying a rule a formula disappears from the sequent, the corresponding connector will get bound in the net that is constructed, and when a formula gets created, a different connector will be associated to it.

Definition 3.1 (Types and Contexts)**.** (1) The set of types is defined by the grammar:

$$A,B\;::=\;\varphi\mid A\rightarrow B.$$

where φ is a basic type of which there are infinitely many.

(2) A *context of sockets* Γ is a finite set of *statements* $x{:}A$ with distinct *subjects* (x). We write Γ_1,Γ_2 for the union of Γ_1 and Γ_2, provided Γ_1 and Γ_2 are compatible (if Γ_1 contains $x{:}A_1$ and Γ_2 contains $x{:}A_2$ then $A_1=A_2$), and write $\Gamma,x{:}A$ for $\Gamma,\{x{:}A\}$.

(3) Contexts of *plugs* Δ are defined in a similar way.

(Simple) type assignment for $\mathcal{X}$ is defined using the following sequent calculus: the Curry-Howard isomorphism for Implicative LK is easily achieved by erasure.

Definition 3.2 (Typing for $\mathcal{X}$)**.** (1) *Type judgements* are expressed as $P :\cdot\ \Gamma \vdash \Delta$, where Γ is a context of *sockets* and Δ is a context of *plugs*, and P is a net, the *witness* of this judgement.
(2) *Type assignment for* $\mathcal{X}$ is defined by the following rules:

$$(cap):\ \frac{}{\langle y{\cdot}\alpha\rangle :\cdot\ \Gamma, y{:}A \vdash \alpha : A, \Delta} \qquad (imp):\ \frac{P :\cdot\ \Gamma \vdash \alpha : A, \Delta \quad Q :\cdot\ \Gamma, x{:}B \vdash \Delta}{P\widehat{\alpha}\,[y]\,\widehat{x}Q :\cdot\ \Gamma, y{:}A{\to}B \vdash \Delta}$$

$$(exp):\ \frac{P :\cdot\ \Gamma, x{:}A \vdash \alpha : B, \Delta}{\widehat{x}P\widehat{\alpha}{\cdot}\beta :\cdot\ \Gamma \vdash \beta{:}A{\to}B, \Delta} \qquad (cut):\ \frac{P :\cdot\ \Gamma \vdash \alpha : A, \Delta \quad Q :\cdot\ \Gamma, x{:}A \vdash \Delta}{P\widehat{\alpha}\dagger\widehat{x}Q :\cdot\ \Gamma \vdash \Delta}$$

We write $P :\cdot\ \Gamma \vdash \Delta$ if there exists a derivation that has this judgement in the bottom line.

Notice that the system does not deal with contraction or weakening; in fact, that's the reason the propagation rules are present in $\mathcal{X}$. For a *linear* version of $\mathcal{X}$ that has explicit contraction and weakening, see [16]. Notice that Γ and Δ carry the types of the free connectors in P, as unordered sets. There is no notion of type for P itself, instead the derivable statement shows how P is connectable.

Example 3.3 (An inhabited proof of Peirce's Law)**.**

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\langle y{\cdot}\delta\rangle :\cdot\ y{:}A \vdash \delta{:}A, \eta{:}B}\,(cap)}{\widehat{y}\langle y{\cdot}\delta\rangle\widehat{\eta}{\cdot}\alpha :\cdot\ \vdash \alpha{:}A{\to}B, \delta{:}A}\,(exp) \qquad \dfrac{}{\langle w{\cdot}\delta\rangle :\cdot\ w{:}A \vdash \delta{:}A}\,(cap)}{(\widehat{y}\langle y{\cdot}\delta\rangle\widehat{\eta}{\cdot}\alpha)\widehat{\alpha}\,[z]\,\widehat{w}\langle w{\cdot}\delta\rangle :\cdot\ z{:}(A{\to}B){\to}A \vdash \delta{:}A}\,(imp)}{\widehat{z}((\widehat{y}\langle y{\cdot}\delta\rangle\widehat{\eta}{\cdot}\alpha)\widehat{\alpha}\,[z]\,\widehat{w}\langle w{\cdot}\delta\rangle)\widehat{\delta}{\cdot}\gamma :\cdot\ \vdash \gamma{:}((A{\to}B){\to}A){\to}A}\,(exp)}$$

The following soundness result is proven in [5]:

Theorem 3.4 (Witness reduction)**.** If $P :\cdot\ \Gamma \vdash \Delta$, and $P \to Q$, then $Q :\cdot\ \Gamma \vdash \Delta$.

4. The relation with the Lambda Calculus

In this section, we will briefly highlight the relation between the λ-calculus and $\mathcal{X}$, from [5]. We assume the reader to be familiar with λ-calculus; the direct encoding of λ-terms into $\mathcal{X}$ is defined by:

Definition 4.1 (Interpreting the λ-calculus)**.** The interpretation of lambda terms into $\mathcal{X}$ in the context α, $\llbracket M \rrbracket^{\lambda}_{\alpha}$, is defined by:

$$\begin{array}{rcll} \llbracket x \rrbracket^{\lambda}_{\alpha} & \triangleq & \langle x{\cdot}\alpha\rangle & \\ \llbracket \lambda x.M \rrbracket^{\lambda}_{\alpha} & \triangleq & \widehat{x}\llbracket M \rrbracket^{\lambda}_{\beta}\widehat{\beta}{\cdot}\alpha, & \beta \textit{ fresh} \\ \llbracket MN \rrbracket^{\lambda}_{\alpha} & \triangleq & \llbracket M \rrbracket^{\lambda}_{\gamma}\widehat{\gamma}\dagger\widehat{x}(\llbracket N \rrbracket^{\lambda}_{\beta}\widehat{\beta}\,[x]\,\widehat{y}\langle y{\cdot}\alpha\rangle), & x, y, \beta, \gamma \textit{ fresh} \end{array}$$

We can even represent substitution explicitly (so represent $\lambda\mathbf{x}$) via $\llbracket \cdot \rrbracket^{\lambda x}$, by adding the clause

$$\llbracket M[N/x] \rrbracket^{\lambda x}_{\alpha} = \llbracket N \rrbracket^{\lambda x}_{\gamma}\widehat{\gamma} \between \widehat{x}\llbracket M \rrbracket^{\lambda x}_{\alpha},\ \gamma \textit{ fresh}$$

Notice that every sub-term of $\llbracket M \rrbracket_{\alpha}$ has exactly one free plug; also, this interpretation is the standard way of encoding natural deduction in the sequent calculus.

