Proof Nets in Process Algebraic Form

Dimitris Mostrous
Department of Informatics, University of Lisbon, Portugal
dimitris@di.fc.ul.pt

Abstract. We present δ-calculus, a computational interpretation of Linear Logic, in the form of a typed process algebra whose structures correspond to Proof Nets, and where typing derivations correspond to linear sequent proofs. Term reduction shares the properties of cut elimination in the logic, and immediately we can obtain a number of inherited qualities, among which are termination, deadlock-freedom, and determinism. We obtain the expected soundness results and provide a propositions-as-types correspondence theorem. We then propose extensions for general recursion and the removal of the additive fragment while allowing to express similar behaviour, contributing to the theory of proof nets.

1 Introduction

Proof Nets \([13,15]\) can be understood as a Classical Natural Deduction for Linear Logic \([13]\). They are an abstract, geometrical representation of sequent proofs that is inherently parallel due to locality of many normalising steps. We present δ-calculus, a new process algebraic computational interpretation of Linear Logic, where typed terms correspond directly to proof nets. We thus obtain a Curry-Howard correspondence relating Linear Logic proofs and typed terms in a highly parallel language with many desirable properties, in particular determinism (semantic confluence), termination (strong normalisation), and consequently deadlock-freedom. It can be argued that these properties are too strong for many kinds of concurrent systems, but they remain highly desirable for parallelising the execution of general purpose (deterministic) programs. Moreover, starting from a canonical interpretation of Linear Logic, concurrency extensions for sharing and non-determinism may be incorporated; this remains as future work. We extend proof nets with a connective for general recursion, and propose an adaptation of the system that allows us to completely eliminate the additive fragment (which induces boxes, i.e., prefixes), simplifying the logic while enabling similar behaviours, even if the ability to superimpose is lost.

We briefly clarify what we understand as parallelism in the context of this work. Parallelism arises when there is locality in reduction, i.e., when reduction rules do not mention an irrelevant (or larger than needed) context. Such contexts can generally contain terms that need not be affected by the actual redex, and could therefore participate in independent reductions which an implementation

\(^1\) From the Greek δίκτυο: «net». 
could safely perform concurrently. This is evident if one considers parallel reduction\cite{25}, since the family of immediate redices reflects the potential parallelism of a term. The $\pi$-calculus imposes sequentiality constraints that impede parallelism, and for this reason is not a good choice as a language for linear proofs. Take for instance the term $x(y).z(k).P \mid \overline{x}(a) \mid \overline{z}(b)$. The redex on $z$ may be safe to contract, but is blocked under the context of $x(y)$. It would be enabled in commutative $\pi$-calculus\cite{27}, where $x(y).z(k).P \equiv z(k).x(y).P$, but then $x$ would be blocked, so the two redices can never be performed in parallel. The culprit is input prefix, which represents an artificial causality, sustained by the need to define a scope for bound variables ($y, k$ above), and does not contribute towards the result of computation (unlike, say, a matching prefix). For this reason, once we abolish bound input, as in Fusion calculus\cite{24}, we can avoid prefixes, as in Solos calculus\cite{22}, obtaining a highly parallel language.

However, some causality can also arise from lack of symmetry: an example which cannot reduce (see\cite{22}) in Solos is $(\nu x)(ax \mid xb)$, although (in a linear setting) it would be safe to (somehow) obtain $\overline{ab}$, and adapting a linear equivalence\cite{28} for $\pi$-calculus, we could reasonably expect that $(\nu x)(ax \mid xb) \equiv \overline{ab}$.

The multiplicative fragment of $\delta$-calculus, which is derived directly from a process interpretation of proof nets, corresponds closely to a linear variation of Solos with explicit substitutions (interpreting logical axioms) that react locally with everything else. Specifically, the axiom link $ab$ can be seen as an explicit fusion $a=b$ (cf. the Explicit Fusions calculus\cite{12}), and we have $(\nu x)(ax \mid xb) \rightsquigarrow ab$, which indicates a high degree of locality, and thus potential parallelism.

**Summary of Main Results** Contrary to other works\cite{5,6}, we make no attempt to interpret proofs in $\pi$-calculus. Instead, we create a syntax as an isomorphic image of proof nets, similarly in some ways to the original interpretation\cite{3}. Consequently, our formulation allows as much parallelism as proof nets (with additive boxes), relaxing many of the sequentiality constraints of $\pi$-calculus, while exhibiting higher symmetry than Solos, due to linearity.

We interpret the fragment consisting of multiplicatives, additives, exponentials, and the multiplicative units. The quantifiers are left for future work, but pose no difficulty. The typing rules, in turn, are basically the sequent presentation of the logic. Proof normalisation (reduction) employs commutations that map closely to the commutative contractions of Proof Nets, therefore we can speak of complete cut-elimination. Hence, we do not specify any reduction strategy, although it is clear that some are better than others for practical purposes. One possible criticism is the presence of boxes (prefixes) for additives, but they are convenient for programming, where reasonable evaluation strategies might treat them as lazy. Nevertheless, we propose an extension that avoids them, and moreover we introduce a new kind of exponential for general recursion.

**Outline** In Section 2 we present the language and the reduction semantics, and in Section 3 we describe the typing system. In Section 4 we state the basic soundness properties, followed by a Curry-Howard correspondence between
typed terms and linear sequent proofs. In Section 5 we outline our extensions of proof nets. In Section 6 we conclude with an analysis of related work. Due to space limitations, proofs are given in an online appendix [1].

