The language $\mathcal{X}$: Circuits, Computations and Classical Logic

(Extended Abstract)

(Ninth Italian Conference on Theoretical Computer Science (ICTCS'05), LNCS 3701, pp. 66-80, 2005)

Steffen van Bakel[1†], Stéphane Lengrand[2], and Pierre Lescanne[3]

[1]Department of Computing, Imperial College London, 180 Queen's Gate London SW7 2BZ, U.K.

[2]School of Computer Science University of St Andrews, North Haugh, St Andrews Fife KY16 9SS, Scotland

[3]École Normale Supérieure de Lyon, 46 Allée d'Italie 69364 Lyon 07, FRANCE

Pierre.Lescanne@ens-lyon.fr

Abstract

We present the syntax and reduction rules for $\mathcal{X}$, an untyped language that is well suited to describe structures which we call "circuits" and which are made of parts that are connected by wires. To demonstrate that $\mathcal{X}$ gives an expressive platform, we will show how, even in an untyped setting, that we can faithfully embed algebraic objects and elaborate calculi, like the naturals, the λ-calculus, Bloe and Rose's calculus of explicit substitutions $\lambda\mathbf{x}$, and Parigot's $\lambda\mu$.

1 Introduction

In the past, the study of the relation between *computation*, *programming languages* and *logic* has concentrated mainly on *natural deduction systems*. In fact, these carry the predicate '*natural*' deservedly; in comparison with, for example, *sequent style systems*, natural deduction systems are easy to understand and reason about. This holds most strongly in the context of *non-classical* logics. For example, the relation between *Intuitionistic Logic* and the *Lambda Calculus* (with types) is well-studied and understood, and has resulted in a vast and well-investigated area of research, resulting in, amongst others, functional programming languages. In an other direction, the deep study of sequent calculus resulted in Linear Logic.

Expressing classical logic in a natural deduction system comes with handicaps. This can be observed by looking at [21], where Gentzen commented that even his intuitionistic Natural Deduction calculus "lacks a certain formal elegance", while its Classical counterpart fares still worse, breaking the symmetry between the introduction and elimination rules. Because of these technical difficulties, Gentzen found that to achieve the main results of [13], "I had to provide a logical calculus especially suited to the purpose. For this the natural [deduction] calculus proved unsuitable."

In this paper, we will try and break a spear for the sequent-style approach, and make some further steps towards the development of a programming language based on cut-elimination for the sequent calculus for classical logic. Essentially following [17], we will present a language called $\mathcal{X}$ that describes circuits, and its reduction rules that join circuits. The logic we will consider contains only implication, but that is mainly because we, in this initial phase, aim for simplicity; cut-elimination in sequent calculi is notorious for the great number of rules, which will only increase manifold when considering more logical connectors.

† Partially supported by École Normale Supérieure de Lyon, France

To break with the natural deduction paradigm comes with a price, in that no longer *abstraction* and *application* (corresponding to *introduction of implication* and *modes ponens*) are the basic tools of the extracted language. In fact, the language we obtain is more of a *continuation* style, that models both the *parameter* as well as the *context* call. However, abstraction and application can be faithfully implemented, and we will show how $\mathcal{X}$ can be used to describe the behaviour of functional programming languages at a very low level of granularity.

$\mathcal{X}$ as a language for describing circuits

The basic pieces of $\mathcal{X}$ can be understood as components with entrance and exit wires and ways to describe how to connect them to build larger circuits. Those component will be quickly surveyed in the introduction and receive a more detailed treatment in Section 2. We call *"circuits"* the structures we build, because they are made of components connected by wires.

$\mathcal{X}$ as a syntax for the sequent calculus

Starting from the proof of Dragalin [12], Herbelin proposed in his PhD [16] a Curry-Howard correspondence; this was more elaborated in [9] leading to the definition of the language $\overline{\lambda}\mu\tilde{\mu}$. Among other approaches we need to mention [15, 10, 11, 7]; more generally, this work has connections with linear logic [14]. The relation between CBN and CBV in the context of $\overline{\lambda}\mu\tilde{\mu}$ was studied in detail in [23].

The origins of the language $\mathcal{X}$ we discuss in this paper lie in an observation made on the structure of derivations in $\overline{\lambda}\mu\tilde{\mu}$ in [9]. This that was picked up by Lengrand [17], who introduced $\mathcal{X}$ and investigated in-depth the relation between $\mathcal{X}$ and $\overline{\lambda}\mu\tilde{\mu}$. Later it became apparent that $\mathcal{X}$ also has strong connections with the notations for the sequent calculus as presented first by Urban in his PhD thesis [22]. With respect to [22] an improvement of this paper is to make the notation more intuitive and readable by moving to an infix notation, and to insist on the computational aspect. This is achieved by studying $\mathcal{X}$ in the context of the normal functional programming languages paradigms, but, more importantly, to cut the link between $\mathcal{X}$ and Classical Logic, in that we also consider circuits that do *not* correspond to proofs.

This main step forward with respect to previous work is achieved by moving to an *untyped* language (both [17] and [22] consider only well-typed objects) which serves as an expressive framework for representing the untyped lambda calculus, the untyped calculus of explicit substitutions and the untyped language $\overline{\lambda}\mu\tilde{\mu}$. In particular, in our setting we can model infinite computations.

In the future, we aim to study $\mathcal{X}$ outside the context of Classical Logic in much the same way as the λ-calculus is studied outside the context of Intuitionistic Logic.

$\mathcal{X}$ as a fine grained operational model of computation

When taking the the λ-calculus as a model for programming languages, the operational behaviour is provided by *β-contraction*. As is well known, β-contraction expresses how to calculate the value of a function applied to a parameter. In this, the parameter is used to instantiate occurrences of the bound variable in the body via the process of *substitution*. This description is rather basic as it says nothing on the actual *cost* of the substitution, which is quite high at run-time. Usually, a calculus of *explicit substitutions* [8, 1, 19, 18] is considered better suited for an accurate account of the substitution process and its implementation. When we refer to the calculus of explicit substitution we rather intend the calculus of *explicit substitution with* *explicit names* $\lambda\mathbf{x}$, due to Bloo and Rose [8]. $\lambda\mathbf{x}$ gives a better account of substitution as it integrates substitutions as first class citizens, decomposes the process of inserting a term into atomic actions, and explains in detail how substitutions are distributed through terms to be eventually evaluated at the variable level.

In this paper, we will show that the level of description reached by explicit substitutions can in fact be greatly refined. In $\mathcal{X}$, we reach a 'subatomic' level by decomposing explicit substitutions into smaller components. At this level, the calculus $\mathcal{X}$ explains how substitutions and terms interact.

The calculus is actually symmetric [5] and, unlike $\lambda\mathbf{x}$ where a substitution is applied to a term, a term in $\mathcal{X}$ can also be applied to a substitution. Their interaction percolates (propagates) subtly and gently through the term or substitution according to the direction that has been chosen. We will see that the these two kinds of interaction have a direct connection with call-by-value and call-by-name reduction, that both have a natural description in $\mathcal{X}$.

A notion of principal contexts for $\mathcal{X}$ has been defined in [4]; a tool [3, 4] to study $\mathcal{X}$ has been developed (see `http://www.doc.ic.ac.uk/~jr200/X`) that allows to input circuits from $\mathcal{X}$ and have fine control over reduction.

The ingredients of the syntax

It is important to note that $\mathcal{X}$ does *not* have variables[1] –like the λ-calculus or $\overline{\lambda}\mu\tilde{\mu}$– as possible places where terms might be inserted; instead, $\mathcal{X}$ has *wires*, also called *connectors*, that can occur free or bound in a term. As for the λ-calculus, the binding of a wire indicates that it is *active* in the computation; other than in the λ-calculus, however, the binding is not part of a term that is involved in the interaction, but is part of the interaction itself.