In [4], the following relation is shown between reduction in λ-calculus and $\mathcal{X}$:

Theorem 4.2 ([4])**.** If $M \to_{\beta} N$, then $\llbracket M \rrbracket_{\alpha} \to \llbracket N \rrbracket_{\alpha}$; this is true also for CBN and CBV.

To strengthen the fact that we consider more than just those nets that represent proofs, it is straightforward to verify that $\llbracket \Delta\Delta \rrbracket^{\lambda}_{\beta} \to \llbracket \Delta\Delta \rrbracket^{\lambda}_{\beta}$.

It is worthwhile to notice that the image of Λ under the interpretation function $\llbracket \cdot \rrbracket^{\lambda}_{\alpha}$ does not generate a confluent sub-calculus, since we can show both

$$\llbracket (\lambda x.xx)(yy) \rrbracket^{\lambda}_{\alpha} \;\to_{\mathrm{V}}\; \langle y{\cdot}\sigma\rangle\widehat{\sigma}\,[y]\,\widehat{z}(\langle z{\cdot}\tau\rangle\widehat{\tau}\,[z]\,\widehat{u}\langle u{\cdot}\alpha\rangle)$$

and also

$$\llbracket (\lambda x.xx)(yy) \rrbracket^{\lambda}_{\alpha} \;\to_{\mathrm{N}}\; \langle y{\cdot}\sigma\rangle\widehat{\sigma}\,[y]\,\widehat{z}((\langle y{\cdot}\sigma\rangle\widehat{\sigma}\,[y]\,\widehat{z}\langle z{\cdot}\tau\rangle)\widehat{\tau}\,[z]\,\widehat{u}\langle u{\cdot}\alpha\rangle)$$

Notice that both reductions return normal forms, and that these are different, so $\llbracket (\lambda x.xx)(yy) \rrbracket^{\lambda}_{\alpha}$ has *two* normal forms (see also [4]); this in fact corresponds to the fact that $(\lambda x.xx)(yy)$ has different normal forms with respect to CBN and CBV reduction.

5. The relation with $\lambda\mu$

In this section we will briefly discuss the result regarding the relation between $\lambda\mu$ and $\mathcal{X}$, as previously presented in [5] and [2]. Parigot's $\lambda\mu$ is a proof-term syntax for classical logic, different in approach from $\mathcal{X}$ in that it is expressed in the setting of Natural Deduction. The typing system of $\lambda\mu$ is isomorphic to the multi-conclusion logical system; as $\mathcal{X}$, it uses two disjoint sets of variables (Roman letters and Greek letters). The sequents typing terms are of the form $\Gamma \vdash A \mid \Delta$, marking the conclusion A as *active*.

Definition 5.1 ($\lambda\mu$ (cf [19, 12])**.** The terms of $\lambda\mu$ are defined by the grammar:

$$M, N \;::=\; x \mid \lambda x.M \mid MN \mid \mu\beta.[\alpha]M$$

Reduction on $\lambda\mu$-terms is defined as the compatible closure of the rules:

$$\begin{array}{rrcl}
\textit{logical}\ (\beta): & (\lambda x.M)N & \to & M[N/x] \\
\textit{structural}\ (\mu): & (\mu\alpha.[\beta]M)N & \to & \mu\gamma.([\beta]M)[N{\cdot}\gamma/\alpha] \\
\textit{renaming}: & \mu\alpha.[\beta](\mu\gamma.[\delta]M) & \to & \mu\alpha.[\delta]M[\beta/\gamma] \\
\textit{erasing}: & \mu\alpha.[\alpha]M & \to & M, \textit{ if } \alpha \textit{ does not occur in } M.
\end{array}$$

where $M[N{\cdot}\gamma/\alpha]$ stands for the term obtained from M in which every (pseudo) sub-term of the form $[\alpha]M'$ is substituted by $[\gamma](M'N)$ (γ is a fresh variable).

The typing rules for $\lambda\mu$ are:

$$\frac{}{\Gamma \vdash_{\lambda\mu} x : A \mid \Delta}\,(x{:}A \in \Gamma) \qquad \frac{\Gamma \vdash_{\lambda\mu} M : A{\to}B \mid \Delta \quad \Gamma \vdash_{\lambda\mu} N : A \mid \Delta}{\Gamma \vdash_{\lambda\mu} MN : B \mid \Delta}$$

$$\frac{\Gamma, x{:}A \vdash_{\lambda\mu} M : B \mid \Delta}{\Gamma \vdash_{\lambda\mu} \lambda x.M : A{\to}B \mid \Delta} \qquad \frac{\Gamma \vdash_{\lambda\mu} M : B \mid \alpha{:}A, \beta{:}B, \Delta}{\Gamma \vdash_{\lambda\mu} \mu\alpha.[\beta]M : A \mid \beta{:}B, \Delta} \qquad \frac{\Gamma \vdash_{\lambda\mu} M : B \mid \alpha{:}B, \Delta}{\Gamma \vdash_{\lambda\mu} \mu\alpha.[\alpha]M : B \mid \Delta}$$

This notion of type assignment is a natural extension of that for the λ-calculus (apart from a trailing Δ, the first three rules are exactly the same), and adds the notion that there is a *main*, or *active*, conclusion, labelled by a term of the calculus, and *alternative* conclusions, labelled by $\alpha, \beta, \ldots$.

We will now define how to interpret $\lambda\mu$ in $\mathcal{X}$.

Definition 5.2 (Interpretation of $\lambda\mu$ in $\mathcal{X}$)**.** We define $\llbracket \cdot \rrbracket^{\lambda\mu}_{\alpha}$ as $\llbracket \cdot \rrbracket^{\lambda}_{\alpha}$, by adding the alternative:

$$\llbracket \mu\delta.[\gamma]M \rrbracket^{\lambda\mu}_{\alpha} \;\triangleq\; \llbracket M \rrbracket^{\lambda\mu}_{\gamma}\widehat{\delta} \dagger\, \widehat{x}\langle x{\cdot}\alpha\rangle$$

Notice that, in the interpretation of the λ-calculus, we can only connect to the plug that corresponds to the name of the term itself, whereas for the $\lambda\mu$-calculus, we can also connect to plugs that occur inside, i.e., to named sub-terms.

The following lemma shows how μ-substitution can be expressed in $\mathcal{X}$.

Lemma 5.3. (1) $\llbracket M \rrbracket_{\delta}^{\lambda\mu} \widehat{\delta} \nearrow \widehat{x}(\llbracket N \rrbracket_{\beta}^{\lambda\mu} \widehat{\beta}\,[x]\,\widehat{y}\langle y{\cdot}\gamma\rangle) \downarrow \llbracket M[N{\cdot}\gamma/\delta]N \rrbracket_{\gamma}^{\lambda\mu}$.

