On the Concurrent Meaning of Logical Correctness

Marc de Visme[1] and Damiano Mazza[2]

[1] LIP, UMR 5668 CNRS - ENS Lyon - UCB Lyon 1 - INRIA 5668
devisme@clipper.ens.fr
[2] CNRS, UMR 7030, LIPN, Université Paris 13, Sorbonne Paris Cité
Damiano.Mazza@lipn.univ-pairs13.fr

Abstract. Following Girard, proof nets naturally suggest the introduction of more general proof-like objects, called nets (or proof structures in Girard's original terminology), which may not be logically correct but which may still have non-trivial computational behavior. For instance, Ehrhard and Laurent's encoding of the pi-calculus in differential interaction nets maps a host of processes to incorrect nets. It is then natural to ask which processes are mapped to proof nets or, in other words, what meaning does logical correctness have when seen through Ehrhard and Laurent's encoding. We answer this question by showing that logical correctness ensures a form of deadlock-freedom.

1 Introduction

Nets in proof theory. Proof nets are one of the main novelties introduced by linear logic in proof theory. They are a graphical syntax for proofs which has, over sequent calculus, the advantage of greatly reducing the "bureaucracy" of the cut-elimination procedure. For example, the composition of three lemmas $A \vdash B$, $B \vdash C$ and $C \vdash D$ yields two syntactically distinct proofs of $A \vdash D$ in sequent calculus, but only one proof net, because proof nets are able to express that the two compositions (technically, the two cuts) actually happen "in parallel".

Proof nets also brought a more radical, albeit less known innovation: they unveiled the existence of "incorrect nets", *i.e.*, proof-like objects which are logically wrong and yet are capable of non-trivial computational interactions, via cut-elimination. More precisely, when one defines proof nets, one is led to naturally introduce a wider class of objects, which we simply call *nets* (*proof structures* in Girard's terminology [10]). Perhaps surprisingly, the cut-elimination procedure is defined on all nets, so in principle all nets exhibit computational behavior. However, cut-elimination is guaranteed to hold (*i.e.*, all cuts may be eliminated) only for those nets which are logically correct, *i.e.*, which come from sequent calculus proofs. These logically correct nets are the ones we call proof nets.

Differential linear logic and the π-calculus. Through the years, the relationship between linear logic and concurrency has been explored from several angles. Let us mention Abramsky's pioneering work [2, 1], Bellin and Scott's encoding [3]

and the recent line of work initiated by Caires and Pfenning [4, 5] and extended by Wadler [23]. Here, we are concerned with the approach brought forth by the introduction of differential linear logic, due to Ehrhard and Regnier [9]. Unlike linear logic nets, the nets associated with such a logical system, called *differential interaction nets*, are naturally endowed with a non-deterministic cut-elimination procedure. This non-determinism is interesting because it is controlled by the exponential modalities (connectives $!(-)$ and $?(-)$), rather than being "wild" as in classical sequent calculus.

There are essentially two results relating differential linear logic and process calculi: an exact correspondence found by Honda and Laurent [11] between a fragment of the private, localized asynchronous π-calculus and a fragment of polarized differential nets; and Ehrhard and Laurent's encoding of the π-calculus in differential nets [8] (which may be more easily understood via an encoding of the solos calculus [7]). Both results make an essential use of the wider notion of net: in general, a process does not correspond to/is encoded by a proof net; rather, it may very well be the case that the induced net is not logically correct.

The above observation is the starting point of this work: what processes correspond/are mapped to proof nets? In other words, what does logical correctness have to say about concurrent processes? Actually, this question is fully answered by Honda and Laurent: in their framework, proof nets precisely match the processes of the *control π-calculus* [12], a deterministic process calculus capable of encoding the $\lambda\mu$-calculus in a fully abstract way. This means that, when restricted to correct nets, Honda and Laurent's correspondence does not cover "really concurrent" processes. Fortunately, Ehrhard and Laurent's encoding is more general than Honda and Laurent's and the answer to our question, which was not known prior to this work, is more interesting when the input of that encoding is taken into account.

Correctness and deadlock-freedom. Answering our question directly for Ehrhard and Laurent's encoding is a bit disappointing: the encoding is quite complex and too many processes end up being mapped to incorrect nets because of trivial reasons. Our first step was therefore to take Honda and Laurent's correspondence, which is extremely well behaved, and amend it in order to integrate the idea behind Ehrhard and Laurent's encoding, which gives it its higher expressiveness. The result is pleasantly simple: on the process side, we have a fragment of the local asynchronous π-calculus (*i.e.*, we no longer restrict to internal mobility) very similar to Honda and Laurent's; on the logical side, we have a fragment of polarized differential linear logic which extends Honda and Laurent's essentially by adding the mix rule; between them, an encoding $[\![-]\!]$ mapping processes to nets which is much simpler than Ehrhard and Laurent's. The presence of mix allows more flexibility for dealing with parallel inputs: for instance, $!x.P \mid !y.Q$ is correct in our framework but not in Honda and Laurent's (for linear logic experts: with mix, sequents are allowed to have more than one positive conclusion).

A high-level answer to our question would be that the main property ensured by logical correctness is a certain form of deadlock-freedom, namely the absence of cyclic input/output dependencies, such as $\nu(x, y)(x.\overline{y} \mid y.\overline{x})$ (Theorem 2). The

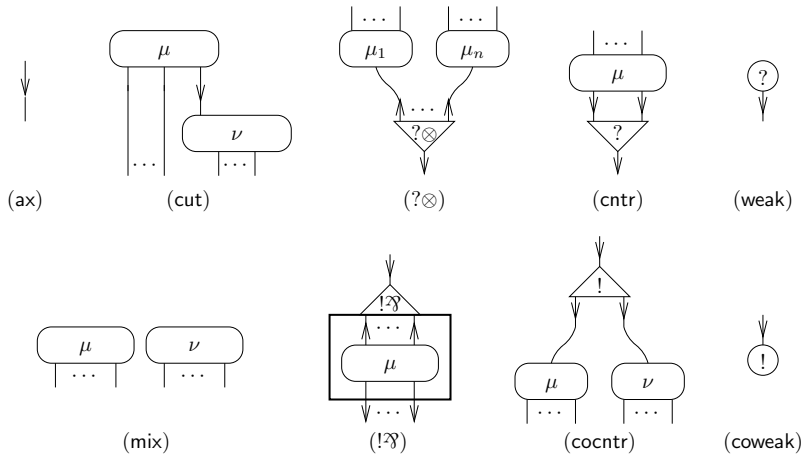


Fig. 1. Sequentializable nets. The case (mix_0) of the empty net is not depicted.

