A proof-theoretic view on scheduling in concurrency

Emmanuel Beffara

A proof-theoretic view on scheduling in concurrency

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

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 logical system, 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 translations between proof systems.

1 Introduction

The extension of the familiar Curry-Howard correspondence outside the intuitionistic and functional worlds has been an active topic for decades, with a variety of approaches to the question of determinism and confluence, or lack thereof. The interactive nature of cut elimination procedures suggested relationships with actually interactive models of computation like games or process algebras. Several systems were proposed based on linear logic [13], following the intuition that it is a logic of interaction. Interpretations of proofs as processes, first sketched by Abramsky [1] and formalized by Bellin and Scott [6], later refined by various people including the author of this paper [3], stressed that proof nets [14] and process calculi have significant similarities in dynamics. At the same time, type systems for concurrency [26] revealed to be equivalent to variants of linear logic [16, 8]. 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.

Getting out of this restriction requires to actually handle the inherent non-determinism in concurrent systems. A natural option is of course to relate processes with proofs in a non-deterministic system. The prototypical such system is the original classical sequent calculus LK, however because of its structural rules it feels too remote from actual process dynamics to offer a direct correspondence (although encodings through intermediate systems have been proposed [2]). Several more direct approaches have been proposed based on linear logic: non-determinism in the style of complexity theory has been modelled using the additives of linear logic [18, 22, 23]; differential logic was recently developed by Ehrhard and Regnier [12] and features a structured form of non-determinism in its cut elimination; its untyped proof formalism was even shown expressive enough to represent the π-calculus [11].

The present work elaborates on a different approach to the topic, first sketched in a previous paper called *Proofs as executions* [5]. 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 be confluent in order to preserve meaning. On the other hand, the meaning of a process is not its final irreducible form but what

Paulo Oliva (Ed.): Classical Logic and Computation 2014
EPTCS 164, 2014, pp. 78–92, doi:10.4204/EPTCS.164.6

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). These schedules are what provides necessary information to make a system deterministic, i.e. to make its execution confluent. Note that despite the use of the word "schedule", we do not claim any precise link with e.g. scheduling in operating systems; the word "strategy" might also be appropriate, although the standard meaning of "reduction strategy" is way more restrictive than what we describe here.

Concretely, we develop this idea in a simple framework, relating schedules of finite processes with proofs in multiplicative linear logic; this is to be considered as a first step towards a wider correspondence, illustrating the idea of our interpretation. The process calculus we use is introduced in Section 2, the logical system is introduced in Section 3. Section 4 presents two logical translations of processes for which the correspondence is proved, once in a synchronous form in Section 4.1 and once in an asynchronous form in Section 4.2, in which logic can abstract away from execution order. Section 5 discusses extensions and directions for future developments of the proofs-as-schedules paradigm, related to semantics of processes (Section 5.1), the status of name hiding and passing (Section 5.2) and the relationships with CPS translations and determinisation in classical logic (Section 5.3).

Note. The previous paper [5] 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.

Related work. 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, 24]) but our approach has different roots, in particular because of the status of cut-elimination, nevertheless proof search in our context looks like the building of a schedule for a fixed process, as illustrated in the technical proofs in this paper. Bruscoli establishes a correspondence between proof search in a variant of MLL and execution of a minimal CCS, in a context of deep-inference [7]. Although our translation of processes into formulas is different that that used by the above works, connections can certainly be established, but we defer this to later developments.

2 Multiplicative CCS

We consider processes in a fragment of the standard language CCS [21]. The fragment we use, hereafter called *multiplicative* CCS (or MCCS), is defined by the following grammar:

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

where a is taken from a given set of *channel names* and ℓ is taken from a given set $\mathscr{L}$ of *locations*. We impose that each location occurs 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 $|$ which is a multiplicative operation (in the sense that it distributes over non-deterministic choice, which is then additive, and not the other way around; besides it is in correspondence with the unit **1** of linear logic and surely not with **0**).

Definition 1. *Structural congruence is the smallest congruence $\equiv$ that makes parallel composition commutative and associative with* 1 *as neutral element.*

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

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

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

Note that, by definition, $P \rightarrow^*_{\emptyset} Q$ holds is when P and Q are structurally congruent. Moreover, the subscript c in $P \rightarrow^*_c Q$ is indeed an involution, i.e. a set of disjoint pairs and not a list of pairs: it does not record the order of interactions, only the information of which actions were synchronised together; as stated in Theorem 4, this does characterize executions up to permutation of independent transitions.

Locations do not affect the execution of processes: they are nothing more than a technical tool used to name occurrences of actions in a formal way. They will be used in Section 4.1 for the correspondence between execution and cut elimination and in Section 5.1 for the discussion of the semantic interpretation of our results. We introduce a few useful notations for this purpose:

Definition 3. *Let P be an MCCS term.*

- *The set of locations occurring in P is written $\mathscr{L}(P)$.*
- *Given $\ell \in \mathscr{L}(P)$, 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 action order of P is the partial order $\leq_P$ over $\mathscr{L}(P)$ such that $\ell <_P m$ for every sub-term $x^{\ell}.Q$ of P with $m \in \mathscr{L}(Q)$.*

Remark that the information of locations, subjects, polarities and action order completely characterizes a class of structural congruence of MCCS terms, so we can consider that providing such information does define a process without ambiguity.