(2) $\llbracket M \rrbracket_{\nu}^{\lambda\mu} \widehat{\delta} \nearrow \widehat{x}(\llbracket N \rrbracket_{\beta}^{\lambda\mu} \widehat{\beta}\,[x]\,\widehat{y}\langle y{\cdot}\gamma\rangle) \downarrow \llbracket M[N{\cdot}\gamma/\delta] \rrbracket_{\nu}^{\lambda\mu}$, if $\delta \neq \nu$.

In [5] it is shown that the encoding of $\lambda\mu$-terms is correct, as stated below; notice that, unlike for the λ-calculus, it is only shown that the interpretation is preserved modulo equivalence, not modulo reduction; a similar restriction holds for the interpretation of $\lambda\mu$ in $\overline{\lambda}\mu\tilde{\mu}$ achieved in [10][1]. We can now show that $\lambda\mu$'s reduction is preserved by our interpretation.

Theorem 5.4 (Simulation of CBN for $\lambda\mu$)**.** If $M \to_{\mathrm{N}} N$ then $\llbracket M \rrbracket_{\alpha}^{\lambda\mu} \downarrow \llbracket N \rrbracket_{\alpha}^{\lambda\mu}$.

We can also show that types are preserved by the interpretation:

Theorem 5.5. If $\Gamma \vdash_{\lambda\mu} M : A \mid \Delta$, then $\llbracket M \rrbracket_{\alpha}^{\lambda\mu} :\cdot\ \Gamma \vdash \alpha{:}A, \Delta$.

We can even go back again, and interpret $\mathcal{X}$ in $\lambda\mu$, as done in [2]. However, since reduction is confluent in $\lambda\mu$, we are forced to consider *confluent sub-reduction* systems of $\mathcal{X}$, like CBN and CBV. The type-preserving properties of these two encodings are achieved via the standard *double negation* translation, followed by *double negation elimination*.

Definition 5.6. First, let Ω be any (fixed) type and, for convenience, $\neg T \equiv T{\to}\Omega$. Also, take

$$(\mathit{force})F \triangleq \mu\tau.[\omega]F\,\lambda t.\mu!.[\tau]t \ =_{\mathrm{V}}\ F^{-} \qquad\qquad (\mathit{delay})t \triangleq \lambda f.f\,t$$

Then $\mathit{force} : \neg\neg T{\to}T$ and $\mathit{delay} : T{\to}\neg\neg T$ and $\mathit{force} \circ \mathit{delay}$ is the identity on T, for every T.

Definition 5.7. The CBN interpretation of T, $\llbracket T \rrbracket_{\mathrm{N}}^{\mu} \triangleq \neg\neg\llbracket T \rrbracket_{\mathrm{N}}^{\mu\prime\prime}$ is defined inductively by:

$$\begin{aligned} \llbracket \phi \rrbracket_{\mathrm{N}}^{\mu\prime\prime} &\triangleq \phi, \\ \llbracket A{\to}B \rrbracket_{\mathrm{N}}^{\mu\prime\prime} &\triangleq \llbracket A \rrbracket_{\mathrm{N}}^{\mu}{\to}\llbracket B \rrbracket_{\mathrm{N}}^{\mu} \end{aligned}$$

Also $\llbracket \Gamma, x{:}T \rrbracket_{\mathrm{N}}^{\mu} \triangleq \llbracket \Gamma \rrbracket_{\mathrm{N}}^{\mu}, x{:}\llbracket T \rrbracket_{\mathrm{N}}^{\mu}$.

Type recovery is possible, owing to the following result.

Lemma 5.8. For any type T, there exist $\varphi_T : \llbracket T \rrbracket_{\mathrm{N}}^{\mu}{\to}T$ and $\psi_T : T{\to}\llbracket T \rrbracket_{\mathrm{N}}^{\mu}$.

The first step of the interpretation is type-free, athough our definition aims at complying with types later; the notation $\mu!.C$ is a shortcut for $\mu\eta.C$ where η is a fresh μ-variable.

Definition 5.9 (Call by name)**.** Let

$$\begin{aligned} \llbracket \langle x{\cdot}\alpha \rangle \rrbracket_{\mathrm{N}}^{\mu\prime\prime} &\triangleq \lambda v.\mu!.[\alpha]\lambda f.f\,v \\ \llbracket P\widehat{\beta}\,[x]\,\widehat{y}Q \rrbracket_{\mathrm{N}}^{\mu\prime\prime} &\triangleq \lambda v.\mu!.[\omega](\lambda y.\mu!.\llbracket Q \rrbracket_{\mathrm{N}}^{\mu})v\,\mu\beta.\llbracket P \rrbracket_{\mathrm{N}}^{\mu} \end{aligned}$$

For P any $\mathcal{X}$-term, we define $\llbracket P \rrbracket_{\mathrm{N}}^{\mu}$ by structural induction

$$\begin{aligned} \llbracket \langle x{\cdot}\alpha \rangle \rrbracket_{\mathrm{N}}^{\mu} &\triangleq [\omega]x\,\llbracket \langle x{\cdot}\alpha \rangle \rrbracket_{\mathrm{N}}^{\mu\prime\prime} \\ \llbracket \widehat{y}P\widehat{\beta}{\cdot}\alpha \rrbracket_{\mathrm{N}}^{\mu} &\triangleq [\alpha]\lambda f.f\,\lambda y.\mu\beta.\llbracket P \rrbracket_{\mathrm{N}}^{\mu} \\ \llbracket P\widehat{\beta}\,[x]\,\widehat{y}Q \rrbracket_{\mathrm{N}}^{\mu} &\triangleq [\omega]x\,\llbracket P\widehat{\beta}\,[x]\,\widehat{y}Q \rrbracket_{\mathrm{N}}^{\mu\prime\prime} \\ \llbracket P\widehat{\alpha} \dagger \widehat{x}Q \rrbracket_{\mathrm{N}}^{\mu} \triangleq \llbracket P\widehat{\alpha} \nwarrow \widehat{x}Q \rrbracket_{\mathrm{N}}^{\mu} &\triangleq [\omega](\lambda x.\mu!.\llbracket Q \rrbracket_{\mathrm{N}}^{\mu})\mu\alpha.\llbracket P \rrbracket_{\mathrm{N}}^{\mu} \\ \llbracket P\widehat{\alpha} \nearrow \widehat{x}Q \rrbracket_{\mathrm{N}}^{\mu} &\triangleq [\omega](\mu\alpha.\llbracket P \rrbracket_{\mathrm{N}}^{\mu})\llbracket Q \rrbracket_{\mathrm{N}}^{\mu\prime\prime} \end{aligned}$$

[1] A corrected version of this paper is available from Herbelin's home page.