2 Proof Nets as Processes

A proof net consists of a collection of links, connectives which have premises and conclusions. The links are wired together, for example a conclusion of one can be a premise of another, and the whole composition forms a graph called a proof structure. When a proof structure corresponds to a sequent derivation, it is called a proof net, and the link conclusions which are not wired as premises to other links provide the conclusions of the sequent derivation. The correctness criterion, a purely geometrical method by which this correspondence is checked, was originally defined by Girard [13] and simplified later by Danos and Regnier [8] (for a fragment of the logic). In this work we check correctness by typing, but note that the complexity of the criterion and that of type-checking may be related. Proof net normalisation [13,14] amounts to the (deterministic) re-linking of the uniquely (but implicitly) identified parts of a net: once we assign names to them, we obtain a process algebra.

We now describe the static and dynamic parts of δ-calculus. Note that the language is explicitly typed, but in this section we ignore annotations for brevity. The syntax of terms π, ρ... is defined in Figure 1. In a slight abuse of notation, we define \( L(a) \) for a link with conclusion \( a \). The reduction relation \( \pi \rightsquigarrow \rho \) and the commutations \( \pi \leftrightarrow \rho \) are shown in Figure 2. We assume a countable set of names \( a, b, x, y \) etc., the usual notions of free and bound names, and \( \alpha \)-conversion. All names appearing in links are free, and are only made bound with scope restriction or one of the other forms with \( \nu \) which we subsequently explain. Parallel composition is associative with unit \( \epsilon \), to avoid confusion with the additive unit 0 of Linear Logic; binding priority is given by \( ., \nu, |, + \). In \( \rightsquigarrow \) we use a context \( C[\pi] \), to signify that reduction can take place in any subterm, without giving the obvious definition: it includes parallel composition,
Symmetric Contraction, $\rightsquigarrow$

\[
\begin{align*}
L(a) & \rightsquigarrow L(c) \mid a^1 \quad a \mid \overrightarrow{a} \rightsquigarrow a^1 \quad (\nu a)(a^1 \mid \pi) \rightsquigarrow \pi \quad \text{(Link – A, } \updownarrow - 1, \text{ Dis)} \\
\text{ L(a) } & b, c \rightsquigarrow b c \mid a^1 \\
\text{ a(\nu a)} & \pi + a_r(\nu a), \pi \mid \overrightarrow{a} \rightsquigarrow \pi \{\overrightarrow{a} \} \mid a^1 \\
\text{ a(\nu a)} & \pi + a_r(\nu a), \pi \mid \overrightarrow{a} \rightsquigarrow \pi \{\overrightarrow{a} \} \mid a^1 \\
\text{ a(\nu a)} & \pi \rightsquigarrow \pi \{\overrightarrow{a} \} \mid a^1 \\
\text{ a(\nu a)} & \pi \rightsquigarrow \pi \{\overrightarrow{a} \} \mid a^1
\end{align*}
\]

\(\text{Commutations, } \Rightarrow\)

\[
\begin{align*}
\pi \Rightarrow \rho & \quad \pi \Rightarrow \pi' \Rightarrow \rho \quad \pi \Rightarrow \pi \mid \varepsilon \\
C[\pi] & \Rightarrow C[\rho] \quad \pi \Rightarrow \rho \quad \pi \Rightarrow \sigma(\pi) \quad \pi \Rightarrow \varepsilon \Rightarrow \pi \quad (\text{Standard}) \\
(\nu a) & \pi \Rightarrow (\nu a)(\pi \mid b) \quad a \notin \text{fn}(\rho) \quad (1 - \nu a) \\
(\nu a)(a_r(\nu a), \pi_1 + a_r(\nu a), \pi_2 \mid b) & \Rightarrow a_r(\nu a)(\pi_1 \mid b) + a_r(\nu a)(\pi_2 \mid b) \quad b, d \notin \text{fn}(\rho) \quad (1 - \nu a) \\
(\nu a)(a_r(\nu a), \pi_1 \mid \pi) & \Rightarrow a_r(\nu a)(\pi_1 \mid \pi) \left(\pi_1 \mid b, d \notin \text{fn}(\pi)\right) \quad (\text{I } \varepsilon - \nu a) \\
(\nu a)(a_r(\nu a), \pi_1 \mid \pi \mid b, x \notin \text{fn}(\pi_1), b \notin \text{fn}(\pi_2) & \Rightarrow \pi \Rightarrow \rho \}
\end{align*}
\]

Fig. 2. Symmetric Contraction and Commutations

(scope), and the prefixed terms (with) and (of course). We write $\overrightarrow{c(*)}$ to mean $\overrightarrow{c_1(*)} \cdots \overrightarrow{c_n(*)}$, similarly for $\overrightarrow{k(\bar{x} \bar{y} \bar{z})}$, and $\overrightarrow{b,c}$ for $\{b,c_1,\ldots,c_n\}$.

A normalisation step takes place when a cut between two links is eliminated, generating new links so as to preserve the correctness of the structure. We could have chosen, as in the original proof nets, to have explicit cuts, e.g., $\text{CUT}(a, b)$ for a cut between $a$ and $b$, but as in Abramsky’s $\pi$-calculus translation (in Bellin and Scott [5]), we prefer to have implicit cuts to reduce the number of connectives. Thus, each name appears once (as conclusion) or twice (in a cut).