When locations are unimportant for a given statement, they will be kept implicit. With this convention, the definition of the reduction relation is the standard one:

$$a.P \,|\, \bar{a}.Q \,|\, R \quad \rightarrow \quad P \,|\, Q \,|\, R$$

This language is very minimalistic and its operational theory is actually very simple (one can prove that its bisimilarity is completely axiomatized by a single rule scheme $(a.P)^{n+1} \simeq a.(P \,|\, (a.P)^n)$ plus structural congruence). The results of this paper extend smoothly to a framework with replication and sum. A much more subtle point is that of name hiding, which is discussed in Section 5.2.

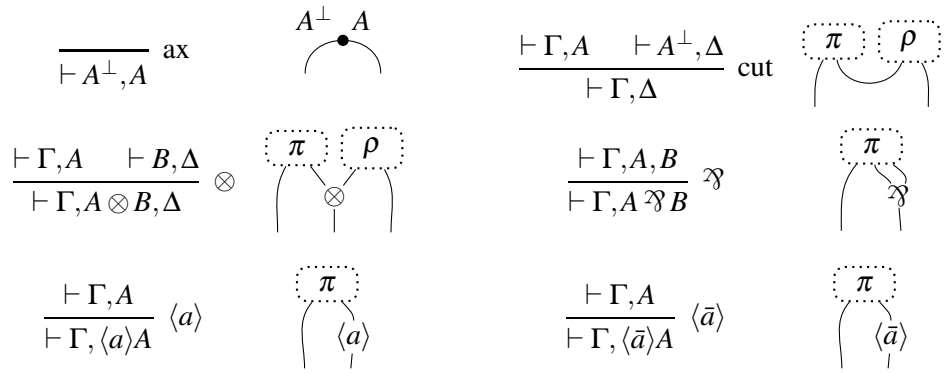


Table 1: Proof rules and proof net syntax for MLLa.

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 [15] logic, however the logic itself 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 modalities, as shown in Table 1. We use the language of proof nets: a *proof structure* is a directed graph whose nodes are labelled by names of proof rules and have the appropriate input and output degrees for the rule they represent (premisses are ingoing edges and conclusions are outgoing, the implicit orientation in the pictures is downwards), a *proof net* is a proof structure that is the translation of a sequent calculus proof using the rules of Table 1. 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 (yet we call them *modalities* since they do have a logical meaning: they impose restrictions on the structure of their possible proofs and this is the key for the results in the present work). For these reasons, standard theory for multiplicative proof nets applies to proof nets of MLLa, including the Danos-Regnier correctness criterion:

Theorem 1 (Danos-Regnier [10])**.** *Let π be a proof structure. Define a* switching graph *of π as any graph obtained by deleting one of the ingoing edges of each $\parr$ node and forgetting about edge orientation. Then π is a proof net if and only if all its switching graphs are connected and acyclic.*

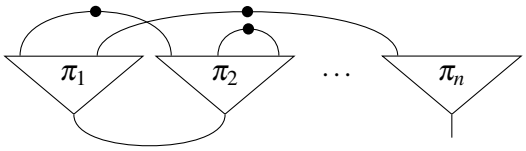
We restrict ourselves to propositional logic, however our construction heavily relies on the instantiation of propositional variables with particular formulas: this substitution mechanism is a key part of interaction as we model it. The same work could be carried out, possibly in a cleaner way, using 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 [23].

In order to ease the formulation of operational correspondence proofs, we assume that each proof structure comes with an injective labelling of its modality links with locations in $\mathscr{L}$.

Definition 4. *Let π be an MLLa proof structure (possibly with cuts).*

- *The set of locations occurring in π is written $\mathscr{L}(\pi)$.*
- *Given $\ell \in \mathscr{L}(\pi)$, the* subject *of ℓ is the name in the modality tagged by ℓ, written* $\mathrm{subj}_\pi \ell$*. The* polarity *of ℓ is that of the modality, written* $\mathrm{pol}_\pi \ell$*, element of* $\{\pm 1\}$.
- *The proof order of π is the partial order $\leq_\pi$ over $\mathscr{L}(\pi)$ such that $\ell <_\pi m$ when the link tagged m appears in the tree of premisses of the link tagged by ℓ.*

These definitions are similar to those for MCCS terms. The partial order $\leq_\pi$ is a bit more complicated to define but can be explained in a simple way. Observe that 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:



Then the partial order $\leq_\pi$ over $\mathscr{L}(\pi)$ such that $\ell \leq_\pi m$ when the modality link labelled ℓ occurs below that labelled m in one of the trees π_i.

4 Execution as implication

In this section, we formally develop the correspondence between operational semantics of processes and proofs in MLLa:

- Each process term P is translated into a cut-free proof in MLLa, where instances of modality rules correspond to action prefixes in P; the conclusion $\lceil P \rceil$ of this proof (which can be considered as the type of P) is deduced syntactically from P.
- A schedule for reducing P to Q is a *multiplicative* proof (possibly with cuts) of the implication $\lceil P \rceil \multimap \lceil Q \rceil$, i.e. this proof may not contain modality rules, which stresses the fact that it is only allowed to relate actions in a given process without introducing new actions.
- An execution of P corresponds to a cut elimination sequence of P cut against a schedule for reducing P to some Q; various cut elimination sequences correspond to different linear orderings of independent events in a given execution.

Hence the concurrent aspect of execution is represented by the variety of proofs for a given implication, while different choices in cut elimination correspond to unimportant ordering decisions, which is consistent with the fact that cut elimination is confluent.

