Proofs as executions

Emmanuel Beffara[1*] and Virgile Mogbil[2†]

[1] IML – FRE3529, CNRS – Université d'Aix-Marseille
[2] LIPN – UMR7030, CNRS – Université Paris 13

Abstract. This paper proposes a new interpretation of the logical contents of programs in the context of concurrent interaction, wherein proofs correspond to valid executions of a processes. A type system based on linear logic is used, in which a given process has many different types, each typing corresponding to a particular way of interacting with its environment and cut elimination corresponds to executing the process in a given interaction scenario. A completeness result is established, stating that every lock-avoiding execution of a process in some environment corresponds to a particular typing. Besides traces, types contain precise information about the flow of control between a process and its environment, and proofs are interpreted as composable schedulings of processes. In this interpretation, logic appears as a way of making explicit the flow of causality between interacting processes.

1 Introduction

The extension of the familiar Curry-Howard correspondence to interactive models of computation has been an active research topic for several decades. Several systems were proposed based on linear logic [11], following the fundamental observation that it is a logic of interaction. Interpretations of proofs as processes, first formalized by Abramsky [1], later refined by various people including the first author [2], stressed that proof nets [12] and process calculi have significant similarities in dynamics. At the same time, type systems for concurrency [24] revealed to be equivalent to variants of linear logic [14, 6]. These approaches successfully stress the fact that concurrent calculi are very expressive and versatile models of interactive behaviour, however they are not satisfactory yet as a proof-theoretical account of concurrency, because they tend to impose determinism in execution, effectively constraining processes to functional behaviour.

Several approaches to the question of non-determinism in proof theory have been proposed using the additives of linear logic [16, 17, 21]. In a different style, differential logic was recently developed by Ehrhard and Regnier [10] and its untyped proof formalism was shown expressive enough to represent the π-calculus [9]. The present work proposes a different approach to the topic, by questioning the "proofs-as-programs" paradigm. Proof theory wants cut elimination to be confluent, because the meaning of proofs lies in their normal forms. On the other

* Supported by the French ANR project LOGOI, ANR-10-BLAN-0213
† Supported by the French ANR project COMPLICE, ANR-08-BLAN-0211-01

hand, reduction in process calculi is execution: the meaning of a term is not its final irreducible form but what happens to get there, as interaction with other processes. Hence we propose to match proofs with executions rather than terms. But this raises the new question of what is the logical meaning of an execution. Here we must remember that cut elimination is a process of *explicitation* and cut-free proofs are explicit, direct reasonings justifying some fact. In our case, the fact is the interaction, which is a scheduling of a set of events in a system. The justification, then, is the *control flow* through the system, specifying when actions happen and when execution jumps from one process to another.

Technically, we illustrate this idea in the very simple setting of finitary CCS with no choice operator, in order to focus on the novel ideas of our approach, but ways to extend these techniques to a larger class of processes are sketched in the perspectives. The corresponding logic is multiplicative linear logic, with a family of modalities à la Hennessy-Milner [13] representing actions.

In our type system, multiplicatives represent causality and independence between parts of a run, using connectedness/acyclicity arguments to describe avoidance of deadlocks. Modalities represent observable transitions, with explicit scheduling constraints using the well-known stratifying effect of boxes in proof nets. Axiom rules have an unusual interpretation: they are void of interactive content (no forwarding or copycat behaviour), but they logically implement the transfer of control flow between different parts of a running process.

Comparison to other work This handling of control flow using the symmetries of linear logic is reminiscent of the work of Mazurak and Zdancewicz [18] who use linear negation as an explicit scheduling operation. Our work differs from theirs and other works on typing for concurrency, in that we proceed "backwards": while Curry-Howard systems for concurrency embed logical systems into concurrent calculi, we embed executions of processes into a logical system.

The idea of matching proofs with executions is reminiscent of the *proof search* approach to computation. Indeed, the relationship between logical linearity and interaction has been explored for instance by Miller and Tiu [19, 22] in sequent calculus and by Bruscoli [5] in deep inference. Our approach has fundamentally different bases: in these works, formulas are programs and proofs are reduction sequences, while in our settings a formula is an interaction scenario and proofs describe how a process can act according to this scenario, following its syntactic structure. Moreover, internal dynamics in processes actually corresponds to cut-elimination, which sets our work closer to proofs-as-programs than proof search.

Outline The paper is organized as follows: Section 2 introduces a logic of schedulings based on linear logic and illustrates our interpretation. Section 3 defines a simple fragment of CCS and a notion of determinisation, used to represent executions as terms. Section 4 presents the proof nets for the logic of schedulings and its cut-elimination. Section 5 shows the typing of executions and the associated subject reduction property, and Section 6 establishes the completeness property that all lock-avoiding executions are typable. Appendices with detailed proofs can be found on the online version [4].

$$\frac{P \vdash \Gamma, A, B}{P \vdash \Gamma, A \parr B}\ (\parr) \qquad \frac{P \vdash \Gamma, A \quad Q \vdash B, \Delta}{P \mid Q \vdash \Gamma, A \otimes B, \Delta}\ (\otimes) \qquad \frac{P \vdash \Gamma, A \qquad Q \vdash A^\perp, \Delta}{P \mid Q \vdash \Gamma, \Delta}\ (\text{cut})$$

$$\frac{}{1 \vdash A, A^\perp}\ (\text{ax}) \qquad \frac{P \vdash \Gamma, A}{a.P \vdash \Gamma, \langle a \rangle A}\ (\text{act}) \qquad \frac{P \vdash \Gamma \quad a \notin \Gamma}{(\boldsymbol{\nu} a)P \vdash \Gamma}\ (\text{new})$$

$$\text{Derived rules:} \begin{cases} \dfrac{P : \Gamma, A \vdash B}{P : \Gamma \vdash A \multimap B}\ (\multimap\text{R}) \qquad \dfrac{P : \Gamma \vdash A \quad Q : \Delta, B \vdash C}{P \mid Q : \Gamma, \Delta, A \multimap B \vdash C}\ (\multimap\text{L}) \\ \dfrac{P : \Gamma \vdash A \multimap B \quad Q : \Delta \vdash A}{P \mid Q : \Gamma, \Delta \vdash B}\ (\text{mp}) \qquad \dfrac{P : \Gamma, A \vdash B}{a.P : \Gamma, \langle \overline{a} \rangle A \vdash B}\ (\text{act}) \end{cases}$$

– Rule (act) applies for names of both polarities.
– In rule (new), $a \notin \Gamma$ means that neither $\langle a \rangle$ nor $\langle \overline{a} \rangle$ occurs in Γ.