We can now first show that the encoding is faithful.

Theorem 5.10. For all $\mathcal{X}$-terms P, Q, if $P \to_{\mathrm{N}}{}^{*} Q$, then $\ulcorner P \urcorner^{\Omega}_{\mathrm{N}} =_{\mathrm{N}} \ulcorner Q \urcorner^{\Omega}_{\mathrm{N}}$.

The main result is that type contexts are preserved.

Theorem 5.11 (Conservation of types in CBN)**.** For any $P :\cdot \Gamma \vdash \Delta$ in $\mathcal{X}$, and type Ω there exists a $\lambda\mu$-term $\ulcorner P \urcorner^{\Omega}_{\mathrm{N}}$ such that $\Gamma \vdash_{\lambda\mu} \ulcorner P \urcorner^{\Omega}_{\mathrm{N}} : \Omega \mid \Delta$.

We can show similar results for CBV-reduction.

Definition 5.12 (Call by value)**.** For P any $\mathcal{X}$-term, we define $\ulcorner P \urcorner^{\mu}_{\mathrm{V}}$ by structural induction:

$$\begin{array}{rcl}
\ulcorner \langle x \cdot \alpha \rangle \urcorner^{\mu}_{\mathrm{V}} & \triangleq & [\alpha]\lambda f.f\ x \\
\ulcorner \widehat{y} P \widehat{\beta} \cdot \alpha \urcorner^{\mu}_{\mathrm{V}} & \triangleq & [\alpha]\lambda f.f\ \lambda y.\mu\beta.\ulcorner P \urcorner^{\mu}_{\mathrm{V}} \\
\ulcorner P \widehat{\beta}\,[x]\,\widehat{y} Q \urcorner^{\mu}_{\mathrm{V}} & \triangleq & [\omega](\mu\beta.\ulcorner P \urcorner^{\mu}_{\mathrm{V}})\lambda v.\mu!.[\omega](x\,v)\lambda y.\mu!.\ulcorner Q \urcorner^{\mu}_{\mathrm{V}} \\
\ulcorner P \widehat{\alpha} \dagger \widehat{x} Q \urcorner^{\mu}_{\mathrm{V}} \triangleq \ulcorner P \widehat{\alpha} \not\succ \widehat{x} Q \urcorner^{\mu}_{\mathrm{V}} & \triangleq & [\omega](\mu\alpha.\ulcorner P \urcorner^{\mu}_{\mathrm{V}})\lambda x.\mu!.\ulcorner Q \urcorner^{\mu}_{\mathrm{V}} \\
\ulcorner P \widehat{\alpha} \not\prec \widehat{x} Q \urcorner^{\mu}_{\mathrm{V}} & \triangleq & [\omega](\lambda x.\mu!.\ulcorner Q \urcorner^{\mu}_{\mathrm{V}})\mu\alpha.\ulcorner P \urcorner^{\mu -}_{\mathrm{V}}
\end{array}$$

The faithfulness result is:

Theorem 5.13. For all $\mathcal{X}$-terms P, P', if $P \to_{\mathrm{V}} P'$, then $\ulcorner P \urcorner^{\Omega}_{\mathrm{V}} =_{\mathrm{V}} \ulcorner P' \urcorner^{\Omega}_{\mathrm{V}}$.

Theorem 5.14 (Conservation of types in CBN)**.** For any $P :\cdot \Gamma \vdash \Delta$ in $\mathcal{X}$, and type Ω there exists a $\lambda\mu$-term $\ulcorner P \urcorner^{\Omega}_{\mathrm{V}}$ such that $\Gamma \vdash_{\lambda\mu} \ulcorner P \urcorner^{\Omega}_{\mathrm{V}} : \Omega \mid \Delta$.

6. The relation with $\overline{\lambda}\mu\tilde{\mu}$

Another proof-system has been proposed for classical sequent calculus is Curien and Herbelin's $\overline{\lambda}\mu\tilde{\mu}$-calculus. We will see that it is possible to relate this formalism to $\mathcal{X}$, in both directions.

The syntax of $\overline{\lambda}\mu\tilde{\mu}$, as presented in [10], has three different categories: commands, terms, and contexts or co-terms. Correspondingly, they are typed by three kinds of sequents: the usual sequents $\Gamma \vdash \Delta$ type commands, while the sequents typing terms (resp. contexts) are of the form $\Gamma \vdash A \mid \Delta$ (resp. $\Gamma \mid A \vdash \Delta$), marking the conclusion (resp. hypothesis) A as *active*, as in $\lambda\mu$.

Definition 6.1. The syntax of $\overline{\lambda}\mu\tilde{\mu}$'s *commands*, *terms* and *contexts* is defined by:

$$\begin{array}{rcll}
c & ::= & \langle v \,|\, e \rangle & (\textit{commands}) \\
e & ::= & \alpha \mid v \cdot e \mid \tilde{\mu} x.c & (\textit{contexts}) \\
v & ::= & x \mid \lambda x.v \mid \mu\beta.c & (\textit{terms})
\end{array}$$

Reduction in $\overline{\lambda}\mu\tilde{\mu}$ is defined by:

$$\begin{array}{rcl}
(\to): \ \langle \lambda x.v_1 \,|\, v_2 \cdot e \rangle & \to & \langle v_2 \,|\, \tilde{\mu} x.\langle v_1 \,|\, e \rangle \rangle \\
(\mu): \ \langle \mu\beta.c \,|\, e \rangle & \to & c[e/\beta] \\
(\tilde{\mu}): \ \langle v \,|\, \tilde{\mu} x.c \rangle & \to & c[v/x]
\end{array}$$