We implement this correspondence in two different ways. The first version is *synchronous* in that it exactly relates provability of implication and cut elimination with step-by-step execution of processes. The second version is *asynchronous* in that it allows scheduling decisions to be made in advance of execution, which leads to a more flexible system, albeit with a more intricate interpretation.

4.1 Synchronous translation

In this translation, each term P is mapped to a formula $\lceil P \rceil_{\mathrm{s}}$ that has a unique cut-free proof, which follows the syntactic structure of P. We then prove, in Theorem 2, that $\lceil P \rceil_{\mathrm{s}} \multimap \lceil Q \rceil_{\mathrm{s}}$ is provable if and only if there is an execution $P \to^* Q$.

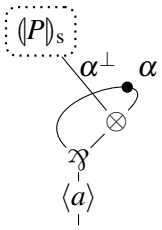
Definition 5 (type assignment)**.** *Terms of MCCS are translated into MLLa formulas as follows, where in each case α is a fresh propositional variable:*

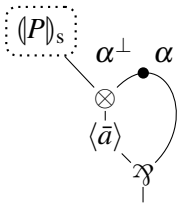
$$\begin{aligned}
\lceil 1 \rceil_{\mathrm{s}} &:= \alpha^\perp \parr \alpha \\
\lceil P \mid Q \rceil_{\mathrm{s}} &:= \lceil P \rceil_{\mathrm{s}} \otimes \lceil Q \rceil_{\mathrm{s}} \\
\lceil a.P \rceil_{\mathrm{s}} &:= \langle a \rangle (\alpha^\perp \parr (\lceil P \rceil_{\mathrm{s}} \otimes \alpha))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

The freshness condition is a way to enforce polymorphism in our translation: each proposition variable occurs exactly 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 simplicity. 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 slight proof-theoretic complications of the units.

Proposition 1 (proof assignment)**.** *For every MCCS term P, there is a unique cut-free proof of $\lceil P \rceil_{\mathrm{s}}$. This proof will be denoted as $(\!|P|\!)_{\mathrm{s}}$.*

Proof. The existence of a proof of $\lceil P \rceil_{\mathrm{s}}$ is proved by a straightforward induction on P. The case of action prefixes is as follows:

$(\!|a.P|\!)_{\mathrm{s}} =$ 

$(\!|\bar{a}.P|\!)_{\mathrm{s}} =$ 

The key point for uniqueness is that the freshness constraint on propositional variables imposes that for each variable α there must be an axiom link between the occurrence of α and that of $\alpha^\perp$, and subsequently that all connectives must be explicitly introduced in a proof of $\lceil P \rceil_{\mathrm{s}}$. □

This proof assignment property is formulated in the absence of locations. We enrich it in the obvious way with location information: the modality link $\langle a \rangle$ that corresponds to a prefix $a^\ell.P$ gets labelled by the location ℓ.

Theorem 2 below states the correspondence between process execution and provability of transitions. In order to go from proofs to executions, we need to read back terms from proofs, in order to get a kind of reverse operation for $(\!|-|\!)_{\mathrm{s}}$. A direct read-back is hard to formulate, however we can define a particular case of read-back for a proof that is derived from some $(\!|P|\!)_{\mathrm{s}}$ if we know P.

Definition 6 (term extraction)**.** *Let P be a term of MCCS. An MLLa proof structure π is said to be* compatible *with P if $\mathscr{L}(\pi) \subseteq \mathscr{L}(P)$, subjects and polarities coincide between P and π and the order $\leq_P$ is included in the order $\leq_\pi$.*

For π compatible with P we define the term $P \upharpoonright \pi$ to be the MCCS term with locations $\mathscr{L}(\pi)$, subjects and polarities as in P and π and action order as in P.

Remark that for all terms P, by construction $(\!|P|\!)_{\mathrm{s}}$ is compatible with P and we have $P \upharpoonright (\!|P|\!)_{\mathrm{s}} \equiv P$. Note also that compatibility implies that the process is *less* constrained than the proof: given a proof π, the set of processes compatible with π contains terms that are more parallel than the structure of π (and in particular the process that is just a parallel composition of all actions that correspond to modality links in π). The stricter ordering in the proof π effectively means that π has made decisions on how P will run, as illustrated by the lemma below and Theorem 2.

Lemma 1. *Assume P is a term and π be a proof structure compatible with P. For any cut-elimination step $\pi \to \pi'$, the structure π' is compatible with P and one of the two following situations occurs:*

- *either $\pi \to \pi'$ is an elimination of modality links $\langle a\rangle$ and $\langle \bar{a}\rangle$ at some locations ℓ and m, then $P \upharpoonright \pi \to_{\{(\ell,m)\}} P \upharpoonright \pi'$ is a valid execution step,*
- *or it is another elimination step and $P \upharpoonright \pi \equiv P \upharpoonright \pi'$.*

Proof. The case of an elimination of modality links follows from the definition of term extraction: the actions in the extracted term are necessarily at top level since the modalities are premisses of a cut so their locations are minimal with respect to $\leq_\pi$ hence with respect to $\leq_P$ too. In π' the location set is the same with ℓ and m removed and the other data (subjects, polarities and proof order) are simply restricted to this new set, so compatibility with P is preserved.

