A logical view on scheduling in concurrency

Emmanuel Beffara

A logical view on scheduling in concurrency

Emmanuel Beffara
I2M, Université d'Aix-Marseille & CNRS

February 25, 2014

Abstract. This paper elaborates on a new approach of the question of the proof-theoretic study of concurrent interaction called "proofs as schedules". Observing that proof theory is well suited to the description of confluent systems while concurrency has non-determinism as a fundamental feature, we develop a correspondence where proofs provide what is needed to make concurrent systems confluent, namely scheduling. In our core logical system (a variant of multiplicative linear logic), processes and schedulers appear explicitly as proofs in different fragments of the proof language and cut elimination between them does correspond to execution of a concurrent system. This separation of roles suggests new insights for the denotational semantics of processes and new methods for the translation of π-calculi into prefix-less formalisms (like solos), as the operational counterpart of a translation between logical systems.

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 [8], following the intuition that it is a logic of interaction. Interpretations of proofs as processes, first formalized by Abramsky [1], later refined by various people including the author of this paper [2], stressed that proof nets [9] and process calculi have significant similarities in dynamics. At the same time, type systems for concurrency [20] revealed to be equivalent to variants of linear logic [11, 4]. 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 essentially functional behaviour.

Several approaches to the question of non-determinism in proof theory have been proposed. For instance, non-determinism in the style of complexity theory has been modelled using the additives of linear logic [13, 16, 17]. In a different style, differential logic was recently developed by Ehrhard and Regnier [7] and features a structured form of non-determinism in its cut elimination; its untyped proof formalism was shown expressive enough to represent the π-calculus [6].

The present work elaborates on a different approach to the topic, first sketched in a previous paper called *Proofs as executions* [3]. Our point of view is that cut elimination and general process execution should not be made to match directly, because their meaning is on different levels. On the one hand, the meaning of proofs lies in their normal forms, hence cut elimination should confluent in order to preserve meaning. On the other hand, the meaning of a process is not its final irreducible form but what happens to get there, as interaction with other processes (hence execution of interactions should definitely not preserve meaning). We thus establish a

correspondence between proofs and interaction plans for processes, hereafter called *schedules* (rather than executions).

The previous paper [3] formalizes this idea by defining a logical system, which appears as a kind of type system for processes, with the crucial property that for every lock-avoiding execution of a process P one can deduce a typing of P, hence a proof, whose cut elimination does correspond to the considered execution. A limitation of this result was that the "type" of a process could be very different depending on its particular environment and execution, which made it difficult to deduce an interpretation of processes in logical terms. The present work improves the previous results in several respects: Firstly, the typing of processes is uniform, and really independent of the particular executions it might exhibit. Subsequently, the contributions of the process and the scheduler in the proof corresponding to a scheduling are clearly identified, as simple fragments of the proof language. This in turn suggests new approaches to the logical study of denotational models of processes.

Despite these references to a previous work, this paper aims to be self-contained. The process calculus is introduced in Section 2, the logical system is introduced in Section 3. Section 4 presents a typing of processes for which a precise characterization of execution by means of provability is obtained. Section 5 presents another typing which induces a more asynchronous correspondence, in which logic can abstract away from particular execution orders. Section 6 discusses implications for the study of the semantics of processes and Section 7 discusses directions for future work.

2 Multiplicative CCS

We consider processes in a fragment of the standard language CCS [15]. The fragment we use, hereafter called *multiplicative* CCS (or MCCS), is defined by the following grammar:

$$\begin{array}{rll} P, Q := & 1 & \text{inactive process} \\ & P \mid Q & \text{parallel composition} \\ & a^\ell.P & \text{positive action prefix} \\ & \bar{a}^\ell.P & \text{negative action prefix} \\ & (\boldsymbol{\nu} a)P & \text{name hiding} \end{array}$$

where a is taken from a given set of *channel names* and ℓ is taken from a given set $\mathcal{L}$ of *locations*. We impose that each location is used at most once in any term: locations are used to identify occurrences of actions in a process. Note that we use 1 for the inactive process instead of the usual 0 because it is the neutral element of $\mid$ which is a multiplicative operation. The construct $(\boldsymbol{\nu} a)$ is a binder for the name a, hence for action prefixes $a.$ and $\bar{a}.$, and terms are considered up to renaming of hidden names.

1 **Definition.** Structural congruence is the smallest congruence $\equiv$ that makes parallel composition commutative and associative with 1 as neutral element, and that implements the standard scoping rules:

$$(\boldsymbol{\nu} a)P \mid Q \equiv (\boldsymbol{\nu} a)(P \mid Q) \qquad\qquad (\boldsymbol{\nu} a)1 \equiv 1$$

where a does not occur free in Q.