There are two kinds of wires: *sockets* and *plugs* (corresponding to *variables* and *covariables*, respectively, in [23]) that are reminiscent of values and continuations. Wires are not supposed to denote a location in a term like variables in the λ-calculus. Rather, they can be *connected* with wires in other components.

One specificity of $\mathcal{X}$ is that syntactic constructors bind *two* wires, one of each kind. In $\mathcal{X}$, bound wires receive a hat, so to show that x is bound we write $\hat{x}$ [24, 25]. That a wire is bound in a term implies, naturally, that this wire is unknown outside that term, but also that it 'interacts' with another 'opposite' wire that is bound into another term. The interaction differs from one constructor to another, and is ruled by basic reductions (see Section 2). In addition to bound wires an introduction rule exhibits a free wire, that is exposed; this can correspond to the creation of the wire, which is then connectable.

Contents of this paper

In this paper we will present the formal definitions for $\mathcal{X}$, via syntax and reduction rules, and will show that the system is well behaved by stating a number of essential properties. We will define a notion of simple type assignment for terms in $\mathcal{X}$, in that we will define a system of derivable judgements for which the terms of $\mathcal{X}$ are witnesses; we will show a soundness result for this system by showing that a subject-reduction result holds.

We will also compare $\mathcal{X}$ with a number of its predecessors. In fact, we will show that a number of well-know calculi are easily, elegantly and surprisingly effectively implementable in $\mathcal{X}$. For anyone familiar with the problem of expressibility, in view of the fact that $\mathcal{X}$ is substitution-free, these result are truly novel. With the exception of the calculus $\overline{\lambda}\mu\tilde{\mu}$, the

[1] We encourage the reader to not become confused by the use of names like x for the class of connectors that are called plugs; these names are, in fact, inherited from $\overline{\lambda}\mu\tilde{\mu}$.

converse is unobtainable. This can easily be understood from the fact that the vast majority of calculi in our area is confluent (Church-Rosser), whereas $\mathcal{X}$ is not.

2 The $\mathcal{X}$-calculus

The circuits that are the objects of $\mathcal{X}$ are built with three kinds of building stones, or constructors, called *capsule*, *export* and *mediator*. We define an operator *cut*, which is handy for describing circuit construction, and which will be eliminated eventually by *rules*. In addition we give *congruence among circuits*.

2.1 The operators

Circuits are connected through *wires* that are named. In our description wires are directed: we know in which direction the 'ether running through our circuits' moves, and can say when a wire provides an entrance to a circuit or when a wire provides an exit. Thus we make the distinction between exit wires which we call *plugs* and entry wires which we call *sockets*; we will use the word *connectors* for either sockets or plugs.

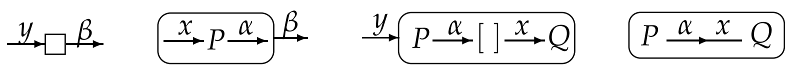
When connecting two circuits P and Q by the operator we may suppose that P has a plug α and Q has a socket x which we want to connect together to create a flow from P to Q. After the link has been established, the wires have been plugged, and the names of the connectors are forgotten; in fact, those names are *bound* in the link. We use the *"hat"*-notation to express binding, writing $\hat{x}$ to say that x is bound, keeping in line with the old tradition of *Principia Mathematica* [24]. The notion of free and bound connectors is defined as usual. We will normally adopt Barendregt's convention (called convention on variables by Barendregt, but here it will be a convention on names). An exception to that convention is the definition of natural numbers in Section 3.

Definition 2.1 (Syntax) The circuits of the $\mathcal{X}$-calculus are defined by the following grammar, where $x, y, \ldots$ range over the infinite set of *sockets*, and $\alpha, \beta, \ldots$ over the infinite set of *plugs*.

$$P, Q ::= \langle y.\beta \rangle \mid \widehat{x}P\widehat{\alpha}\cdot\beta \mid P\widehat{\alpha}\;[y]\;\widehat{x}Q \mid P\widehat{\alpha}\dagger\widehat{x}Q$$

Notice that, using Barendregt's convention, for example, the connector α in $P\widehat{\alpha}\;[y]\;\widehat{x}Q$ is supposed not to occur free in Q.

Diagrammatically, we represent the basic circuits as:



2.2 The reduction rules

The calculus, defined by the reduction rules below, explains in detail how cuts are distributed through circuits to be eventually erased at the level of capsules.

It is important to know when a connector is introduced, i.e. is connectable, i.e. is exposed and unique; this will play an important role in the reduction rules. Informally, a circuit P introduces a socket x if P is constructed from subcircuits which do not contain x as free socket, so x only occurs at the "top level." This means that P is either a mediator with a middle connector $[x]$ or a capsule with left part x. Similarly, a circuit introduces a plug α if it is an export that "creates" α or a capsule with right part α. We say now formally what it means for a terms to *introduce* a connector.

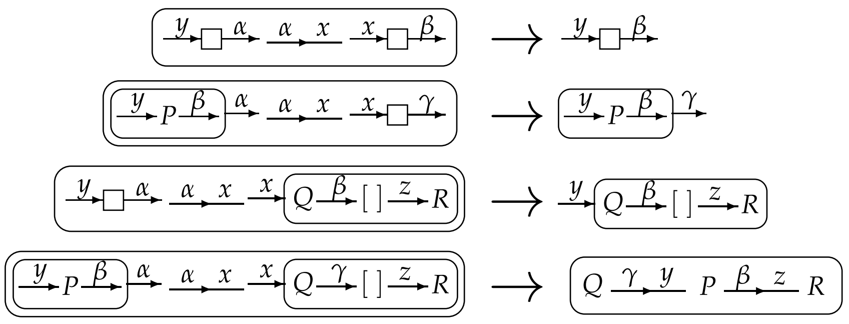


Figure 1: The diagrammatical representation for the logical rules

Definition 2.2 (Introduction)

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

We first present a simple family of reduction rules, that specify how to reduce a cut with sub-circuits that both introduce the connectors mentioned in the cut.

Definition 2.3 (Logical Reduction) Assume that the terms of the left-hand sides of the rules *introduce the socket x and the plug* α.

$$\begin{array}{lrcl}
(var): & \langle y.\alpha\rangle\widehat{\alpha}\dagger\widehat{x}\langle x.\beta\rangle & \to & \langle y.\beta\rangle \\
(exp): & (\widehat{y}P\widehat{\beta}\cdot\alpha)\widehat{\alpha}\dagger\widehat{x}\langle x.\gamma\rangle & \to & \widehat{y}P\widehat{\beta}\cdot\gamma \\
(med): & \langle y.\alpha\rangle\widehat{\alpha}\dagger\widehat{x}(Q\widehat{\beta}\,[x]\,\widehat{z}R) & \to & Q\widehat{\beta}\,[y]\,\widehat{z}R \\
(ins): & (\widehat{y}P\widehat{\beta}\cdot\alpha)\widehat{\alpha}\dagger\widehat{x}(Q\widehat{\gamma}\,[x]\,\widehat{z}R) & \to & Q\widehat{\gamma}\dagger\widehat{y}P\widehat{\beta}\dagger\widehat{z}R
\end{array}$$

The diagrammatical representation of these rules is given in Figure 1.

Notice that, in rule (*ins*), in addition to the conditions for introduction of the connectors that are active in the cut ($\alpha \notin fp(P)$ and $x \notin fs(Q,R)$) we can also state that $\beta \notin fp(Q)\backslash\{\gamma\}$, as well as that $y \notin fs(R)\backslash\{z\}$, due to Barendregt's convention.