An axiom, written $ab$, is the only link with two distinct conclusions, $a$ and $b$, and it has no premise, and it is symmetric: $ab \Rightarrow ba$. It can be seen as an explicit fusion [24], or a symmetric adaptation of a linear forwarder [11]. Axioms contract with any link with which they share a conclusion. For example, using (Link – A) and (Dis): $(\nu x)(a x \mid x c) \rightarrow (\nu x)(a c \mid x !) \rightarrow ac$. Notice how this reduction has the effect of a local substitution, since there is no irrelevant context, which immediately increases the parallelism that can be attained. Recall also that this term cannot reduce in Solos (see the introduction to [22]) because the monadic communication reacts in the standard way, without symmetry (i.e., $\overrightarrow{b} \mid a x \rightarrow \overrightarrow{a x}$).
but $xa \mid \pi b \not\rightarrow$), when in $\delta$-calculus axioms react with everything; and indeed this is the behaviour of (an explicit, linear) substitution.

Thus, in $\delta$-calculus there is no substitution per se, but in the reduction rules we introduce a convenient abbreviation: $\pi \{ a/x \} \overset{\text{def}}{=} (\nu x)(\pi \mid ax)$, from which we obtain $ax \{ a/x \}$ (or equivalently, $xc \{ a/x \}$) for the above example.

The residual $x^!$ is released so as to interact (and this is another kind of cut) with the enclosing scope of $(\nu x)$, in order to destroy it, since it is needed no more. It can be understood as an asynchronous «deallocation». In fact, contrary to usual process algebras, here a scope restriction defines a box which in proof nets can be understood as an arbitrary sequent $[13]$. Typically, in proof nets both the main and the auxiliary interface (the ports, or doors) are shown, but in $\delta$-calculus we do not need to mention the auxiliary part which coincides with the free names of the boxed term. In particular, the scooping of a name using $(\nu x)\pi$ should be understood as a special type of box with no main door, and with an auxiliary interface equal to $\text{fn}(\pi) \setminus x$. The implicit identification of auxiliary ports serves to facilitate a dynamic nesting of boxes, which drives cut elimination in a style similar (but not equal) to the structural congruence of $\pi$-calculus. In $\delta$-calculus, this restructuring is performed with commutations, see $\Rightarrow$ in Figure 2 to which we return after describing the remaining connectives.

The binary multiplicative links of proof nets, tensor and par, are written $\pi(b, c)$ and $a(b, c)$, respectively, where $a$ is the conclusion and $b, c$ are the premises. Their contraction, with rule $(\otimes - \otimes)$, produces two new axioms, which connect directly the premises of each through the (implicit) cuts that they enable. The significant aspect of this reduction is that it is completely local, and therefore can be performed in parallel to any other redex that may be enabled.

The units are: $\pi$, for tensor, and $a$, for par.

In the case of the additives there is a phenomenon of superposition, as exactly one of the sides will be chosen, left with $\pi_l(b)$ or right with $\pi_r(b)$. It can be seen as a deterministic summation, restricted to terms guarded by the same name. An example of the left case is: $\pi_l(b) \mid a_l(\nu c)cz + a_r(\nu d)c \rho \rightsquigarrow cz \{ b/c \} \rightsquigarrow \supset b^z$. Both components of the summation are boxed, marking a moment of sequentiality, or laziness. Each has a main port which is a bound name, here $c$ and $d$. Once a side is chosen, in that case applying $(\otimes_1 - \otimes)$, the bound name $c$ is extruded and wired to $b$, becoming a scope $(\nu c)(bc \mid cz)$ which is abbreviated as before to $cz \{ b/c \}$. The other side, $a_r(\nu d)c \rho$, vanishes, which is why additives are the natural choice for if-then-else constructs, or objects as summations of methods.

Exponential links are, more or less, the equivalent of replicated terms in $\pi$-calculus and they are responsible for the expressive power of Linear Logic, given that only these can be copied. They can be understood as the embodiment

\[1\] Note that in the intuitionistic interpretation of $\mathbf{I}$, the axiom $a \leftrightarrow x$ is understood as a forwarer, and it does not replace substitution: $(\nu x)(P \mid a \leftrightarrow x) \rightarrow P[a/x]$.\n\[3\] A reference to this type of reaction can also be found in Honda’s work $[16]$, but it is neither typed nor developed in an interpretation, although it utilises the idea of making names explicit in proof net links; it follows Abramsky’s interpretation $[3]$.\n\[4\] A historical note: Milner’s replication $!P$ was inspired by exponentials $[23]$.\n
of storage. The general principle is well-known: these boxes may only contain (free) references to other exponentials, and in particular conclusions of ?-links, a property ensured by typing. Here we can observe an asymmetry, as there is one constructor, $\overrightarrow{\pi}(\nu^c)\pi$, and three destructors of dereliction $?a(b)$, contraction $?a(b\#c)$, and weakening $?a(*)$.