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

$$(\boldsymbol{\nu} a_1) \dots (\boldsymbol{\nu} a_n)(\bar{a}^\ell.P \mid a^m.Q \mid R) \quad \to_{\{(\ell,m)\}} \quad (\boldsymbol{\nu} a_1) \dots (\boldsymbol{\nu} a_n)(P \mid Q \mid R)$$

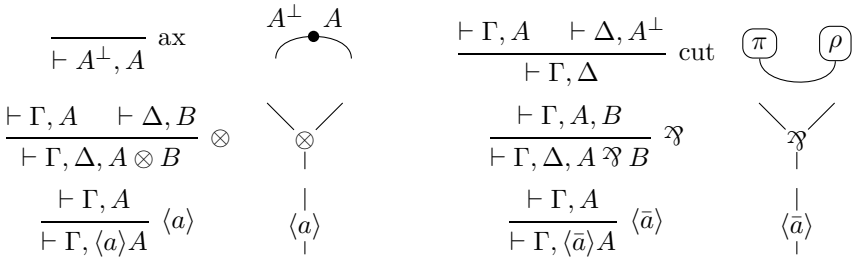


Table 1: Proof rules and proof net syntax for MLLa.

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

Locations will be used mainly in Section **??** for the discussion of the semantic interpretation of our results. When locations are unimportant for a given statement, they will be kept implicit, and in this case the reduction relation is the standard one:

$$(\boldsymbol{\nu} a_1) \dots (\boldsymbol{\nu} a_n)(a.P \mid \bar{a}.Q \mid R) \quad \to \quad (\boldsymbol{\nu} a_1) \dots (\boldsymbol{\nu} a_n)(P \mid Q \mid R)$$

3 MLL with actions

Formulas of propositional multiplicative linear logic with actions (in short MLLa) are generated from the following grammar:

$$\begin{array}{rll} A, B := \alpha, \ \alpha^\perp & & \text{propositional variable} \\ & A \otimes B, \ A \parr B & \text{conjunction, disjunction} \\ & \langle a \rangle A, \ \langle \bar{a} \rangle A & \text{action modalities} \end{array}$$

where α is taken from a given set of propositional variables and a is taken from the set of channel names. Negation is defined inductively on formulas in the standard way, with $(\langle a \rangle A)^\perp = \langle \bar{a} \rangle (A^\perp)$; linear implication $A \multimap B$ is defined as a shorthand for $A^\perp \parr B$.

The modalities are reminiscent of those of Hennessy-Milner [10] logic, however the logic itself and is very different, in particular because of linearity. While Hennessy-Milner modalities $\langle a \rangle A$ and $[a]A$ mean, respectively, "I can do a and then satisfy A" and "whenever I do a, I will satisfy A", in our logic the formula $\langle a \rangle A$ means something like "I will do a and then exhibit behaviour A". There is no analogue of the modality $[a]$ because all the information we provide is positive. The duality induced by linear negation $(\cdot)^\perp$ is not a classical negation but a change of roles: $A^\perp$ is the type of behaviours that interact correctly with behaviours of type A. The connectives $\otimes$ and $\parr$ are not classical conjunction and disjunction either, but spatial ones representing causality and independence between parts of a run, using connectedness/acyclicity arguments to describe avoidance of deadlocks.

Proof rules are the standard rules of linear logic, extended with modality, as shown in Table 1. We use the language of proof nets, with the notations from the same table. As far as proof theory is concerned, these modalities are innocuous: they commute with all other rules, their cancellation rule in cut elimination is the obvious one, the types A and $\langle a \rangle A$ are isomorphic. Indeed, they are nothing more than markers in formulas. For this reasons, standard theory for multiplicative proof nets [5] applies to proof nets of MLLa.

We restrict ourselves to propositional logic, however our construction heavily relies on the instantiation of propositional variables with particular formulas: this substitution mechanism is

a key part of interaction as we model it. The same work could be carried out, possibly in a cleaner way, using second-order quantification, however we decided not to use this quantification explicitly because it makes proof theory more complicated and we do not need its full power in the present work. A similar approach was taken by Terui in his translation of boolean circuits into MLL proofs [17].

4 Execution as implication

In this section, we consider the fragment of MCCS *without name hiding*, for which we formulate a characterization of execution by implication in MLLa. Formally, each term P is mapped to a formula $\lceil P \rceil_s$ that has a unique cut-free proof, which follows the syntactic structure of P. We then prove, in Theorem 7, that $\lceil P \rceil_s \multimap \lceil Q \rceil_s$ is provable if and only if there is an execution $P \to^* Q$.