Still in rule (*ins*) the reader may have noticed that we did not put parenthesis in the expression $Q\widehat{\gamma}\dagger\widehat{y}P\widehat{\beta}\dagger\widehat{z}R$, which therefore is officially not a circuit. Instead, we should have given both the circuits $(Q\widehat{\gamma}\dagger\widehat{y}P)\widehat{\beta}\dagger\widehat{z}R$ and $Q\widehat{\gamma}\dagger\widehat{y}(P\widehat{\beta}\dagger\widehat{z}R)$ as result of the rewriting. However there is, in fact, a kind of associativity at play which means that we can omit the parenthesis; this will be made more clear in the next section.

We now need to define how to reduce a cut when one of its sub-circuits does not introduce a connector mentioned in the cut. This requires to extend the syntax with two new operators that we call *activated* cuts:

$$P ::= \ldots \mid P\widehat{\alpha}\not\dagger\widehat{x}Q \mid P\widehat{\alpha}\nwarrow\widehat{x}Q$$

Definition 2.4 (Activating the cuts)

$$\begin{array}{ll}
(act\text{-}\textsc{l}): & P\widehat{\alpha}\dagger\widehat{x}Q \to P\widehat{\alpha}\not\dagger\widehat{x}Q,\ \textit{if P does not introduce } \alpha \\
(act\text{-}\textsc{r}): & P\widehat{\alpha}\dagger\widehat{x}Q \to P\widehat{\alpha}\nwarrow\widehat{x}Q,\ \textit{if Q does not introduce } x
\end{array}$$

Notice that both side-conditions can be valid simultaneously, thereby validating both rewrite rules at the same moment. This gives, in fact, a *critical pair* or *superposition* for our notion of reduction, and is the cause for the loss of confluence. This notion of activation is related to

Left propagation

$$
\begin{array}{rrcll}
(d\textsc{l}): & \langle y.\alpha\rangle\widehat{\alpha}\not\nearrow\widehat{x}P & \to & \langle y.\alpha\rangle\widehat{\alpha}\dagger\widehat{x}P & \\
(\textsc{l}1): & \langle y.\beta\rangle\widehat{\alpha}\not\nearrow\widehat{x}P & \to & \langle y.\beta\rangle, & \beta\neq\alpha \\
(\textsc{l}2): & (\widehat{y}Q\widehat{\beta}\cdot\alpha)\widehat{\alpha}\not\nearrow\widehat{x}P & \to & (\widehat{y}(Q\widehat{\alpha}\not\nearrow\widehat{x}P)\widehat{\beta}\cdot\gamma)\widehat{\gamma}\dagger\widehat{x}P, & \gamma\ \textit{fresh} \\
(\textsc{l}3): & (\widehat{y}Q\widehat{\beta}\cdot\gamma)\widehat{\alpha}\not\nearrow\widehat{x}P & \to & \widehat{y}(Q\widehat{\alpha}\not\nearrow\widehat{x}P)\widehat{\beta}\cdot\gamma, & \gamma\neq\alpha \\
(\textsc{l}4): & (Q\widehat{\beta}\,[z]\,\widehat{y}R)\widehat{\alpha}\not\nearrow\widehat{x}P & \to & (Q\widehat{\alpha}\not\nearrow\widehat{x}P)\widehat{\beta}\,[z]\,\widehat{y}(R\widehat{\alpha}\not\nearrow\widehat{x}P) & \\
(\textsc{l}5): & (Q\widehat{\beta}\dagger\widehat{y}R)\widehat{\alpha}\not\nearrow\widehat{x}P & \to & (Q\widehat{\alpha}\not\nearrow\widehat{x}P)\widehat{\beta}\dagger\widehat{y}(R\widehat{\alpha}\not\nearrow\widehat{x}P) &
\end{array}
$$

Right propagation

$$
\begin{array}{rlcll}
(d\textsc{r}): & P\widehat{\alpha}\not\nwarrow\widehat{x}\langle x.\beta\rangle & \to & P\widehat{\alpha}\dagger\widehat{x}\langle x.\beta\rangle & \\
(\textsc{r}1): & P\widehat{\alpha}\not\nwarrow\widehat{x}\langle y.\beta\rangle & \to & \langle y.\beta\rangle, & y\neq x \\
(\textsc{r}2): & P\widehat{\alpha}\not\nwarrow\widehat{x}(\widehat{y}Q\widehat{\beta}\cdot\gamma) & \to & \widehat{y}(P\widehat{\alpha}\not\nwarrow\widehat{x}Q)\widehat{\beta}\cdot\gamma & \\
(\textsc{r}3): & P\widehat{\alpha}\not\nwarrow\widehat{x}(Q\widehat{\beta}\,[x]\,\widehat{y}R) & \to & P\widehat{\alpha}\dagger\widehat{z}((P\widehat{\alpha}\not\nwarrow\widehat{x}Q)\widehat{\beta}\,[z]\,\widehat{y}(P\widehat{\alpha}\not\nwarrow\widehat{x}R)), & z\ \textit{fresh} \\
(\textsc{r}4): & P\widehat{\alpha}\not\nwarrow\widehat{x}(Q\widehat{\beta}\,[z]\,\widehat{y}R) & \to & (P\widehat{\alpha}\not\nwarrow\widehat{x}Q)\widehat{\beta}\,[z]\,\widehat{y}(P\widehat{\alpha}\not\nwarrow\widehat{x}R), & z\neq x \\
(\textsc{r}5): & P\widehat{\alpha}\not\nwarrow\widehat{x}(Q\widehat{\beta}\dagger\widehat{y}R) & \to & (P\widehat{\alpha}\not\nwarrow\widehat{x}Q)\widehat{\beta}\dagger\widehat{y}(P\widehat{\alpha}\not\nwarrow\widehat{x}R) &
\end{array}
$$

Figure 2: The propagation rules.

the notion of colour in [11].

Circuits where cuts are not activated are called *pure* (the diagrammatical representation of activated cuts is the same as that for not activated cuts). Activated cuts are propagated through the terms, to reach a position where a logical rule can be applied.

We will now define how to propagate a cut through sub-circuits. The direction of the activating shows in which direction the cut should be propagated, hence the two sets of reduction rules.

Definition 2.5 (Propagation) The rules of propagation are given in Figure 2.

We will subscript the arrow that represents our reduction to indicate certain sub-systems, defined by a sub-reduction: for example, we will write $\to_{\text{A}}$ for the reduction that uses only rules in *Left propagation* or *Right propagation*. In fact, $\to_{\text{A}}$ is the reduction that pushes $\not\nearrow$ and $\not\nwarrow$ inward.

The rules ($\textsc{l}2$) and ($\textsc{r}3$) deserve some attention. For instance, in the left-hand side of ($\textsc{l}2$), α is not introduced, hence α occurs more than once in $\widehat{y}Q\widehat{\beta}\cdot\alpha$, that is once after the dot and again in Q. The occurrence after the dot is dealt with separately by creating the new name γ. Note that the cut associated with that γ is then unactivated; this is because, after the activated cut has been pushed through $\widehat{y}(Q\widehat{\alpha}\not\nearrow\widehat{x}P)\widehat{\beta}\cdot\gamma$ (so leaves a circuit with no activated cut), the resulting term $(\widehat{y}R\widehat{\beta}\cdot\gamma)\widehat{\gamma}\dagger\widehat{x}P$ needs to be considered in its entirety: although we now that now γ is introduced, we do not know if x is. So, in any case, it would be wrong to activate the cut before the result of $Q\widehat{\alpha}\not\nearrow\widehat{x}P$ (i.e. R) is known. The same thing holds for x in ($\textsc{r}3$) and a new name z is created and the external cut is unactivated.

2.3 Structural congruences

By viewing $\mathcal{X}$ as a calculus, a natural questions to ask, and which has not been addressed in the past, is that if certain terms could be considered to be equivalne. To that purpose, we will define two congruences for $\dagger$ that look like associativity and commutativity.