Typing for $\overline{\lambda}\mu\tilde{\mu}$ is defined by:

$$(\textit{cut}): \frac{\Gamma \vdash_{\overline{\lambda}\mu\tilde{\mu}} v : A \mid \Delta \quad \Gamma \mid e : A \vdash_{\overline{\lambda}\mu\tilde{\mu}} \Delta}{\langle v \,|\, e \rangle : \Gamma \vdash_{\overline{\lambda}\mu\tilde{\mu}} \Delta} \quad (\tilde{\mu}): \frac{c : \Gamma, x : A \vdash_{\overline{\lambda}\mu\tilde{\mu}} \Delta}{\Gamma \mid \tilde{\mu} x.c : A \vdash_{\overline{\lambda}\mu\tilde{\mu}} \Delta} \quad (\mu): \frac{c : \Gamma \vdash_{\overline{\lambda}\mu\tilde{\mu}} \alpha : A, \Delta}{\Gamma \vdash_{\overline{\lambda}\mu\tilde{\mu}} \mu\alpha.c : A \mid \Delta}$$

$$(Ax\text{-}c): \quad \frac{}{\Gamma \mid \alpha : A \vdash_{\overline{\lambda}\mu\tilde{\mu}} \alpha{:}A, \Delta} \qquad (Ax\text{-}t): \quad \frac{}{\Gamma, x{:}A \vdash_{\overline{\lambda}\mu\tilde{\mu}} x : A \mid \Delta}$$

$$(LI): \frac{\Gamma \vdash_{\overline{\lambda}\mu\tilde{\mu}} v : A \mid \Delta \quad \Gamma \mid e : B \vdash \Delta}{\Gamma \mid v \cdot e : A{\to}B \vdash_{\overline{\lambda}\mu\tilde{\mu}} \Delta} \qquad (RI): \frac{\Gamma, x{:}A \vdash_{\overline{\lambda}\mu\tilde{\mu}} v : B \mid \Delta}{\Gamma \vdash_{\overline{\lambda}\mu\tilde{\mu}} \lambda x.v : A{\to}B \mid \Delta}$$

With conventional notations about contexts, $v \cdot e$ is to be thought of as $e[[\,]\,v]$.

We see here how a term (context) is built either by introducing '$\to$' on the right-hand side (left-hand side) of a sequent, or just by activating one conclusion (hypothesis) from a sequent typing a command: $\mu\alpha.c$ is inherited from $\lambda\mu$, and $\tilde{\mu}x.c$ is to be thought as $\mathsf{let}\ x = [\,]\ \mathsf{in}\ c$. Note that the type of a context is the type that a term is expected to have in order to fill the hole, much like the import net in $\mathcal{X}$.

The system has a critical pair $\langle \mu\alpha.c_1 \mid \tilde{\mu}x.c_2 \rangle$ and applying in this case rule (μ) gives a call-by-value evaluation, whereas applying rule $(\tilde{\mu})$ gives a call-by-name evaluation. As can be expected, the system with both rules is not confluent.

We can show that there exists an obvious translation from $\mathcal{X}$ into $\overline{\lambda}\mu\tilde{\mu}$:

Definition 6.2 (Translation of $\mathcal{X}$ into $\overline{\lambda}\mu\tilde{\mu}$ [15])**.**

$$\begin{array}{llll} [\![\langle x{\cdot}\alpha \rangle]\!]^{\mathcal{X}} \triangleq \langle x \mid \alpha \rangle & & [\![P\widehat{\alpha}\,[x]\,\widehat{y}Q]\!]^{\mathcal{X}} \triangleq \langle x \mid (\mu\alpha.[\![P]\!]^{\mathcal{X}}){\cdot}(\tilde{\mu}y.[\![Q]\!]^{\mathcal{X}}) \rangle \\ [\![\widehat{x}P\widehat{\alpha}{\cdot}\beta]\!]^{\mathcal{X}} \triangleq \langle \lambda x.\mu\alpha.[\![P]\!]^{\mathcal{X}} \mid \beta \rangle & & [\![P\widehat{\alpha} \dagger \widehat{x}Q]\!]^{\mathcal{X}} \triangleq \langle \mu\alpha.[\![P]\!]^{\mathcal{X}} \mid \tilde{\mu}x.[\![Q]\!]^{\mathcal{X}} \rangle \end{array}$$

Definition 6.3 (Translation of $\overline{\lambda}\mu\tilde{\mu}$ into $\mathcal{X}$ [15])**.**

$$[\![\langle v \mid e \rangle]\!]^{\overline{\lambda}\mu\tilde{\mu}} \triangleq [\![v]\!]^{\overline{\lambda}\mu\tilde{\mu}}_{\alpha}\widehat{\alpha} \dagger \widehat{x}[\![e]\!]^{\overline{\lambda}\mu\tilde{\mu}}_{x}$$

$$\begin{array}{llll} [\![x]\!]^{\overline{\lambda}\mu\tilde{\mu}}_{\alpha} \triangleq \langle x{\cdot}\alpha \rangle & & [\![\alpha]\!]^{\overline{\lambda}\mu\tilde{\mu}}_{x} \triangleq \langle x{\cdot}\alpha \rangle \\ [\![\lambda x.v]\!]^{\overline{\lambda}\mu\tilde{\mu}}_{\alpha} \triangleq \widehat{x}[\![v]\!]^{\overline{\lambda}\mu\tilde{\mu}}_{\beta}\widehat{\beta}{\cdot}\alpha & & [\![v{\cdot}e]\!]^{\overline{\lambda}\mu\tilde{\mu}}_{x} \triangleq [\![v]\!]^{\overline{\lambda}\mu\tilde{\mu}}_{\alpha}\widehat{\alpha}\ [x]\ \widehat{y}[\![e]\!]^{\overline{\lambda}\mu\tilde{\mu}}_{y} \\ [\![\mu\beta.c]\!]^{\overline{\lambda}\mu\tilde{\mu}}_{\alpha} \triangleq [\![c]\!]^{\overline{\lambda}\mu\tilde{\mu}}\widehat{\beta} \dagger \widehat{x}\langle x{\cdot}\alpha \rangle & & [\![\tilde{\mu}y.c]\!]^{\overline{\lambda}\mu\tilde{\mu}}_{x} \triangleq \langle x{\cdot}\beta \rangle\widehat{\beta} \dagger \widehat{y}[\![c]\!]^{\overline{\lambda}\mu\tilde{\mu}} \end{array}$$

Also these interpretations respect reduction and types.

7. The Pi-calculus with pairing

In the rest of this overview we will summarise the results of [3], that studies the relation between $\mathcal{X}$ and the π-calculus. The notion of π-calculus that is considered in that paper is slightly different from other systems studied in the literature. The reason for this change lies directly in the calculus that is going to be interpreted, $\mathcal{X}$: since we are going to model sending and receiving pairs of names as interfaces for functions, we consider the π-calculus with pairing, inspired by [1].

To ease the definition of the interpretation function of circuits in $\mathcal{X}$ to processes, we deviate slightly from the normal practice, and write either Greek characters $\alpha, \beta, \upsilon, \ldots$ or Roman characters $x, y, z, \ldots$ for channel names; we use n for either a Greek or a Roman name, and '$\cdot$' for the generic variable. We also introduce a structure over names, such that not only names but also pairs of names can be sent (but not a pair of pairs). We also introduce the let-construct to deal with inputs of pairs of names that get distributed over the continuation.