The other cut elimination steps can be either multiplicative eliminations or axiom eliminations. In these cases the location set is unchanged. For multiplicative steps the proof order is unchanged too, while in the case of axiom eliminations the proof order is possibly strengthened, since one tree of links in the proof is merged on top of another. In both cases, this preserves compatibility with P. □

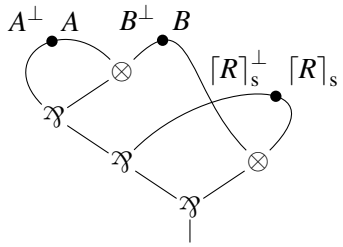
Remark that the strengthening of the proof order in the case of axiom elimination, i.e. the fact that if $\pi \to \pi'$ by an axiom elimination then $\leq_\pi$ may be strictly included in $\leq_{\pi'}$, is the very reason why we need term extraction: in this case the syntactic structure of the proof is more constrained than that of the process term P we are observing, so we cannot read back a reduct of P without extra information.

Theorem 2. *Let P and Q be two MCCS terms. There is an execution $P \to^* Q$ if and only if $\lceil P\rceil_{\mathrm{s}} \multimap \lceil Q\rceil_{\mathrm{s}}$ is provable in MLL (without modality rules) for some instantiation of the propositional variables of $\lceil P\rceil_{\mathrm{s}}$.*

Proof. For the direct implication, remark that for a structural congruence $P \equiv Q$ at top level (i.e. not under prefixes), an implication $\lceil P\rceil_{\mathrm{s}} \multimap \lceil Q\rceil_{\mathrm{s}}$ is provided by a standard isomorphism for the associativity and commutativity of the 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

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

so we get a proof as follows, if we instantiate α as $\lceil Q\rceil_{\mathrm{s}}$ and β as $\lceil P\rceil_{\mathrm{s}} \otimes \lceil Q\rceil_{\mathrm{s}}$:



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

For the reverse implication, consider an MLL proof π of $\lceil P \rceil_{\mathrm{s}}^\perp, \lceil Q \rceil_{\mathrm{s}}$ for some instantiation of the variables of $\lceil P \rceil_{\mathrm{s}}$. If we cut this proof against $(\!|P|\!)_{\mathrm{s}}$ (where the type variables are instantiated as in π), we get a proof ρ of $\lceil Q \rceil_{\mathrm{s}}$. By construction, the cut-elimination procedure reduces each proof into a cut-free proof with the same conclusion, so cut-elimination in ρ reaches a cut-free proof of $\lceil Q \rceil_{\mathrm{s}}$; by Proposition 1 there is only one such proof, hence cut-elimination of ρ reaches $(\!|Q|\!)_{\mathrm{s}}$. Since π is an MLL proof, it contains no modality link, hence ρ is compatible with P according to Definition 6 and by Lemma 1 we know that each cut-elimination step of ρ induces either a structural congruence or an execution step in terms extracted from P by intermediate proofs, so from a cut-elimination sequence $(\!|P|\!)_{\mathrm{s}} \to^* (\!|Q|\!)_{\mathrm{s}}$ we can extract a CCS-execution sequence $P \to^* Q$, and we also deduce that $P \upharpoonright (\!|Q|\!)_{\mathrm{s}} = Q$. □

4.2 Asynchronous executions

The tight correspondence of Theorem 2 is obtained thanks to the fact that the action order $\leq_P$ of processes is mapped to proof order $\leq_{(\!|P|\!)_{\mathrm{s}}}$ because of syntactic structure of the type $\lceil P \rceil_{\mathrm{s}}$, in which modalities are nested as in P. The downside of this correspondence is that it does not leave much space for modular reasoning about the behaviour on different channels in processes. Besides, it does not account for proofs extracted from executions as in our previous work [5].

In this section, in order to generalise this result, we provide an asynchronous variant of the correspondence. We discuss below the relevance of this variant.

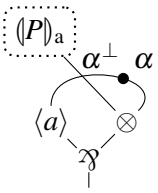
Definition 7 (asynchronous type assignment)**.** *Terms of MCCS are translated into MLLa formulas as follows, where in each case α is a propositional variable:*

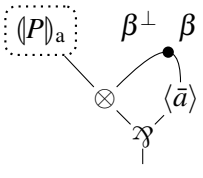
$$\begin{aligned}
\lceil 1 \rceil_{\mathrm{a}} &:= \alpha^\perp \parr \alpha \\
\lceil P \,|\, Q \rceil_{\mathrm{a}} &:= \lceil P \rceil_{\mathrm{a}} \otimes \lceil Q \rceil_{\mathrm{a}} \\
\lceil a.P \rceil_{\mathrm{a}} &:= \langle a \rangle \alpha^\perp \parr (\lceil P \rceil_{\mathrm{a}} \otimes \alpha) && = \langle \bar{a} \rangle \alpha \multimap (\lceil P \rceil_{\mathrm{a}} \otimes \alpha) \\
\lceil \bar{a}.P \rceil_{\mathrm{a}} &:= (\lceil P \rceil_{\mathrm{a}} \otimes \alpha^\perp) \parr \langle \bar{a} \rangle \alpha && = (\lceil P \rceil_{\mathrm{a}} \multimap \alpha) \multimap \langle \bar{a} \rangle \alpha
\end{aligned}$$

Note that this translation is the same as the synchronous translation of Definition 5, except for the position of modalities.