Table 1. Inference rules for MLL with action modalities (MLL$_\mathrm{a}$)

2 A logic of schedulings

We first present the logic we use to describe interactions and schedulings. It corresponds to the multiplicative fragment of linear logic [11], augmented with a family of modalities that describe actions. Note that this section introduces the logic in sequent calculus for simplicity, but the proper formalism for establishing our results is that of proof nets, presented in section 4; this choice of presentation is (hopefully) more pedagogical and has no technical consequences.

Definition 1 (MLL$_\mathbf{a}$). *The formulas of MLL$_a$ are built by the grammar*

$$A, B ::= \alpha \ \mid\ \alpha^\perp \ \mid\ A \otimes B \ \mid\ A \parr B \ \mid\ \langle a \rangle A \ \mid\ \langle \overline{a} \rangle A$$

where the α are literals and the a are CCS names. The negation $A^\perp$ *of a non-literal formula A is defined by de Morgan duality as $(A \otimes B)^\perp = A^\perp \parr B^\perp$ and $(\langle a \rangle A)^\perp = \langle \overline{a} \rangle A^\perp$. A* type *$(\Gamma, \Delta \ldots)$ is a finite multiset of formulas. Derivations are built from the rules of table 1, where the left side of $\vdash$ is a CCS term up to structural congruence (as of section 3).*

Although it is formulated as a type system for processes, this logic should be interpreted as a calculus for building schedulings. To explain this interpretation, we adopt a few notations that stress the functional aspect of the system: $P : A_1, \ldots, A_n \vdash B$ represents the judgement $P \vdash A_1^\perp, \ldots, A_n^\perp, B$ and the binary connective $A \multimap B$ stands for $A^\perp \parr B$. We easily get the derived rules of table 1: ($\multimap$R) and ($\multimap$L) are respectively a reformulation of ($\parr$) and ($\otimes$), and (mp) is *modus ponens* for linear implication, obtained with (ax), ($\otimes$) and (cut). The (ax) and (cut) rules have natural two-sided counterparts. This is an intuitionistic or implicative formulation, but we do need the full expressiveness of the MLL$_\mathrm{a}$ for the developments of the following sections.

A formula specifies a way for a process to interact with its environment and a proof provides a way to justify this interaction. A judgement $P : A_1, \ldots, A_n \vdash B$

then denotes a function that combines n interactions of types A_i for independent processes Q_i into an interaction of type B of the process $Q_1 \mid \cdots \mid Q_n \mid P$.

- A modality $\langle a\rangle A$ means doing the action a and then acting according to A. To lighten notations, we will represent successive modalities as a single one: $\langle abc\rangle\alpha$ means $\langle a\rangle\langle b\rangle\langle c\rangle\alpha$. Note that the silent action τ is not represented in types, since it is not part of the interactive behaviour of processes.
- Implication $A \multimap B$ is an interaction that provides a behaviour B expecting A from the environment, as made explicit by the rule (mp). The rule ($\multimap$R) means that some context may actually be provided by the environment.
- A variable α is a behaviour not known from the considered term. An interaction of type α means jumping to a continuation of type α, necessarily provided by the context: indeed, since a scheduling of this type may not provide any behaviour, it effectively gives control to some other process.

As we will formalize later on, the term P on the left side of a judgement is guaranteed to be able to provide the behaviour computed by the proof, and this behaviour will consume all the actions of P. Reciprocally, all the behaviours that consume all actions of P correspond to some proof.

Let us illustrate this by examining the possible ways of typing a term like $a.b.1 \mid c.1$. This term has three possible ways of interacting: each interleaving of the sequence (a, b) with the sequence (c) is a valid trace. A simple interleaving is the sequential execution of one part followed by the other, as (a, b, c). E.g.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{1 : C \vdash C}\,(\text{ax})}{b.1 : C \vdash \langle b\rangle C}\,(\text{act})}{a.b.1 : C \vdash \langle ab\rangle C}\,(\text{act}) \qquad \dfrac{\dfrac{}{1 : \alpha \vdash \alpha}\,(\text{ax})}{c.1 : \alpha \vdash \langle c\rangle\alpha}\,(\text{act})}{a.b.1 \mid c.1 : \alpha \vdash \langle abc\rangle\alpha}\,(\text{cut}) \qquad \text{with } C = \langle c\rangle\alpha.$$

The important point is the choice of the axiom on C: it stands for the fact the $a.b.1$ finally hands control to $c.1$ for which we have type C.

The interleaving (a, c, b) is more subtle: now $c.1$ will have to get control from $a.b.1$ after a and give back control to it after doing c. We can write this as

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{1 : \alpha \vdash \alpha}\,(\text{ax})}{b.1 : \alpha \vdash \langle b\rangle\alpha}\,(\text{act}) \qquad \dfrac{}{1 : C \vdash C}\,(\text{ax})}{b.1 : \alpha, \langle b\rangle\alpha \multimap C \vdash C}\,(\multimap\text{L})}{a.b.1 : \alpha, \langle b\rangle\alpha \multimap C \vdash \langle a\rangle C}\,(\text{act})}{a.b.1 : \alpha \vdash (\langle b\rangle\alpha \multimap C) \multimap \langle a\rangle C}\,(\multimap\text{R}) \qquad \dfrac{\dfrac{\dfrac{}{1 : B \vdash B}\,(\text{ax})}{c.1 : B \vdash \langle c\rangle B}\,(\text{act})}{c.1 \vdash B \multimap \langle c\rangle B}\,(\multimap\text{R})}{a.b.1 \mid c.1 : \alpha \vdash \langle acb\rangle\alpha}\,(\text{mp}) \qquad \text{with } \begin{cases} B = \langle b\rangle\alpha \\ C = \langle cb\rangle\alpha \end{cases}$$

Again, the choice of the right types for the axioms is crucial because it depends on the continuation in interaction. Indeed, we have three steps (a, c, b) and as many types for continuations: $\langle cb\rangle\alpha$, $\langle b\rangle\alpha$ and α.