Definition 2.6

$$\begin{array}{rlll}(\dagger\text{-}assoc)\colon & (P\widehat{\alpha}\dagger\widehat{x}Q)\widehat{\beta}\dagger\widehat{y}R & \overset{\mathcal{X}}{=} P\widehat{\alpha}\dagger\widehat{x}(Q\widehat{\beta}\dagger\widehat{y}R) & \text{if } \beta\notin fp(P)\ \&\ x\notin fs(R)\\ (left\text{-}comm)\colon & P\widehat{\alpha}\dagger\widehat{x}(Q\widehat{\beta}\dagger\widehat{y}R) & \overset{\mathcal{X}}{=} Q\widehat{\beta}\dagger\widehat{y}(P\widehat{\alpha}\dagger\widehat{x}R) & \text{if } x\notin fs(Q)\ \&\ y\notin fs(P)\\ (right\text{-}comm)\colon & (P\widehat{\alpha}\dagger\widehat{x}Q)\widehat{\beta}\dagger\widehat{y}R & \overset{\mathcal{X}}{=} (P\widehat{\beta}\dagger\widehat{y}R)\widehat{\alpha}\dagger\widehat{x}Q & \text{if } \alpha\notin fp(R)\ \&\ \beta\notin fp(Q)\end{array}$$

Rule ($\dagger$-*assoc*) allows us to write $P\widehat{\alpha}\dagger\widehat{x}Q\widehat{\beta}\dagger\widehat{y}R$ (provided the side-condition is fulfilled) since the order of the applications of cuts is irrelevant. Notice that the side-condition for this rule is the one we have indicated for (*ins*), and comes from the variable convention. This is consistent with the parenthesis-free notation we have used for the right-hand side of (*ins*). Notice that now writing $P\widehat{\alpha}\dagger\widehat{x}Q\widehat{\alpha}\dagger\widehat{x}R$ is licit. The second rule is left-commutativity and the third rule is right-commutativity.

There is another rule asserting the associativity of the mediators, given by:

$$\begin{array}{rl}(med\text{-}assoc)\colon P\widehat{\alpha}\ [z]\ \widehat{x}(Q\widehat{\beta}\ [u]\ \widehat{y}R) \overset{\mathcal{X}}{=} & (P\widehat{\alpha}\ [z]\ \widehat{x}Q)\widehat{\beta}\ [u]\ \widehat{y}R\\ & \text{if } \beta\notin fp(P)\backslash\{\alpha\}, x\notin fs(R)\backslash\{y\}\end{array}$$

Observe that, unsurprisingly, the side-condition is the same as for the cut. We will freely write $P\widehat{\alpha}\ [z]\ \widehat{x}Q\widehat{\beta}\ [u]\ \widehat{y}R$.

2.4 Call-by-name and call-by-value

In this section we will define two sub-systems of reduction, that have a strong connection to call-by-value (CBV) and call-by-name (CBN) reduction. Notice that this is essentially different from the approach of [23], where, as in $\overline{\lambda}\mu\tilde{\mu}$, only one notion of reduction is defined; the CBN-CBV result there was obtained via different interpretation functions from CBN/CBV calculi.

As mentioned above, when P does not introduce α and Q does not introduce x, $P\widehat{\alpha}\dagger\widehat{x}Q$ is a *superposition*, meaning that two rules, namely (*act*-L) and (*act*-R), can both be fired which can lead to different irreducible terms: $\rightarrow$ is *not confluent*. The sub-systems of reduction we will introduce explicitly favour one kind of activating whenever the above critical pair occurs; these were shown to be confluent in [17] when restricted to typeable terms, and we conjecture that this result can be extended to untyped terms.

Definition 2.7
- We write $P \rightarrow_{\text{V}} Q$ for the sub-reduction system that only activates a cut via (*act*-L) when it could be activated in two ways.
- Likewise, we write $P \rightarrow_{\text{N}} Q$ for the sub-reduction system that only activates such a cut via (*act*-R).

The reason to use the names CBV and CBN in fact comes from the fact that these systems successfully implement their counterparts in the λ-calculus (see Theorem 5.5). And, in fact, the use of the terminology CBV is justifiable, when at the same time calling *values* those circuits that introduce a plug. But, in contrast to the case for the λ-calculus, our two systems are really *dual*, and we actually should use a terminology like 'call-by-$*$', where '$*$' is a name for those circuits that introduce a socket. At the moment, there is no clear idea on what '$*$' should be, so we will use the (misnomer) CBN.

We will now state some basic properties, which essentially show that the calculus is well behaved. Recall that a term is pure if it contains no activated cuts.

Lemma 2.8 (CANCELLATION) *i)* $P\widehat{\alpha}\dagger\widehat{x}Q \rightarrow_{\text{V}} P$ *if* $\alpha\notin fp(P)$ *and* P *is pure.*
ii) $P\widehat{\alpha}\dagger\widehat{x}Q \rightarrow_{\text{N}} Q$ *if* $x\notin fs(Q)$ *and* Q *is pure.*

We will now show that a cut with a capsule leads to renaming.

Lemma 2.9 (Renaming) *i)* $P\widehat{\delta}\dagger\widehat{z}\langle z.\alpha\rangle \rightarrow P[\alpha/\delta]$, *if* P *is pure.*
ii) $\langle z.\alpha\rangle\widehat{\alpha}\dagger\widehat{x}P \rightarrow P[z/x]$, *if* P *is pure.*

These results motivate the extension (in both sub-systems) of the reduction rules, formulating new rules in the shape of the above results.

3 Expressing the natural numbers in $\mathcal{X}$

The example of expressing natural numbers into $\mathcal{X}$ that we will give in this section is interesting in two respects. Firstly, it shows how a basic structure can be embedded in $\mathcal{X}$. Secondly, it shows many features and among them α-conversion.

A natural number is represented in $\mathcal{X}$ by a sequence of capsules connected by mediating sockets, i.e., with the same used names.

We assume that natural numbers have two free sockets x and f and one free plug α, with x as entry socket, α as exit plug and f as mediating socket. We define 0 as $\langle x.\alpha\rangle$ and $succ(N)$ as $N\widehat{\alpha}\ [f]\ \widehat{x}\langle x.\alpha\rangle$ where N itself is a natural number (which violates Barendregt's convention and should be removed in actual use). By induction it follows that x and α have both a unique occurrence in each natural number.

Lemma 3.1 If N *is a natural number, then* $N\widehat{\alpha}\ [f]\ \widehat{x}\langle x.\alpha\rangle = \langle x.\alpha\rangle\widehat{\alpha}\ [f]\ \widehat{x}N$.

Lemma 3.2 If N_1 *and* N_2 *are natural numbers, then (* $\rightarrow_{\text{S}}$ *is either* $\rightarrow_{\text{V}}$ *or* $\rightarrow_{\text{N}}$ *)*
i) $(N_1\widehat{\alpha}\ [f]\ \widehat{x}\langle x.\alpha\rangle)\widehat{\alpha}\dagger\widehat{x}N_2 \ \rightarrow_{\text{S}}\ N_1\widehat{\alpha}\ [f]\ \widehat{x}N_2$
ii) $N_1\widehat{\alpha}\dagger\widehat{x}(N_2\widehat{\alpha}\ [f]\ \widehat{x}\langle x.\alpha\rangle) \ \rightarrow_{\text{S}}\ N_1\widehat{\alpha}\ [f]\ \widehat{x}N_2$

Definition 3.3 (Addition and multiplication)

$$add(N_1,N_2) = N_1\widehat{\alpha}\dagger\widehat{x}N_2 \qquad times(N_1,N_2) = (\widehat{x}N_1\widehat{\alpha}\cdot\beta)\widehat{\beta}\dagger\widehat{f}N_2$$

Using this definition, we can show that the normal properties for addition hold.

