

Multiparty Sessions based on Proof Nets

Dimitris Mostrous

LaSIGE, Department of Informatics, Faculty of Engineering
University of Lisbon, Portugal.
dimitris@di.fc.ul.pt

We interpret Linear Logic Proof Nets in a term language based on Solos calculus. The system includes a synchronisation mechanism, obtained by a conservative extension of the logic, that enables to define non-deterministic behaviours and multiparty sessions.

1 Introduction

Since their inception, sessions [11, 19] and multiparty sessions [12] have been gaining momentum as a very useful foundation for the description and verification of *structured interactions*. Interestingly, recent works have established a close correspondence between typed, synchronous pi-calculus processes and sequent proofs of a variation of Intuitionistic Linear Logic [3]. This particular interpretation of Linear proofs is considered a sessions system because it has (for practical purposes) the same type constructors but with a clear logical motivation. In this paper we outline a system based on an interpretation of the proof objects of Classical Linear Logic, namely Proof Nets [8], improving our previous work [16]. The process language resembles Solos [14] and exhibits asynchrony in both input and output. Proof Nets have a number of advantages over sequent proofs, such as increased potential for parallelism and a very appealing graphical notation that could be seen as a new kind of *global type* [12]. Nevertheless, accurate logical interpretations are typically deterministic, which limits their applicability to concurrent programming. However, with a very modest adaptation that enables *synchronisation*, non-deterministic behaviours can be allowed without compromising the basic properties of interest, namely strong normalisation and deadlock-freedom.

Let us distinguish *multiparty behaviours* and the *multiparty session types* (global types) of [12]. A multiparty behaviour emerges when more than two processes can be part of the same session, and this is achieved at the operational level by a synchronisation mechanism such as the multicast request $\bar{a}[2..n](\vec{s}).P$ [12]. We propose a similar mechanism in the form of replications with synchronisation, $!a_1(x_1) \cdots a_n(x_n).P$, which allow a service to be activated with multiple parties. A global type captures the interactions and sequencing constraints of the complete protocol of a program. In our proposal, the equivalent to a global type is the *proof net* of the program. Although our approach is technically very different, we believe that the logical foundations and simpler meta-theory are appealing. We show how pi-calculus channels with i/o type and a multi-party interaction from [12] can be encoded.

2 The Process Interpretation

Syntax The language is inspired by proof nets except that connectives have explicit locations (names). Types are ranged over by A, B, C , with type variables ranging over X, Y . We assume a countable set of *names*, ranged over by a, b, c, x, y, z, r, k . Then, \tilde{b} stands for a sequence b_1, \dots, b_n of length $|\tilde{b}| = n$, and

similarly for types. Processes, P, Q, R , are defined as follows:

$$P ::= \bar{a}(\tilde{A}, \tilde{x}) \quad | \quad a(\tilde{X}, \tilde{y}) \quad | \quad \bar{a} \triangleleft i(b) \quad | \quad a \triangleright \{i(x_i:A_i).P_i\}_{i \in I} \quad | \quad ?\bar{a}(b) \quad | \quad !a_i(x_i:A_i)_{i \in I}.P \\ | \quad ab \quad | \quad X \mapsto A \quad | \quad (\nu a:A)P \quad | \quad (\nu X)P \quad | \quad (P|Q) \quad | \quad \mathbf{0}$$

There are two kinds of *solos*-like [14] communication devices: $a(\tilde{X}, \tilde{y})$ and $\bar{a}(\tilde{A}, \tilde{x})$. In typed processes, we will be using the nullary signals a for $a()$ and \bar{a} for $\bar{a}()$,¹ the binary input $a(b, c)$ (resp. output $\bar{a}(b, c)$) and the *asynchronous* polymorphic input $a(X, b)$ (resp. output $\bar{a}(B, b)$). The explicit substitution, ab , interprets Linear Logic axioms; this is standard in related works [16, 2]. The type alias $X \mapsto A$ is simply a typing device, and the scope of X is restricted with $(\nu X)P$. The branching connective $a \triangleright \{i(x_i:A_i).P_i\}_{i \in I}$, with $I = \{1, \dots, n\}$, written also in the form $a \triangleright \{1(x_1:A_1).P_1 \parallel \dots \parallel n(x_n:A_n).P_n\}$, offers an indexed sequence of alternative behaviours. One of these can be selected using $\bar{a} \triangleleft k(b)$ with $k \in I$. Our notion of replication enables synchronisation, similarly to a multiparty “accept” (cf. [12]). The notation is $!a_i(x_i:A_i)_{i \in I}.P$, written also as $!a_1(x_1:A_1) \cdots a_n(x_n:A_n).P$ with $n \geq 1$. Dually, $?\bar{a}(b)$ can be thought as a “request.”