The other crucial point is the introduction of a in front of $b.1$, as the succession of rules (ax), ($\multimap$L), (act), ($\multimap$R). The conclusion type reads as "if using $\langle b\rangle\alpha$

the environment can do C, then, by combining with it, $a.b.1$ can do a then C". Operationally, $a.b.1$ starts by doing a, then jumps to C (the behaviour of the environment), and at some point the environment will give control back from C (that is the negative occurrence of C) and $b.1$ will then perform $\langle b\rangle\alpha$. This part is generic in C: we could use the same reasoning for any type C, including a type variable γ. In a more concise way, $(B \multimap \gamma) \multimap \langle a\rangle\gamma$ is an interruptible version of the modality $\langle a\rangle B$. Similarly, the typing of $c.1$ is generic in B. We only need to choose B and C appropriately for the (mp) rule, so that types unify properly.

Another aspect is when parallel composition is typed by a cut which means that a synchronisation (send/receive) happens between the composed processes:

$$\dfrac{\dfrac{\dfrac{}{1 : \varepsilon \vdash \varepsilon}\,(\mathrm{ax})}{e.1 : \varepsilon \vdash \langle e\rangle\varepsilon}\,(\mathrm{act}) \quad \dfrac{\dfrac{\dfrac{\dfrac{}{1 : \alpha \vdash \alpha}\,(\mathrm{ax})}{\bar{e}.1 : \langle e\rangle\alpha \vdash \alpha}\,(\mathrm{act})}{d.\bar{e}.1 : \langle e\rangle\alpha \vdash \langle d\rangle\alpha}\,(\mathrm{act}) \quad \dfrac{\dfrac{}{1 : \delta \vdash \delta}\,(\mathrm{ax})}{\bar{d}.1 : \langle d\rangle\delta \vdash \delta}\,(\mathrm{act})}{d.\bar{e}.1 \mid \bar{d}.1 : \langle e\rangle\alpha \vdash \alpha}\,(\mathrm{cut})}{e.1 \mid d.\bar{e}.1 \mid \bar{d}.1 : \alpha \vdash \alpha}\,(\mathrm{cut}) \quad \text{with } \begin{cases} \delta = \alpha \\ \varepsilon = \alpha \end{cases}$$

Here the conclusion type is a simple interaction with the environment. This term has different proofs providing the same type, e.g. using a intermediate trace for $e.1|d.\bar{e}.1$ instead of $d.\bar{e}.1|\bar{d}.1$ as in the proof above. Such variants are irrelevant in scheduling and will be removed by switching to proof nets in the next sections.

3 CCS runs as pairings

We consider processes of the standard language CCS [20]. The general language is defined by the following grammar. Note that we use 1 for the inactive process instead of the usual 0 because it is the neutral element of $|$ which is a multiplicative operation. Moreover, actions a are decorated by *locations* ℓ:

$$P, Q ::= a^{\ell}.P \;\mid\; \bar{a}^{\ell}.P \;\mid\; 1 \;\mid\; (P \,|\, Q) \;\mid\; P + Q \;\mid\; *P \;\mid\; (\boldsymbol{\nu} a)P$$

where a is taken from an infinite set $\mathcal{N}$ of names and ℓ is taken from an infinite set $\mathcal{L}$ of locations. Each location is used at most once in any term. The main source of non-determinism is the fact that a given action name may occur several times in a given term, and locations are used to name the different occurrences.

For the purpose of the present study, we actually restrict to the following fragment. The reason for this will be explained in the following development.

Definition 2 (MCCS). *Multiplicative CCS is the fragment of CCS using neither choice (+) nor replication ($*$). Structural congruence is the smallest congruence $\equiv$ that makes parallel composition associative commutative and* 1 *neutral.*

The set of locations occurring in P is written $\mathcal{L}(P)$. Given $\ell \in \mathcal{L}(P)$, the *subject* of ℓ is the name tagged by ℓ, written $\mathrm{subj}_P\,\ell$. The *polarity* of ℓ is that of the action tagged by its subject, written $\mathrm{pol}_P\,\ell$, element of $\{\pm 1\}$. Intuitively, a negative action $\bar{a}$ represents the sending of a signal on a channel a, and a positive action a represents the reception of such a signal.

Definition 3 (execution). *Execution is the relation over structural congruence classes, labelled by partial involutions over $\mathcal{L}$, defined by the rule*

$$\bar{a}^{\ell}.P \mid a^{m}.Q \mid R \quad \rightarrow_{ex}^{\{(\ell,m)\}} \quad P \mid Q \mid R$$

Let $\rightarrow_{ex*}$ be the reflexive transitive closure of $\rightarrow_{ex}$, with the annotations defined as $P \rightarrow_{ex*}^{\emptyset} P$ and if $P \rightarrow_{ex*}^{c} Q \rightarrow_{ex*}^{d} R$ then $P \rightarrow_{ex*}^{c \cup d} R$.

The annotation c in $P \rightarrow_{ex}^{c} Q$ describes which occurrences interact in the execution step, we write $P \rightarrow_{ex} Q$ if c is unimportant. Similarly, we keep locations implicit when they do not matter. Remark that, for a given P and c, there is at most one Q such that $P \rightarrow_{ex}^{c} Q$, since c describes the interaction completely.

Definition 4 (pairing). *A* pairing *of a term P is a partial involution c over $\mathcal{L}(P)$ such that for all $\ell \in \operatorname{dom} c$, $\operatorname{subj} c(\ell) = \operatorname{subj} \ell$ and $\operatorname{pol} c(\ell) = -\operatorname{pol} \ell$.*

Let $\sim_c$ be the smallest equivalence that contains c. Let $\leq_P$ be the partial order over $\mathcal{L}(P)$ such that $\ell <_P m$ for every subterm $x^{\ell}.Q$ of P with $m \in \mathcal{L}(Q)$. c is consistent *if $\operatorname{dom} c$ is downward closed for $\leq_P$ and $\sim_c <_P \sim_c$ is acyclic.*

Example 1. The total pairings of $P = a^1.c^2 \mid b^3.\bar{a}^4 \mid \bar{b}^5.\bar{c}^6 \mid a^7.\bar{b}^8 \mid b^9 \mid \bar{a}^0$ are $c_1 = \{(9,5),(1,0),(2,6),(3,8),(4,7)\}$, $c_2 = \{(3,5),(1,4),(2,6),(7,0),(9,8)\}$, $c_3 = \{(1,4),(3,8),(7,0),(9,5),(2,6)\}$, $c_4 = \{(1,0),(3,5),(7,4),(9,8),(2,6)\}$. Only c_1 is inconsistent as there is a cycle induced by $\{(3,8),(4,7)\}$. The maximal consistent pairing included in c_1 is $\{(9,5),(1,0),(2,6)\}$.