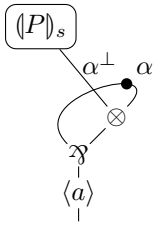
3 **Definition (type assignment).** Terms of MCCS are translated into MLLa formulas as follows, where α and β are fresh propositional variables:

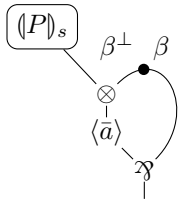
$$\begin{aligned}
\lceil 1 \rceil_s &:= \alpha^\perp \parr \alpha \\
\lceil P \mid Q \rceil_s &:= \lceil P \rceil_s \otimes \lceil Q \rceil_s \\
\lceil a.P \rceil_s &:= \langle a \rangle (\alpha^\perp \parr (\lceil P \rceil_s \otimes \alpha)) && = \langle a \rangle (\alpha \multimap (\lceil P \rceil_s \otimes \alpha)) \\
\lceil \bar{a}.P \rceil_s &:= \langle \bar{a} \rangle (\lceil P \rceil_s \otimes \beta^\perp) \parr \beta && = \langle a \rangle (\lceil P \rceil_s \multimap \beta) \multimap \beta
\end{aligned}$$

The freshness condition is a way to enforce polymorphism in our translation: each proposition variable occurs twice, once in each polarity. As observed in Section 3, the intended meaning is indeed a universal quantification over these variables, in a context where we chose not to use second order quantification. For the same reason, the type for 1 is the type of an identity, where we could have used the multiplicative unit $\mathbf{1}$ of linear logic (indeed, $\mathbf{1}$ and $\forall\alpha(\alpha^\perp \parr \alpha)$ are equivalent formulas): we deliberately restrict ourselves to a unit-free logic in order to avoid the proof-theoretic complications of the units.

4 **Proposition (proof assignment).** *For every MCCS term P, there is a unique cut free proof of $\lceil P \rceil_s$. This proof will be denoted as $(\!|P|\!)_s$.*

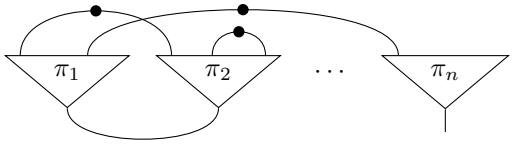
Proof. By straightforward induction on terms. The case of action prefixes is as follows:

$(\!|a.P|\!)_s =$ 

$(\!|\bar{a}.P|\!)_s =$ 

The axioms are necessarily in this position because the variables are fresh: no other occurrences appear anywhere else so they must be introduced by the same axiom rule. □

5 **Definition (term extraction).** A proof structure π can always be decomposed as a family of trees of links together with a set of axioms between leafs and possibly a set of cuts between roots:



Each tree π_i is translated into a CCS term $\lfloor \pi_i \rfloor_s$ by the following procedure:

- if a $\otimes$ link is immediately above a modality link, delete it and connect its left premiss to the modality link, turn its right premiss into a conclusion;
- in each tree π_i of the resulting structure, turn each binary link into a parallel composition, each modality link $\langle a \rangle$ into a prefix $a.$, each axiom into a neutral process, which defines a term $\lfloor \pi_i \rfloor_s$.

The term $\lfloor \pi \rfloor_s$ is defined as the parallel composition $\lfloor \pi_1 \rfloor_s \mid \cdots \mid \lfloor \pi_n \rfloor_s$.

Remark that, by construction, for all term P we have $\lfloor (\!| P |\!)_s \rfloor_s \equiv P$.

6 **Lemma.** *For any cut elimination step $\pi \to \pi'$, one of the two following situations occurs:*

- *either $\pi \to \pi'$ is an elimination of modality links $\langle a \rangle$ and $\langle \bar{a} \rangle$, then $\lfloor \pi \rfloor_s \to \lfloor \pi' \rfloor_s$ is a valid CCS execution step involving the prefixes $a.$ and $\bar{a}.$,*
- *or it is another elimination step and $\lfloor \pi \rfloor_s \equiv \lfloor \pi' \rfloor_s$.*

Proof. The case of an elimination of modality links is obvious by definition of term extraction: the actions in the extracted CCS term are necessarily at top level since the modalities are premisses of a cut, and the suffixes of the involved actions are the subtrees above them.