reader acquainted with correctness criteria for proof nets will not be surprised: correctness is all about absence of cycles! What logical correctness does *not* ensure is the absence of configurations such as $\nu x\overline{x}$, in which a buffer is waiting forever to be emptied. Usually, such configurations are also considered as deadlocked [13]; in our framework, one needs to introduce some kind of type system (on top of correctness) to exclude them.

2 Nets and Proof Nets

Definition 1 (net, sequentializable net). *A net is a graph-like object whose edges are directed and called* wires *and whose nodes are called* cells*. Wires are not necessarily attached to cells, i.e., a wire is allowed to be cyclic, or to have one or both of its ends unattached. These "dangling" ends are called* free ports *of the net. Free ports have a polarity which depends on the orientation of the wire to which they belong: they are* positive *if it is incoming,* negative *if it is outgoing. Each cell carries either a* negative symbol*,* $?\otimes$ *or* $?$*, or a* positive symbol*,* $!\parr$ *or* $!$*. Negative (resp. positive) cells have a distinguished outgoing (resp. incoming) wire attached, called* principal port*; the number and orientation of the other attached wires must be as follows: any* $n \in \mathbb{N}$ *for* $?\otimes$ *(outgoing) and* $!\parr$ *(incoming),* 2 *or* 0 *for* $?$ *(incoming) and* $!$ *(outgoing). Cells of kind* $?$ *(resp.* $!$*) are called* contractions *(resp.* cocontractions*) when binary,* weakenings *(resp.* coweakenings*) when nullary. Furthermore, each* $!\parr$ *cell comes with a* box*, which contains a subnet whose free ports are all negative.*

Nets are always considered up to associativity of (co)contractions and neutrality of (co)weakenings with respect to (co)contractions.

A net is sequentializable *if it may be obtained inductively by the rules of Fig. 1 (including the rule, not depicted, stating that the empty net is sequentializable).*

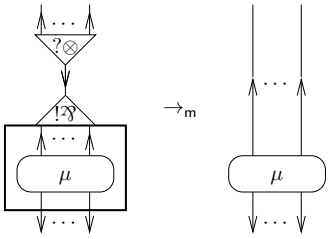


Fig. 2. Multiplicative reduction; the arities of the ?⊗ and !⅋ cells are required to match.

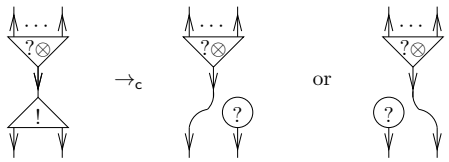


Fig. 3. Choice reduction; this reduction is non-deterministic.

That figure showcases the graphical representation we adopt for nets, with the following conventions: when the orientation of a free port is not drawn, it means that it may be either negative or positive; the principal port of a cell is at the "tip" of the triangle representing it, or the only port in the case of (co)weakenings, which are represented as circles; finally, the (!⅋) case shows how we depict boxes.

The inductive cases of Fig. 1 exactly correspond to the sequent calculus rules of (a polarized fragment of) differential linear logic with mix. In this respect, sequentializable nets are to be seen as nothing more than a graphical syntax for sequent calculus proofs, with formulas/types erased.

Definition 2 (reduction). *Reduction (or* cut-elimination*) for nets is defined by means of the graph-rewriting rules of Figures 2 through 5. In each case, the rule may be applied anywhere within a net, as long as it is not inside a box. One removes the left hand side of the rule (the* redex*) and replaces it with the right hand side, an operation which is sound because the free ports and their orientation are preserved by the rules.*

Note that, because of Fig. 3, reduction of nets is not confluent. It is also non-terminating: the pure λ-calculus may be easily encoded in nets, essentially via Girard's translation of intuitionistic logic in linear logic [10]. Later we will also see an encoding in nets of a fragment of the π-calculus, which is expressive enough to embed the λ-calculus (and more). Also observe that reduction is defined for all nets, not just the sequentializable ones.

Definition 3 (proof net). *Let μ be a net. A* shallow path *of μ is a directed path no portion of which is contained in a box of μ. A* correctness path *of μ is*

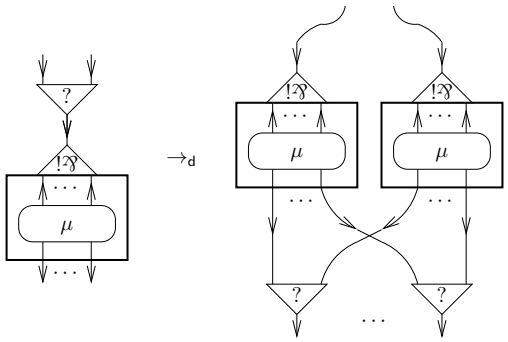


Fig. 4. Duplication reduction.

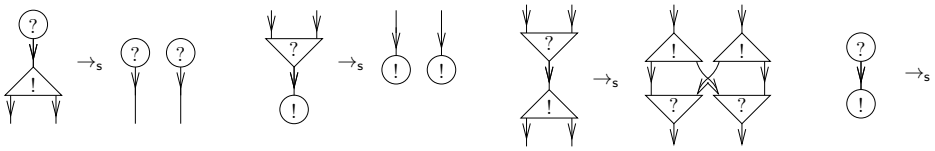


Fig. 5. Structural reduction.

a sequence $\varphi_1, \ldots, \varphi_n$ *of shallow paths of* μ *such that, for all* $1 \leq i < n$, φ_i *ends in the principal port of a* $!\parr$ *cell* c *and* φ_{i+1} *starts with one of the outgoing ports of the box associated with* c. *A* proof net *is a net* μ *such that:*

- μ *contains no cyclic correctness path;*
- *inductively, the net contained in each box of* μ *is a proof net.*

It is immediate from the definition that a subnet of a proof net is a proof net. This fact will be useful in the sequel.