Observe that pairings and consistency are preserved by structural congruence, as a direct consequence of the fact that subjects, polarities and prefixing are preserved by structural congruence.

Proposition 1. *A pairing c of a term P is consistent if and only if there is a term Q such that $P \rightarrow_{ex*}^{c} Q$.*

Proof (sketch). In an execution $P_0 \rightarrow_{ex}^{c_1} P_1 \rightarrow_{ex}^{c_2} \cdots \rightarrow_{ex}^{c_n} P_n$, the c_i are disjoint, so their union is a pairing, and consistency is ensured by the fact that executions respect prefixing. Conversely, write $c = c_1 \uplus \cdots \uplus c_n$ with the c_i atomic. By definition, if c is consistent then $\leq_P$ induces a partial order over the domains of the c_i. Assume that the considered enumeration respects this order, then we can prove by recurrence that there is an execution sequence $P = P_0 \rightarrow_{ex}^{c_1} P_1 \cdots \rightarrow_{ex}^{c_n} P_n$, since each c_i joins two actions of P_{i-1} that are minimal for $\leq_{P_{i-1}}$.

We easily get the following (for a proof see the appendix [4, B.1]).

Proposition 2. *Let P be a term. Any two executions $P \rightarrow_{ex*}^{c} Q$ and $P \rightarrow_{ex*}^{c} R$ with the same pairing are permutations of each other, and in this case $Q \equiv R$.*

We will thus consider consistent pairings as the proper notion of execution for CCS terms. Maximal consistent pairings represent executions of processes until a state where no more execution is possible.

A useful tool in the study of pairings is the following notion of determinisation, by which we can turn a pairing of a term into a term that has no other pairing. In other words, determinisation is a way to represent a run of a term in the language of MCCS itself.

Definition 5 (deterministic term). *A term P is* deterministic *if it has at most one occurrence of each action.*

The pairings of a deterministic term form a lattice, consistent pairings too, so there is a unique maximal consistent pairing for any deterministic term.

The restriction operator $(\boldsymbol{\nu} a)$ serves two purposes: it limits the scope of a name, and it makes it possible to have names local to each copy of a subterm in the presence of replication; both these features are useless in the deterministic case, hence we leave it out on determinisation. We abide by Barendregt's convention that each bound channel is named distinctly from each other channel.

Definition 6 (determinisation). *Assume an injective map $\delta : \mathcal{N} \times \{\pm 1\} \times \mathcal{L} \to \mathcal{N}$. Given a partial involution c,* determinisation *along c is the operator ∂_c which commutes with parallel composition such that $\partial_c((\boldsymbol{\nu} a)P) = \partial_c(P)$ and*

$$\partial_c\left(a^\ell.P\right) = \delta(a,+1,\ell)^\ell.\partial_c(P), \quad \partial_c\left(\bar{a}^\ell.P\right) = \begin{cases} \overline{\delta(a,+1,\ell)}^\ell.\partial_c(P) & \text{if } \ell \in \operatorname{dom} c, \\ \delta(a,-1,\ell)^\ell.\partial_c(P) & \text{otherwise.} \end{cases}$$

By construction, $\partial_c(P)$ is deterministic, the pairings of $\partial_c(P)$ are the restrictions of c, consistency preserved, so c is the unique maximal pairing of $\partial_c(P)$.

Example 2. For the term P and the pairings of example 1, we obtain the following determinisations (with $\delta(a,+1,7) = d$ and $\delta(b,+1,9) = e$):
$c_3 = \{(1,4),(3,8),(7,0),(9,5),(2,6)\}$ induces $\partial_{c_3}(P) = a.c \mid b.\bar{a} \mid \bar{e}.\bar{c} \mid d.\bar{b} \mid e \mid \bar{d}$,
$c_4 = \{(1,0),(3,5),(7,4),(9,8),(2,6)\}$ induces $\partial_{c_4}(P) = a.c \mid b.\bar{d} \mid \bar{b}.\bar{c} \mid d.\bar{e} \mid e \mid \bar{a}$.

If we extended our study to the whole of CCS, determinisations would still be in MCCS, but the theory of pairings would have to be refined: external choice requires a notion of conflict in the space of locations (as in event structures [23]), replications requires the introduction of indices to distinguish copies.

4 Proof nets for MLL with action modalities

Proofs in sequent calculus are well suited to inductive reasoning, however their use in proof theory is uneasy because their rigid structure obscures many arguments, like those below in particular. For this reason, we will turn to proof nets, using the standard machinery of linear logic [12, 7]. Modality rules are represented using boxes (like promotions in standard linear logic, but with different typing rules). The only extra information we add to standard proof structures is the location of each box, to reflect the use of locations in CCS terms in the sequel. For readers not familiar with the standard definitions for proof nets, these are put in appendix [4, A.1]. We detail here specificities of $\mathrm{MLL_a}$.

Definition 7 (proof structure). *A* proof structure *consists of an ordered forest of nodes labelled by formulas, denoted x^A, with a set Ax of axiom links (pairs of leaves), a set Cut of cuts (pairs of roots) and a set Box of modality boxes, labelled by action modalities, such that each box β has a unique location $\ell(\beta)$. The roots that are not part of a cut are called the* conclusion nodes*. The* conclusion type *is the multiset of the labels of the conclusion nodes.*

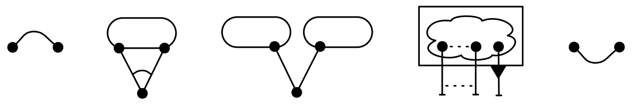


Fig. 1. Representation of proof structures: axiom link, ⅋ node, ⊗ node, boxes, cut.