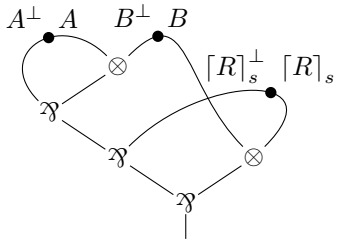
The other cut elimination steps can be either multiplicative eliminations, for which the result is immediate, or axiom eliminations. In this case it might happen that a new premiss appears above a $\langle \bar{a} \rangle$ modality link, and the technical "switching" operation for tensors above modalities in Definition 5 is used precisely to make sure that, when reading back a term, we do not consider this right premiss as a new part of the term prefixed by $\bar{a}$. □

7 **Theorem.** *Let P and Q be two MCCS terms. There is an execution $P \to^* Q$ if and only if $\lceil P \rceil_s \multimap \lceil Q \rceil_s$ is provable in MLL (without modality rules) for some instantiation of the propositional variables of $\lceil P \rceil_s$.*

Proof. For the direct implication, remark that for a structural congruence $P \equiv Q$ at top level (i.e. not under prefixes), an implication $\lceil P \rceil_s \multimap \lceil Q \rceil_s$ is provided by a standard isomorphism for that associativity and commutativity of tensor, hence it is enough to handle the case of a reduction $(a.P \mid \bar{a}.Q) \mid R \to (P \mid Q) \mid R$. The expected type is

$$\lceil (a.P \mid \bar{a}.Q) \mid R \rceil_s \multimap \lceil (P \mid Q) \mid R \rceil_s =$$
$$(((\langle \bar{a} \rangle (\alpha \otimes (\lceil P \rceil_s^\perp \parr \alpha^\perp)) \parr (\langle a \rangle (\lceil Q \rceil_s^\perp \parr \beta) \otimes \beta^\perp)) \parr \lceil R \rceil_s^\perp) \parr ((\lceil P \rceil_s \otimes \lceil Q \rceil_s) \otimes \lceil R \rceil_s)$$

so we get a proof as follows, if we instantiate α as $\lceil Q \rceil_s$ and β as $\lceil P \rceil_s \otimes \lceil Q \rceil_s$:



with $\begin{cases} A = \langle a \rangle (\lceil Q \rceil_s^\perp \parr (\lceil P \rceil_s \otimes \lceil Q \rceil_s)) \\ B = \lceil P \rceil_s \otimes \lceil Q \rceil_s \end{cases}$

For the reverse implication, consider an MLL proof of some $\lceil P \rceil_s \multimap \lceil Q \rceil_s$. If we cut this proof against $(\!|P|\!)_s$, we get a proof of $\lceil Q \rceil_s$. Cut elimination reaches the unique cut free proof of $\lceil Q \rceil_s$, namely $(\!|Q|\!)_s$. By Lemma 6, we know that each cut elimination step induces either a structural congruence or an execution step, so from a cut elimination sequence $(\!|P|\!)_s \to^* (\!|Q|\!)_s$ we can extract a CCS execution sequence $P \to^* Q$. □

5 Asynchronous executions

In this section, we still consider the fragment of MCCS without name hiding, but we provide an asynchronous variant of Theorem 7. We discuss below the relevance of this variant.

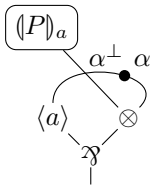
8 **Definition (asynchronous type assignment).** Terms of MCCS are translated into MLLa formulas as follows, where α and β are fresh propositional variables:

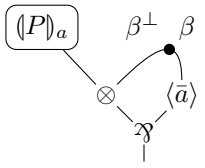
$$\begin{aligned}
\lceil 1 \rceil_a &:= \alpha^\perp \parr \alpha \\
\lceil P \mid Q \rceil_a &:= \lceil P \rceil_a \otimes \lceil Q \rceil_a \\
\lceil a.P \rceil_a &:= \langle a \rangle \alpha^\perp \parr (\lceil P \rceil_a \otimes \alpha) && = \langle a \rangle \alpha \multimap (\lceil P \rceil_a \otimes \alpha) \\
\lceil \bar{a}.P \rceil_a &:= (\lceil P \rceil_a \otimes \beta^\perp) \parr \langle \bar{a} \rangle \beta && = (\lceil P \rceil_a \multimap \beta) \multimap \langle a \rangle \beta
\end{aligned}$$

Note that this translation is the same as the synchronous translation of Definition 3, except for the position of modalities.