Lemma 3.4 (Properties of *add*)

$$\begin{aligned} add(\langle x.\alpha\rangle,N) &\rightarrow_{\text{A}} N \\ add(N,\langle x.\alpha\rangle) &\rightarrow_{\text{A}} N \\ add(N_1\widehat{\alpha}\ [f]\ \widehat{x}\langle x.\alpha\rangle,N_2) &\rightarrow_{\text{A}} add(N_1,N_2)\widehat{\alpha}\ [f]\ \widehat{x}\langle x.\alpha\rangle \\ add(N_1,N_2\widehat{\alpha}\ [f]\ \widehat{x}\langle x.\alpha\rangle) &\rightarrow_{\text{A}} add(N_1,N_2)\widehat{\alpha}\ [f]\ \widehat{x}\langle x.\alpha\rangle \end{aligned}$$

From Lemma 3.4 we get, by induction:

$$\begin{array}{ll} add(0,N) = N & add(succ(N_1),N_2) = succ(add(N_1,N_2)) \\ add(N,0) = N & add(succ(N_1),N_2) = succ(add(N_1,N_2)). \end{array}$$

From them we can prove:

$$\begin{aligned} add(N_1,N_2) &= add(N_2,N_1) \\ add(N_1,add(N_2,N_3)) &= add(add(N_1,N_2),N_3). \end{aligned}$$

The key point of the definition of *times* is that the *ins* rule copies N_2 into N_1 at each of the occurences of f.

Lemma 3.5 (Properties of *times*)

$$\begin{aligned} times(N,0) &\rightarrow_{\text{A}} 0 \\ times(N_1,succ(N_2)) &\rightarrow_{\text{A}} add(times(N_1,N_2),N_1) \end{aligned}$$

4 Typing for $\mathcal{X}$

The notion of type assignment on $\mathcal{X}$ that we present in this section is the basic implicative system for Classical Logic (Gentzen system LK). The Curry-Howard property is easily achieved by erasing all term-information.

Definition 4.1 (Types and Contexts) *i*) The set of types is defined by the grammar:

$$A,B ::= \varphi \mid A{\rightarrow}B\,.$$

The types considered in this paper are normally known as *simple* (or *Curry*) types.

ii) A *context of sockets* Γ is a mapping from sockets to types, denoted as a finite set of *statements* $x{:}A$, such that the *subject* of the statements (x) are distinct. When we write Γ_1,Γ_2 we mean the union of Γ_1 and Γ_2 when Γ_1 and Γ_2 are coherent (if Γ_1 contains $x{:}A_1$ and Γ_2 contains $x{:}A_2$ then $A_1 = A_2$).

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

Definition 4.2 (Typing for $\mathcal{X}$) *i*) *Type judgements* are expressed via a ternary relation $P :\cdot \Gamma \vdash \Delta$, where Γ is a context of *sockets* and Δ is a context of *plugs*, and P is a circuit. We say that P is the *witness* of this judgement.

ii) *Type assignment for* $\mathcal{X}$ is defined by the following sequent calculus:

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

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

We write $P :\cdot \Gamma \vdash \Delta$ if there exists a derivation for this judgement.

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.

We can now provide the type naturals.

- The type of natural numbers in $\mathcal{X}$ is $N :\cdot x{:}A, f{:}A \rightarrow A \vdash \alpha{:}A$.

The soundness result of simple type assignment with respect to reduction is stated as usual:

Theorem 4.3 (Witness reduction) *i) If $P :\cdot \Gamma \vdash \Delta$, and $P \rightarrow Q$, then $Q :\cdot \Gamma \vdash \Delta$.*

ii) If $P :\cdot \Gamma \vdash \Delta$, and $P \overset{\mathcal{X}}{=} Q$, then $Q :\cdot \Gamma \vdash \Delta$.

Theorem 4.4 (Strong normalisation [22]) *If $P :\cdot \Gamma \vdash \Delta$, then P is strongly normalising.*

5 Interpreting the λ-calculus

In this section, we illustrate the expressive power of $\mathcal{X}$ by showing that we can faithfully interpreted the the λ-calculus [6], and in the following sections we will show a similar result for $\lambda\mathbf{x}$ and $\lambda\mu$. Using the notion of Curry type assignment, we will show that assignable types are preserved by the interpretation.

In part, the interpretation results could be seen as variants of similar results obtained by Curien and Herbelin in [9]. Indeed, we could have defined our mappings using the mappings of the λ-calculus and $\lambda\mu$ into $\overline{\lambda}\mu\tilde{\mu}$, and concatenating those to the mapping from $\overline{\lambda}\mu\tilde{\mu}$ to $\mathcal{X}$, but our encoding is more detailed and precise than that, and deals with explicit substitution as well. In fact, we will show that our interpretation encompasses CBV and CBN reduction, something that has not been achieved in [9], and will argue that $\mathcal{X}$ in fact does more than that, like expressing explicit substitution.

One should notice that for [9] the preservation of the CBV-evaluation and CBN-evaluation relies on two *distinct* translations of terms. For instance, the CBV- and CBN-the λ-calculus can both be encoded into CPS [2], and there it is clear that what accounts for the distinction CBV/CBN is the encodings themselves, and not the way CPS reduces the encoded terms.

So, when encoding the λ-calculus in $\overline{\lambda}\mu\tilde{\mu}$, the distinction between CBV and CBN mostly relies on Curien and Herbelin's two distinct encodings rather than the features of $\overline{\lambda}\mu\tilde{\mu}$ (the same holds for [23]). Whereas there the CBN-translation seems intuitive, they apparently need to twist it in a more complex way in order to give an accurate interpretation of the CBV-the λ-calculus, since the CBV-interpretation of a term M reduces to its CBN-interpretation. This is a bit disappointing since the CBN-encoding turns out to be more refined than the CBV-encoding, breaking the nice symmetry.

In contrast, in $\mathcal{X}$ we have no need of two separate interpretation functions, but will define only *one*. Combining this with the two sub-reduction systems $\rightarrow_{\mathrm{V}}$ and $\rightarrow_{\mathrm{N}}$ we can encode the the CBV- and CBN-the λ-calculus. We can compare this to what is done by Danos, Joinet, Shellinx, when they write (before section 3.1.2: "Note that choosing colours has nothing to do with imposing a strategy. We don not select redexes, but rather the way we want to reduce them ..."

We first define the direct encoding of the λ-calculus into $\mathcal{X}$:

Definition 5.1 (INTERPRETATION OF THE λ-CALCULUS IN $\mathcal{X}$)

$$\begin{aligned}
\llbracket x \rrbracket^{\lambda}_{\alpha} &= \langle x.\alpha\rangle \\
\llbracket \lambda x.M \rrbracket^{\lambda}_{\alpha} &= \widehat{x}\llbracket M\rrbracket^{\lambda}_{\beta}\widehat{\beta}\cdot\alpha \\
\llbracket MN \rrbracket^{\lambda}_{\alpha} &= \llbracket M\rrbracket^{\lambda}_{\gamma}\widehat{\gamma}\dagger\widehat{x}(\llbracket N\rrbracket^{\lambda}_{\beta}\widehat{\beta}\;[x]\;\widehat{y}\langle y.\alpha\rangle)
\end{aligned}$$

Observe that every sub-term of $\llbracket M\rrbracket^{\lambda}_{\alpha}$ has exactly one free plug.