Free, passive, and active names The *free names* ($\text{fn}(P)$) are defined in the standard way. We just note that the only bound names are a in $(\nu a:A)P$ and the x_i in $a \triangleright \{i(x_i:A_i).P_i\}_{i \in I}$ and $!a_i(x_i:A_i)_{i \in I}.P$. The *passive names* ($\text{pn}(P)$) are defined similarly to $\text{fn}(P)$ except for:

$$\text{pn}(a(\tilde{X}, \tilde{x})) = \text{pn}(\bar{a}(\tilde{A}, \tilde{x})) = \{\tilde{x}\} \quad \text{pn}(\bar{a} \triangleleft i(b)) = \text{pn}(?\bar{a}(b)) = \{b\} \quad \text{pn}(ab) = \emptyset \\ \text{pn}(a \triangleright \{i(x_i:A_i).P_i\}_{i \in I}) = \cup_{i \in I} (\text{pn}(P_i) \setminus \{x_i\}) \quad \text{pn}(!a_i(x_i:A_i)_{i \in I}.P) = \text{pn}(P) \setminus \cup_{i \in I} \{x_i\}$$

The *active names* ($\text{an}(P)$) are defined by $\text{an}(P) = \text{fn}(P) \setminus \text{pn}(P)$. For example, y in $(\nu x)(a(x, y) | xy)$ is not active. (As usual, we assume the name convention.)

Structure Equivalence With \equiv we denote the least congruence on processes that is an equivalence relation, equates processes up to α -conversion, satisfies the abelian monoid laws for parallel composition, the usual laws for scope extrusion, and satisfies the following axioms:²

$$(\nu X)\mathbf{0} \equiv \mathbf{0} \quad (\nu X)P | Q \equiv (\nu X)(P | Q) \quad (X \notin \text{ftv}(Q)) \\ (\nu X)(\nu Y)P \equiv (\nu Y)(\nu X)P \quad (\nu X)(\nu a:A)P \equiv (\nu a:A)(\nu X)P \quad (X \notin \text{ftv}(A)) \\ ab \equiv ba \quad ab | !a_1(x_1:A_1) \cdots a(x:A) \cdots a_n(x_n:A_n).P \equiv ab | !a_1(x_1:A_1) \cdots b(x:A) \cdots a_n(x_n:A_n).P$$

The most notable axiom is the last one, which effects a forwarding, e.g., $?\bar{a}(x) | ab | !b(y).P \equiv^2 ab | ?\bar{a}(x) | !a(y).P$. As can be seen next, the term on the right can now reduce.

Reduction “ \longrightarrow ” is the smallest binary relation on terms such that:

$$\bar{a}(\tilde{A}, \tilde{y}) | a(\tilde{X}, \tilde{x}) \longrightarrow \tilde{X} \mapsto \tilde{A} | \tilde{x}\tilde{y} \quad |\tilde{A}| = |\tilde{X}|, |\tilde{x}| = |\tilde{y}| \quad (\text{R-Com}) \\ \bar{a} \triangleleft k(b) | a \triangleright \{i(x_i:A_i).P_i\}_{i \in I} \longrightarrow P_k\{b/x_k\} \quad k \in I \quad (\text{R-Sel}) \\ \prod_{i \in I} ?\bar{a}_i(b_i) | !a_i(x_i:A_i)_{i \in I}.P \longrightarrow P\{b_i/x_i\}_{i \in I} | !a_i(x_i:A_i)_{i \in I}.P \quad (\text{R-Sync}) \\ (\nu a:A)(ab | P) \longrightarrow P\{b/a\} \quad a \neq b, a \in \text{an}(P) \quad (\text{R-Ax}) \\ P \equiv P' \longrightarrow Q' \equiv Q \Rightarrow P \longrightarrow Q \quad (\text{R-Str}) \quad P \longrightarrow Q \Rightarrow C[P] \longrightarrow C[Q] \quad (\text{R-Ctx}) \\ \text{Contexts in (R-Ctx):} \quad C[\cdot] ::= \cdot \quad | \quad (C[\cdot] | P) \quad | \quad (\nu a:A)C[\cdot] \quad | \quad (\nu X)C[\cdot]$$

¹They have no computational content, but without them reduction leaves garbage axioms of unit type.

²The free type variables ($\text{ftv}(P)$) are defined in a standard way, noting that $\text{ftv}((\nu X)P) = \text{ftv}(P) \setminus X$. The free type variables of a type A ($\text{ftv}(A)$) are also standard and arise from \forall/\exists .