9 **Proposition (asynchronous proof assignment).** *For every MCCS term P, there is a unique cut free proof of $\lceil P \rceil_a$. This proof will be denoted as $(\!|P|\!)_a$.*

Proof. By straightforward induction on terms. The case of action prefixes is as follows:

$(\!|a.P|\!)_a =$ 

$(\!|\bar{a}.P|\!)_a =$ 

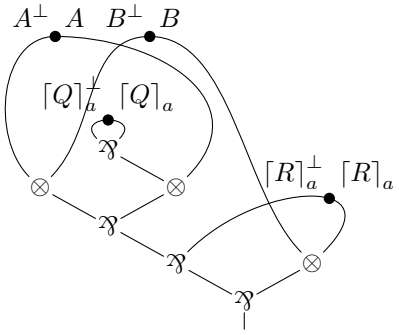
□

10 **Proposition.** *For all MCCS reduction $P \to Q$, the formula $\lceil P \rceil_a \multimap \lceil Q \rceil_a$ is provable in MLL for some instantiation of the variables of $\lceil P \rceil_a$.*

Proof. For any structural congruence $P \equiv Q$, the implication $\lceil P \rceil_a \to \lceil Q \rceil_a$ is proved by a standard isomorphism for the monoidal structure of the tensor. For an execution step $(a.P \mid \bar{a}.Q) \mid R \to (P \mid Q) \mid R$, we prove the implication:

$$\begin{aligned}
&\lceil (a.P \mid \bar{a}.Q) \mid R \rceil_a \multimap \lceil (P \mid Q) \mid R \rceil_a = \\
&\qquad ((\langle \bar{a} \rangle \alpha \otimes (\lceil P \rceil_a^\perp \parr \alpha^\perp) \parr ((\lceil Q \rceil_a^\perp \parr \beta) \otimes \langle a \rangle \beta^\perp)) \parr \lceil R \rceil_a^\perp) \parr ((\lceil P \rceil_a \otimes \lceil Q \rceil_a) \otimes \lceil R \rceil_a)
\end{aligned}$$

with the following proof:



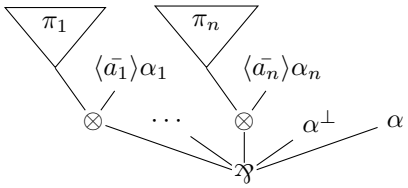
with $\begin{cases} \alpha = \lceil Q \rceil_a \\ \beta = \lceil Q \rceil_a \\ A = \langle a \rangle \lceil Q \rceil_a^\perp \\ B = \lceil P \rceil_s \otimes \lceil Q \rceil_s \end{cases}$

□

11 **Lemma.** *Let P be an MCCS term with at least one action prefix. If $\lceil P \rceil_a \multimap \lceil 1 \rceil_a$ is provable in MLL for some instantiation of the propositional variables, then there exists a reduction $P \to Q$ such that $\lceil Q \rceil_a \multimap \lceil 1 \rceil_a$ is also provable.*

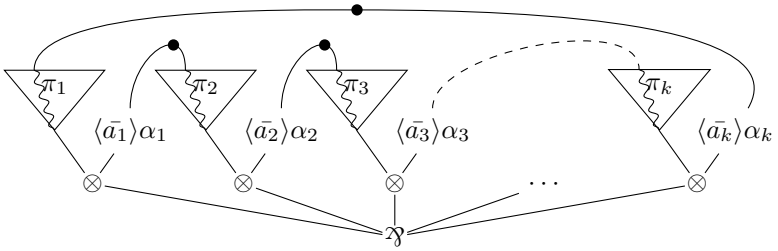
Proof. Consider a proof π of $\lceil P \rceil_a^\perp \parr \lceil 1 \rceil_a$ for some instantiation of the propositional variables of $\lceil P \rceil_a$. MLL admits cut elimination and expansion of axiom links, so we may assume without loss of generality that π is cut-free and that all multiplicative connectives in $\lceil P \rceil_a^\perp$ are introduced by π.

The term P can be written as a composition of action prefixes $a_1.P_1 \mid \cdots \mid a_n.P_n$ (where the a_i are actions of arbitrary polarities) so π is decomposed as follows, with some tensors possibly reversed depending on the polarities of the a_i:



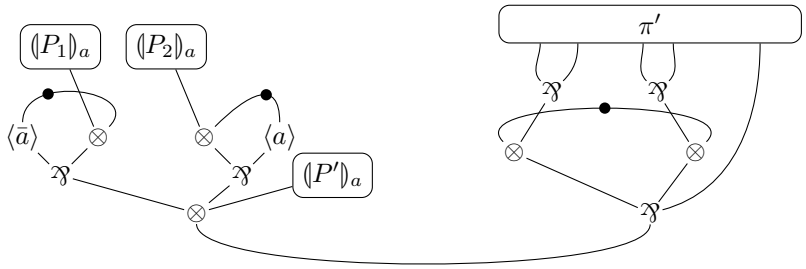
with axiom links on top (the $\parr$ link of arity $n+2$ in the picture stands for a tree of binary $\parr$ links, since our language only includes binary connectives). Note that, since π does not use modality rules, all premisses $\langle \bar{a_i} \rangle \alpha_i$ must be introduced by axiom links.