Proposition 2 (asynchronous proof assignment)**.** *For every MCCS term P, there is a unique cut-free proof of $\lceil P \rceil_{\mathrm{a}}$. This proof will be denoted as $(\!|P|\!)_{\mathrm{a}}$.*

Proof. The argument is the same as in Proposition 1. The case of action prefixes is now as follows:

$(\!|a.P|\!)_{\mathrm{a}} =$ 

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

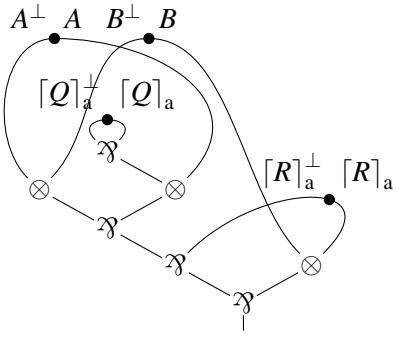
□

Proposition 3. *For all MCCS reduction $P \to Q$, the formula $\lceil P \rceil_{\mathrm{a}} \multimap \lceil Q \rceil_{\mathrm{a}}$ is provable in MLL for some instantiation of the variables of $\lceil P \rceil_{\mathrm{a}}$.*

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

$$\lceil (a.P \,|\, \bar{a}.Q) \,|\, R \rceil_{\mathrm{a}} \multimap \lceil (P \,|\, Q) \,|\, R \rceil_{\mathrm{a}} =$$
$$((\langle \bar{a} \rangle \alpha \otimes (\lceil P \rceil_{\mathrm{a}}^{\perp} \parr \alpha^{\perp}) \parr ((\lceil Q \rceil_{\mathrm{a}}^{\perp} \parr \beta) \otimes \langle a \rangle \beta^{\perp})) \parr \lceil R \rceil_{\mathrm{a}}^{\perp}) \parr ((\lceil P \rceil_{\mathrm{a}} \otimes \lceil Q \rceil_{\mathrm{a}}) \otimes \lceil R \rceil_{\mathrm{a}})$$

with the following proof:



$$\text{with} \begin{cases} \alpha = \lceil Q \rceil_{\mathrm{a}} \\ \beta = \lceil Q \rceil_{\mathrm{a}} \\ A = \langle a \rangle \lceil Q \rceil_{\mathrm{a}}^{\perp} \\ B = \lceil P \rceil_{\mathrm{s}} \otimes \lceil Q \rceil_{\mathrm{s}} \end{cases}$$

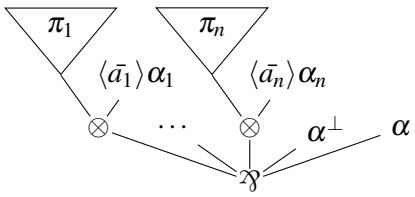
□

Lemma 2. *Let P be an MCCS term with at least one action prefix. If $\lceil P \rceil_{\mathrm{a}} \multimap \lceil 1 \rceil_{\mathrm{a}}$ is provable in MLL for some instantiation of the propositional variables, then there exists a reduction $P \to Q$ such that $\lceil Q \rceil_{\mathrm{a}} \multimap \lceil 1 \rceil_{\mathrm{a}}$ is also provable.*

Note that the condition on P is that it is not already congruent to 1, otherwise $\lceil P \rceil_{\mathrm{a}} \multimap \lceil 1 \rceil_{\mathrm{a}}$ is provable in MLL (as well as the reverse implication) but obviously P does not reduce. However, the statement does *not* require that P must reducible (i.e. that it is not in "normal form" for execution): the existence of a proof of $\lceil P \rceil_{\mathrm{a}} \multimap \lceil 1 \rceil_{\mathrm{a}}$ does imply this fact.

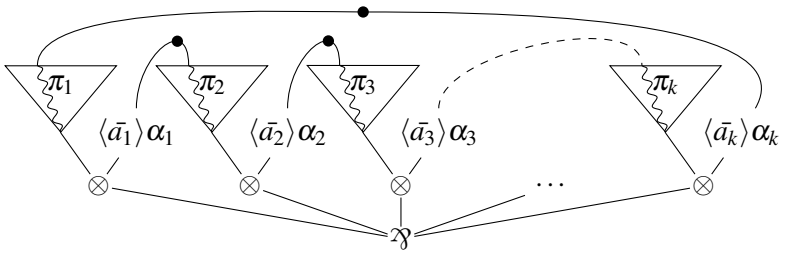
Proof. Consider a proof π of $\lceil P \rceil_{\mathrm{a}}^{\perp} \parr \lceil 1 \rceil_{\mathrm{a}}$ for some instantiation of the propositional variables of $\lceil P \rceil_{\mathrm{a}}$. MLL admits cut elimination and expansion of axiom links, so we may assume without loss of generality that π is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

The term P can be written as a composition of action prefixes $a_1.P_1 \,|\, \cdots \,|\, a_n.P_n$ (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:



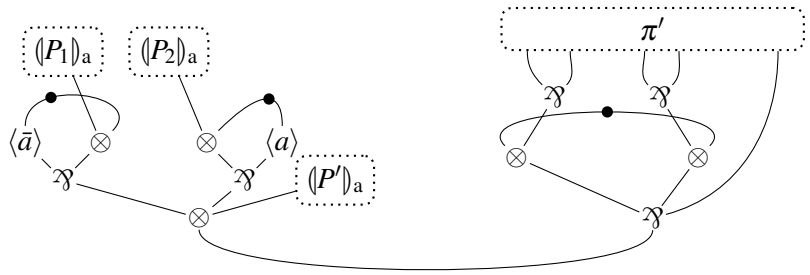
with axiom links on top (the $\parr$ link of arity $n+2$ in the picture actually stands for some tree of binary $\parr$ links, corresponding to the bracketing of parallel compositions in P, since our language only includes binary connectives). Note that, since π does not use modality rules, all premisses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

We now argue that two of these are actually connected by an axiom rule. Towards a 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 $1, f(1), f^2(1), \dots$ is eventually periodic. Consider a minimal cycle $i, f(i), f^2(i), \dots, f^k(i) = i$. Up to the reordering of sub-terms of P, let us assume this sequence is $1, 2, \dots, k, 1$. 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 $\parr$ link. Hence there is a $\parr$-switching of π that has a cycle, which violates the Danos-Regnier correctness criterion for multiplicative proof nets [10].

