WoLLIC 2006

Relating State-Based and Process-Based Concurrency through Linear Logic

Iliano Cervesato [1]

Deductive Solutions
Annandale, VA — USA
`iliano@deductivesolutions.com`

Andre Scedrov [2]

Mathematics Department, University of Pennsylvania
Philadelphia, PA — USA
`scedrov@math.upenn.edu`

Abstract

This paper has the purpose of reviewing some of the established relationships between logic and concurrency, and of exploring new ones.

Concurrent and distributed systems are notoriously hard to get right. Therefore, following an approach that has proved highly beneficial for sequential programs, much effort has been invested in tracing the foundations of concurrency in logic. The starting points of such investigations have been various idealized languages of concurrent and distributed programming, in particular the well-established state-transformation model inspired to Petri nets and multiset rewriting, and the prolific process-based models such as the π-calculus and other process algebras. In nearly all cases, the target of these investigations has been linear logic, a formal language that supports a view of formulas as consumable resources. In the first part of this paper, we review some of these interpretations of concurrent languages into linear logic.

In the second part of the paper, we propose a completely new approach to understanding concurrent and distributed programming as a manifestation of logic, which yields a language that merges those two main paradigms of concurrency. Specifically, we present a new semantics for multiset rewriting founded on an alternative view of linear logic. The resulting interpretation is extended with a majority of linear connectives into the language of *ω-multisets*. This interpretation drops the distinction between multiset elements and rewrite rules, and considerably enriches the expressive power of standard multiset rewriting with embedded rules, choice, replication, and more. Derivations are now primarily viewed as open objects, and are closed only to examine intermediate rewriting states. The resulting language can also be interpreted as a process algebra. For example, a simple translation maps process constructors of the asynchronous π-calculus to rewrite operators, while the structural equivalence corresponds directly to logically-motivated structural properties of ω-multisets (with one exception).

Key words: Linear logic, multiset rewriting, process algebra.

1 Introduction

In his seminal paper [26], Girard anticipated the potential for linear logic to act as a model for concurrency, but left the task of precisely pinpointing this relationship to the research community. This challenge was soon taken up by numerous researchers who explored the link between the then new and promising formalism and various understandings of the notion of concurrency and distributed computing.

The *state-transition model* of concurrency [17,35,41,51], epitomized by place-transition Petri nets and propositional multiset rewriting (the two formalisms being syntactic variants of each other), was almost immediately given an interpretation in linear logic in the work of numerous researchers. Asperti [6] and Gunter and Gehlot [28,27] independently explored the relation from a proof-theoretic point of view, noticing that once Petri nets were interpreted as logical theories in the multiplicative fragment of linear logic, their computation amounted to proofs. Kanovich [32] followed a similar path to study the complexity of sublanguages of linear logic. Instead, Martí-Oliet and Meseguer [37,38] and Brown and Gurr [13] approached the issue from a categorical perspective, motivating the use of additional linear connectives as net operators. Engberg and Winskel [21] reached a similar conclusion using quantales, an early model of linear logic. A few years later, Cervesato [14] compiled a comparison of a number of encodings of linear logic. In this paradigm, concurrent computation takes place on a global state shared by all agents. Each agent can act on portions of this state by applying transformations which are often modeled as rewrite rules. Rules operating on disjoint portions of the state can be applied in any order, possibly concurrently. Iterating the application of rules will produce a succession of states. This leads to the natural notion of reachability among states. A number of actual programming and specification languages have been based on this notion of concurrency, the most prominent being Maude [19,41] (which actually mechanizes a broader form of rewriting), Colored Petri Nets [31], and the programming language GAMMA [35]. The interpretation of the state transition model of concurrency in linear logic relies on two ob-

[1] Cervesato was partially supported by OSD/ONR CIP/SW URI “Software Quality and Infrastructure Protection for Diffuse Computing” through ONR Grant N00014-01-1-0795 and by NRL under contract N00173-00-C-2086.