The following is one of the central results in the theory of polarized proof nets, originally due to Laurent [15].

Lemma 1. *A net is sequentializable iff it is a proof net.*

Proof. The forward direction is a straightforward induction. For the converse, Laurent's proof works without changes by treating cocontraction cells as his tensor cells (these are like binary $?\otimes$ cells but with the same orientation as cocontraction cells), which have the same behavior with respect to polarity and sequentialization, as is obvious by comparing ($?\otimes$) and (cocntr) of Fig. 1. □

As observed by Vaux [22], the above result is false in full polarized differential linear logic because of codereliction cells, which we carefully avoided here. In that framework, one needs to resort to the usual (and more complex) Danos-Regnier criterion.

The following is another key property of proof nets: correctness is preserved by reduction. In light of Lemma 1, point 1 is trivial (reduction is cut-elimination in sequent calculus), but it is easily proved independently of it. Point 2 states that correctness is also preserved by *expansion* if we restrict to duplication and structural steps; this is less often remarked but will be useful for us.

Lemma 2. *Let μ be a proof net. Then:*

1. *if $\mu \to \mu'$, then μ' is a proof net;*
2. *if $\mu' \to_{\mathsf{sd}} \mu$, then μ' is a proof net.*

Proof. For point 1, observe that reduction preserves directed paths from right to left, *i.e.*, a directed path crossing the right hand side of a rule induces a directed path in the left hand side, between the same free ports. Then, if there is a cycle in μ', there is a cycle in μ. For point 2, observe that the converse is true: the reductions in Figures 4 and 5 preserve directed paths from left to right. □

Note that the proof of point 2 actually works for *every* reduction rule. This is a very unusual situation in nets, which is due to the polarized framework. In full linear logic, an incorrect net *may* reduce to a proof net (normally some paths *are* lost from left to right in the multiplicative step). Since we do not need such a strong "reversibility" property, we state point 2 in its present, restricted form, which holds in full generality.

3 Processes as Nets

We fix a countably infinite set of *names* $x, y, z \ldots$ and denote by $\widetilde{x}$ finite sequences of names. We consider a fragment of the asynchronous localized polyadic π-calculus ALπ [20]:

$$P, Q ::= \mathbf{0} \;\Big|\; \overline{x}\langle\widetilde{y}\rangle \;\Big|\; !x(\widetilde{y}).P \;\Big|\; P \mid Q \;\Big|\; \nu x P,$$

where, in $!x(\widetilde{y}).P$, we ask that no $z \in \mathsf{fn}(P)$ is the subject of an input (usually this is required only for the names in $\widetilde{y}$). Structural equivalence is defined as expected: the parallel operator is associative, commutative and has unit $\mathbf{0}$; $\nu x \nu y P \equiv \nu y \nu x P$ and both processes are written more concisely as $\nu(x, y)P$; and we have the scope extrusion rule $\nu x P \mid Q \equiv \nu x(P \mid Q)$ as long as $x \notin \mathsf{fn}(Q)$. Reduction is also defined as customary: it is the closure of the rule

$$\overline{x}\langle\widetilde{y}\rangle \mid !x(\widetilde{z}).P \longrightarrow P\{\widetilde{y}/\widetilde{z}\} \mid !x(\widetilde{z}).P$$

(where it is required that $|\widetilde{y}| = |\widetilde{z}|$) under the evaluation contexts $C ::= \{\cdot\} \;\Big|\; C \mid P \;\Big|\; \nu x C$ and under structural congruence: $P \equiv P' \longrightarrow Q' \equiv Q$ implies $P \longrightarrow Q$.

We observe that the above fragment of ALπ is essentially the one considered by Honda and Laurent [11], the only difference being the liberalization to arbitrary asynchronous outputs (Honda and Laurent only consider *bound* asynchronous outputs $\nu\widetilde{y}(\overline{x}\langle\widetilde{y}\rangle \mid P)$, *i.e.*, they further restrict to internal interaction).

We say that an occurrence of a name $x \in \mathsf{fn}(P)$ is

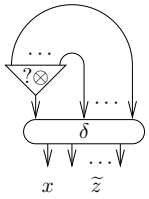
– *positive* if it is the subject of an input;
– *negative* if it is the subject or object of an output.

Furthermore, we say that $x \in \mathsf{fn}(P)$ appears *neutrally* if it occurs both positively and negatively.

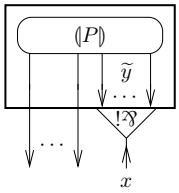
Let us see now how processes may be encoded as nets. First, we define a pre-encoding, denoted by $(\!|\cdot|\!)$, having the following property: given a process P, $(\!|P|\!)$ has exactly one negative (resp. positive) free port for each free name of P occurring negatively (resp. positively). Such names are annotated next to the corresponding free port. Therefore, if x occurs neutrally in P, $(\!|P|\!)$ will have two free ports labelled by x, one negative and one positive.

The pre-encoding is defined inductively:

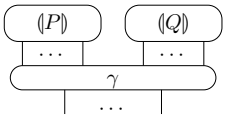
– $(\!|\mathbf{0}|\!)$ is the empty net.
– $(\!|\overline{x}\langle y_1, \ldots, y_n\rangle|\!)$ is the net



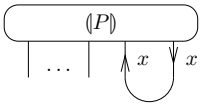
where we write $\widetilde{z}$ for the sequence without repetitions containing the names appearing in $y_1, \ldots, y_n$ except x. The net δ consists, if necessary, of contractions joining the wires corresponding to those names among $x, y_1, \ldots, y_n$ which are equal.
– $(\!|!x(\widetilde{y}).P|\!)$ is the net



– $(\!|P \mid Q|\!)$ is the net



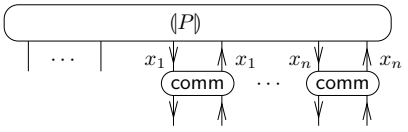
where γ consists, if necessary, of contractions and cocontractions joining the wires corresponding to the free names shared by P and Q.
– $(\!|\nu x P|\!)$ is the net



If there is no negative (resp. positive) free port labelled by x in $(\!|P|\!)$ (because x does not occur negatively (resp. positively) in P) then such a free port is introduced by adjoining a weakening (resp. coweakening) cell.