A modality box β is a set of nodes (the ports) associated to a proof structure S whose conclusions are in bijection with the ports. If the modality of β is $\langle a \rangle$, then the *principal port* is labelled $\langle a \rangle A$ and matches a conclusion of S labelled A, while *auxiliary ports* have the same label as their matching conclusion in S.

The graphical notation of proof structures is presented in figure 1. By definition there are arcs only to multiplicative nodes, moreover proof structures can be drawn considering the top-bottom orientation of arcs, so we keep arc orientation implicit by this convention. Arcs to a ⅋ node are joint by a circle on the side of this node. By construction, the conclusion labels suffice to deduce all labels, so we keep most of this information implicit.

Definition 8 (proof net). *A* proof net *is a proof structure built following MLL_a sequent calculus rules. An* immediate subnet *of a proof net π is an induced subgraph of π that is itself a proof net. A* subnet *of π is either an immediate subnet of π or (inductively) a subnet of a box of π.*

Well known correctness criteria [7, 12, 8] apply to characterise proof nets among proof structures by combinatorial means like acyclicity and connectedness, which allows the definition of proof nets without any reference to sequent calculus. We will not elaborate on this aspect because it is essentially independent from the present work.

Definition 9 (cut elimination). Annotated cut elimination *is the relation $\to_{ce}^{c}$ over proof structures, labelled by partial involutions c over $\mathcal{L}$, that is the reflexive transitive closure of the rules below (such that if $\pi \to_{ce}^{c} \pi' \to_{ce}^{d} \pi''$ then $\pi \to_{ce}^{c \cup d} \pi''$). We have $\pi \to_{ce}^{c} \pi'$ if π contains a cut $\kappa = \{x, y\}$ either at top level or inside a box and one of the following cases occurs:*

- *Multiplicative step and Axiom step: standard definition, with $c = \emptyset$.*
- *Modality step: If x and y are principal ports of two boxes β, β', then c permutes $\ell(\beta)$ and $\ell(\beta')$ and π' is obtained by replacing each box with its associated proof structure.*
- *Commutation step: If x is the auxiliary port of a box β, then $c = \emptyset$, and the cut and a subnet of π that contains y are moved inside β.*

Our proof system enjoys a standard cut-elimination theorem using this definition: if $\pi \to_{ce}^{c} \pi'$ and π is a proof net, then π' is a proof net with the same conclusion (this is proved by standard arguments using correctness criteria, hence we will not develop this point); if a proof π is irreducible by $\to_{ce}$, then it has no cut link (this is an immediate case analysis). Note however that $\to_{ce}$ is not confluent, because of commutation steps.

Definition 10 (head reduction). *Head reduction is the annotated relation $\rightarrow_h^c$ over proof structures defined as the restriction of $\rightarrow_{ce}^c$ that only applies at top level and does not use the commutation step of cut elimination.*

This particular strategy is relevant because it does not reduce inside boxes, that is under prefixes, it only affects cuts in active position (from the point of view of processes). However, this strategy does not eliminate all cuts in general.

In the analysis of proofs, the following notion of path will be useful. It describes a way to traverse arcs and axioms/cuts in a proof structure while respecting the logical meaning of formulas.

Definition 11 (path). *A* path *in a proof structure S is an alternating path in the underlying graph of S, such that alternations occur only at axioms, cuts and boxes. Each move between ports x and y of a box β must be associated with a path between the corresponding conclusions in β. We further require a typing constraint: a path can only move up a left (resp. right) branch if has moved down a left (resp. right) branch before, with a natural well-bracketing condition.*

For instance, a path starting from an axiom with type α may move down the tree of nodes, reach a cut, move up the other side of the cut, always in the branches that contain α, reach an axiom, and so on.

5 Typing executions of MCCS terms

Proofs in $\mathrm{MLL_a}$ will serve as a type system. Although this can be formulated in usual sequent style (as in table 1), the natural notion rather relates proof nets and structural congruence classes of terms.

Definition 12 (term assignment). *Let S be a proof structure. The MCCS term $\lfloor S \rfloor$ assigned to π is the parallel composition of the $\lfloor \beta \rfloor$ for each box β in S. In turn, for a box β with location ℓ and associated structure S_β, the term $\lfloor \beta \rfloor$ is $a^\ell.\lfloor S_\beta \rfloor$ if the principal port of β has modality $\langle a \rangle$ and $\bar{a}^\ell.\lfloor S_\beta \rfloor$ if the principal port of β has modality $\langle \overline{a} \rangle$. A term P is said to have type Γ if there is a proof net π of conclusion Γ such that $\lfloor \pi \rfloor \equiv P$. In this case we write $\pi : P \vdash \Gamma$.*

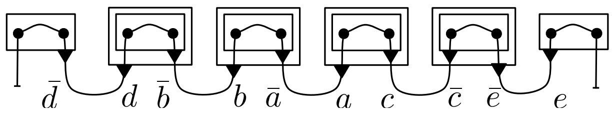
A proof net is a proof structure that is built using the rules of table 1, ignoring the terms on the left of the $\vdash$ symbols. It is obvious that these terms do reflect the definition of term assignment: A term P has type Γ if and only if there is a type derivation with conclusion $P \vdash \Gamma$ using the rules of table 1.

We now establish the correspondence between cut elimination in a proof and execution steps in the assigned terms. The first result justifies head reduction:

Proposition 3. *Let π be a proof structure. For every head reduction $\pi \rightarrow_h^c \pi'$ there is an execution $\lfloor \pi \rfloor \rightarrow_{ex*}^c \lfloor \pi' \rfloor$.*

Proof (sketch). Axiom and multiplicative cut elimination steps do not affect the assigned terms, besides their annotation is empty, so the result holds immediately for them. When a modality step applies, it reduces a cut between boxes with dual modalities (because of typing), hence the associated terms are ready to interact; the reduct is easily seen to be the assigned term of the reduct proof.

Example 3. Let π be the following proof net.



We have $\lfloor\pi\rfloor = a.c \mid b.\bar{a} \mid \bar{e}.\bar{c} \mid d.\bar{b} \mid e \mid \bar{d}$. (It is $\partial_{c_3}(P)$ of previous examples). As it is a deterministic term, we abusively identify locations with names. We consider the head reduction sequence $\pi \rightarrow_h^z \pi'$ (where π' is an axiom link) for $z = \{(d,\bar{d}),(b,\bar{b}),(a,\bar{a}),(e,\bar{e}),(c,\bar{c})\}$. We have $\lfloor\pi\rfloor \rightarrow_{ex*}^z \lfloor\pi'\rfloor \equiv 1$.