Dereliction, see (?D − !), expanding the abbreviation for $\pi\{b/c\}$, reduces as: $?a(b) | \overrightarrow{\pi}(\nu^c)\pi \rightsquigarrow (\nu^c) (bc | \pi) | a^\dagger$. The exponential box turns into a scope, with an extrusion of $\nu^c$ that encompasses $b$, so that $bc$ can be wired, providing «access» to $\pi$ via $b$; thus the exponential becomes linear. Consider a weakening contraction, using rule (?W − !):

$$\overrightarrow{\pi}(\nu^b).\left(?c(*) | (\nu x \{d(x) | \pi\}) \right) \ ho \rightsquigarrow \ ?c(*) | ?d(*) | a^\dagger \ | \rho$$

We can assume that the exponentials $\overrightarrow{x}$ and $\overrightarrow{y}$ occur in $\rho$, as they must be outside of $\overrightarrow{\pi}$ for typing. For correctness they must also be deleted, which explains the creation of a weakening link for each, irrespective of how they would be used should the box had not been deleted; for example $\overrightarrow{x}$ would be destroyed anyway by $?c(*)$. When there is no auxiliary interface, i.e., no free names, the reduction simply deletes the box as in: $?a(*) | \overrightarrow{\pi}(\nu x).\pi \rightsquigarrow a^\dagger$.

Contraction is implemented with the rule (?C − !): from $\overrightarrow{\pi}(\nu^d).\pi$ we obtain $\overrightarrow{\pi}(\nu^d).\pi \{\overrightarrow{k}^\dagger\}$ and $\overrightarrow{\pi}(\nu^d).\pi \{\overrightarrow{\gamma}^\dagger\}$. The contractions $?k_1(x_1\#y_1) | \cdots | ?k_n(x_n\#y_n)$, abbreviated to $?k(x\#y)$, for each of the free names $\{k_1, \ldots, k_n\}$ (i.e., $\overrightarrow{k}$) of $\overrightarrow{\pi}$, are needed to preserve linearity. This explains the re-binding of $\overrightarrow{k}$ inside the new boxes, so that they refer to the future copies of $\overrightarrow{k}$ at $\overrightarrow{x}$ and $\overrightarrow{y}$, respectively. Besides its inherent complexity, this rule mimics the standard version [13].

Commutations The commuting conversions of Figure 2 serve to enable reduction, similarly to $\Rightarrow$ in $\pi$-calculus. However, in the case of proof nets they facilitate complete cut-elimination [13], i.e., they can also enable reductions that are separated by box boundaries, which in $\pi$-calculus would allow an action guarded under prefix to interact with another from the outside scope. This is not normally allowed, and it follows that even typed $\pi$-calculus terms can deadlock, which immediately excludes the possibility of progress even for some terms that would otherwise be deterministic. Moreover, the standard scope transformations of $\pi$-calculus, extrusion and restriction, are generally unsound in $\delta$-calculus. For example $((\nu a)(\nu b)\pi | \overrightarrow{\pi}(a, d) | dz)$ could be typable, but after a scope extrusion of $\nu b$, $((\nu a)(\nu b)(\pi | \overrightarrow{\pi}(a, d)) | dz)$ would be untypable, because to type the $\otimes$-link $\overrightarrow{\pi}(a, d)$ we need both $\pi$ and $dz$ in its immediate context, and in the resulting term $dz$ is left outside (cf. Figure 3).

This motivates a formulation where — and this is how it reflects in sequent derivations — complete typing environments are commuted, so everything that moves into a scope comes with all the context (the terms) that are needed to type it. The procedure is similar, but coarser, to the standard technique with empires [13]. In the previous example we can bring the outside scope $((\nu a)$ into the inside scope $((\nu b)$ using $(I - \nu\nu)$, obtaining $((\nu a)(\nu b)\pi | \overrightarrow{\pi}(a, d) | dz)$ ⇒
On \( \nu \)-calculus, prefixing, and linear substitution

Perhaps the most attractive feature of prefixing is that it naturally defines a scope for bound names. Indeed, we can recover a familiar notation reminiscent of (asynchronous) \( \pi \)-calculus with a combination of \( \nu \) and axiom, \( \exists \), or other links:

\[
(a | \pi) \xrightarrow{\text{def}} (\nu x)(ax | \pi) \quad \quad \quad (a, y \mid \pi) \xrightarrow{\text{def}} (\nu x y)(a(x, y) | \pi)
\]

Encoded prefixes react to implement the transitive closure of substitutions; written differently, this is monadic communication:

\[
(a \cdot b)(c \cdot \pi) \xrightarrow{\pi} a(c \cdot \pi) \quad \text{or} \quad \pi \{a b|c\}\{c a\} \xrightarrow{\pi} \pi \{a c\}
\]

Polyadic reaction is similar: \((\nu a)(\pi(b, c) \mid a(y, z) \cdot \pi) \xrightarrow{\pi} (\nu y z)(by \mid cz \mid \pi)\). The intrusion of a scope leads to movement of a link \( \pi(b, c) \) closer to some \( \ell(a) \) which,
due to the boundary of $a$, must be in $\pi$; this ensures cut-elimination:

$$(\nu a) (\nu(b,c) \mid x(y, z) \cdot \pi) \Rightarrow (\nu a) x(y, z) \cdot (\nu(b,c) \mid \pi)$$

In conclusion, it seems straightforward to encode a (terminating, deterministic) fragment of a linear $\pi$-calculus with explicit substitutions. To simulate general replication we need a more powerful kind of contraction on a self-replicating exponential, which we outline in Section 3. Non-deterministic summation, sharing, and matching are more involved, and remain as future work.

### 3 Linear Sequent Typing

The typing system of $\delta$-calculus is, modulo the necessary adaptation to handle the explicit identity of formulas, the Linear Sequent Calculus.