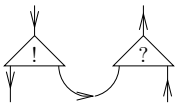
We then define the encoding $[\![\cdot]\!]$ as

$[\![P]\!]$:=



where $x_1, \ldots, x_n$ are exactly the names occurring neutrally in P and

comm :=

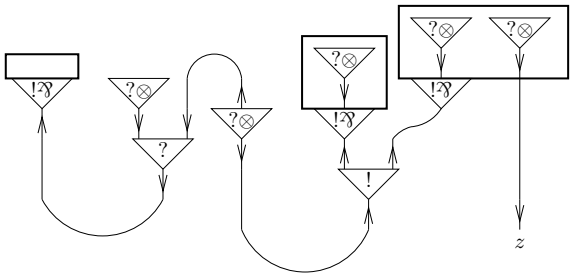


For example, if

$$E := \nu(x,y)(!y \mid \overline{y} \mid \overline{x}\langle y\rangle \mid !x(w).\overline{w} \mid !x(w).(\overline{w} \mid \overline{z})),$$

then

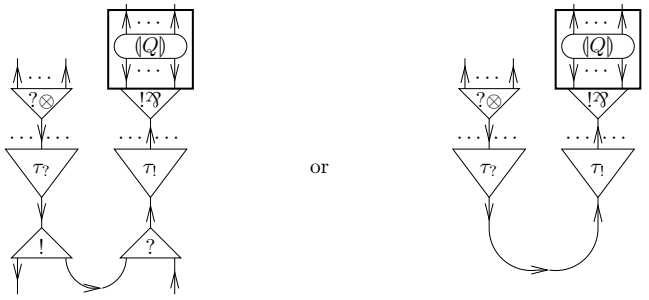
$[\![E]\!] = (\!|E|\!) =$



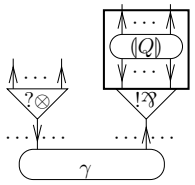
The above encoding is a blend of the correspondence between processes and nets shown by Honda and Laurent [11] and the encoding of Ehrhard and Laurent [8]. This is its main property:

Proposition 1. *If $P \longrightarrow P'$, then $[\![P]\!] \to^* \mu \,{}^*_{\mathsf{sd}}\!\leftarrow [\![P']\!]$.*

Proof. Let $P \longrightarrow P'$ because of an interaction between an output $\overline{x}\langle\widetilde{y}\rangle$ and an input $!x(\widetilde{z}).Q$. It is not hard to verify that, under this hypothesis, $[\![P]\!]$ contains a subnet of the form



according to whether x is free (left) or not (right), where $\tau_?$ (resp. $\tau_!$) is a tree of ? (resp. !) cells. In both cases, the net reduces, by means of $\to_{\mathsf{s}}$ steps, to a net of the form



where γ is essentially what Ehrhard and Laurent call a *communication zone* [8]. As in their work, the ?⊗ and !⅋ cell are able to "meet" through the communication zone, by means of $\to_{\mathsf{d}}$ steps for !⅋ and $\to_{\mathsf{c}}$ for ?⊗ (making the suitable non-deterministic choices), so that the actual interaction may take place by means of a $\to_{\mathsf{m}}$ step. In case $\tau_?$ and $\tau_!$ are both equal to a wire, the net thus obtained is equal to $[\![P']\!]$. Otherwise, we must "undo" the extra copies of the !⅋ cell (and its box) that may have been produced by the interaction with $\tau_?$ (via ${}_{\mathsf{d}}\!\leftarrow$ steps) and "undo" the communication zone (via ${}_{\mathsf{s}}\!\leftarrow$ steps). □

We observe that the converse property fails: $[\![P]\!]$ may perform reductions having no counterpart in P. This is an unavoidable problem [16], but it does not prevent the encoding to be meaningful: it merely tells us that correctness will only be a sufficient and not necessary criterion for ensuring deadlock-freedom (or any dynamic property at all). Typically, $[\![P]\!]$ may deadlock, whereas P does not.

4 Processes and Correctness

We consider *correctness judgments* of the form $\Gamma \vdash_{\tilde{d}} P : \widetilde{p}$, where:

- P is a process;
- $\widetilde{p}$ and $\tilde{d}$ are repetition-free sequences of names;
- Γ is a sequence of pairs of the form $(x; \widetilde{y})$ such that:
  - $\widetilde{y}$ is a (possibly empty) repetition-free subsequence of $\widetilde{p}$;
  - for any other pair $(x'; \widetilde{y}')$ occurring in Γ, $x \neq x'$;
- if $\Gamma = (x_1; \widetilde{y}_1), \ldots, (x_n; \widetilde{y}_n)$ and $\widetilde{x} := x_1, \ldots, x_n$, then $\widetilde{x}$, $\widetilde{d}$ and $\widetilde{p}$ are pairwise disjoint.

Furthermore, all sequences are treated modulo arbitrary reordering, so they are treated like finite sets.