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 $(\!|P|\!)_{\mathrm{a}}$ 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 $(\!|P_1|\!)_{\mathrm{a}}$, $(\!|P_2|\!)_{\mathrm{a}}$ and $(\!|P'|\!)_{\mathrm{a}}$ plus a cut between the two remaining ports of π'. This reduct is also a reduct of $(\!|P_1 \mid P_2 \mid P'|\!)_{\mathrm{a}}$ against π' plus two $\parr$ rules. The latter is thus an MLL proof of $\lceil P_1 \mid P_2 \mid P' \rceil_{\mathrm{a}} \multimap \lceil 1 \rceil_{\mathrm{a}}$. □

Theorem 3. *Let P be an MCCS term. There is an execution $P \to^* 1$ if and only if $\lceil P \rceil_{\mathrm{a}} \multimap \lceil 1 \rceil_{\mathrm{a}}$ is provable in MLL (without modality rules) for some instantiation of the propositional variables.*

Proof. For the direct implication, Proposition 3 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 2 provides the induction step. □

The interpretation of Theorem 3 is more subtle than for Theorem 2, because in this variant there is no step-by-step correspondence between execution and cut elimination. Of course, Lemma 2 does prove that there is a *strategy* for eliminating cuts that corresponds to an execution, but in general the normalisation of $(\!|P|\!)_{\mathrm{a}}$ against a particular proof of $\lceil P\rceil_{\mathrm{a}} \multimap \lceil 1\rceil_{\mathrm{a}}$ may consume modalities in P in an arbitrary order. This is why Theorem 3 requires the final type to be $\lceil 1\rceil_{\mathrm{a}}$: the statement like of Theorem 2 is actually false for the asynchronous variant.

Definition 8. *Let $\Rightarrow$ be the relation over processes such that $P \Rightarrow Q$ holds whenever $\lceil P\rceil_{\mathrm{a}} \multimap \lceil Q\rceil_{\mathrm{a}}$ is provable in MLL for some instantiation of the propositional variables of $\lceil P\rceil_{\mathrm{a}}$.*

By Proposition 3 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.

5 Discussion

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

$$\lceil !P\rceil_{\mathrm{a}} = !\lceil P\rceil_{\mathrm{a}} \qquad \lceil P+Q\rceil_{\mathrm{a}} = \lceil P\rceil_{\mathrm{a}} \,\&\, \lceil Q\rceil_{\mathrm{a}}$$

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.

5.1 Semantics of processes

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 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.

Definition 9 (pairing)**.** *A* pairing *of a term P is a partial involution c over $\mathscr{L}(P)$ 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. c is consistent *if $\mathrm{dom}\, c$ 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. In our previous work [5], we established the following precise relationship between pairings and executions:

Theorem 4. *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 $\lceil P \rceil_{\mathrm{a}} \multimap \lceil 1 \rceil_{\mathrm{a}}$. In a cut elimination process of $(\!|P|\!)_{\mathrm{a}}$ against π, eventually each modality link in $(\!|P|\!)_{\mathrm{a}}$ will be eliminated by a dual link also from $(\!|P|\!)_{\mathrm{a}}$, and this association between modality links is of course independent of the cut elimination sequence since proof normalisation is confluent. Hence π induces a pairing between modality links in $(\!|P|\!)_{\mathrm{a}}$, and since modality links are in bijection with locations in P, this in turn induces a pairing of P. This pairing is maximal since it reaches all locations in P, and by Theorem 3 it is consistent. Reciprocally, every maximal consistent pairing of P induces a class of proofs of $\lceil P \rceil_{\mathrm{a}} \multimap \lceil 1 \rceil_{\mathrm{a}}$.

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 $\lceil P \rceil_{\mathrm{a}} \multimap \lceil 1 \rceil_{\mathrm{a}}$, as observed in the proof of Lemma 2. 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, moreover the instantiation of propositional variables is far from obvious since, as in the proof of Proposition 3, it may include the whole type of a term involved in the execution. We defer the precise study of this relationship to future work.

The next step in semantics is the study of denotational models of proofs in our system, which should provide new insights for the denotational semantics of processes. For instance, it is a trivial remark that coherence spaces [13] have a formal similarity with event structures [25], although a precise relationship is hard to formulate, especially in a proofs-as-processes approach. Since our constructions maps processes to types, i.e. spaces, and schedules to proofs i.e. cliques or configurations in spaces, it should be possible to formalize such a relationship in our setting in a meaningful way. A difficulty in this respect is of course to define a non-trivial interpretation of modalities: this is needed in order to identify cliques that are interpretations of multiplicative proofs, as opposed to general proofs with modalities.

5.2 Name hiding and passing

The question of how to integrate name hiding in our system is crucial as it is necessary for expressiveness and modularity. However, Theorems 2 and 3, which characterize execution in the proof system, do not extend to a calculus with name hiding. Indeed, a reasonable type assignment for a term $(\nu a)P$ should not let a appear in the formula $\lceil (\nu a)P \rceil$, yet actions on this bound a should be scheduled.