Subject reduction does not hold in general. Indeed, a given proof may hold several occurrences of a given modality, corresponding to different occurrences of an action in the term, and the structure of cuts may not match a given execution step. This is not a defect, since we actually intend to type pairings rather than processes: we do get subject reduction if we restrict to proofs that describe deterministic terms.

Definition 13 (linear proof). *A proof structure S is called* linear *if*

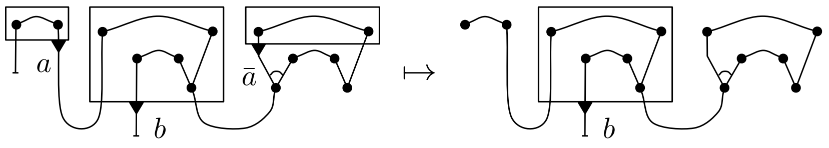
- *S contains at most one box for each modality,*
- *for each a, all occurrences of $\langle\bar{a}\rangle A$ in the labels in S have the same immediate subformula A, and if $\langle\bar{a}\rangle A$ and $\langle a\rangle B$ occur then A and B are dual,*
- *if S contains a box for both $\langle a\rangle A$ and $\langle\bar{a}\rangle A^\perp$, then neither formula occurs in the conclusion type of S.*

The essence of the linearity condition is the first constraint. Intuitively, the second and third constraints serve to guarantee that the property is preserved by composition. Indeed, if a formula $\langle a\rangle A$ occurs in the conclusion of a proof π, then the proof may be cut against a proof that contains a modality box for $\langle\bar{a}\rangle A^\perp$, which breaks linearity if π already contains a box for some $\langle\bar{a}\rangle B$. Note that the fact of being a linear proof is preserved by cut elimination.

Theorem 1 (subject reduction). *Let π be a linear proof of conclusion $P \vdash \Gamma$. For every execution $P \rightarrow_{ex*}^c P'$ there is a linear proof $\pi' : P' \vdash \Gamma$.*

Proof (sketch). An execution step $\lfloor\pi\rfloor \rightarrow_{ex}^{(\ell,m)} P$ involves immediate subterms $a^\ell.Q$ and $\bar{a}^m.R$ for $a \in \mathcal{N}$. Then π contains two top level boxes with respective principal ports $x^{\langle a\rangle A}$ and $y^{\langle\bar{a}\rangle A^\perp}$, for $A \in MLL_{\mathrm{a}}$. Since π is linear, x and y are elimination boxes for each other, ending a path ρ (as of definition 11) whose axioms contain modalities of x and y in their types. Let π' be the rewriting of π where such modalities are removed (boxes are replaced by their contents, axioms on $\langle a\rangle A$ by axioms on A). Clearly π' is a linear proof of conclusion $P' \vdash \Gamma$.

This theorem states that types are preserved by execution in deterministic terms. However, the proof uses a rewriting of the typing proofs that does not correspond to cut elimination in general. Indeed, consider the following example of typing, call π the l.h.s.:



Then the proof is linear, irreducible by head cut elimination, but the assigned term $\lfloor\pi\rfloor = \bar{a} \mid \bar{b} \mid a$ does execute into $\bar{b}$. In π, this involves a cut on the axiom inside the middle box. As done in theorem 1 the rewriting of π in a linear proof π' assigned to $\bar{b}$ is the r.h.s..

We can get a precise correspondence between execution and head cut elimination by imposing an additional constraint on the shape of proofs. In the statement below, an axiom is *immediately contained* in a box if it is an immediate subnet of the structure associated with this box.

Definition 14 (regular proof). *An axiom link immediately contained in a box β is* anchored *if there is a path from one of its conclusions to an auxiliary port of β and a path from its other conclusion to the principal port. A proof structure π is* regular *if all its axioms are anchored and for every pair of boxes with dual modalities, one of the boxes does not immediately contain any axiom.*

Theorem 2 (strong subject reduction). *Let π be a regular linear proof net. For every execution $\lfloor\pi\rfloor \to^{c}_{ex*} P$ there is a regular linear proof π' such that $\pi \to^{c}_{h} \pi'$ and $\lfloor\pi'\rfloor = P$.*

Proof (sketch). Consider an execution step $\lfloor\pi\rfloor \to^{(\ell,m)}_{ex} P$. As in the proof of theorem 1, linearity implies that there are boxes at top level and a path ρ between their principal ports $x^{\langle a\rangle A}$ and $y^{\langle\bar{a}\rangle A^{\perp}}$ for immediate subterms $a^{\ell}.Q$ and $\bar{a}^{m}.R$ of $\lfloor\pi\rfloor$. Since x is cut at top level, ρ traverses no box, otherwise linearity or regularity would be contradicted. Then ρ is a multiplicative cut path whose cut elimination $\to^{\emptyset}_{h}$ until x and y preserves $\lfloor\pi\rfloor$ as well as regularity and linearity.

6 Anti-execution and completeness

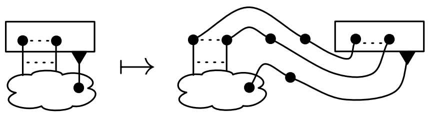
In this section we establish our correspondence theorem relating typings and executions. To achieve this goal we first provide a kind of reciprocal statement for subject reduction: if a term T can reduce into a typed term T', then we can type T with a proof that reduces to the typing of T'. Because we want logically correct proof structures, this operation requires some care.

Example 4. Consider the term $P := a.\bar{b} \mid b.\bar{c} \mid \bar{a}.c$. We cannot type each thread with a simple type like $\langle a\rangle\alpha$, $\langle\bar{b}\rangle\alpha^{\perp}$ and then introduce a cut for each interaction, since we would get a cyclic proof structure, which is incorrect.

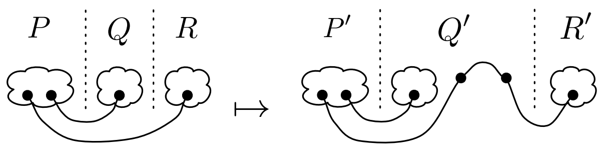
We now describe a general method for deducing a typing by "anti-execution" of a proof. We stay at a partly informal level for clarity, all formal statements are detailed in the appendix [4, B.3].