Definition 4 (correctness). *A process P is* correct *if a judgment of the form $\Gamma \vdash P : \widetilde{p}$ may be derived from the rules of Fig. 6.*

The rules of Fig. 6 are implicitly assumed to ensure that the derived judgments are valid, *i.e.*, that all linearity constraints are met. So, for instance, in the out rule, $x, \widetilde{y}$ is supposed to be repetition-free. This means that, if we want

$$\frac{}{\vdash \overline{x}\langle\widetilde{y}\rangle : x, \widetilde{y}}\ \mathsf{out} \qquad \frac{\vdash P : \widetilde{p}, \widetilde{y}}{(x;\widetilde{p}) \vdash\ !x(\widetilde{y}).P : \widetilde{p}}\ \mathsf{!in}$$

$$\frac{}{\vdash \mathbf{0} :}\ \mathsf{mix}_0 \qquad \frac{\Gamma \vdash_{\tilde{d}} P : \widetilde{p} \quad \Delta \vdash_{\tilde{e}} Q : \widetilde{q}}{\Gamma, \Delta \vdash_{\tilde{d},\tilde{e}} P \mid Q : \widetilde{p}, \widetilde{q}}\ \mathsf{par} \qquad \frac{\Gamma \vdash_{\tilde{d},x} P : \widetilde{p}}{\Gamma \vdash_{\tilde{d}} \nu x P : \widetilde{p}}\ \mathsf{res}$$

$$\frac{\Gamma \vdash_{\tilde{d}} P : \widetilde{p}}{\Gamma \vdash_{\tilde{d}} P : \widetilde{p}, x}\ \mathsf{weak} \qquad \frac{\Gamma \vdash_{\tilde{d}} P : \widetilde{p}, x, y}{\Gamma \vdash_{\tilde{d}} P\{z/x, y\} : \widetilde{p}, z}\ \mathsf{cntr}$$

$$\frac{\Gamma \vdash_{\tilde{d}} P : \widetilde{p}}{\Gamma, (x;) \vdash_{\tilde{d}} P : \widetilde{p}}\ \mathsf{coweak} \qquad \frac{\Gamma, (x;\widetilde{p}), (y : \widetilde{q}) \vdash_{\tilde{d}} P : \widetilde{o}}{\Gamma, (z;\widetilde{p},\widetilde{q}) \vdash_{\tilde{d}} P\{z/x, y\} : \widetilde{o}}\ \mathsf{cocntr}$$

$$\frac{\Gamma, (x^+;\widetilde{p}), (y_1;\widetilde{q}_1, x^-), \ldots, (y_n;\widetilde{q}_n, x^-) \vdash_{\tilde{d}} P : \widetilde{o}, x^- \quad x^- \notin \widetilde{p}, \Gamma}{\Gamma, (y_1;\widetilde{q}_1,\widetilde{p}), \ldots, (y_n;\widetilde{q}_n,\widetilde{p}) \vdash_{\tilde{d},x} P\{x/x^+, x^-\} : \widetilde{o}}\ \mathsf{com}$$

Fig. 6. Correctness rules for processes.

to prove $P := \overline{x}\langle x, x\rangle$ correct, we need to first introduce $\overline{x}\langle x_1, x_2\rangle$ and then derive the correctness of P by means of two cntr rules. Similarly, in the par rule, and in light of the observation below concerning free names, P and Q are assumed to share no free name. If we want to prove the correctness of $x \mid \overline{x}$, we must first consider $x' \mid \overline{x}$ and then apply a com rule. With this in mind, it is a straightforward exercise to show that the example process E of p.8 is correct.

The following invariants are immediate; they are not so useful but help understanding the system. Let $(x_1;\widetilde{y}_1), \ldots, (x_n;\widetilde{y}_n) \vdash_{\tilde{d}} P : \widetilde{p}$ be derivable. Then:

- $\mathsf{fn}(P) \subseteq \{x_1, \ldots, x_n, \widetilde{d}, \widetilde{p}\}$;
- every $x_i \in \mathsf{fn}(P)$ appears only positively;
- every $y \in \mathsf{fn}(P) \cap \widetilde{p}$ appears only negatively;
- every $z \in \mathsf{fn}(P) \cap \widetilde{d}$ appears neutrally.

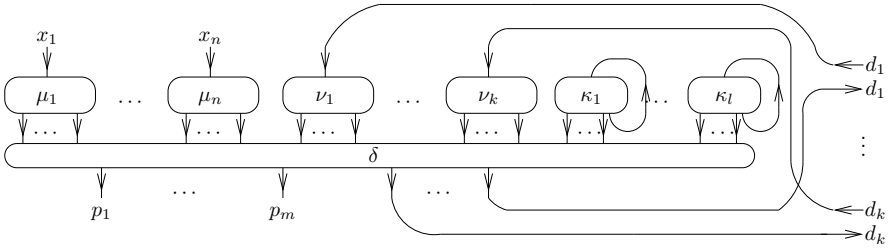
So, to insist on the above remark on linearity, the rules cocntr, cntr and com are used to contract names that occur positively, negatively and neutrally, respectively.

Intuitively, $(x_1;\widetilde{y}_1), \ldots, (x_n;\widetilde{y}_n) \vdash_{\tilde{d}} P : \widetilde{p}$ means that P uses the names $\widetilde{x}, \widetilde{d}, \widetilde{p}$ with the polarity indicated above and, moreover, the names $\widetilde{y}_i$ (which are in $\widetilde{p}$, hence used as subject or object of output prefixes in P) are "guarded" by x_i, in the sense that they are under a top-level input prefix of subject x_i. A more precise description is given in the following proof, which shows that the rules of Fig. 6 exactly capture logical correctness, through the encoding of Sect. 3:

Theorem 1. *A process P is correct iff $[\![P]\!]$ is a proof net.*

Proof. For the forward implication, we start by observing that, if $\widetilde{d}$ are the names occurring neutrally in P, then $[\![P]\!]$ is a proof net iff $(\!|\nu(\widetilde{d})P|\!)$ is a proof net. This is because the comm nets introduced to obtain $[\![P]\!]$ from $(\!|P|\!)$ provide exactly

the same connections/paths as the wires introduced to obtain $(\!|\nu(\widetilde{d})P|\!)$ from $(\!|P|\!)$. Suppose now that, if $(x_1;\widetilde{y}_1),\ldots,(x_n;\widetilde{y}_n) \vdash_{\tilde{d}} P : \widetilde{p}$, with $\widetilde{p} = p_1,\ldots,p_m$ and $\widetilde{d} = d_1,\ldots,d_k$, then $(\!|P|\!)$ has the following shape:



Under this hypothesis, call *virtual correctness path* of $(\!|P|\!)$ a sequence $\varphi_1,\ldots,\varphi_h$ of correctness paths of $(\!|P|\!)$ such that, for all $1 \leq i < h$, φ_i ends in an outgoing free port labelled by d_i and φ_{i+1} starts in the incoming free port labelled by d_i itself. In other words, the virtual correctness paths of $(\!|P|\!)$ are exactly the correctness paths of $(\!|\nu(\widetilde{d})P|\!)$. Now, we claim that, if there is a derivation of $(x_1;\widetilde{y}_1),\ldots,(x_n;\widetilde{y}_n) \vdash_{\tilde{d}} P : \widetilde{p}$, then $(\!|P|\!)$ has the above shape and:

1. it is a proof net;
2. δ consists of wires and contraction cells only;
3. for all $1 \leq i \leq k$, there is no virtual correctness path from the incoming free port labelled by d_i to the outgoing free port labelled by d_i itself.