Definition 7.1. Channel names and data are defined by:

$$a, b, c, d \ ::= \ x \mid \alpha \quad \textit{names} \qquad p \ ::= \ a \mid \langle a, b \rangle \quad \textit{data}$$

Notice that pairing is *not* recursive. Processes are defined by:

$$\begin{array}{lllll} P,Q ::= & 0 & \textit{Nil} & \mid a(x).P & \textit{Input} \\ \mid & P \mid Q & \textit{Composition} & \mid \overline{a}\langle p\rangle.P & \textit{Output} \\ \mid & !\,P & \textit{Replication} & \mid \textit{let}\,\langle x,y\rangle = z\ \textit{in}\ P & \textit{Let construct} \\ \mid & (\nu a)\,P & \textit{Restriction} & & \end{array}$$

We abbreviate $a(x).\,\textit{let}\,\langle y,z\rangle = x\ \textit{in}\ P$ by $a(\langle y,z\rangle).P$, and $(\nu m)\,(\nu n)\,P$ by $(\nu m,n)\,P$.

Definition 7.2 (Congruence)**.** The structural congruence is the smallest equivalence relation closed under contexts defined by the following rules:

$$\begin{array}{rclcrcl} P \mid \mathbf{0} & \equiv & P & & (\nu n)\,\mathbf{0} & \equiv & \mathbf{0} \\ P \mid Q & \equiv & Q \mid P & & (\nu m,n)\,P & \equiv & (\nu n,m)\,P \\ (P \mid Q) \mid R & \equiv & P \mid (Q \mid R) & & (\nu n)\,(P \mid Q) & \equiv & P \mid (\nu n)\,Q \quad \textit{if}\ n \notin \textit{fn}(P) \\ !\,P & \equiv & P \mid !\,P & & \textit{let}\,\langle x,y\rangle = \langle a,b\rangle\ \textit{in}\ R & \equiv & R[a/x,b/y] \end{array}$$

Definition 7.3. (1) The *reduction relation* of the π-calculus is defined by following rules:

$$\begin{array}{rl} (\textit{synchronisation}): & \overline{a}\langle b\rangle.P \mid a(x).Q \ \rightarrow_\pi \ P \mid Q[b/x] \\ (\textit{binding}): & P \rightarrow_\pi P' \ \Rightarrow \ (\nu n)\,P \rightarrow_\pi (\nu n)\,P' \\ (\textit{composition}): & P \rightarrow_\pi P' \ \Rightarrow \ P \mid Q \rightarrow_\pi P' \mid Q \\ (\textit{congruence}): & P \equiv Q \ \ \&\ \ Q \rightarrow_\pi Q' \ \ \&\ \ Q' \equiv P' \ \Rightarrow \ P \rightarrow_\pi P' \end{array}$$

(2) We write $\rightarrow_\pi^*$ for the reflexive and transitive closure of $\rightarrow_\pi$.
(3) We write $P \downarrow n$ if $P \equiv (\nu p_1 \ldots p_m)\,(\alpha.R \mid Q)$, where $\alpha = \overline{n}\langle b\rangle$ or $\alpha = n(x)$ and $n \neq p_1 \ldots p_m$ for some R, Q.
(4) We write $Q \Downarrow n$ if there exists P such that $Q \rightarrow_\pi^* P$ and $P \downarrow n$.

Notice that $\overline{a}\langle\langle b,c\rangle\rangle.P \mid a(\langle x,y\rangle).Q \ \rightarrow_\pi^* \ P \mid Q[b/x,c/y]$.

Definition 7.4. *Barbed contextual simulation* is the largest relation $\preceq_\pi$ such that $P \preceq_\pi Q$ implies:

- for each name n, if $P \downarrow n$ then $Q \Downarrow n$;
- for any context C, if $C[P] \rightarrow_\pi P'$, then for some Q', $C[Q] \rightarrow_\pi^* Q'$ and $P' \preceq_\pi Q'$.

8. Interpreting $\mathcal{X}$ into π

In this section, we define an encoding from nets in $\mathcal{X}$ onto processes in π. Since in π it is impossible to reduce under an input, as in [17, 20, 14], we cannot fully encode reduction in $\mathcal{X}$, but have to limit the notion of reduction in that reduction is not possible under an import. We show that this limited reduction in $\mathcal{X}$ is preserved by the encoding, as well as is type assignment.

As mentioned in the introduction, we add pairing to the π-calculus in order to be able to deal with arrow types. Notice that using the polyadic π-calculus would not be sufficient: since we would like the interpretation to respect reduction, in particular we need to be able to reduce the interpretation of $(\widehat{x}P\widehat{\alpha}\cdot\beta)\widehat{\beta} \dagger \widehat{z}\langle z\cdot\gamma\rangle$ to that of $\widehat{x}P\widehat{\alpha}\cdot\gamma$ (with β not free in P). So, choosing to encode the export of x and α over β as $\overline{\beta}\langle x,\alpha\rangle$ would force the interpretation of $\langle z\cdot\gamma\rangle$ to receive a pair of names. But requiring for a capsule to always deal with pairs of names is too restrictive, it is desirable to allow capsules to deal with single names as well. So, rather than moving towards the polyadic π-calculus, we opt for the following: communication will take place sending a single

item, which is either a name or a pair of names. This implies that a process sending a pair can also successfully communicate with a process not explicitly demanding to receive a pair.

Definition 8.1 (Notation)**.** In the definition below, we use '$\cdot$' for the generic variable, to separate plugs and sockets (and their interpretation) from the 'internal' variables of π. Also, although the departure point is to view Greek names for outputs and Roman names for inputs, by the very nature of the π-calculus (it is only possible to communicate using the *same* channel for in and output), in the implementation we are forced to use Greek names also for inputs, and Roman names for outputs; in fact, we need to explicitly convert '*an output sent on α is to be received as input on x*' via '$\alpha(\cdot)\overline{x}\langle\cdot\rangle$' (so α is now also an input, and x also an output channel), which for convenience is abbreviated into $\alpha{=}x$.