(R-Com) resembles solos reduction [14] but with explicit fusions [7, 16]. Specifically, given two vectors \tilde{x} and \tilde{y} of length n , the notation $\widetilde{x\tilde{y}}$ stands for $x_1y_1 \mid \cdots \mid x_ny_n$ or $\mathbf{0}$ if the vectors are empty. For polymorphism we create type aliases: $\widetilde{X \mapsto A}$ stands for $X_1 \mapsto A_1 \mid \cdots \mid X_n \mapsto A_n$. Combined type and name communication appears also in a synchronous setting [18]. (R-Sel) is standard. In (R-Sync) we synchronise on all a_i , obtaining a form of multi-party session against $?a_1(b_1) \mid \cdots \mid ?a_n(b_n)$. (R-Ax) effects a capture-avoiding name substitution, defined in the standard way. The side-condition $a \neq b$ guarantees that no bound name becomes free; $a \in \text{an}(P)$ ensures that the cut is applied correctly, that is, against two (or more) conclusions.

The Caires-Pfenning axiom reduction (R-Ax) is based on $(\text{va})([a \leftrightarrow b] \mid P) \longrightarrow P\{b/a\}$ ($a \neq b$) from [17], which is similar to the ‘‘Cleanup’’ rule of [1]. However, in an asynchronous language, this rule breaks subject reduction, which motivates our side-condition $a \in \text{an}(P)$. For example, $Q \doteq (\text{vx}, y)(\bar{a}\langle x, y \rangle \mid [x \leftrightarrow b] \mid [y \leftrightarrow c])$ is typable in the system of [5], with conclusion $b : A, c : B \vdash Q :: a : A \otimes B$, but it reduces to $(\text{vy})(\bar{a}\langle b, y \rangle \mid [y \leftrightarrow c])$ which is not typable.³

Types and duality The types, ranged over by A, B, C, D, \dots , are linear logic formulae [8]:

$$A ::= \mathbf{1} \mid \perp \mid A \otimes B \mid A \wp B \mid A \& B \mid A \oplus B \mid !_m A \mid ?_m A \mid \forall X.A \mid \exists X.A \mid X \mid \sim X$$

The *mode* m can be ε (empty) or \star (synchronising): ε is a formality and is never shown; \star is used to enforce some restrictions, but does not generally alter the meaning of types. Negation $\sim(\cdot)$, which corresponds to duality, is an involution on types ($\sim(\sim A) = A$) defined in the usual way (we use the notation from [10]):

$$\begin{aligned} \sim \mathbf{1} &\doteq \perp & \sim \perp &\doteq \mathbf{1} & \sim(A \otimes B) &\doteq \sim A \wp \sim B & \sim(A \wp B) &\doteq \sim A \otimes \sim B \\ \sim(A \& B) &\doteq \sim A \oplus \sim B & \sim(A \oplus B) &\doteq \sim A \& \sim B & \sim(!_m A) &\doteq ?_m \sim A & \sim(?_m A) &\doteq !_m \sim A \\ \sim(\forall X.A) &\doteq \exists X.\sim A & \sim(\exists X.A) &\doteq \forall X.\sim A & \sim(\sim X) &\doteq X \end{aligned}$$

The multiplicative conjunction $A \otimes B$ (with unit $\mathbf{1}$) is the type of a channel that communicates a name of type A and a name of type B , offered by disconnected terms; it can be thought as an ‘‘output.’’ The multiplicative disjunction $A \wp B$ (with unit \perp) is only different in that the communicated names can be offered by one term; this possibility of dependency makes it an ‘‘input.’’ In a standard way, the additive conjunction $A \& B$ is an external choice (branching), and dually additive disjunction $A \oplus B$ is an internal choice (selection). Ignoring modes, the exponential types $!A$ and $?A$ can be understood as a decomposition of the ‘‘shared’’ type in sessions: $!A$ is assigned to a persistent term that offers A ; dually, $?A$ can be assigned to any name with type A so that it can communicate with $!\sim A$. The second-order types $\forall X.A$ and $\exists X.A$ are standard, as is type substitution: $A[B/X]$ stands for A with B for X , and $\sim B$ for $\sim X$.

Judgements and interfaces A judgement $P \triangleright \Gamma$ denotes that term P can be assigned the *interface* Γ . Interfaces, ranging over Γ, Δ , are sequences with possible repetition, defined by:

$$\Gamma ::= \emptyset \mid \Gamma, a : A \mid \Gamma, [a : A] \mid \Gamma, X$$

$a : A$ is standard. A *discharged occurrence* $[a : A]$ indicates that a has been used as A : it serves to protect linearity, since a can no longer be used. Γ, X records that X appears free in the term, ensuring freshness of type variables. $\tilde{a} : \tilde{A}$ stands for $a_1 : A_1, \dots, a_n : A_n$. $? \Gamma$ stands for $\tilde{a} : ?\tilde{A}$, i.e., $a_1 : ?A_1, \dots, a_n : ?A_n$. Similarly, $[\Gamma]$ means $[\tilde{a} : \tilde{A}]$. Let $\text{fn}(a : A) = \text{fn}([a : A]) = a$ and $\text{fn}(X) = \emptyset$, plus the obvious definition for