**Definition 1 (Linear Types).** The types are given as follows:

$$A, B, C ::= A \otimes B \mid A \notimes B \mid 1 \mid 1 \notimes A \mid 1 \notimes B \mid A \notimes B \mid !A \notimes ?A$$

for tensor, $\notimes$, tensor and $\notimes$ units, with, plus, of course, why not.

**Definition 2 (Duality).** Duality (negation) is defined as follows:

$$1^\bot \overset{\text{def}}{=} \bot, \bot^\bot \overset{\text{def}}{=} 1, (A)^\bot \overset{\text{def}}{=} A^\bot, (A^\bot)^\bot \overset{\text{def}}{=} A, (A \notimes B)^\bot \overset{\text{def}}{=} A^\bot \notimes B^\bot, (A \otimes B)^\bot \overset{\text{def}}{=} A^\bot \otimes B^\bot, (A \notimes B)^\bot \overset{\text{def}}{=} A^\bot \notimes B^\bot$$

Fig. 3. Linear Sequent Typing
Typing judgements denote that term $\pi$ can be assigned the interface $\Gamma$:

$$\text{Sequent: } \vdash \pi \triangleright \Gamma \quad \text{Interface: } \Gamma ::= \emptyset \mid \Gamma, a : A \mid \Gamma, a : [A]$$

When we write $\Gamma, \Delta$ the domains must be disjoint, and $?\Gamma$ means that all elements of $\Gamma$ are of the form $a : ?A$. The rules are shown in Figure 3.

A notable element is that of discharged occurrences $a : [A]$, where $A$ is a type but $[A]$ is an interface component, for which duality is not defined, and stays internal to sequents. These discharged typings should be understood as «pseudo-conclusions», because they are only used to signify that a name has been used and it remains in the environment until it is eliminated, in rule (New), by its binder. Note that since cuts are symmetric, we do not know which type was recorded ($A$ or $A^\perp$), and the correct approach in (New) is to allow either.

The introduction of discharged formulas takes place through several rules. In (Open), they type the residue $a^\dagger$ of a reduction on $a$. In (Cut), to discharge the active name $a$. In remaining links, to discharge a name used in a suspended cut, i.e., a cut between a conclusion of some link and a premise of another; that is the idea of wiring them together. For example, when typing a $\otimes$-link $\pi(b, c)$ in the context of $\pi$ and $\rho$, it should be understood that $b : A$ and $c : B$ come from conclusions of links in those nets (a simple inspection of the rules is convincing of this), and by their composition in the premises of the tensor link they are «used». In fact, these two cuts between the premises of the $\otimes$-link and the two subnets $\pi, \rho$ can never take place before a cut with $\forall$ on $a$; normalisation takes place in levels, with each cut enabling others, and this is the source of causality.

Now we can observe that, if we erase pseudo-conclusions, the interfaces in premises and conclusions of each rule match exactly the Linear Sequent Calculus. After all, the only rule that requires (eliminates) some $[A]$ is the extra-logical (New). To make the point clear, it is easy to verify that the following can be derived (note the abbreviated syntax) using (New) and $(\text{Ax}, \forall, \otimes)$:

$$\vdash \pi \triangleright \Gamma, x : A \quad \vdash \pi \triangleright \Gamma, x : A, y : B \quad \vdash \pi \triangleright \Gamma, x : A \quad \rho \triangleright \Delta, y : B \quad \vdash \pi(x, y) \cdot (\pi | \rho) \triangleright \Gamma, \Delta, a : A \otimes B$$

4 Basic Properties

Proposition 1 The interface of a well-typed term cannot consist solely of discharged formulas.

Proof. By structural induction on the type derivation. A discharged formula is always introduced with non-discharged ones: in all rules, e.g., (Cut), ($\otimes$), we obtain the result by the induction hypothesis, and for (Ax), (1) the result is immediate. This is the general pattern for connectives that have a context.

The following lemma is required for Soundness of $\Rightarrow$ and $\sim\sim$. For $\Rightarrow$ it is applied on the premises of the last rule, which is always (New), e.g., in $(\nu b) ((\nu a) \pi | \rho)$ with $(1 - \nu \nu)$, and similarly for the other cut intrusions.
Lemma 1 (Intrusion).

(a) If $\vdash (\nu a) \pi \mid \rho \triangleright \Gamma$ and $a \notin \text{fn}(\rho)$ then $\vdash (\nu a) (\pi \mid \rho) \triangleright \Gamma$

(b) If $\vdash a_1(\nu c), \pi_1 + a_2(\nu d), \pi_2 \mid \rho \triangleright \Gamma$ and $a, c, d \notin \text{fn}(\rho)$ then $\vdash a_1(\nu c), (\pi_1 \mid \rho) + a_2(\nu d), (\pi_2 \mid \rho) \triangleright \Gamma$

(c) if $\vdash !x(\nu b), \pi_1 \mid !x(\nu d), \pi_2 \mid \rho \triangleright \Gamma, x: [D]$ and $x \in \text{fn}(\pi_1)$ and $b \notin \text{fn}(\pi_2)$, then $\vdash !x(\nu b), (\pi_1 \mid !x(\nu d), \pi_2 \mid \rho) \triangleright \Gamma$

Proof. By induction on the typing derivation, and case analysis on the last rule.