This is obviously enough to infer that $(\!|\nu(\widetilde{d})P|\!)$ is a proof net, hence we conclude by the above observation. The proof of the above claim is by induction on the last rule of the derivation, which is straightforward as long as one does a little bit of "induction loading", *i.e.*, we need to add the following point:

4. for all $1 \leq i \leq n$, $q \in \widetilde{y}_i$ iff there exists $1 \leq j \leq m$ such that $q = p_j$ and there is a virtual correctness path from an outgoing free port of μ_i to the outgoing free port labelled by p_j.

(As anticipated above, this explains more thoroughly the meaning of typing judgments).

For the converse, we start by making the following claim: if $[\![P]\!]$ is a proof net, then $(\!|P|\!)$ is a proof net. This is a straightforward induction on P. Then, we suppose $[\![P]\!]$ to be a proof net and reason again by induction on P. The cases $P = \mathbf{0}$ and $P = \overline{x}\langle\widetilde{y}\rangle$ are trivial (these processes are always correct). The case $P = !x(\widetilde{y}).Q$ is immediate, thanks to the above claim. Let $P = Q \mid R$. By inspecting the definition of $[\![Q \mid R]\!]$, we see that, if we consider suitable renamings Q', R' of Q, R ("unsharing" the common free names), we have that $[\![P']\!]$ and $[\![Q]\!]'$ are proof nets. We then apply the induction hypothesis and easily derive a correctness judgment for P. Let $P = \nu x Q$. By the above claim, $(\!|P|\!)$ is a proof net. Now, the only difference between $(\!|P|\!)$ and $[\![Q]\!]$ is that in, the latter, one finds a comm net where there is a wire in the former. Since this does not affect cycles, $[\![Q]\!]$ is a proof net, and we conclude by applying the induction hypothesis. □

Definition 5 (circular deadlock). *A* dependency cycle *in a process P is a non-empty sequence $x_1, \dots, x_n \in \mathsf{fn}(P)$ such that $P \equiv \nu\widetilde{z}(!x_1(\widetilde{y}_1).Q_1 \mid \cdots \mid !x_n(\widetilde{y}_n).Q_n \mid R)$ and, for all $1 \leq i < n$, $x_i \in \mathsf{fn}(Q_{i+1})$ and $x_n \in \mathsf{fn}(Q_1)$. A* circular deadlock *is a closed process $P \equiv \nu\widetilde{x}Q$ such that $\widetilde{x}$ is a dependency cycle in Q.*

Let us conclude this section by stating the main property ensured by correctness.

Theorem 2. *Let P be correct. Then:*

1. *no subprocess of P has a dependency cycle;*
2. *if $P \longrightarrow P'$, then P' is correct.*

Proof. For point 1, it is immediate to see that, if P has a dependency cycle, then $[\![P]\!]$ has a cycle, therefore it is not a proof net, and therefore P cannot be correct by Theorem 1. Since subnets of proof nets are proof nets, this also applies to arbitrary subprocesses of P.

For point 2, Proposition 1 tells us that $[\![P]\!] \to^* \mu \overset{*}{\underset{\mathsf{sd}}{\leftarrow}} [\![P']\!]$; now, by Theorem 1, $[\![P]\!]$ is a proof net, so $[\![P']\!]$ is a proof net by Lemma 2, hence P' is correct again by Theorem 1. □

Corollary 1 (deadlock freedom). *Let P be a closed correct process. Then, if $P \longrightarrow^* Q$, Q is not a circular deadlock.*

Proof. By induction on the number n of reduction steps from P to Q. If $n = 0$, we conclude by point 1 of Theorem 2. If $n > 0$, we have $P \longrightarrow P'$, and we may conclude by applying the induction hypothesis to P', thanks to point 2 of Theorem 2. □

5 Examples

The aim of this section is to acquire an idea of how much we can capture with correct processes. Let us start by observing that correct processes need not be deterministic: for instance, we invite the reader to check that $\nu x(\overline{x} \mid !x.\overline{a} \mid !x.\overline{b})$ is correct.

5.1 Functional programming

We consider a $\lambda\mu$-calculus with a parallel operator and a "multi-application":

$$M, N ::= x \;\Big|\; \lambda x.M \;\Big|\; M[N_1, \dots, N_k] \;\Big|\; \mu a.[b]M \;\Big|\; \mathbf{0} \;\Big|\; M \mid N$$