³The reduction rule is not mentioned in [5], but the type rule is given and one of the authors relayed to me that reduction is assumed to be the same as in [17].

$$\begin{array}{c}
\text{(New)} \quad \frac{P \triangleright \Gamma, [a: A]}{(va: A)P \triangleright \Gamma} \quad \text{(NewX)} \quad \frac{P \triangleright \Gamma, X \quad X \notin \text{ftv}(\Gamma)}{(vX)P \triangleright \Gamma} \quad \text{(Sub)} \quad \frac{\Gamma \preceq \Delta \quad P \triangleright \Delta}{P \triangleright \Gamma} \quad \text{(Str)} \quad \frac{P \equiv Q \quad Q \triangleright \Gamma}{P \triangleright \Gamma} \\
\text{(Ax)} \quad \frac{}{ab \triangleright a: A, b: \sim A} \quad \text{(Cut)} \quad \frac{P \triangleright \Gamma, a: A \quad Q \triangleright \Delta, a: \sim A}{P|Q \triangleright [a: A], \Gamma, \Delta} \quad \text{(OpenCut)} \quad \frac{P \triangleright \Gamma, a: !_m A \quad Q \triangleright \Delta, a: ?_m \sim A}{P|Q \triangleright \Gamma, \Delta, a: !_m A} \\
\text{(CoMix)} \quad \frac{P \triangleright \Gamma, \Theta \quad Q \triangleright \Delta, \Theta \quad \Theta \subseteq \{a: ?_x A\}}{P|Q \triangleright \Gamma, \Delta, \Theta} \quad \text{(1)} \quad \frac{}{\bar{a}() \triangleright a: \mathbf{1}} \quad \text{(L)} \quad \frac{P \triangleright \Gamma}{a() | P \triangleright \Gamma, a: \mathcal{L}} \\
\text{(\otimes)} \quad \frac{P \triangleright \Gamma, b: A \quad Q \triangleright \Delta, c: B}{\bar{a}(b, c) | P|Q \triangleright [b: A, c: B], \Gamma, \Delta, a: A \otimes B} \quad \text{(\wp)} \quad \frac{P \triangleright \Gamma, b: A, c: B}{a(b, c) | P \triangleright [b: A, c: B], \Gamma, a: A \wp B} \\
\text{(\oplus_1)} \quad \frac{P \triangleright \Gamma, b: A}{\bar{a} \triangleleft 1(b) | P \triangleright [b: A], \Gamma, a: A \oplus B} \quad \text{(\&)} \quad \frac{P \triangleright \Gamma, b: A \quad Q \triangleright \Gamma, c: B \quad b, c \notin \text{fn}(\Gamma)}{a \triangleright \{1(b:A).P \parallel 2(c:B).Q\} \triangleright \Gamma, a: A \& B} \\
\text{(\forall)} \quad \frac{P \triangleright \Gamma, b: A \quad X \notin \text{ftv}(\Gamma)}{a(X, b) | P \triangleright [b: A], X, \Gamma, a: \forall X.A} \quad \text{(TyAl)} \quad \frac{P[A/X] \triangleright \Gamma}{X \mapsto A | P \triangleright \Gamma, X} \quad \text{(\exists)} \quad \frac{P \triangleright \Gamma, b: C \quad C = A[B/X]}{\bar{a}(B, b) | P \triangleright [b: C], \Gamma, a: \exists X.A} \\
\text{(!)} \quad \frac{P \triangleright ?\Gamma, \{x_i: A_i\}_{i \in I} \quad \forall i \in I. x_i \notin \text{fn}(?\Gamma) \quad I \neq \emptyset \quad \forall i \geq 2. m_i = \star}{!a_i(x_i: A_i)_{i \in I}. P \triangleright ?\Gamma, \{a_i: !_m_i A_i\}_{i \in I}} \quad \text{(?D)} \quad \frac{P \triangleright \Gamma, b: A}{? \bar{a}(b) | P \triangleright [b: A], \Gamma, a: ?_m A}
\end{array}$$

Figure 1: Linear Logic Typing with Multiparty Promotion

free type variables ($\text{ftv}(\Gamma)$). We consider *well-formed interfaces*, in which only $a: ?A$ can appear multiple times, but $a: ?_x A$ cannot. Moreover, (Γ, X) is well-formed when Γ is well-formed and $\exists \sigma. \sigma(\Gamma) = [\Delta], \Sigma$ such that $X \notin \text{ftv}(\Sigma)$. For example in the (\forall) rule the conclusion is $\Gamma, [x: A], X, a: \forall X.A$ with X possibly free in A .