We now argue that two of these are actually connected by an axiom rule. For contradiction, assume that it is not the case: every $\langle \bar{a_i} \rangle \alpha_i$ is introduced by an axiom link whose other conclusion is a leaf in one of the subproofs $\pi_{f(i)}$. Since there are finitely many subproofs, the sequence $1, f(1), f^2(1), \ldots$ is eventually periodic. Consider a minimal cycle $i, f(i), f^2(i), \ldots, f^k(i) = i$. Up to the reordering of subterms of P, let us assume this sequence is $1, 2, \ldots, k, 1$. Then we have the following situation:



Call t_i the tensor link under the node $\langle \bar{a_i} \rangle \alpha_i$, and x_i the axiom link that has $\langle \bar{a_i} \rangle \alpha_i$ as one of its premisses. Then we have a cycle that goes from t_1 up to x_1 into π_2, down to t_2, up to x_2 into π_3 and so on until t_k, up to x_k and back into π_1, down to t_1. This cycle traverses each tree π_i straight from a leaf to the root, so it only goes through at most one premiss of each $⅋$ link. Hence there is a $⅋$-switching of π that has a cycle, which violates the Danos-Regnier correctness criterion for multiplicative proof nets [5].

Hence there is an axiom link between two of the $\langle \bar{a_i} \rangle \alpha_i$. For simplicity, assume that the indices are 1 and 2. Then a_1 and a_2 are dual, so P can be written $\bar{a}.P_1 \mid a.P_2 \mid P'$, and we have $P \to P_1 \mid P_2 \mid P'$. Moreover the cut between $(\!|P|\!)_a$ and π has the following shape:



After a few steps of cut elimination (five pairs of multiplicatives, three axioms and a pair of modalities) this proof eventually reduces into π' cut against $(\!|P_1|\!)_a$, $(\!|P_2|\!)_a$ and $(\!|P'|\!)_a$ plus a cut between the two remaining ports of π'. This reduct is also a reduct of $(\!|P_1 \mid P_2 \mid P'|\!)_a$ against π' plus two $⅋$ rules. The latter is thus an MLL proof of $\lceil P_1 \mid P_2 \mid P' \rceil_a \to \lceil 1 \rceil_a$. □

12 **Theorem.** *Let P be an MCCS term. There is an execution $P \to^* 1$ if and only if $\lceil P \rceil_a \multimap \lceil 1 \rceil_a$ is provable in MLL (without modality rules) for some instantiation of the propositional variables.*

Proof. For the direct implication, Proposition 10 provides the implication for one step of execution. The result follows immediately since linear implications properly compose and the correctness of proofs is preserved by instantiation of propositional variables.

For the reverse implication, we can reason by induction on the number of action prefixes in P. The base case of a term with no action is vacuous since the neutral process 1 is the only such term, up to structural congruence. Lemma 11 provides the induction step. □

The interpretation of Theorem 12 is more subtle than for Theorem 7, because in this variant there is no step-by-step correspondence between execution and cut elimination. Of course, Lemma 11 does prove that there is a *strategy* for eliminating cuts that corresponds to an execution, but in general the normalization of $(\!|P|\!)_a$ against a particular proof of $\lceil P \rceil_a \multimap \lceil 1 \rceil_a$ may consume modalities in P in an arbitrary order. This is why Theorem 12 requires the final type to be $\lceil 1 \rceil_a$: the statement like of Theorem 7 is actually false for the asynchronous variant.

13 **Definition.** Let $\Rightarrow$ be the relation over processes such that $P \Rightarrow Q$ holds whenever $\lceil P \rceil_a \multimap \lceil Q \rceil_a$ is provable in MLL for some instantiation of the propositional variables of $\lceil P \rceil_a$.

By Proposition 10 we know that $P \to^* Q$ implies $P \Rightarrow Q$, but other rules are easily provable, like the following:

$$\frac{P \to Q}{P \Rightarrow Q} \qquad \frac{P \Rightarrow Q}{a.P \Rightarrow a.Q} \qquad u.a.P \mid v.\bar{a}.Q \Rightarrow u.P \mid v.Q$$

Therefore $\Rightarrow$ can make decisions in advance about the execution of a process, and update the visible part of the process accordingly. In other words, $\Rightarrow$ is an fully asynchronous form of

execution: it allows to perform an interaction as soon as this interaction may be part of some full execution.

6 Semantic interpretation

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. 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 $\{\pm 1\}$.