A syntactic way of handling name hiding could be to allow quantification over channel names in the logic, so that (νa) in a process would become a quantification 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. However, the nature of such a quantifier is unclear:

- A universal $\forall a$ is not an option, since it would allow the scheduler to provide any name for a in $(\nu a)P$, including one that P or its environment already knows, which would produce new possible interactions.
- An existential $\exists a$ is better suited since it forces the scheduler to behave independently of the actual name used, effectively treating a as fresh. However, a proper correspondence can only be obtained only if, by construction, $\exists a$ is only introduced with a premiss where a is fresh, and such a restriction is very unnatural as it does not respect the meaning of connectives in the target logic.
- Using a specific quantifier for freshness, like Miller and Tiu's ∇ [20], would probably solve this issue but at the price of introducing an ad-hoc construct with its particular theory in an otherwise well-studied logic, with the consequence that it would make the semantic explorations mentioned above more difficult.

A very different approach, more semantic than the use of a quantifier, would be to decide that placing a (νa) 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 (i.e. a proof without modality rules).

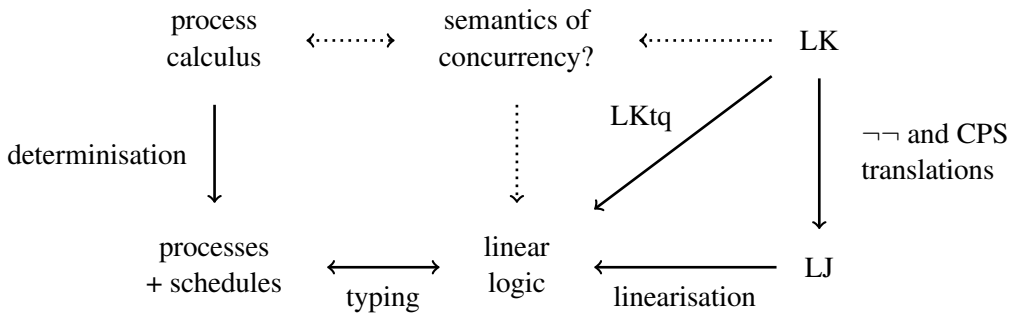
- Only the asynchronous translation of section 4.2 can handle this approach, since Theorem 2 proves that any partial schedule in the synchronous variant is the beginning of an execution. This is consistent with the fact that it amounts to scheduling a channel in advance of actual execution.
- It blurs the distinction between a process and a scheduler, since now a process does include scheduling decisions. This issue could be solved by typing a process $(\nu a)P$ with something like $S \multimap \lceil P' \rceil_{\mathrm{a}}$ where S is the type of a scheduler for a channel (depending on how a is used in P) and P' is a type for P where a is hidden.

The latter approach would most likely require second-order quantification in order to specify S. Further study of this point should also be the way to approach the problem of extending our work to name passing calculi: once we know how to properly specify what a channel is for restricting it, we can investigate how to communicate it through another channel.

5.3 CPS translations and determinisation

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 $\lfloor \bar{a}.P \rfloor = \forall \alpha.(\lfloor P \rfloor \multimap \alpha) \multimap \alpha$, which is a weak form of double-negation, so weak in fact that this formula, without modalities, is isomorphic to $\lfloor P \rfloor$. This suggests a kind of (linear) continuation-passing-style translation of processes into 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 desequentialisation of processes (like translations into solos [4, 17]) as a translation between logical systems, in a way similar to CPS translations from classical logic into intuitionistic logic.

This idea of CPS-like translations indeed suggest a step in a wider programme for the approach of non-determinism in logic. It is well known that the classical sequent calculus LK is non confluent, to the point that any two proofs of the same formula can be identified by conversion through cut-elimination, which makes any direct denotational semantics of proofs degenerate, and that interesting semantics can be obtained by means of double-negation translations into intuitionistic logic. Such translations are effective because they impose constraints on proof reduction, which on the computational side corresponds to fixing reduction strategies like call-by-name or call-by-value in languages with control. It is a form of determinisation, and such translations have been extensively studied. The systematic study of decompositions of LK into linear logic by Danos, Joinet and Schellinx in the system LKtq [9] is of particular interest with respect to the present work since it combines determinisation and linearisation. Reading these translations from the computational point of view provides translations of classical proofs into a deterministic process calculus.



Our approach of non-determinism by means of scheduling suggest a different take on these questions: would it be possible to provide a form of linearisation of classical proofs that would preserve the intrinsic non-determinism of LK, so that various evaluations strategies could be obtained by effectively choosing different types of schedulers? The semantic study of our proofs-as-schedules correspondence could be an approach to this kind of question by means of a study of the semantics of processes.

References

[1] Samson Abramsky (1994): *Proofs as processes*. *Theoretical Computer Science* 135(1), pp. 5–9, doi:10.1016/0304-3975(94)00103-0.

[2] Steffen van Bakel, Luca Cardelli & Maria Grazia Vigliotti (2014): *Classical cut-elimination in the π-calculus*.

[3] Emmanuel Beffara (2006): *A concurrent model for linear logic*. In: *21st International Conference on Mathematical Foundations of Programming Semantics (MFPS)*, *Electronic Notes in Theoretical Computer Science* 155, pp. 147–168, doi:10.1016/j.entcs.2005.11.055.

[4] Emmanuel Beffara & François Maurel (2006): *Concurrent nets: A study of prefixing in process calculi*. *Theoretical Computer Science* 356(3), pp. 356–373, doi:10.1016/j.tcs.2006.02.009.