Subtyping The usual structural rules of Linear Logic are incorporated into the relation $\Gamma \preceq \Delta$:

$$\begin{array}{c}
\Gamma, [a: A] \preceq \Gamma, [a: \sim A] \quad \Gamma \preceq \sigma(\Gamma) \quad \Gamma, a: ?_m A \preceq \Gamma \\
\Gamma, a: ?A \preceq \Gamma, a: ?A, a: ?A \quad \Gamma, a: ?A \preceq \Gamma, a: ?_x A \quad \Gamma, a: !_x A \preceq \Gamma, a: !A
\end{array}$$

The first rule identifies the type of a discharged occurrence and its dual, matching a type annotation which may be $(va: A)$ or $(va: \sim A)$. Then we have *exchange*, *weakening*, *contraction*. The last two axioms alter the mode: we can forget \star in $?_x A$, and dually we can record it on $!A$.

Typing rules can be found in Fig. 1. We type modulo structure equivalence, a possibility suggested by [15] and used in [3]. This is because associativity of “ $|$ ” does not preserve typability, i.e., a cut between P and $(Q|R)$ may be untypable as $(P|Q)|R$; (va) causes similar problems.

In (Cut) the name a is discharged and can then be closed with (New) . (OpenCut) was added for two reasons. First, it is intuitive, since we are not required to close the name, i.e., to fix the number of clients of $!A$, departing from the de facto interpretation of “cut as composition under name restriction.” Second, it is needed for soundness. Take $R \doteq (va: !A)(ab | ?\bar{a}(x) | P | !a(y).Q)$ typed with $R \triangleright \Gamma, b: !A$.

Using (R-Ax) we obtain $R \longrightarrow ?\bar{b}(x) \mid P\{b/a\} \mid !b(y).Q$, which is *only* typable with the same interface by using (OpenCut); with (Cut) we obtain $\Gamma, [b: !A]$.⁴

Asynchronous messages can encode standard sessions (see [4, 5]): $\bar{a}(b, c)$ with type $A \otimes B$ maps to the session type $!_s \sim A.B$ or $!_s \sim B.A$. Dually, $a(x, y)$ with $\sim A \wp \sim B$ maps to $?_s \sim A.\sim B$ or $?_s \sim B.\sim A$. To write processes in standard sessions style, with reuse of names (e.g., $\bar{a}b; a(x); \mathbf{0}$), we introduce abbreviations that use the second component for a 's continuation:

$$\begin{array}{ll} \bar{a}b; P \doteq (\nu x, y)(\bar{a}(x, y) \mid xb \mid P\{y/a\}) & a(x); P \doteq (\nu x, y)(a(x, y) \mid P\{y/a\}) \\ a \triangleleft l_k; P \doteq (\nu x)(\bar{a} \triangleleft k(x) \mid P\{x/a\}) & a \triangleright \{l_i.P_i\}_{i \in I} \doteq a \triangleright \{i(x_i).P_i\{x_i/a\}\}_{i \in I} \\ \bar{a}B; P \doteq (\nu x)(\bar{a}(B, x) \mid P\{x/a\}) & a(X); P \doteq (\nu X, x)(a(X, x) \mid P\{x/a\}) \end{array}$$

It is easy to check that linear redices commute with all other redices, and therefore a “real” prefix would not have any effect on computation except to make it more sequential.

The rule (!) implements an extension of the logic:

$$\frac{? \Gamma, A}{? \Gamma, !A} \text{ (promotion)} \quad \text{becomes} \quad \frac{? \Gamma, A_1, \dots, A_n}{? \Gamma, !A_1, \dots, !A_n} \text{ (multi-promotion)}$$

Actually we need to employ some restrictions on this rule, which is why all conclusions except the first must have a \star -mode. Since there is no contraction for $?_*$,⁵ all the $a_i: ?_* \sim A_i$ ($i \geq 2$) will come from terms with just one call to the session.⁶ The first conclusion, $a_1: !_m A_1$, can have standard mode ($m_1 = \varepsilon$), which allows a client's call with $a_1: ? \sim A_1$ to be connected to (i.e., to depend on) other calls on a_1 . In this way we provide a *hook* for one client to participate in another instance of the same session, and this facilitates a form of *dynamic join*. We return to this concept in the first example.

Finally, the sidecondition in (&, !) forbids premises from having multiple copies of a name (e.g., $x: ?A, x: ?A$) which should be removed in the conclusion; essentially it forces contractions (by \preceq). Other rules are immune by the well-formedness of $\Gamma, [a: A]$.⁷

Expressiveness & Properties The system is an extension of proof nets, in process form, so it can encode System F, inductive sessions (using second-order features), etc. Due to space limitations we only show two examples: (a) how shared channels can be simulated with synchronisation; (b) the (two Buyer, one Seller) protocol from [12].