The annotation c in an execution step $P \to_c Q$ describes which occurrences interact. Remark that, for a given P and c, there is at most one Q such that $P \to_c Q$, since c describes the interaction completely.

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

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 $\mathrm{dom}\, c$ is downward closed for $\leq_P$ and $\sim_c <_P \sim_c$ is acyclic.

15 *Example.* The total pairings of $P = a^1.c^2|b^3.\bar{a}^4|\bar{b}^5.\bar{c}^6|a^7.\bar{b}^8|b^9|\bar{a}^0$ 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. In our previous work [3], we established the following precise relationship between pairings and executions:

16 **Theorem.** *Let P be an MCCS term and c a pairing of P.*

- *c is consistent if and only if there is a term Q such that $P \to_c^* Q$,*
- *any two executions $P \to_c^* Q$ and $P \to_c^* R$ with the same pairing are permutations of each other, and in this case $Q \equiv R$.*

Maximal consistent pairings represent executions of processes until a state where no more execution is possible.

We can actually relate pairings and logical schedules in a precise way. Consider a term P and proof π of $\lceil P \rceil_a \multimap \lceil 1 \rceil_a$. In a cut elimination process of $(\!|P|\!)_a$ against π, eventually each modality link in $(\!|P|\!)_a$ will be eliminated by a dual link also from $(\!|P|\!)_a$, and this association between modality links is of course independent of the cut elimination sequence since proof normalization is confluent. Hence π induces a pairing between modality links in $(\!|P|\!)_a$, and since modality links are in bijection with locations in P, this in turn induces a pairing of P. This pairing is maximal, and by Theorem 12 it is consistent. Reciprocally, every maximal consistent pairing of P induces a class of proofs of $\lceil P \rceil_a \multimap \lceil 1 \rceil_a$.

Moreover, this relationship between pairings and proofs is actually a correspondence between pairings of P and possible ways of positioning axiom links on top of a canonical multiplicative proof structure for $\lceil P \rceil_a \multimap \lceil 1 \rceil_a$. However, this correspondence is not straightforward (like one axiom for each pair) because there are more leaves in proofs of this type than there are actions in P. We defer the precise study of this relatinship to future work.

7 Conclusion and future work

We thus have defined two type assignment systems for MCCS processes into MLLa that enjoy good properties relating cut-elimination and execution. It appears that processes correspond to cut-free proofs using modality links, while schedules correspond to pure MLL proofs without modality links.

Although the technical details of the paper are limited to multiplicative CCS, it is clear that the same approach extends to the full calculus, including replication and sum, with a type assignment like

$$\lceil !P \rceil_a = ! \lceil P \rceil_a \qquad\qquad \lceil P + Q \rceil_a = \lceil P \rceil_a \,\&\, \lceil Q \rceil_a$$

so that, for replication, a schedule has to decide the number of copies of P it will use, and for the sum it will have to decide which side of the additive conjunction will be effectively used. This leads to a correspondence using full propositional linear logic; the theory of proof nets is harder in this case but the principles of the correspondence remain the same.

We now conclude with ideas for research directions open by the present work.

Denotational semantics The study of the denotational semantics of proofs in our system should provide new insights for the denotational semantics of processes. For instance, it is a trivial remark that coherence spaces [8] have a formal similarity with event structures [19], although a precise relationship is hard to formulate. Since our constructions maps processes to types, i.e. spaces, and schedules as proofs i.e. cliques or configurations in spaces, it should be possible to formalize such a relationship in our setting.

CPS translations Both our type assignments actually translate actions and processes into types of *intuitionistic* MLL formulas, therefore proofs extracted from processes could be interpreted as linear functions. Moreover, the structure of the translations, up to the position of modalities, is $\lfloor \bar{a}.P \rfloor = (\lfloor P \rfloor \multimap \beta) \multimap \beta$, which is a weak form of double-negation. This suggests a kind of (linear) continuation-passing-style translation of processes info a linear λ-calculus, which could be another setting for the semantic and operational study of processes and schedules. In relation with other forms of proofs-as-processes correspondences, this might suggest an approach to the question of desequentialization of processes (like translations into solos [2, 12]) as a translation between logical systems, in a way similar to CPS translations from classical logic into intuitionistic logic.

Name hiding and passing Although Theorem 7 characterizes execution, it has a fundamental limitation: it does not extend to a calculus with name restriction. Indeed, a reasonable type assignment for a term $(\boldsymbol{\nu} a)P$ should not let a appear in the formula $\lceil (\boldsymbol{\nu} a)P \rceil_s$. This limitation is less present in Theorem 12 because of the asynchronous nature of this variant.