Consider a generic execution step $P \mid a.Q \mid \bar{a}.R \to_{ex} P \mid Q \mid R$. Assume the reduct is typed by some proof π. We want to put the parts of π that correspond to Q and R into boxes, with a cut between them, while rewriting the proof to avoid cycles. For this purpose, we proceed in four steps:

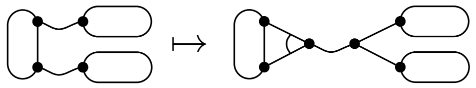
Selection consists in moving each box belonging to Q or R away from the main proof, by means of an axiom/cut pair, so that Q and R are represented by simple sets of boxes, cut with the main proof (which corresponds to P), with no multiplicative connectives:



Chaining consists in introducing an extra axiom/cut pair in the middle of each cut between P and R, so that there are cuts only between P and Q or Q and R, and not between P and R directly:

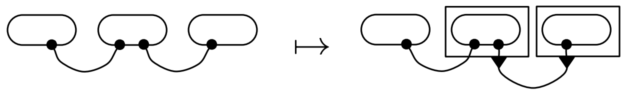


Simplification consists in making sure that there is actually exactly one cut between P and Q and one between Q and R, by multiplexing multiple cuts through multiplicatives:



Correctness criteria guarantee that we can always find two cuts for which there is one connected component on one side, two on the other.

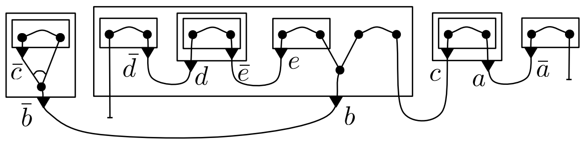
Boxing consists in putting Q and R into boxes, cut together, so that Q has one auxiliary port to P and R has no auxiliary port:



Following this method, we prove the following statement:

Proposition 4 (anti-execution). *Let $T_1 \to^c_{ex} T_2$ be an execution step and let $\pi_2 : T_2 \vdash \Gamma$ be a typing. There exists a typing $\pi_1 : T_1 \vdash \Gamma$ such that $\pi_1 \to^c_h \pi_2$.*

Example 5. Consider the term of P of example 1. We consider the execution $e = (a, \bar{a})(b, \bar{b})(c, \bar{c})(d, \bar{d})(e, \bar{e})$ of the determinized term $\partial_{c_4}(P) = a.c | b.\bar{d} | \bar{b}.\bar{c} | d.\bar{e} | e | \bar{a}$ for the (total and consistent) pairing $c_4 = \{(1, 0), (3, 5), (7, 4), (9, 8), (2, 6)\}$. A typing synthesized by the construction of proposition 4 is the following.



Lemma 1 (preserved regularity). *In the construction of proposition 4, if π_2 is regular, then so is π_1. If π_2 is linear and T_2 is deterministic, then π_1 is linear.*

Proof (sketch). Let $T_2 = P \mid Q \mid R$. If an axiom is introduced by anti-execution rewrite steps, used in proposition 4 then: i) it is added to P by selection and it will not be boxed, or ii) it is added to Q by chaining and becomes anchored by simplification and boxing. No axiom is introduced on the side of R, Q only contains chaining axioms, so regularity is satisfied for the new axioms. Besides, regularity is not broken for axioms previously present in the proof.

Example 6. In the previous example 5, one can also start execution by $(b, \bar{b})(a, \bar{a})$ as seen in the typing. All execution permutation of $\partial_{c_4}(P)$ in the pairing c_4 is allowed by the typing proof synthesized from the execution e.

We now summarize the previous results, about subject reduction and the reverse operation, into a precise statement relating typings and execution.

Lemma 2 (initial typing). *Every linear MCCS term where no name occurs with both modalities is typable by a cut-free regular proof.*

Proof. We simply build a proof of $T \vdash A_T, B_T$ with A_T non-modal by induction on T. For $T = 1$, use the axiom rule to get $1 \vdash \alpha^\perp, \alpha$. For $T = P \mid Q$, deduce $T \vdash A_P \parr A_Q, B_P \otimes B_Q$ by the tensor rule. For $T = a.P$, deduce $T \vdash A_P, \langle a \rangle B_P$ by the action rule, similarly for $\bar{a}.P$. The proof thus built is obviously regular since every axiom is at top level or anchored, and there are no pairs of boxes with dual modalities.

Theorem 3 (completeness). *For every execution $P \rightarrow^c_{ex*} Q$ there are typings $\pi_P : P \vdash \Gamma$ and $\pi_Q : Q \vdash \Gamma$ such that $\pi_P \rightarrow^c_h \pi_Q$. Moreover, for every execution sequence $P \rightarrow^{c_1}_{ex} P_1 \cdots \rightarrow^{c_n}_{ex} P_n = Q$ with $c_1 \cup \cdots \cup c_n = c$, there is a cut elimination sequence $\pi_P \rightarrow^{c_1}_h \pi_1 \cdots \rightarrow^{c_n}_h \pi_n = \pi_Q$, with $\lfloor \pi_i \rfloor = P_i$ for all i.*

Proof. By definition, the term $\partial_c(Q)$ is linear and has no dual actions, so by lemma 2 we can find a cut-free regular proof $\pi'_Q : \partial_c(Q) \vdash \Gamma$. If we apply proposition 4 repeatedly to π'_Q with the steps of the considered execution $\partial_c(P) \rightarrow^c_{ex*} \partial_c(Q)$, we get a proof $\pi'_P : \partial_c(P) \vdash \Gamma$ that reduces to π'_Q by a head reduction sequence labelled c. Let π_P and π_Q be the relabellings of π'_P and π'_Q by the inverse of ∂_c, then we have $\pi_P : P \vdash \Gamma$, $\pi_Q : Q \vdash \Gamma$ and $\pi_P \rightarrow^c_h \pi_Q$.

Every execution sequence of P with label c is an execution sequence of $\partial_c(P)$ with the same label. By lemma 1, π'_P enjoys strong subject reduction as of theorem 2, hence every run of $\partial_c(P)$ labelled by c corresponds to a head reduction sequence in π'_P labelled by c. By relabelling with ∂_c^{-1}, every run of P labelled by c corresponds to a head reduction sequence $\pi_P \rightarrow^c_h \pi_Q$.