a) channels Non-determinism can be expressed by sharing a channel between multiple competing processes trying to send and receive messages. This is impossible with existing logical sessions systems, and more generally if we follow the logic “by the book.” A channel a with i/o type $(A, \sim A)$, i.e., that exports two complementary capabilities A and $\sim A$, can be encoded by the two names a_1 and a_2 in $!a_1(x:A)a_2(y:\sim A).xy \triangleright a_1: !A, a_2: !_* \sim A$. The channel is used by terms with $a_1: ? \sim A$ or $a_2: ?_* A$, and there can be multiple instances of each, giving rise to critical (non-deterministic) pairs. Moreover, A can be linear, i.e., we can communicate linear values through shared channels, which is a novel feature. For example:

$$\begin{array}{l} ?\bar{a}_1(b_1) \mid ?\bar{a}_2(b_2) \mid ?\bar{a}_2(b_3) \mid !a_1(x:A)a_2(y:\sim A).xy \mid P \mid R \mid S \\ \longrightarrow^{(a)} b_1 b_2 \mid ?\bar{a}_2(b_3) \mid !a_1(x:A)a_2(y:\sim A).xy \mid P \mid R \mid S \\ \longrightarrow^{(b)} b_1 b_3 \mid ?\bar{a}_2(b_2) \mid !a_1(x:A)a_2(y:\sim A).xy \mid P \mid R \mid S \end{array}$$

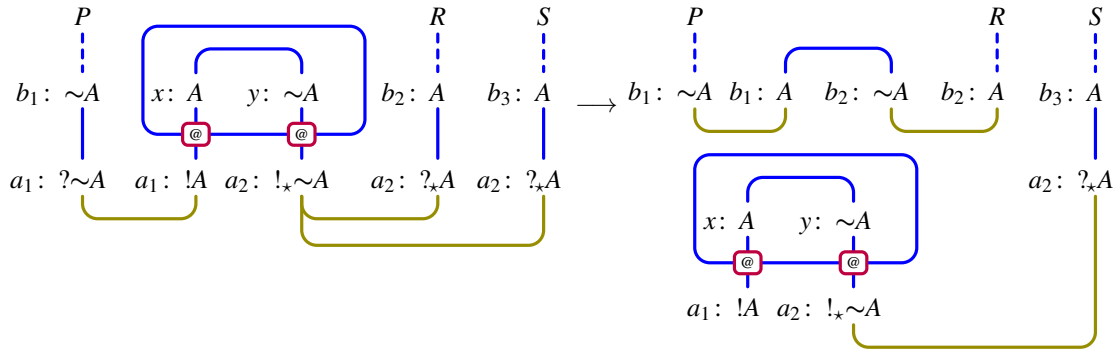
⁴ Several works [17, 5, 20, 2] would not enjoy subject reduction if this example could be transferred: their cut rule requires $(\nu b)(\dots)$, which is here missing. These works don't have “Mix” (here: (CoMix)), which we used in the example; but this should be checked, since “Mix” can be encoded with a new conclusion $c: \mathcal{L} \otimes \mathcal{L}$ [8, p. 100].

⁵More accurately: contraction of $?_* \sim A_i$ is *multiplicative*.

⁶In the sense that two calls can never depend on each other.

⁷It is subtle but due to the variable convention, (&) is actually immune too; the condition serves for clarity.

First, note that confluence is lost: assume P, R, S cannot reduce and it becomes obvious. The graphical notation with a reduction of the first possibility is depicted below.



It is possible that P has another call to a_1 , but by the restriction on $?_*$ -types there cannot be a “trip” from b_2 to a_2 , as this would lead to a cycle. Concretely, if R has another call to a_2 , then it is from a part disconnected to b_2 , and similarly for S ; see (CoMix).⁸

b) multiparty interactions The (two Buyer, one Seller) protocol from [12] is shown below, with insignificant adaptations, using the previously explained abbreviations (we omit some signals for $\mathbf{1}/\mathcal{L}$):

$$\begin{aligned}
 \text{Buyer1} &\doteq (vb_1) (? \bar{a}_1(b_1) \mid \bar{b}_1 \text{“The Art of War”}; b_1(\text{quote}); (vz)(\bar{b}_1 z; \mathbf{0} \mid \bar{z} \text{quote}/2; P_1)) \\
 \text{Buyer2} &\doteq (vb_2) (? \bar{a}_2(b_2) \mid b_2(\text{quote}); b_2(z); z(\text{contrib}); b_2 \triangleleft \text{ok}; \bar{b}_2 \text{“SW12 3AZ”}; b_2(\text{date}); P_2) \\
 \text{Seller} &\doteq !a_1(x_1) a_2(x_2). \left(\begin{array}{l} x_1(\text{title}); \bar{x}_1 \in 20; \bar{x}_2 \in 20; x_1(z); \bar{x}_2 z; \\ x_2 \triangleright \{\text{ok}.x_2(\text{address}); \bar{x}_2 \text{“7/Feb”}; Q \parallel \text{quit}.\mathbf{0}\} \end{array} \right)
 \end{aligned}$$