Theorem 1 (Soundness of Commutation). If $\vdash \pi \triangleright \Gamma$ and $\pi \Rightarrow \rho$ then $\vdash \rho \triangleright \Gamma$

Proof. By induction on the typing derivation, and case analysis on the last $\Rightarrow$ rule applied.

Lemma 2 (Encoded Substitution). If $\vdash \pi \triangleright \Gamma, x: A$ and $a \notin \text{fn}(\pi)$ then $\vdash \pi \{a/x\} \triangleright \Gamma, a: A$

Proof. By definition $\pi \{a/x\} \overset{\text{def}}{=} (\nu x) (\pi \mid ax)$, which can be typed with (New) followed by (Cut) on $x$, and for the cut it must be that $\vdash ax \triangleright a: A, x: A^\bot$. Then $\vdash \pi \mid ax \triangleright \Gamma, x: [A], a: A$, and we apply (New) to obtain the result.

Theorem 2 (Soundness of Reduction). If $\vdash \pi \triangleright \Gamma$ and $\pi \rightsquigarrow \rho$ then $\vdash \rho \triangleright \Gamma$

Proof. By induction on the typing derivation, and case analysis on the last $\rightsquigarrow$ rule applied. The absence of substitution is a simplifying factor.

Let $\langle I \rangle$ be $\Gamma$ with names hidden and discharged formulas completely removed, and let $\Gamma$ be a sequent in Linear Logic. Then:

Theorem 3 (Static Correspondence). (a) If $\vdash \pi \triangleright \Gamma$ then $\vdash \langle I \rangle$; and (b) if $\vdash \Gamma$ then there exists $\pi$ such that $\vdash \pi \triangleright \Gamma$ and $\langle I \rangle = \Gamma$

Proof. For part (a), we proceed by induction. For (b), we construct the $\delta$-term (hence a proof net) that corresponds to a sequent proof $\Pi$ of $\Gamma$, by induction, as in Theorem 2.7 of the original article on Linear Logic [13].

Theorem 4 (Cut-Elimination). If $\vdash \pi \triangleright \Gamma$ with $\Gamma = \{a_i: A_i \mid i \in I\}$ then there exists $\rho$ s.t. $\vdash \rho \triangleright \Gamma$, $\pi \rightsquigarrow \rho$, and there are no links in $\rho$ (pairwise) sharing the same name as their conclusion (hence, there is no reducible cut in $\rho$).

Proof. A complete proof of strong normalisation (sN) is beyond the scope of this work, but we can be confident based on the following observations. First, note that by Theorem 2 the interface of a term is preserved under reduction. Then, as our commutation steps clearly allow a cut scope to enter any box: additive, exponential, or another cut scope, it follows that links that cut can be brought at the same level. Note that the interface $\Gamma$ is cut-closed, i.e., there are no discharged formulas $a: [A]$, so cut intrusions (which require an outer scope) can
be applied for all cuts. To establish sN, we ensure that we have not introduced any new device which may cause a term (a) to be stuck, or (b) to produce a contractum of size larger than its generating redex. For (a), we recall the remark on commutations, and for (b), it suffices to observe that the contraction for scope deallocation, \((\nu a)(\pi, a^\dagger) \leadsto \pi\), causes the term to strictly diminish in size. Otherwise, connectives and reduction map exactly to proof nets, but with implicit cuts. However, for any well-typed term \(\pi\) \(\alpha\)-converted s.t. all bound names are distinct, we can obtain a proof net translation such that whenever there are two links \(\lambda_1, \lambda_2\) sharing a name \(a\), we add a cut link \(\text{CUT}(a, b)\) and apply \(\lambda_2[b/a]\), where \(b\) is globally fresh. This is always sound, as names appear twice to facilitate implicit cuts; the extra-logical parts are discarded to complete the translation. It is therefore reasonable to expect that \(\delta\)-calculus is sN.

5 Extensions

Non-terminating processes There is a simple way to allow processes to run forever, by introducing a new kind of exponential that refers to a (future) copy of itself. We add the type rule (cf. Figure 3):

\[
\vdash \pi \vdash \Gamma, a : ? A, b : A \quad \vdash \mu \pi, \nu b : A, \pi \vdash \Gamma, a : ! A \quad (\mu \Pi)
\]

where the (free) name \(a\) appears in \(\pi\) with a type that allows to perform a cut with (a copy of) itself; now names may appear thrice.

Then, taking \(\text{fn}(\pi) = \text{ad}k\), and \(?k(x@y) : \pi \overset{def}{=} (\nu \vec{x} \vec{y}) (\overline{k(x@y)} | \pi)\), we obtain:

\[
?a(b) | \mu \pi, \nu d, \pi \leadsto \overline{k(x@y)} \cdot (\pi{\{b \vec{x}/d \vec{k}\}} | \mu \pi, \nu d, \pi{\{\vec{y}/k\}}) \quad (?, D - \mu!)
\]

\[
?a(c) | \mu \pi, \nu d, \pi \leadsto \overline{k(x@y)} \cdot (\mu \nu \bar{b}, \pi{\{b \vec{x}/d \vec{k}\}} | \mu \pi, \nu d, \pi{\{c \vec{y}/a \vec{k}\}}) | a^\dagger \quad (?, C - \mu!)
\]

\[
?a(\ast) | \mu \pi, \nu d, \pi \leadsto ?k(\ast) | a^\dagger \quad (?, W - \mu!)
\]