This suggests a way of handling name hiding in our framework: placing a $(\boldsymbol{\nu} a)$ in front of a process would mean something like making decisions about the scheduling that will happen on name a, by cutting the relevant conclusions against a partial schedule. Further study of this point should also be the way to approach the problem of extending our work to name passing calculi.

Another, more syntactic way of handling name hiding would be to allow quantification over channel names in the logic, so that $(\boldsymbol{\nu} a)$ in a process would become $\exists a$ in the type. A schedule would then have to handle actions on this name without knowing anything about it, hence considering it effectively as a fresh name.

Proof search 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 [14, 18]). Our approach has different roots than this, in particular because of the status of cut-elimination in our work, nevertheless proof search in our context looks like the building of a schedule for a fixed process. Although our translation of processes into formulas is different that that used by the above works, connections can certainly be established.

References

[1] Samson Abramsky. *Proofs as processes.* Theoretical Computer Science, 135(1):5–9, December 1994.

[2] Emmanuel Beffara and François Maurel. *Concurrent nets: a study of prefixing in process calculi.* Theoretical Computer Science, 356(3):356–373, May 2006.

[3] Emmanuel Beffara and Virgile Mogbil. *Proofs as executions.* In Jos C. M. Baeten, Tom Ball and Frank S. de Boer, editors, Proceedings of IFIP TCS, Lecture Notes in Computer Science, volume 7604, pages 280–294. Springer, 2012.

[4] Luís Caires and Frank Pfenning. *Session types as intuitionistic linear propositions.* In Proceedings of the 21st International Conference on Concurrency Theory, Paris, France, August 2010.

[5] Vincent Danos and Laurent Regnier. *The structure of multiplicatives.* Archive for Mathematical Logic, 28:181–203, 1989.

[6] Thomas Ehrhard and Olivier Laurent. *Interpreting a finitary pi-calculus in differential interaction nets.* Information and Computation, 208(6):606–633, June 2010.

[7] Thomas Ehrhard and Laurent Regnier. *Differential interaction nets.* Theoretical Computer Science, 364(2):166–195, 2006.

[8] Jean-Yves Girard. *Linear logic.* Theoretical Computer Science, 50:1–102, 1987.

[9] Jean-Yves Girard. *Proof-nets: the parallel syntax for proof-theory.* In Paolo Agliano and Aldo Ursini, editors, Logic and Algebra. M. Dekker, New York, 1996.

[10] Matthew Hennessy and Robin Milner. *Algebraic laws for nondeterminism and concurrency.* Journal of the ACM, 32(1):137–161, January 1985.

[11] Kohei Honda and Olivier Laurent. *An exact correspondence between a typed π-calculus and polarised proof-nets.* Theoretical computer science, 411(22–24):2223–2238, May 2010.

[12] Cosimo Laneve and Björn Victor. *Solos in concert.* Mathematical Structures in Computer Science, 13(5):657–683, October 2003.

[13] François Maurel. *Nondeterministic light logics and NP time.* In Martin Hofmann, editor, Typed Lambda Calculi and Applications (TLCA), 6th International Conference, LNCS, pages 241–255. Springer, 2003.

[14] Dale Miller. *The π-calculus as a theory in linear logic: preliminary results.* In Evelina Lamma and Paola Mello, editors, Extensions of Logic Programming, Third International Workshop (WELP'92), Lecture Notes in Computer Science, volume 660, pages 242–264. Springer, 1992.

[15] Robin Milner. *Communication and concurrency.* Prentice Hall, 1989.

[16] Virgile Mogbil. *Non-deterministic boolean proof nets.* In Marko van Eekelen and Olha Shkaravska, editors, Foundational and Practical Aspects of Resource Analysis, number 6324 in Lecture Notes in Computer Science, pages 131–145. Springer Berlin Heidelberg, January 2010.

[17] Kazushige Terui. *Proof nets and boolean circuits.* In Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science, LICS '04, page 182–191, Washington, DC, USA, 2004. IEEE Computer Society.

[18] Alwen Tiu and Dale Miller. *A proof search specification of the π-calculus.* ENTCS, 138(1):79–101, 2005.

[19] Glynn Winskel. *Event structures.* In W. Brauer, W. Reisig and G. Rozenberg, editors, Petri Nets: Applications and Relationships to Other Models of Concurrency, number 255 in Lecture Notes in Computer Science, pages 325–392. Springer Berlin Heidelberg, January 1987.

[20] Nobuko Yoshida, Martin Berger and Kohei Honda. *Strong normalisation in the π-Calculus.* In 16th IEEE Symposium on Logic in Computer Science (LICS), pages 311–322, 2001.