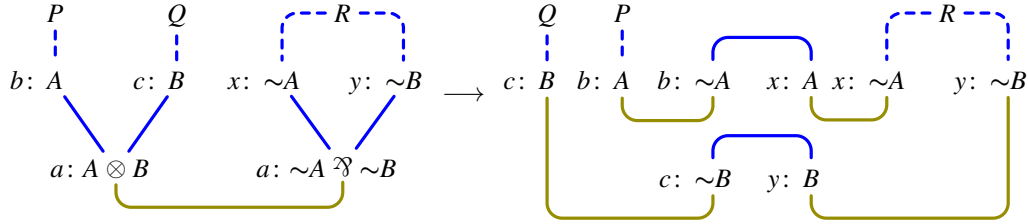
We note that the simplicity of the example has not been sacrificed, compared to the code in [12]. One difference is that we passed z from Buyer1 to Buyer2 *through* Seller using $x_1(z); \bar{x}_2 z$, when in [12] all names are known to all participants. We do not employ the *global types* of [12], but there is a *proof net* for Buyer1 | Buyer2 | Seller, not shown due to space constraints, and we postulate that:

The proof net can serve as an alternative notion of global type.

Outline of results The expected soundness result for reduction, $P \triangleright \Gamma$ and $P \longrightarrow P'$ implies $P' \triangleright \Gamma$, is obtained in a standard way, but fails without the \star -mode. Strong Normalisation (sN), i.e., $P \triangleright \Gamma$ implies that *all* reduction sequences from P are finite, is shown by an adaptation of the reducibility candidates technique from [8]. The loss of confluence complicates the proof, which is in fact obtained for an extended (confluent) reduction relation using a technique of [6], from which we derive as a corollary the result. For sN we prove the (initially) stronger property of *reducibility* [8], which can also serve as a very strong progress guarantee.

A Curry-Howard correspondence can be obtained easily for a fragment of the language. For the multiplicative, additive, and second-order cut-elimination we only need to perform extra axiom cuts (i.e., substitutions). For exponentials, we restrict replications to a single input and simulate the actual copying (with contraction links) that takes place in proof nets with sharing and sequentialised cut-elimination steps. Indeed, there is still a loss of parallelism compared to standard proof nets, but the term language is more realistic. We show just one case of cut-elimination, the cut $(\otimes \text{---} \wp)$, implemented by $\bar{a}(b, c) \mid a(x, y) \longrightarrow bx \mid cy$, adding appropriate contexts (P, Q, R) :

⁸In general, derelictions can be connected through their premises; try with two copies of $a: ?(A \oplus \sim A)$.



3 Conclusion

We claim that our language is simpler and proof-theoretically more appealing than related works such as [3]: structured interactions take place as expected (*fidelity*), but parallelism is not inhibited by the use of prefix, which cannot anyway alter the result in a deterministic setting. It is really a question of proof nets vs. sequent proofs, and in logic the first are almost always preferable. Even with synchronisation and the induced non-determinism, the system we propose retains good properties, for example it seems to be the first approach to multiparty behaviours that enjoys strong normalisation. Finally, our notion of *proof net as global type* seems to be a reasonable solution for logically founded multiparty sessions.

In relation to Abramsky's interpretation [1], it is close to proof nets *with boxes*, i.e., to a completely synchronous calculus. Moreover, it is not so friendly syntactically, it does not have a notion of bound name, copying of exponentials is explicit (no *sharing*), and of course it is completely deterministic. An interesting future direction would be to obtain a *light* variation of our system, e.g., following [9]. Then we could speak of implicit complexity for multiparty sessions, similarly to what has been done in [13] for binary sessions. Due to space restrictions, more examples and all proofs have been omitted. These will appear in a longer version, see <http://www.di.fc.ul.pt/~dimitris/>.