Notice that the scope \((\nu a)\) is only deallocated by weakening and contraction, and dereliction unfolds the box, which implies a contractive effect. The commutations present no difficulty. The solution is not perfect: once opened, our \(\mu\)-boxes can only be «internally» reused, so \(\mu \pi, (\nu x : !A), \pi{\{a(\ast)\}} | \pi\) is typable, but \(\mu \pi, (\nu x : !A), \pi x\) is not, as can be easily verified. This is reminiscent of the situation in objects with methods that return \text{self} \[2\], and requires second-order quantification and recursive types. Observe also that by the above typable example, the standard exponential boxes can be simulated, so we could remove them altogether: the (unfolded) copy can be deleted after the box is opened for the first time. However, this comes at the cost of some useless contractions with dereliction, so it is not efficient. Finally, note that a recursive box can refer to standard boxes, since everything is copied as expected and linearity is preserved, so this extension integrates very well with the standard system: it represents a new \textit{execution device}, the feasibility of which was mentioned by Girard in \[13\].
Eliminating the additives  The &-links induce boxes (i.e., prefixes), which are useful as a lazy feature but ultimately impede parallelism. Proof-theoretically, this constitutes a defect of proof nets, which are meant to abstract the bureaucratic commutations that belong in the sequent world. We propose an alternative approach, which is not a real solution (c.f. [15]), but has the advantage of being extremely simple and can be summarised as: completely remove the additive fragment and weakening, and allow anything to be deleted. For this, we add the new connective $a \top$ (deletion, cf. $\epsilon$-link in Interaction Nets [20]) and the type rules (Mix), ($\top$):

\[
\begin{align*}
\vdash \pi \triangleright \Gamma & \quad \vdash \rho \triangleright \Delta \\
\vdash \pi \mid \rho \triangleright \Gamma, \Delta & \quad \vdash a \top \triangleright a : A \\
\end{align*}
\]

(Mix, $\top$)

\[
\begin{align*}
\vdash a^\dagger \triangleright a : [A] & \quad \vdash \epsilon \triangleright \emptyset \\
\vdash a \triangleright a : \bot & \\
\end{align*}
\]

(Open, Nil, $\bot$)

Rule (Mix) immediately enables the simplification of rules that require a «jump» context (cf. Figure 3): (Open), (Nil), ($\bot$). We add a reduction rule for $a \top$:

\[
a \top \mid L(a) \rightsquigarrow \overrightarrow{x^\dagger} \mid a^\dagger \quad \text{fn}(L(a)) = a\overrightarrow{x}
\]

($\top - \text{Link}$)

The function of deletion is simple: it eliminates anything it cuts with, and propagates, preserving typability. For example, $a \top \mid \pi(b, c) \mid \rho \rightsquigarrow b \top \mid c \top \mid a^\dagger \mid \pi \mid \rho$. Assume that the $\otimes$-link is typed with $\pi$ in the left premise and $\rho$ in the right; then the contractum is typed with (Open), (Mix), and two instances of (Cut) for $\pi \mid b \top$ and $\rho \mid c \top$, respectively. Additive behaviours can be encoded with multiplicatives, e.g., $a(x, y) \mid x \top \mid \pi$ will delete all terms connected to $x$, after a cut with a $\otimes$-link. For ($\mu$-)boxes we obtain: $a \top \mid \mu ! \pi(x) \mid \pi \rightsquigarrow \overrightarrow{c^\dagger} \mid a^\dagger$, where \( \text{fn}(\pi) = a\overrightarrow{x^\dagger} \). Note that $a \top \mid a \top \rightsquigarrow a^\dagger$ as expected, and $a \top \mid a^\dagger$ is untypable. A limitation is that there is no more superposition, so choice is multiplicative.

Summary  We shall not elaborate on these extensions, due to space limitations, but the soundness theorems hold (and for the first extension, also Theorem 3, since ($\mu P$) and (P) map to the same logical rule). With $\mu$-boxes, we can no longer speak of cut-elimination, but we conjecture that a cut-progress/deadlock-freedom property holds; we delegate the detailed proofs to a longer version.

6 Related Work and Conclusions

Abramsky [3] was the first to study the relationship between Linear Logic and processes, in the parallel language of proof expressions (PE). In this original interpretation the syntax is not $\pi$-calculus, and indeed the constructs are close to proof net links, but the logical relationship is not as strong as in $\delta$-calculus. For example, there are no commutative contractions, and for this reason reduction evaluates to canonical (i.e., lazy) and not normal forms, from which it follows that it does not enjoy cut-elimination, but the weaker property of determinacy. Also, there is no scope restriction and therefore no bound names, which inhibits
modularity since all names must be chosen *globally* fresh. Moreover, with scope restriction (which effectively defines a box) some commutations would be needed even for lazy evaluation, and it is not obvious how to introduce them in PE. Axioms are interpreted as variables, and do not have the operational meaning of an explicit substitution, contrary to δ-calculus (and proof net) axioms that wire two names together (the conclusions), enabling their *identification*. This explains why substitution is needed in PE but not in δ-calculus.