Definition 8.2. The interpretation of circuits is defined by:

$$\begin{aligned}
\ulcorner\langle x{\cdot}\alpha\rangle\urcorner_\pi &= x(\cdot).\,\overline{\alpha}\langle\cdot\rangle\\
\ulcorner\widehat{y}Q\widehat{\beta}{\cdot}\alpha\urcorner_\pi &= (\nu y,\beta)\,(\ulcorner Q\urcorner_\pi \mid \overline{\alpha}\langle\langle y,\beta\rangle\rangle)\\
\ulcorner P\widehat{\alpha}\,[x]\,\widehat{y}Q\urcorner_\pi &= x(\langle v,d\rangle).\,(\nu\alpha)\,(!\ulcorner P\urcorner_\pi \mid !\,\alpha{=}v) \mid (\nu y)\,(!\,d{=}y \mid !\ulcorner Q\urcorner_\pi)\\
\ulcorner P\widehat{\alpha}\dagger\widehat{x}Q\urcorner_\pi &= \ulcorner P\widehat{\alpha}\not\curlywedge\widehat{x}Q\urcorner_\pi = \ulcorner P\widehat{\alpha}\curlywedge\widehat{x}Q\urcorner_\pi = (\nu\alpha,x)\,(!\ulcorner P\urcorner_\pi \mid !\,\alpha{=}x \mid !\ulcorner Q\urcorner_\pi)
\end{aligned}$$

Notice that the interpretation of the inactive cut is the same as that of activated cuts.

The need to restrict reduction in $\mathcal{X}$, as mentioned in the beginning of this section, is clear after the definition of encoding. The alternative for the import $P\widehat{\alpha}\,[x]\,\widehat{y}Q$ (and *not*, perhaps surprisingly, the export, which corresponds to a function) creates a process that inputs a pair, over a combination of processes, including the interpretation of P and Q; therefore, all cuts that appear in either P or Q are inactive after the interpretation. Since if $P \to P'$ and $Q \to Q'$, then $P\widehat{\alpha}\,[x]\,\widehat{y}Q \to P'\widehat{\alpha}\,[x]\,\widehat{y}Q'$, this reduction cannot be mimicked by the encoding, and therefore has to be blocked.

One way to overcome this shortcoming would be to allow *equivalence* between processes in our reduction system for π, generated by *silent* actions (normally called τ-actions), which are communications between processes that are hidden from the context, as are reductions in $\mathcal{X}$ (or in the λ-calculus, for that matter). This would allow for a rule like:

$$\frac{P \to_\tau Q}{x(\cdot).\,P \equiv x(\cdot).\,Q}$$

When we use this kind of equivalence in our system, we can simulate *full* cut-elimination. Notice that, by construction of the encoding, we are actually using the *asynchronous* π-calculus as a model for *cut*-elimination.

The correctness result for the encoding essentially states that the image of the encoding in π contains some extra behaviour that can be disregarded. The precise formulation of the correctness lemma is stated below.

Lemma 8.3 (Correctness)**.** If $P \to P'$, then for some Q, $\ulcorner P\urcorner_\pi \to_\pi^* Q$ and $\ulcorner P'\urcorner_\pi \preceq_\pi Q$.

One of the main goals we aimed for with our interpretation was: if α does not occur free in P, and x does not occur free in Q, then both $\ulcorner P\widehat{\alpha}\dagger\widehat{x}Q\urcorner_\pi \to_\pi \ulcorner P\urcorner_\pi$ and $\ulcorner P\widehat{\alpha}\dagger\widehat{x}Q\urcorner_\pi \to_\pi \ulcorner Q\urcorner_\pi$. However, we have not achieved this; we can at most show that $\ulcorner P\widehat{\alpha}\dagger\widehat{x}Q\urcorner_\pi$ reduces to a process that contains $\ulcorner P\urcorner_\pi \mid \ulcorner Q\urcorner_\pi$. It is as yet not clear what this say about either $\mathcal{X}$, or LK, or π, or simply about the encoding. The problem seems to be linked to the fact that π does not have an automatic *cancellation*: since communication is based on the exchange of channel names, processes that do not communicate with each other just 'sit next to each other'. In $\mathcal{X}$, a process that wants to be

'heard', but is not 'listened' to, disappears; this corresponds to a proof contracting to a proof, not to two proofs for the same sequent. But, when moving to *linear* $\mathcal{X}$, or $*\mathcal{X}$, studied in [16], this all changes. Since there reduction can generate non connected nets, it seems promising to explore an encoding of $*\mathcal{X}$ in π.

9. Classic type assignment for π

We will now introduce a notion of type assignment for processes in π, that describes the '*input-output interface*' of a process. This notion is novel in that it assigns to channels the type of the input or output that is sent over the channel; in that it differs from normal notions, that would state:

$$\frac{P :\cdot \Gamma, b{:}A \vdash \Delta}{\overline{a}\langle b\rangle . P :\cdot \Gamma, b{:}A \vdash a{:}\mathsf{ch}(A), \Delta}$$

In order to be able to encode LK, types in our system will not be decorated with channel information.

As for the notion of type assignment on $\mathcal{X}$ terms, in the typing judgements we always write channels used for input on the left and channels used for output on the right; this implies that, if a channel is both used to send and to receive, it will appear on both sides.

Definition 9.1 (Type assignment)**.** The types and contexts we consider for the π-calculus are defined like those of Definition 3.1, generalised to names.

Type assignment for π-calculus is defined by the following sequent system:

$$(0) : \frac{}{0 :\cdot \Gamma \vdash_\pi \Delta} \qquad (\textit{pair-out}) : \frac{P :\cdot \Gamma, b{:}A \vdash_\pi c{:}B, \Delta}{\overline{a}\langle\langle b, c\rangle\rangle . P :\cdot \Gamma, b{:}A \vdash_\pi a{:}A{\to}B, c{:}B, \Delta}$$

$$(!) : \frac{P :\cdot \Gamma \vdash_\pi \Delta}{!\, P :\cdot \Gamma \vdash_\pi \Delta} \qquad (\textit{let}) : \frac{P :\cdot \Gamma, y{:}B \vdash_\pi x{:}A, \Delta}{\textit{let}\,\langle x, y\rangle = z\ \textit{in}\ P :\cdot \Gamma, z{:}A{\to}B \vdash_\pi \Delta}$$

$$(\nu) : \frac{P :\cdot \Gamma, a{:}A \vdash_\pi a{:}A, \Delta}{(\nu a)\, P :\cdot \Gamma \vdash_\pi \Delta} \qquad (\textit{in}) : \frac{P :\cdot \Gamma, x{:}A \vdash_\pi x{:}A.\Delta}{a(x) . P :\cdot \Gamma, a{:}A \vdash_\pi \Delta}$$

$$(|) : \frac{P :\cdot \Gamma \vdash_\pi \Delta \quad Q :\cdot \Gamma \vdash_\pi \Delta}{P \mid Q :\cdot \Gamma \vdash_\pi \Delta} \qquad (\textit{out}) : \frac{P :\cdot \Gamma, b{:}A \vdash_\pi b{:}A, \Delta}{\overline{a}\langle b\rangle . P :\cdot \Gamma, b{:}A \vdash_\pi a{:}A, b{:}A, \Delta}$$

Notice that it is possible to derive $\overline{a}\langle a\rangle :\cdot \vdash_\pi a{:}A$, but that this is not generated by the encoding.