As usual, the parallel operator is assumed to be associative, commutative and with neutral element $\mathbf{0}$. Reduction is defined from the rules (we write $\boldsymbol{N}$ for $[N_1, \dots, N_k]$)

$$(\lambda x.M)\boldsymbol{N} \longrightarrow M\{N_i/x\}, \qquad \mathbf{0}\boldsymbol{N} \longrightarrow \mathbf{0}, \qquad (M \mid N)\boldsymbol{P} \longrightarrow M\boldsymbol{P} \mid N\boldsymbol{P},$$

$$(\mu a.[b]C\{[a]M_1,\dots,[a]M_n\})\boldsymbol{N} \longrightarrow \mu a.[b]C\{[a]M_1N_{i_1},\dots,[a]M_nN_{i_n}\},$$

closed under weak call-by-name evaluation contexts $E ::= \{\cdot\} \mid EM \mid \mu a.[b]E \mid E \mid M$. In the first and last rule, $i, i_1, \dots, i_n \in \{1,\dots,k\}$ are arbitrary and not necessarily distinct: the first rule non-deterministically picks one N_i and substitutes it to x; the last rule picks n terms among $N_1,\dots,N_k$, possibly using the same term more than once and discarding some terms. Because of these rules, the calculus is not confluent. This extension of the $\lambda\mu$-calculus may be encoded in our fragment of ALπ as follows:

$$\begin{aligned}
[\![x]\!]u &:= \overline{x}\langle u\rangle \\
[\![\lambda x.M]\!]u &:= \nu a(\overline{u}\langle a\rangle \mid !a(x,v).[\![M]\!]v) \\
[\![M[N_1,\dots,N_k]]\!]u &:= \nu v([\![M]\!]v \mid !v(a).\nu z(\overline{a}\langle z,u\rangle \mid !z(w).[\![N_1]\!]w) \\
&\qquad\qquad \mid \cdots \mid !v(a).\nu z(\overline{a}\langle z,u\rangle \mid !z(w).[\![N_k]\!]w)) \\
[\![\mu a.[b]M]\!]u &:= [\![M]\!]b\{u/a\} \\
[\![\mathbf{0}]\!]u &:= \mathbf{0} \\
[\![M \mid N]\!]u &:= [\![M]\!]u \mid [\![N]\!]u
\end{aligned}$$

The replicated prefixes $!a(x,v)$ in the encoding of abstraction and $!v(a)$ in the encoding of application are not necessary, *i.e.*, one would normally use $a(x,v)$ and $v(a)$, but these are not available in our fragment of ALπ. Of course this is observationally equivalent because those replicated prefixes are actually used linearly.

Proposition 2. *For all M, $[\![M]\!]u$ is correct.*

Proof. By induction on M. □

Modulo the superfluous replications, the above encoding is just an extension of (the asynchronous variant of) Sangiorgi's encoding of the call-by-name λ-calculus given in [20]. Indeed, although the above extension of the $\lambda\mu$-calculus was obtained by looking at proof nets, another way of looking at it is to see it as a "reverse engineering" of a parallel liberalization of Sangiorgi's encoding: one takes the image of that encoding, closes it under parallel composition and adds the possibility of having an arbitrary number of concurrent arguments in the encoding of an application (which is quite natural from the π-calculus point of view). From this, one may "read back" a λ-calculus mapping to such processes, treating parallel composition homomorphically. Adding μ is straightforward.

So (classical) functional programs are always correct, and non-determinism is compatibe with correctness. Apart from this, the above extension of the $\lambda\mu$-calculus does not have much interest *per se*, except for the possibility, which we have not investigated but that we do not find unlikely, that it actually characterizes the expressiveness of the "correct ALπ", in the same sense as in Honda, Yoshida and Berger's work for $\lambda\mu$ [12], *i.e.*, we would not be suprised if the above embedding is fully abstract in the correct fragment of ALπ.

5.2 Locks

A particularly economic way of implementing a lock in the π-calculus is to see it as a buffer containing at most one value: in $\nu l(P_1 \mid \cdots \mid P_n \mid \bar{l})$, each P_i competes to read from l, and puts back a value once it has done. Unfortunately, this simple solution is not correct from our point of view: P_i will typically be of the form $l.Q_i\{\bar{l}\}$, *i.e.*, Q_i contains $\bar{l}$, inducing a dependency cycle. Consider instead

$$\mathsf{Lock} := !a(z).\nu v(\overline{p}\langle v\rangle \mid v.\overline{z}\langle z\rangle).$$

A process using the above lock will be of the form $P_i := !p(v).Q_i\{\overline{v}\}$, *i.e.*, it waits for the signal from the lock, then performs some operations, upon completion of which sends a release signal via the channel received from the lock process (the replicated input prefix in P_i is forced by our syntax, it would of course be more natural to use a non-replicated input). It is easy to check that the process

$$\nu(p,a)(P_1 \mid \cdots \mid P_n \mid \overline{a}\langle a\rangle \mid \mathsf{Lock})$$

is correct (assuming the Q_i are). It reduces to

$$\nu(p,a,v)(P_1 \mid \cdots \mid P_n \mid Q_i\{\overline{v}\} \mid v.\overline{a}\langle a\rangle \mid \mathsf{Lock})$$

in which a non-deterministically chosen process is ready to execute, while the others are waiting for it to release the lock.

6 Discussion

Related work. Deadlock-freedom is extensively studied in the framework of the π-calculus, let us mention at least [13, 14, 18]. These are sophisticated type systems for deadlock-detection and our simple framework cannot hope to compete with them. In this respect, it is important to stress that the motivation of this work is to investigate the concurrent meaning of logical correctness, without any a priori idea of what this may be. We did not aim at deadlock-detection; it is naturally given by logical correctness and it is not surprising that systems designed *ad hoc* for that purpose perform better.

We have amply mentioned Honda and Laurent's [11] and Ehrhard and Laurent's [8] work, on which ours is based; we hope the relationship is clear. Concerning Caires and Pfenning's line of work [4, 5], as well as Wadler's classical version [23], the approach here is entirely different: we explicitly avoid using types, because our goal is to study "pure" logical correctness, in the structural sense given by proof nets, and this does not need formulas/types to be expressed. In particular, there is no claim here that linear logic has anything to do with session types (although, of course, we know it does!). However, it is worthwhile mentioning that our correctness is more expressive than Caires and Pfenning's or Wadler's. In particular, conflict (as in two processes competing to receive one message) is fully allowed.

Perspectives. The results of this paper may (and should) be extended, along several different directions. First of all, it must be mentioned that restricting to replicated prefixes is not anodyne: it has the effect of making our fragment of ALπ *confusion-free*, in the sense of [19, 21]. While this works well with differential linear logic [16], it is arguably unreasonable from the perspective of concurrency. This restriction is actually a design choice: non-replicated inputs need codereliction, and we already observed how this breaks the usual principles of polarization, making correctness more complex to formulate, so we preferred to stick to the simpler, easier-to-present formulation adopted here.