In PE there are explicit cut-links (*co-equations*) \( t \perp u \), where \( t, u \) are terms. Then, one can understand a tensor link \( \overline{a}(b, c) \) as \( a \perp b \otimes c \), which can interact with \( a \perp x \overline{y} \) resulting in \( b \otimes c \perp x \overline{y} \) and then \( b \perp x, c \perp y \). Instead of the cut-link \( b \perp x \) (resp. \( c \perp y \)), in δ-calculus we use an axiom \( bx \) that nevertheless results in an implicit cut. In the additive fragment the differences are more significant: the \( \oplus \)-link \( a \perp \text{inl}(t) \) indicates that the (arbitrary) term \( t \) is prefixed, which in proof net terms corresponds to a box. In δ-calculus, following a more accurate correspondence, \( \oplus \)-links are more fine-grained and no irrelevant context is attached: the same term becomes \( \overline{a}(b) \mid [b \perp t] \), allowing the parts to interact independently. In summary, it is accurate to say that PE are in principle the first process language that is close to proof nets, but there are important differences which establish the advantages of δ-calculus.

Bellin and Scott [5] studied a π-calculus translation of linear proofs, defined in unpublished lectures by Abramsky; in fact, Milner had also obtained a similar translation, albeit never published [4]. This interpretation is actually quite different than the original proof expressions, and specifically, the terms are not primitive any more, in the sense that for a single connective, a composition of binding, sequencing, and input (resp. output) is used. For instance, a sequent proof with a \( \otimes \)-conclusion maps to a term \((\nu xy)\pi(x,y).((P \mid Q)\mid [P \otimes Q]((P \mid Q)\mid [P \otimes Q]))\), one with a \( \exists \)-conclusion becomes \( a(x,y).P \), and axioms are *forwarders*, \( a(x).b(x) \). Therefore, it represents a regression wrt parallelism due to π-calculus sequentiality (prefixes) and non-local (standard) substitution. Still, one important element of this work was the identification of a more canonical process-algebraic correlate for the symmetric cut, in the form of parallel composition under name restriction. The exponentials were given a rather complicated encoding using *summation*, which indicates that typing with Linear Logic would be problematic, as summation would also map to the additive conjunction rule.

Recently, Honda and Laurent formalised a close correspondence between *Polarised* Proof Nets and π-calculus [17]. However, the variety of proof nets that they used consist solely of exponentials and the type system is not Linear Logic, so their work is not an interpretation. Nevertheless, they illuminate the relationship with the non-deterministic behaviours of π-calculus, which may inform extensions of δ-calculus. In terms of graph-based models of computation, Lafont’s Interaction Nets [19][20] were also inspired by the connectives of (simple sorted) proof nets, and have been used in efficient (parallel) functional reduction. Subsequently, these have evolved into the Differential Interaction Nets of Ehrhard and Regnier [9], which have strong connections with a variation of Linear Logic and have also been applied in resourceful λ-calculi. Ehrhard and Regnier [10] also
presented a very close correspondence between the Acyclic Solos calculus and Differential Interaction Nets, which is not typed with Linear Logic, but shares with δ-calculus the notion of binary actions (solos) corresponding to the multiplicative links. In preliminary work by Laneve, Parrow, and Victor [21], Solo diagrams were briefly shown to be related to untyped, multiplicative proof nets. Indeed, at the untyped multiplicative level Solos is our closest relative, due to the absence of prefix, but it does not support summation, it lacks explicit axioms, and it employs a standard structure congruence that, as we have seen, would not preserve soundness in a linear sequent typing (cf. δ-calculus intrusions).

Caires and Pfenning [6] (with Toninho in [7]) interpreted sessions [26,18] in a variation of Intuitionistic Linear Logic, and obtained a Curry-Howard correspondence, using an almost standard π-calculus (πDILL) as the representation of proofs. In this (monadic) system the output π(y).P induces a tensor type x: A ⊗ B composed of the type A of y (the object of communication) and the type B of the remaining usage of (the subject) x in P. In other words, their interpretation of x: A ⊗ B is sequential: first send some A on x, then do some B on x. Typing is given modulo structural congruence, which results from the need to combine restriction with input/output etc, in their type rules. Operationally, sessions cannot achieve parallelism (in the sense of immediate redices) matching that of δ-calculus (proof nets), and when sessions are interleaved only one of them can immediately reduce. For example, in πDILL the term:

\[(νxy)(π(a).π(b).P | π(m).Q) | x(c).R | y(k).S)\]

must first react on x, after which both x, y may have independent redices. Session variables are not strictly linear, e.g., here π(·) appears twice, which is another manifestation of a causal relationship that cannot be safely broken: in typing the output π(a)..., x is not only a conclusion, but also a premise (along with a) from the usage π(m).Q under the prefix, and likewise for any x in Q. On the logical side, the defect wrt parallelism follows from the fact that every action (a prefix) forces a sequentialisation (a box), and reduction steps correspond to proof normalisation of intuitionistic sequent proofs.

Finally, note that none of the above mentioned interpretations (of Linear Logic) have provisions for unbounded computation or the highly parallel prefix-free additive encoding, which we show how to incorporate in our language without compromising type soundness, even if (in the first case) strong normalisation would be lost. Moreover, both our formulation and the proposed extensions constitute contributions to proof net theory.

Acknowledgments Thanks to Vasco Vasconcelos, Nobuko Yoshida, Kohei Honda, Simon Gay, the members of Gloss (University of Lisbon), and to Bernardo Toninho for clarifications on the example of the Caires-Pfenning interpretation.

References

1. Online appendix of this paper: http://www.di.fc.ul.pt/~dimitris