Definition 5.2 (CURRY TYPE ASSIGNMENT FOR THE λ-CALCULUS)

$$(Ax): \frac{}{\Gamma, x{:}A \vdash_{\lambda} x : A} \qquad (\rightarrow I): \frac{\Gamma, x{:}A \vdash_{\lambda} M : B}{\Gamma \vdash_{\lambda} \lambda x.M : A\rightarrow B}$$

$$(\rightarrow E): \frac{\Gamma \vdash_{\lambda} M : A\rightarrow B \quad \Gamma \vdash_{\lambda} N : A}{\Gamma \vdash_{\lambda} MN : B}$$

We can now show that typeability is preserved by $\llbracket\cdot\rrbracket^{\lambda}_{\alpha}$:

$$
\begin{array}{ll}
\llbracket \Delta\Delta \rrbracket^{\lambda}_{\beta} \;\triangleq\; \llbracket \lambda x.xx \rrbracket^{\lambda}_{\gamma}\widehat{\gamma}\dagger\widehat{z}(\llbracket \lambda x.xx \rrbracket^{\lambda}_{\gamma}\widehat{\gamma}\ [z]\ \widehat{y}\langle y.\beta\rangle) & \triangleq \\
(\widehat{x}\llbracket xx \rrbracket^{\lambda}_{\alpha}\widehat{\alpha}\cdot\delta)\widehat{\delta}\dagger\widehat{z}(\llbracket \lambda x.xx \rrbracket^{\lambda}_{\gamma}\widehat{\gamma}\ [z]\ \widehat{y}\langle y.\beta\rangle) & \rightarrow (\mathit{ins}) \\
\llbracket \lambda x.xx \rrbracket^{\lambda}_{\gamma}\widehat{\gamma}\dagger\widehat{x}(\llbracket xx \rrbracket^{\lambda}_{\alpha}\widehat{\alpha}\dagger\widehat{y}\langle y.\beta\rangle) & \rightarrow (2.9\text{-}(i)) \\
\llbracket \lambda x.xx \rrbracket^{\lambda}_{\gamma}\widehat{\gamma}\dagger\widehat{x}\llbracket xx \rrbracket^{\lambda}_{\beta} & \triangleq \\
\llbracket \lambda x.xx \rrbracket^{\lambda}_{\gamma}\widehat{\gamma}\dagger\widehat{x}(\langle x.\delta\rangle\widehat{\delta}\ [x]\ \widehat{y}\langle y.\beta\rangle) & \rightarrow (\mathit{act}\text{-}\textsc{r}) \\
\llbracket \lambda x.xx \rrbracket^{\lambda}_{\gamma}\widehat{\gamma}\dagger\!\!\!\nwarrow\widehat{x}(\langle x.\delta\rangle\widehat{\delta}\ [x]\ \widehat{y}\langle y.\beta\rangle) & \rightarrow (\textsc{r}3) \\
\llbracket \lambda x.xx \rrbracket^{\lambda}_{\gamma}\widehat{\gamma}\dagger\widehat{z}((\llbracket \lambda x.xx \rrbracket^{\lambda}_{\gamma}\widehat{\gamma}\dagger\!\!\!\nwarrow\widehat{x}\langle x.\delta\rangle)\widehat{\delta}\ [z]\ \widehat{y}(\llbracket \lambda x.xx \rrbracket^{\lambda}_{\gamma}\widehat{\gamma}\dagger\!\!\!\nwarrow\widehat{x}\langle y.\beta\rangle)) & \rightarrow (\textsc{l}1) \\
\llbracket \lambda x.xx \rrbracket^{\lambda}_{\gamma}\widehat{\gamma}\dagger\widehat{z}((\llbracket \lambda x.xx \rrbracket^{\lambda}_{\gamma}\widehat{\gamma}\dagger\!\!\!\nwarrow\widehat{x}\langle x.\delta\rangle)\widehat{\delta}\ [z]\ \widehat{y}\langle y.\beta\rangle) & \rightarrow (d\textsc{r}\ \&\ 2.9\text{-}(i)) \\
\llbracket \lambda x.xx \rrbracket^{\lambda}_{\gamma}\widehat{\gamma}\dagger\widehat{z}(\llbracket \lambda x.xx \rrbracket^{\lambda}_{\delta}\widehat{\delta}\ [z]\ \widehat{y}\langle y.\alpha\rangle) & \triangleq \llbracket \Delta\Delta \rrbracket^{\lambda}_{\alpha}
\end{array}
$$

Figure 3: Reduction of the interpretation of the lambda term $(\lambda x.xx)(\lambda x.xx)$.

Theorem 5.3 *If $\Gamma \vdash_{\lambda} M : A$, then $\llbracket M \rrbracket^{\lambda}_{\alpha} :\cdot\ \Gamma \vdash \alpha{:}A$.*

When encoding the CBV-the λ-calculus, we also use the $\llbracket\cdot\rrbracket^{\lambda}_{\alpha}$ interpretation. And, in contrast to $\overline{\lambda}\mu\tilde{\mu}$, we can get an accurate interpretation of the CBV-the λ-calculus into $\mathcal{X}$ by using the $\rightarrow_{\text{V}}$ system, which we can reformulate as the reduction system obtained by replacing rule (act-R) by:

$$(\mathit{act}\text{-}\textsc{r}') : P\widehat{\alpha}\dagger\widehat{x}Q \;\rightarrow\; P\widehat{\alpha}\dagger\!\!\!\nwarrow\widehat{x}Q, \textit{ if } P \textit{ introduces } \alpha \textit{ and } Q \textit{ does not introduce } x$$

Lemma 5.4 $\llbracket N \rrbracket_{\delta}\widehat{\delta}\dagger\!\!\!\nwarrow\widehat{x}\llbracket M \rrbracket_{\alpha} \rightarrow_{\text{A}} \llbracket M[N/x] \rrbracket_{\alpha}$.

Theorem 5.5 (Simulation of the λ-calculus) *i) If $M \rightarrow_{\text{V}} N$ then $\llbracket M \rrbracket^{\lambda}_{\gamma} \rightarrow_{\text{V}} \llbracket N \rrbracket^{\lambda}_{\gamma}$.*
ii) If $M \rightarrow_{\text{N}} N$ then $\llbracket M \rrbracket^{\lambda}_{\gamma} \rightarrow_{\text{N}} \llbracket N \rrbracket^{\lambda}_{\gamma}$.

Now notice that $(\lambda x.M)(PQ)$ is not an redex in the CBV-λ-calculus. We get

$$\llbracket(\lambda x.M)(PQ)\rrbracket^{\lambda}_{\alpha} \rightarrow (\llbracket P \rrbracket^{\lambda}_{\sigma}\widehat{\sigma}\dagger\widehat{t}(\llbracket Q \rrbracket^{\lambda}_{\tau}\widehat{\tau}\ [t]\ \widehat{u}\langle u.\gamma\rangle))\widehat{\gamma}\dagger\widehat{x}\llbracket M \rrbracket^{\lambda}_{\alpha}$$

In particular, γ is not introduced in the outer-most cut, so (act-L) can be applied. What the call-by-value reduction should block, however, is that (act-R$'$) can be applied; then the propagation of $\llbracket P \rrbracket^{\lambda}_{\sigma}\widehat{\sigma}\dagger\widehat{t}(\llbracket Q \rrbracket^{\lambda}_{\tau}\widehat{\tau}\ [t]\ \widehat{u}\langle u.\gamma\rangle)$ into $\llbracket M \rrbracket^{\lambda}_{\alpha}$ is blocked (which would produce $\llbracket M[(PQ)/x] \rrbracket^{\lambda}_{\alpha}$). Notice that we can only apply rule (act-R$'$) if both $\llbracket P \rrbracket^{\lambda}_{\sigma}\widehat{\sigma}\dagger\widehat{t}(\llbracket Q \rrbracket^{\lambda}_{\tau}\widehat{\tau}\ [t]\ \widehat{u}\langle u.\gamma\rangle)$ introduces γ and $\llbracket M \rrbracket^{\lambda}_{\alpha}$ does not introduce x. This is not the case, since the first test fails.