Of course, by the second author's results [16], introducing confusion in the source π-calculus is not anodyne either: differential interaction nets cannot express confusion. However, this is not necessarily a dead-end: on the one hand, as already pointed out after Proposition 1, the limited expressiveness does not prevent us from formulating sufficient criteria for deadlock freedom; on the other hand, should one wish to try and achieve also necessity, then *multiport* differential interaction nets (in the style of [6]), which are able to soundly encode the π-calculus, may be an interesting route to pursue, although one must first tackle the (interesting!) question of correctness in presence of multiports.

Another direction of further research concerns the addition of types: one may of course immediately endow our framework with a simply-typed discipline. This provides clash-avoidance (processes of the form $\overline{x}\langle\widetilde{y}\rangle \mid x(\widetilde{z}).P$ which cannot reduce because $|\widetilde{y}| \neq |\widetilde{z}|$), but not much else. A more interesting perspective is to consider the general approach to intersection types recently given in [17]. Such an approach yields, via our encoding, an intersection type system for (a fragment of) the π-calculus, which is likely to be able to detect also deadlocks of the form $\nu x\overline{x}$.

We want to stress that such type systems may be formulated as type decorations of the "sequent-style" rules of Fig. 6. This shows the importance of Theorem 1: it allows us to formulate correctness directly on the π-calculus syntax, without referring to proof nets. Indeed, this paper could have been written without ever mentioning proof nets, basing everything on Fig. 6. The usefulness of proof nets crops up in Theorem 2: a direct proof of this result from Fig. 6 would be quite laborious.

Let us end with a note on the nature of deadlocks captured by logical correctness. It is important to understand that this is a "may" notion of deadlock, not "must". For instance, a process like $\nu(x,y)(!x.\overline{y} \mid !y.\overline{x} \mid \overline{x})$ is a circular deadlock as per Definition 5 but would perhaps not be considered deadlocked since the circular dependency may be broken by the rightmost $\overline{x}$. This is an inevitable consequence of sticking to logical correctness: an incorrect net cannot become a proof net by embedding it in a bigger net (a logical mistake cannot be corrected by continuing the proof). So, if one wants to capture "must" deadlock-freedom, something other than pure logic must be sought.

Acknowledgements. The second author was partially supported by French ANR project ELICA (ANR-14-CE25-0005).

References

1. Abramsky, S.: Computational Interpretations of Linear Logic. Theoretical Computer Science 111(1-2), 3–57 (1993)
2. Abramsky, S.: Proofs as processes. Theoretical Computer Science 135(1), 5–9 (1994)
3. Bellin, G., Scott, P.J.: On the pi-Calculus and Linear Logic. Theoretical Computer Science 135(1), 11–65 (1994)
4. Caires, L., Pfenning, F.: Session Types as Intuitionistic Linear Propositions. In: Gastin, P., Laroussinie, F. (eds.) Proceedings of CONCUR 2010. Lecture Notes in Computer Science, vol. 6269, pp. 222–236. Springer (2010)
5. Caires, L., Pfenning, F., Toninho, B.: Linear logic propositions as session types. Mathematical Structures in Computer Science 26(3), 367–423 (2016)
6. Dorman, A., Mazza, D.: A hierarchy of expressiveness in concurrent interaction nets. In: D'Argenio, P.R., Melgratti, H.C. (eds.) Proceedings of CONCUR. Lecture Notes in Computer Science, vol. 8052, pp. 197–211. Springer (2013)
7. Ehrhard, T., Laurent, O.: Acyclic Solos and Differential Interaction Nets. Logical Methods in Computer Science 6(3) (2010)
8. Ehrhard, T., Laurent, O.: Interpreting a Finitary Pi-Calculus in Differential Interaction Nets. Information and Computation 208(6), 606–633 (2010)
9. Ehrhard, T., Regnier, L.: Differential Interaction Nets. Theoretical Computer Science 364(2), 166–195 (2006)
10. Girard, J.Y.: Linear Logic. Theor. Comput. Sci. 50(1), 1–102 (1987)
11. Honda, K., Laurent, O.: An exact correspondence between a typed pi-calculus and polarised proof-nets. Theor. Comput. Sci 411(22-24), 2223–2238 (2010)
12. Honda, K., Yoshida, N., Berger, M.: Process types as a descriptive tool for interaction - control and the pi-calculus. In: Proceedings of RTA-TLCA. pp. 1–20 (2014)
13. Kobayashi, N.: A type system for lock-free processes. Inf. Comput. 177(2), 122–159 (2002)
14. Kobayashi, N.: A new type system for deadlock-free processes. In: Proceedings of CONCUR. pp. 233–247 (2006)
15. Laurent, O.: Syntax vs. semantics: A polarized approach. Theor. Comput. Sci. 343(1-2), 177–206 (2005)
16. Mazza, D.: The true concurrency of differential interaction nets. Mathematical Structures in Computer Science FirstView, 1–29 (11 2016)
17. Mazza, D., Pellissier, L., Vial, P.: Polyadic approximations, fibrations and intersection types. In: Proceedings of POPL (2018), to appear. Available on the second author's web page
18. Padovani, L.: Deadlock and lock freedom in the linear π-calculus. In: Proceedings of CSL-LICS. pp. 72:1–72:10 (2014)
19. Rozenberg, G., Engelfriet, J.: Elementary net systems. In: Dagstuhl Lectures on Petri Nets, Lecture Notes in Computer Science, vol. 1491, pp. 12–121. Springer (1996)
20. Sangiorgi, D., Walker, D.: The π-calculus. A Theory of Mobile Processes. Cambridge University Press (2001)
21. Varacca, D., Völzer, H., Winskel, G.: Probabilistic event structures and domains. Theoretical Computer Science 358(2-3), 173–199 (2006)
22. Vaux, L.: Differential linear logic and polarization. In: Proceedings of TLCA. pp. 371–385 (2009)
23. Wadler, P.: Propositions as sessions. J. Funct. Program. 24(2-3), 384–418 (2014)