References

- [1] Samson Abramsky (1993): *Computational Interpretations of Linear Logic*. *Theor. Comput. Sci.* 111(1-2), pp. 3–57. Available at [http://dx.doi.org/10.1016/0304-3975\(93\)90181-R](http://dx.doi.org/10.1016/0304-3975(93)90181-R).
- [2] Luís Caires, Jorge A. Pérez, Frank Pfenning & Bernardo Toninho (2013): *Behavioral Polymorphism and Parametricity in Session-Based Communication*. In Matthias Felleisen & Philippa Gardner, editors: *ESOP, Lecture Notes in Computer Science 7792*, Springer, pp. 330–349. Available at http://dx.doi.org/10.1007/978-3-642-37036-6_19.
- [3] Luís Caires & Frank Pfenning (2010): *Session Types as Intuitionistic Linear Propositions*. In Paul Gastin & François Laroussinie, editors: *CONCUR, Lecture Notes in Computer Science 6269*, Springer, pp. 222–236. Available at http://dx.doi.org/10.1007/978-3-642-15375-4_16.
- [4] Romain Demangeon & Kohei Honda (2011): *Full Abstraction in a Subtyped pi-Calculus with Linear Types*. In Joost-Pieter Katoen & Barbara König, editors: *CONCUR, Lecture Notes in Computer Science 6901*, Springer, pp. 280–296. Available at http://dx.doi.org/10.1007/978-3-642-23217-6_19.
- [5] Henry DeYoung, Luís Caires, Frank Pfenning & Bernardo Toninho (2012): *Cut Reduction in Linear Logic as Asynchronous Session-Typed Communication*. In Patrick Cégielski & Arnaud Durand, editors: *CSL, LIPIcs 16, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik*, pp. 228–242. Available at <http://dx.doi.org/10.4230/LIPIcs.CSL.2012.228>.
- [6] Thomas Ehrhard & Olivier Laurent (2010): *Interpreting a finitary pi-calculus in differential interaction nets*. *Inf. Comput.* 208(6), pp. 606–633. Available at <http://dx.doi.org/10.1016/j.ic.2009.06.005>.

- [7] Philippa Gardner, Cosimo Laneve & Lucian Wischik (2007): *Linear forwarders*. *Inf. Comput.* 205(10), pp. 1526–1550. Available at <http://dx.doi.org/10.1016/j.ic.2007.01.006>.
- [8] Jean-Yves Girard (1987): *Linear Logic*. *Theor. Comput. Sci.* 50, pp. 1–102. Available at [http://dx.doi.org/10.1016/0304-3975\(87\)90045-4](http://dx.doi.org/10.1016/0304-3975(87)90045-4).
- [9] Jean-Yves Girard (1998): *Light Linear Logic*. *Inf. Comput.* 143(2), pp. 175–204. Available at <http://dx.doi.org/10.1006/inco.1998.2700>.
- [10] Jean-Yves Girard (2011): *The Blind Spot*. European Mathematical Society. Available at <http://dx.doi.org/10.4171/088>.
- [11] Kohei Honda, Vasco Thudichum Vasconcelos & Makoto Kubo (1998): *Language Primitives and Type Discipline for Structured Communication-Based Programming*. In Chris Hankin, editor: *ESOP, Lecture Notes in Computer Science* 1381, Springer, pp. 122–138. Available at <http://dx.doi.org/10.1007/BFb0053567>.
- [12] Kohei Honda, Nobuko Yoshida & Marco Carbone (2008): *Multiparty asynchronous session types*. In George C. Necula & Philip Wadler, editors: *POPL*, ACM, pp. 273–284. Available at <http://doi.acm.org/10.1145/1328438.1328472>.
- [13] Ugo Dal Lago & Paolo Di Giamberardino (2011): *Soft Session Types*. In Bas Luttik & Frank Valencia, editors: *EXPRESS, EPTCS* 64, pp. 59–73. Available at <http://dx.doi.org/10.4204/EPTCS.64.5>.
- [14] Cosimo Laneve & Björn Victor (2003): *Solos In Concert*. *Mathematical Structures in Computer Science* 13(5), pp. 657–683. Available at <http://dx.doi.org/10.1017/S0960129503004055>.
- [15] Robin Milner (1992): *Functions as Processes*. *Mathematical Structures in Computer Science* 2(2), pp. 119–141. Available at <http://dx.doi.org/10.1017/S0960129500001407>.
- [16] Dimitris Mostrous (2012): *Proof Nets in Process Algebraic Form*. Available at <http://www.di.fc.ul.pt/~dimitris/>.
- [17] Jorge A. Pérez, Luís Caires, Frank Pfenning & Bernardo Toninho (2012): *Linear Logical Relations for Session-Based Concurrency*. In: *ESOP '12*, pp. 539–558. Available at http://dx.doi.org/10.1007/978-3-642-28869-2_27.
- [18] Benjamin C. Pierce & Davide Sangiorgi (2000): *Behavioral equivalence in the polymorphic pi-calculus*. *Journal of the ACM* 47(3), pp. 531–584. Available at <http://doi.acm.org/10.1145/337244.337261>.
- [19] Kaku Takeuchi, Kohei Honda & Makoto Kubo (1994): *An Interaction-based Language and its Typing System*. In Constantine Halatsis, Dimitris G. Maritsas, George Philokyprou & Sergios Theodoridis, editors: *PARLE, Lecture Notes in Computer Science* 817, Springer, pp. 398–413. Available at http://dx.doi.org/10.1007/3-540-58184-7_118.
- [20] Philip Wadler (2014): *Propositions as sessions*. *J. Funct. Program.* 24(2-3), pp. 384–418. Available at <http://dx.doi.org/10.1017/S095679681400001X>.