On the other hand, if N is a λ-value (i.e. either a variable or an abstraction) then $\llbracket N \rrbracket^{\lambda}_{\alpha}$ introduces α (in fact, N is a value if and only if $\llbracket N \rrbracket^{\lambda}_{\alpha}$ introduces α). Then $\llbracket N \rrbracket^{\lambda}_{\gamma}\widehat{\gamma}\dagger\widehat{x}\llbracket M \rrbracket^{\lambda}_{\alpha}$ cannot be reduced by rule (act-L), but by either rule (act-R) or a logical rule. This enables the reduction

$$\llbracket N \rrbracket^{\lambda}_{\gamma}\widehat{\gamma}\dagger\widehat{x}\llbracket N \rrbracket^{\lambda}_{\alpha} \;\rightarrow_{\text{V}}\; \llbracket N[M/x] \rrbracket^{\lambda}_{\alpha}.$$

So CBV-reduction for the λ-calculus is respected by the interpretation function, using $\rightarrow_{\text{V}}$.

It is worthwhile to notice that the interpretation function $\llbracket\cdot\rrbracket^{\lambda}_{\alpha}$ does not generate a confluent sub-calculus. Indeed, we have both

$$
\begin{array}{ll}
\llbracket(\lambda x.xx)(yy)\rrbracket^{\lambda}_{\alpha} \rightarrow \langle y.\beta\rangle\widehat{\beta}\ [y]\ \widehat{x}(\langle x.\gamma\rangle\widehat{\gamma}\ [x]\ \widehat{v}\langle v.\alpha\rangle) & \text{and} \\
\llbracket(\lambda x.xx)(yy)\rrbracket^{\lambda}_{\alpha} \rightarrow \langle y.\beta\rangle\widehat{\beta}\ [y]\ \widehat{a}((\langle y.\gamma\rangle\widehat{\gamma}\ [y]\ \widehat{b}\langle b.\delta\rangle)\widehat{\delta}\ [a]\ \widehat{c}\langle c.\alpha\rangle) &
\end{array}
$$

both normal forms. This is of course not surprising, seen that $(\lambda x.xx)(yy)$ has different normal forms with respect to CBN and CBV reduction.

To conclude this section, and illustrate the expressive power of $\mathcal{X}$ as abstract machine for reduction, Figure 3 shows an infinite reduction sequence in $\mathcal{X}$.

6 Interpreting $\lambda\mathbf{x}$

We will now interpret a calculus of explicit substitutions, namely $\lambda\mathbf{x}$ [8], where any β-reduction of the λ-calculus can be split into several more atomic steps of computation. In this section we show that $\mathcal{X}$ has a fine level of atomicity as it simulates each reduction step by describing how the explicit substitutions interact with terms.

We briefly recall here the calculus $\lambda\mathbf{x}$.

Definition 6.1 ($\lambda\mathbf{x}$) The syntax of $\lambda\mathbf{x}$ is an extension of that of the λ-calculus:

$$M ::= x \mid \lambda x.M \mid M_1 M_2 \mid M\langle x = N\rangle$$

The reduction relation is defined by the following rules

$$\begin{array}{rcll rcll}
(\lambda x.M)P & \to & M\langle x{=}P\rangle & (\mathsf{B}) & x\langle x{=}P\rangle & \to & P & (\mathsf{VarI}) \\
(MN)\langle x{=}P\rangle & \to & M\langle x{=}P\rangle N\langle x{=}P\rangle & (\mathsf{App}) & y\langle x{=}P\rangle & \to & y & (\mathsf{VarK}) \\
(\lambda y.M)\langle x{=}P\rangle & \to & \lambda y.(M\langle x{=}P\rangle) & (\mathsf{Abs}) & M\langle x{=}P\rangle & \to & M, \text{ if } x \notin fv(M) & (\mathsf{gc})
\end{array}$$

Notice that the notion of reduction $\lambda\mathbf{x}$ is obtained by deleting rule (gc), and the notion of reduction $\lambda\mathbf{x}_{\mathsf{gc}}$ is obtained by deleting rule (VarK). The rule (gc) is called 'garbage collection', as it removes useless substitutions. We will write $\to_{\mathrm{x}}$ for either reduction system.

Definition 6.2 (Interpretation of $\lambda\mathbf{x}$ in $\mathcal{X}$) We define $[\![\cdot]\!]^{\mathsf{x}}_{\alpha}$ as the interpretation $[\![\cdot]\!]^{\lambda}_{\alpha}$, by adding:

$$[\![M\langle x = N\rangle]\!]^{\mathsf{x}}_{\alpha} = [\![N]\!]^{\mathsf{x}}_{\beta}\widehat{\beta} \nwarrow \widehat{x}[\![M]\!]^{\mathsf{x}}_{\alpha}.$$

Now we show that the reductions can be simulated, preserving the evaluation strategies. Our notion of CBV-$\lambda\mathbf{x}$ is naturally inspired by that of the λ-calculus: in a CBV-β-reduction, the argument must be a value, so that means that when it is simulated by CBV-$\lambda\mathbf{x}$, all the substitutions created are of the form $M\langle x = N\rangle$ where N is a value, that is, either a variable or an abstraction, just as in the λ-calculus. Hence, we build the CBV-$\lambda\mathbf{x}$ by a syntactic restriction:

$$M ::= x \mid \lambda x.M \mid M_1 M_2 \mid M\langle x = \lambda x.N\rangle \mid M\langle x = y\rangle.$$

Now notice that, again, N is a value if and only if $[\![N]\!]^{\mathsf{x}}_{\alpha}$ introduces α.

Theorem 6.3 (Simulation of rule (B)) (CBN:): $[\![(\lambda x.M)N]\!]^{\mathsf{x}}_{\alpha} \to_{\mathrm{N}} [\![M\langle x = N\rangle]\!]^{\mathsf{x}}_{\alpha}$
(CBV:): $[\![(\lambda x.M)N]\!]^{\mathsf{x}}_{\alpha} \to_{\mathrm{V}} [\![M\langle x = N\rangle]\!]^{\mathsf{x}}_{\alpha}$ *iff N is a value.*

Theorem 6.4 (Simulation of the other rules) *Let $M \to N$ by any of the rules (App), (Abs), (VarI), (VarK), (gc), then $[\![M]\!]^{\mathsf{x}}_{\gamma} \to_{\mathrm{V}} [\![N]\!]^{\mathsf{x}}_{\gamma}$ and $[\![M]\!]^{\mathsf{x}}_{\gamma} \to_{\mathrm{N}} [\![N]\!]^{\mathsf{x}}_{\gamma}$.*

We can now state that $\lambda\mathbf{x}$-reduction is preserved by interpretation of terms into $\mathcal{X}$.