The '*input-output interface of a π-process*' property is nicely preserved by all the rules; it also explains how the type system confines the handling of pairs to the rules $(\textit{let})$ and $(\textit{pair-out})$.

It should be remarked that this notion of type assignment does not (directly) relate back to LK. For example, rules $(|)$ and $(!)$ do not change the contexts, so do not correspond to any rule in the logic, not even to a $\lambda\mu$-style activation step. Moreover, rule (ν) just *hides* a formula.

Example 9.2. We can derive

$$\dfrac{\dfrac{\dfrac{\quad}{P :\cdot \Gamma, y{:}B \vdash_\pi x{:}A, \Delta}}{\textit{let}\,\langle x, y\rangle = z\ \textit{in}\ P :\cdot \Gamma, z{:}A{\to}B \vdash_\pi \Delta}}{a(z) . \textit{let}\,\langle x, y\rangle = z\ \textit{in}\ P :\cdot \Gamma, a{:}A{\to}B \vdash_\pi \Delta}$$

so the following rule is derivable:

$$(\textit{pair-in}) : \frac{P :\cdot \Gamma, y{:}B \vdash_\pi x{:}A, \Delta}{a(\langle x, y\rangle).P :\cdot \Gamma, a{:}A \to B \vdash_\pi \Delta}$$

Notice that the rules $(\textit{pair-out})$ and $(\textit{pair-in})$ correspond to the logical rules $(\Rightarrow R)$ and $(\Rightarrow L)$. We now come to the main results for our notion of type assignment.

Theorem 9.3 (Witness reduction)**.** If $P :\cdot \Gamma \vdash_\pi \Delta$ and $P \to_\pi Q$, then $Q :\cdot \Gamma \vdash_\pi \Delta$.

The following theorem states that the encoding preserves types.

Theorem 9.4. If $P :\cdot \Gamma \vdash \Delta$, then $\ulcorner P \urcorner_\pi :\cdot \Gamma \vdash_\pi \Delta$.

Conclusions. We have seen that $\mathcal{X}$ is a very powerful formalism, capable of expressing a large variety of (essentially different) calculi, preserving types, and that computation in $\mathcal{X}$ looks very much like synchronisation in π, so that that calculus is a good system to study classical logic.

Acknowledgement. Of course this work would not exist without my co-authors and collaborators: I would like to thank Pierre Lescanne, Stéphane Lengrand, Alexander Summers, Jayshan Raghunandan, Philippe Audebaud, Luca Cardelli and Maria Grazia Vigliotti for their effort and energy.

REFERENCES

[1] M. Abadi and A. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. In *Proceedings of the Fourth ACM Conference on Computer and Communications Security*, pages 36–47. ACM Press, 1997.
[2] P. Audebaud and S. van Bakel. Understanding $\mathcal{X}$ with $\lambda\mu$. Consistent interpretations of the implicative sequent calculus in natural deduction. Submitted.
[3] S. van Bakel, L. Cardelli, and M.G. Vigliotti. From $\mathcal{X}$ to π; representing the classical sequent calculus in π-calculus. Submitted.
[4] S. van Bakel, S. Lengrand, and P. Lescanne. The language $\mathcal{X}$: circuits, computations and classical logic. In *ICTCS'05*, LNCS 3701, pages 81–96, 2005.
[5] S. van Bakel and P. Lescanne. Computation with classical sequents. *Mathematical Structures of Computer Science*, 2008. To appear.
[6] S. van Bakel and J. Raghunandan. Implementing $\mathcal{X}$. In *TermGraph'04*, ENTCS, 2005.
[7] S. van Bakel and J. Raghunandan. Capture avoidance and garbage collection for $\mathcal{X}$. Submitted.
[8] H. Barendregt. *The Lambda Calculus: its Syntax and Semantics*. North-Holland, Amsterdam, revised edition, 1984.
[9] R. Bloo and K.H. Rose. Preservation of strong normalisation in named lambda calculi with explicit substitution and garbage collection. In *CSN'95*, pages 62–72, 1995.
[10] P.-L. Curien and H. Herbelin. The Duality of Computation. In *Proceedings of the 5 th ACM SIGPLAN International Conference on Functional Programming (ICFP'00)*, pages 233–243. ACM, 2000.
[11] G. Gentzen. Untersuchungen über das Logische Schliessen. *Mathematische Zeitschrift*, 39:176–210 and 405–431, 1935.
[12] Ph. de Groote. On the relation between the $\lambda\mu$-calculus and the syntactic theory of sequential control. In *LPAR'94*, LNCS 822, pages 31–43, 1994.
[13] H. Herbelin. *C'est maintenant qu'on calcule: au cœur de la dualité*. Mémoire de habilitation, Univ. Paris 11, 2005.
[14] K. Honda, N. Yoshida, and M. Berger. Control in the π-calculus. In *Proc. Fourth ACM-SIGPLAN Continuation Workshop (CW'04)*, 2004.
[15] S. Lengrand. Call-by-value, call-by-name, and strong normalization for the classical sequent calculus. In *WRS 2003*, ENTCS 86, 2003.
[16] P. Lescanne and D. Žunić. $*\mathcal{X}$: a diagrammatic calculus with a classical fragrance. Manuscript.
[17] R. Milner. Function as processes. *Mathematical Structures in Computer Science*, 2(2):269–310, 1992.
[18] M. Parigot. An algorithmic interpretation of classical natural deduction. In *LPAR'92*, LNCS 624, pages 190–201, 1992.

[19] M. Parigot. Classical proofs as programs. In *Kurt Gödel Colloquium*, pages 263–276, 1993.
[20] D. Sangiorgi and D. Walker. On barbed equivalences in the π-calculus. In *CONCUR'01*, LNCS 2154, 2001.
[21] A. Summers and S. van Bakel. Approaches to polymorphism in classical sequent calculus. In *ESOP'06*, LNCS 3924, pages 84 – 99, 2006.
[22] C Urban. Strong Normalisation for a Gentzen-like Cut-Elimination Procedure'. In *TLCA'01*, LNCS 2044, pages 415–429, 2001.
[23] C. Urban and G. M. Bierman. Strong normalisation of cut-elimination in classical logic. *Fundamentae Informaticae*, 45(1,2):123–155, 2001.
[24] Christian Urban. *Classical Logic and Computation*. PhD thesis, Univ. Cambridge, 2000.
[25] Philip Wadler. Call-by-Value is Dual to Call-by-Name. In *Proceedings of the eighth ACM SIGPLAN international conference on Functional programming*, pages 189 – 201, 2003.