[5] Emmanuel Beffara & Virgile Mogbil (2012): *Proofs as executions*. In Jos C. M. Baeten, Tom Ball & Frank S. de Boer, editors: *Proceedings of IFIP TCS*, *Lecture Notes in Computer Science* 7604, Springer, pp. 280–294, doi:10.1007/978-3-642-33475-7_20.

[6] Gianluigi Bellin & Philip J. Scott (1994): *On the π-calculus and linear logic*. *Theoretical Computer Science* 135(1), pp. 11–65, doi:10.1016/0304-3975(94)00104-9.

[7] Paola Bruscoli (2002): *A purely logical account of sequentiality in proof search*. In Peter J. Stuckey, editor: *International Conference on Logic Programming (ICLP)*, *Lecture Notes in Computer Science* 2401, Springer, pp. 302–316, doi:10.1007/3-540-45619-8_21.

[8] Luís Caires & Frank Pfenning (2010): *Session types as intuitionistic linear propositions*. In Paul Gastin & François Laroussinie, editors: *Proceedings of the 21st International Conference on Concurrency Theory*, *Lecture Notes in Computer Science* 6269, Springer Berlin Heidelberg, Paris, France, pp. 222–236, doi:10.1007/978-3-642-15375-4_16.

[9] Vincent Danos, Jean-Baptiste Joinet & Harold Schellinx (1997): *A new deconstructive logic: linear logic*. *The Journal of Symbolic Logic* 62(3), pp. 755–807, doi:10.2307/2275572.

[10] Vincent Danos & Laurent Regnier (1989): *The structure of multiplicatives*. *Archive for Mathematical Logic* 28(3), pp. 181–203, doi:10.1007/BF01622878.

[11] Thomas Ehrhard & Olivier Laurent (2010): *Interpreting a finitary π-calculus in differential interaction nets*. *Information and Computation* 208(6), pp. 606–633, doi:10.1016/j.ic.2009.06.005.

[12] Thomas Ehrhard & Laurent Regnier (2006): *Differential interaction nets*. *Theoretical Computer Science* 364(2), pp. 166–195, doi:10.1016/j.tcs.2006.08.003.

[13] Jean-Yves Girard (1987): *Linear logic*. *Theoretical Computer Science* 50(1), pp. 1–101, doi:10.1016/0304-3975(87)90045-4.

[14] Jean-Yves Girard (1996): *Proof-nets: the parallel syntax for proof-theory*. In Paolo Aglianò & Aldo Ursini, editors: *Logic and Algebra*, *Lecture Notes in Pure and Applied Mathematics* 180, Marcel Dekker, New York, pp. 97–124.

[15] Matthew Hennessy & Robin Milner (1985): *Algebraic laws for nondeterminism and concurrency*. *Journal of the ACM* 32(1), pp. 137–161, doi:10.1145/2455.2460.

[16] Kohei Honda & Olivier Laurent (2010): *An exact correspondence between a typed π-calculus and polarised proof-nets*. *Theoretical Computer Science* 411(22–24), pp. 2223–2238, doi:10.1016/j.tcs.2010.01.028.

[17] Cosimo Laneve & Björn Victor (2003): *Solos in concert*. *Mathematical Structures in Computer Science* 13(5), pp. 657–683, doi:10.1017/S0960129503004055.

[18] François Maurel (2003): *Nondeterministic light logics and NP time*. In Martin Hofmann, editor: *Typed Lambda Calculi and Applications*, *Lecture Notes in Computer Science* 2701, Springer, pp. 241–255, doi:10.1007/3-540-44904-3_17.

[19] Dale Miller (1992): *The π-calculus as a theory in linear logic: Preliminary results*. In Evelina Lamma & Paola Mello, editors: *Extensions of Logic Programming, Third International Workshop (WELP'92)*, *Lecture Notes in Computer Science* 660, Springer, pp. 242–264, doi:10.1007/3-540-56454-3_13.

[20] Dale Miller & Alwen Tiu (2005): *A proof theory for generic judgments*. *ACM Transactions on Computational Logic (TOCL)* 6(4), p. 749–783, doi:10.1145/1094622.1094628.

[21] Robin Milner (1989): *Communication and concurrency*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[22] Virgile Mogbil (2010): *Non-deterministic boolean proof nets*. In Marko van Eekelen & Olha Shkaravska, editors: *Foundational and Practical Aspects of Resource Analysis*, Lecture Notes in Computer Science, Springer, pp. 131–145, doi:10.1007/978-3-642-15331-0_9.

[23] Kazushige Terui (2004): *Proof nets and boolean circuits*. In: *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science*, LICS '04, IEEE Computer Society, Washington, DC, USA, p. 182–191, doi:10.1109/LICS.2004.34.

[24] Alwen Tiu & Dale Miller (2005): *A proof search specification of the π-calculus*. *Electronic Notes in Theoretical Computer Science* 138(1), pp. 79–101, doi:10.1016/j.entcs.2005.05.006.

[25] Glynn Winskel (1987): *Event structures*. In W. Brauer, W. Reisig & G. Rozenberg, editors: *Petri Nets: Applications and Relationships to Other Models of Concurrency*, *Lecture Notes in Computer Science* 255, Springer Berlin Heidelberg, pp. 325–392, doi:10.1007/3-540-17906-2_31.

[26] Nobuko Yoshida, Martin Berger & Kohei Honda (2001): *Strong normalisation in the π-calculus*. In: *16th Annual IEEE Symposium on Logic in Computer Science, 2001.*, pp. 311–322, doi:10.1109/LICS.2001.932507.