In other words, every execution of a term can be exactly characterized up to permutation by typing, in the sense that the execution sequences of the term within the same pairing will be exactly the head reduction sequences of the associated typing proof. By combining determinisation (definition 6) and strong subject reduction (theorem 2) we get that, conversely, each regular typing of a term defines a set of executions stable by permutation.

7 Conclusion and further works

In this work we have developed, in the simple framework of multiplicative CCS, a precise logical description of executions of processes. A key technical tool is the use of pairings, by which we separate non-determinism in communication from the multiplicity of equivalent schedulings; this technique extends well to more expressive frameworks (full CCS, π-calculus, etc.). The logical interpretation we propose moves beyond the traditional Curry-Howard for concurrency by accepting non-deterministic terms, albeit with a change of interpretation in the correspondence. Indeed, the logic we use is well studied and has a wide range of existing tools (efficient correctness criteria, proof search, etc.) but its interpretation in our paradigm of proof-as-executions is new.

Logical expressiveness The restriction to purely multiplicative objects, in MCCS and MLL, lets us concentrate on the precise role of multiplicatives and axioms as descriptions of how a process interacts with its environment but hides the complexity inherent to the other defining features of concurrent systems like choice, recursion, name passing, etc. It should be stressed that extending the calculus or the logic are two different things. Extending the calculus enriches the set of possible executions, by introducing more subtle synchronization possibilities: choice allows for conflict between actions, replication allows for arbitrarily large runs with some uniformity, value passing allows for communication of ground values, name passing allows the set of synchronizable pairs to evolve along execution. After determinisation, all these features essentially disappear and deterministic runs can still be formulated in MCCS. On the other hand, enriching the logic leads to richer descriptions of the control flow in processes, for instance using a first order language with predicates to describe properties of continuations.

Causality A crucial feature of our work is the interpretation of axioms as a way to transfer causality. This idea suggests new ways of analyzing causality in interactive systems, and the fact that the flow of causality is often as complicated as the flow of information. Besides, a similar fact is illustrated by the expressiveness of solos [15, 3], where communication is used to carry all prefixing information in processes. Our interpretation may provide a logical insight on this matter.

Cut elimination In the present work, as in proofs-as-programs formalisms, composition of processes is represented by the cut rule and execution corresponds to a particular cut elimination strategy. An interesting direction for future work is the study of the meaning of full cut-elimination, from the proofs-as-executions point of view. The operationally relevant part is the elimination of dual actions, which means executing all internal transitions in advance. This implies making choices with respect to synchronisation. In other words, eliminating cuts in a $\mathrm{MLL_a}$ proof yields a more deterministic process that can still exhibit the behaviour given by the considered type.

References

1. S. Abramsky. Proofs as processes. *TCS*, 135(1):5–9, 1994.
2. E. Beffara. A concurrent model for linear logic. *ENTCS*, 155:147–168, 2006.
3. E. Beffara and F. Maurel. Concurrent nets: a study of prefixing in process calculi. *TCS*, 356(3):356–373, 2006.
4. Emmanuel Beffara and Virgile Mogbil. Proofs as executions. Technical Report 00586459, HAL, July 2012. http://hal.archives-ouvertes.fr/hal-00586459.
5. P. Bruscoli. A purely logical account of sequentiality in proof search. In Peter J. Stuckey, editor, *ICLP*, volume 2401 of *LNCS*, pages 302–316. Springer, 2002.
6. L. Caires and F. Pfenning. Session types as intuitionistic linear propositions. In *CONCUR*, volume 6269 of *LNCS*, pages 222–236. Springer, 2010.
7. V. Danos and L. Regnier. The structure of multiplicatives. *Archive for Math. Logic*, 28(3):181–203, 1989.
8. P. Jacobé de Naurois and V. Mogbil. Correctness of linear logic proof structures is NL-complete. *TCS*, 412(20):1941–1957, 2011.
9. T. Ehrhard and O. Laurent. Interpreting a finitary π-calculus in differential interaction nets. In *CONCUR*, volume 4703 of *LNCS*, pages 333–348. Springer, 2007.
10. T. Ehrhard and L. Regnier. Differential interaction nets. *TCS*, 364(2):166–195, 2006.
11. J.-Y. Girard. Linear logic. *TCS*, 50(1):1–102, 1987.
12. J.-Y. Girard. Proof-nets : the parallel syntax for proof theory. *Logic and Algebra*, 180, 1996.
13. M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, jan 1985.
14. K. Honda and O. Laurent. An exact correspondence between a typed π-calculus and polarised proof-nets. *TCS*, 411(22–24):2223–2238, 2010.
15. C. Laneve and B. Victor. Solos in concert. In J. Wiederman, P. Boas, and M. Nielsen, editors, *ICALP*, volume 1644, pages 513–523. Springer Verlag, 1999.
16. H. Mairson and K. Terui. On the computational complexity of cut-elimination in linear logic. In *ICTCS*, volume 2841 of *LNCS*, pages 23–36. Springer, 2003.
17. F. Maurel. Nondeterministic light logics and NP time. In M. Hofmann, editor, *TLCA*, volume 2701 of *LNCS*, pages 241–255. Springer, 2003.
18. K. Mazurak and S. Zdancewic. Lolliproc: to concurrency from classical linear logic via curry-howard and control. In *ICFP*, pages 39–50, 2010.
19. D. Miller. The π-calculus as a theory in linear logic: preliminary results. In *WELP*, volume 660 of *LNCS*, pages 242–264. Springer, 1992.
20. R. Milner. *Communication and concurrency.* Prentice Hall, 1989.
21. V. Mogbil. Non-deterministic boolean proof nets. In M. van Eekelen and O. Shkaravska, editors, *FOPARA*, volume 6324 of *LNCS*, pages 131–145. Springer, 2010.
22. A. Tiu and D. Miller. A proof search specification of the π-calculus. *ENTCS*, 138(1):79–101, 2005.
23. G. Winskel. Event structures. In *Advances in Petri nets: applications and relationships to other models of concurrency*, pages 325–392. Springer Verlag, 1987.
24. N. Yoshida, M. Berger, and K. Honda. Strong normalisation in the π-calculus. In *LICS*, pages 311–322, 2001.