Theorem 6.5 (Simulation of $\lambda\mathbf{x}$) *i) If $M \to_{\mathrm{V}} N$ then $[\![M]\!]^{\mathsf{x}}_{\gamma} \to_{\mathrm{V}} [\![N]\!]^{\mathsf{x}}_{\gamma}$*
ii) If $M \to_{\mathrm{N}} N$ then $[\![M]\!]^{\mathsf{x}}_{\gamma} \to_{\mathrm{N}} [\![N]\!]^{\mathsf{x}}_{\gamma}$

7 Interpreting $\lambda\mu$

Parigot's $\lambda\mu$-calculus [20] is yet another proof-term syntax for classical logic, but expressed in the setting of Natural Deduction. Curien and Herbelin [9] have shown how the normalisation in $\lambda\mu$ can be interpreted as the cut-elimination in $\overline{\lambda}\mu\tilde{\mu}$. Using that mapping, and the interpretation of $\overline{\lambda}\mu\tilde{\mu}$ into $\mathcal{X}$ from [17], we can generate the following:

Definition 7.1 (Interpretation of $\lambda\mu$ in $\mathcal{X}$) We define $\llbracket\cdot\rrbracket^{\lambda\mu}_{\alpha}$ as the interpretation $\llbracket\cdot\rrbracket^{\lambda}_{\alpha}$, by adding:

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

Similarly to the previous sections, we can add:

$$\begin{aligned}\llbracket M[N/x]\rrbracket^{\lambda\mu}_{\alpha} &= \llbracket N\rrbracket^{\lambda\mu}_{\beta}\widehat{\beta}\dagger\widehat{x}\llbracket M\rrbracket^{\lambda\mu}_{\alpha}\\ \llbracket(\mu\delta.[\gamma]M)[N\cdot\delta/\delta]\rrbracket^{\lambda\mu}_{\alpha} &= \llbracket M\rrbracket^{\lambda\mu}_{\gamma}\widehat{\delta}\dagger\widehat{x}(\llbracket N\rrbracket^{\lambda\mu}_{\beta}\widehat{\beta}\,[x]\,\widehat{y}\langle y.\alpha\rangle)\end{aligned}$$

Notice that the last alternative is justified, since

Lemma 7.2 The following rule is admissible:

$$\llbracket(\mu\delta.[\gamma]M))N\rrbracket^{\lambda\mu}_{\alpha} \to \llbracket M\rrbracket^{\lambda\mu}_{\gamma}\widehat{\delta}\dagger\widehat{x}(\llbracket N\rrbracket^{\lambda\mu}_{\beta}\widehat{\beta}\,[x]\,\widehat{y}\langle y.\alpha\rangle)$$

Notice also the striking similarity between $\llbracket MN\rrbracket^{\lambda\mu}_{\alpha}$ and the result of running $\llbracket(\mu\delta.[\gamma]M))N\rrbracket^{\lambda\mu}_{\alpha}$; the difference lies only in a bound socket.

The main result for this interpretation now becomes:

Theorem 7.3 (Simulation of $\lambda\mu$ in $\mathcal{X}$) *i) If $M\to_{\mathrm{V}} N$ then $\llbracket M\rrbracket^{\lambda\mu}_{\alpha}\to_{\mathrm{V}}\llbracket N\rrbracket^{\lambda\mu}_{\alpha}$.*
ii) If $M\to_{\mathrm{N}} N$ then $\llbracket M\rrbracket^{\lambda\mu}_{\alpha}\to_{\mathrm{N}}\llbracket N\rrbracket^{\lambda\mu}_{\alpha}$.

8 Conclusions and future work

We have seen that $\mathcal{X}$ is a continuation-style formal language that provides a Curry-Howard-de Bruijn isomorphism for a sequent calculus for implicative classical logic. But, of more interest, we have seen $\mathcal{X}$ is very well-suited as generic abstract machine for the running of (applicative) programming languages, by building not only an interpretation for λ, $\lambda\mu$ (for $\overline{\lambda}\mu\tilde{\mu}$, see [17]), but also for $\lambda\mathbf{x}$.

A wealth of research lies in the future, of which this paper is but the first step, the seed. We intend to study normalisation, and confluence of the CBN and CBV strategies, to extend $\mathcal{X}$ in order to represent the other logical connectives, study the relation with linear logic, proofnets (both typed and untyped), the relation with π-calculus, how to express recursion, functions, etc, etc.

Acknowledgements

We would like to thank Alexander Summers, Daniel Hirschkoff, Dragisa Zunic, Harry Mairson, Jamie Gabbay, Jayshan Raghunandan, Luca Cardelli, Luca Roversi, Maria Grazia Vigliotti, Mariangiola Dezani-Ciancaglini, Philippe Audebaud, and Simona Ronchi della Rocca for many fruitful discussions on the topic of this paper.

References

[1] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit Substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.

[2] A.W. Appel and T. Jim. Continuation-passing, closure-passing style. In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of Programming languages*, pages 293–302. ACM Press, 1989.

[3] S. van Bakel and J. Raghunandan. Implementing $\mathcal{X}$. In *Electronic Proceedings of Second International Workshop on* Term Graph Rewriting 2004 *(TermGraph'04), Rome, Italy*, Electronic Notes in Theoretical Computer Science, 2005.

[4] S. van Bakel, J. Raghunandan, and A. Summers. Term Graphs, α-conversion and Principal Types for $\mathcal{X}$. *Submitted*, 2005.

[5] F. Barbanera and S. Berardi. A Symmetric Lambda Calculus for Classical Program Extraction. *Information and Computation*, 125(2):103–117, 1996.

[6] H. Barendregt. *The Lambda Calculus: its Syntax and Semantics*. North-Holland, Amsterdam, revised edition, 1984.

[7] H.P. Barendregt and S Ghilezan. Lambda terms for natural deduction, sequent calculus and cut-elimination. *Journal of Functional Programming*, 10(1):121–134, 2000.

[8] R. Bloo and K.H. Rose. Preservation of Strong Normalisation in Named Lambda Calculi with Explicit Substitution and Garbage Collection. In *CSN'95 – Computer Science in the Netherlands*, pages 62–72, 1995.

[9] P.-L. Curien and H. Herbelin. The Duality of Computation. In *Proceedings of the 5th ACM SIGPLAN International Conference on Functional Programming (ICFP'00)*, volume 35.9 of *ACM Sigplan Notices*, pages 233–243. ACM, 2000.

[10] V. Danos, J.-B. Joinet, and H. Schellinx. Computational isomorphisms in classical logic (extended abstract). *Electronic Notes in Theoretical Computer Science*, 3, 1996.

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

[12] A.G. Dragalin. *Mathematical Intuitionism: Introduction to Proof Theory*, volume 67 of *Translations of Mathematical Monographs*. American Mathematical Society, 1987.

[13] G. Gentzen. Untersuchungen über das Logische Schliessen. *Mathematische Zeitschrift*, 39:176–210 and 405–431, 1935.

[14] J.-Y Girard. Linear Logic. *Theoretical Computer Science*, 50:1–102, 1987.

[15] J.-Y. Girard. A new constrcutive logic: classical logic. *Mathematical Structures in Computer Science*, 1(3):255–296, 1991.

[16] H. Herbelin. *Séquents qu'on calcule : de l'interprétation du calcul des séquents comme calcul de λ-termes et comme calcul de stratégies gagnantes*. Thèse d'université, Université Paris 7, Janvier 1995.

[17] S. Lengrand. Call-by-value, call-by-name, and strong normalization for the classical sequent calculus. In B. Gramlich and S. Lucas, editors, *3rd Workshop on Reduction Strategies in Rewriting and Programming (WRS 2003)*, volume 86 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2003.

[18] S. Lengrand, P. Lescanne, D. Dougherty, M. Dezani-Ciancaglini, and S. van Bakel. Intersection types for explicit substitutions. *Information and Computation*, 189(1):17–42, 2004.

[19] P. Lescanne. From $\lambda\sigma$ to λv: a Journey Through Calculi of Explicit Substitutions. In *POPL'94*, pages 60–69, 1994.

[20] M. Parigot. An algorithmic interpretation of classical natural deduction. In *Proceedings of 3rd International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'92)*, volume 624 of *Lecture Notes in Computer Science*, pages 190–201. Springer Verlag, 1992.

[21] M.E. Szabo, editor. *The Collected Papers of Gerhard Gentzen*. Studies in Logic and the Foundations of Mathematics. North-Holland, 1969.

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

[23] P. 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.

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

[25] A.N. Whitehead and B. Russell. *Principia Mathematica to *56*. Cambridge Mathematical Library. Cambridge University Press, 2nd edition, 1997.