[2] Scedrov was partially supported by OSD/ONR CIP/SW URI “Software Quality and Infrastructure Protection for Diffuse Computing” through ONR Grant N00014-01-1-0795 and OSD/ONR CIP/SW URI “Trustworthy Infrastructure, Mechanisms, and Experimentation for Diffuse Computing” through ONR Grant N00014-04-1-0725. Additional support from NSF Grants CNS-0429689 and CNS-0524059.

servations: first, this formalism embeds connectives that have the same monoidal algebraic structure as multisets; second, its ability to "consume" context formulas during the construction of a derivation ideally models the non-monotonic nature of rule application. This permits simulating multiset reachability by derivability in linear logic. This basic interpretation has been extended to more expressive languages based on the state transition model. In particular, we have enriched it in [16] to support a first-order notion of multiset rewriting with existentials which we have extensively used to model cryptographic protocols [15,17,20], an eminently subtle type of distributed systems.

The alternative *process-based model* of concurrency identifies each agent with a process and communications between agents replace the global state as the vehicle of computation. Languages following this model include CSP [29], CCS and the π-calculus [49,55], the join calculus [25], and a large number of other process algebras, each characterized by subtle differences in behavior. The correlation between logic and process algebra has been investigated along two planes, with occasional contacts. The first approach encodes process operators as term constructors so that a process is represented by a term in the logic. Within this *process-as-term* model, process computation takes the shape of term reduction. Abramsky [2] and Bellin and Scott [9] rely on classical linear logic for this purpose. Miller et al. have performed a similar investigation using intuitionistic linear logic [39], and more recently using a refinement of linear logic with a new quantifier that resembles name generation [45,57]. Abramsky has recently suggested extracting processes from proofs [3]. The process-as-terms approach provides a simple way to logically express relations between processes, such as bisimulation, although capturing both may- and must-properties of processes has remained a challenge. The alternative encoding, known as *process-as-formula*, maps process constructors to logical connectives and quantifiers, with the intended effect of identifying computation with derivability. Bisimulation, structural equivalence and other process relations now correspond to meta-level properties of the logic itself. Linear logic has proved a suitable candidate for this purpose, although some issues are not satisfactorily resolved yet. This approach, which goes back to early work by Andreoli and Pareschi [5], has been applied to the π-calculus by several authors [18,39,42] and to the study of security protocols [16]. A few researchers have compared the process-as-term and process-as-formulas approaches [39] or used them together [18]. Readers interested in a broader perspective of the research on process algebra and (linear) logic may start from the web page of a recent workshop [1] dedicated to this lively topic.

The first part of this paper has the purpose of reviewing some of the process-as-formula interpretations of concurrency into linear logic in a methodical way. While the treatment of the state-transformation model will be fairly complete, we refrain from any claim of exhaustiveness in relation to the many process-based languages as active research in underway to achieve a unified understanding of their subtle semantic differences (we postulate however that logic could be the appropriate middle ground to frame these differences). Furthermore, we will not discuss at all the proof-as-term approach.

The second part of the paper builds on this tutorial introduction to the field and reports on recent research whose intent is to explore an alternative interpretation of the relationship between concurrency and (linear) logic. It stems from the observation that although the aforementioned efforts have drawn useful bridges between linear logic and concurrency, they often make a rather limited use of the logic and often target limited aspects of concurrency. Indeed, adopting derivability as a meta-theoretic target for the interpretation has the effect of reducing the semantics of concurrency to finitary concepts such as reachability (with [39] being a partial exception). Instead, a concurrent system is typically open-ended, meant to have infinite computations. In this paper, we postulate that the traditionally static notion of derivation is insufficient to fully capture the semantics of a concurrent system. Instead, we investigate the use of standard logical inference rules to build open, possibly infinite, proofs that closely model the infinitary behavior that characterizes concurrent systems. Moreover, nearly all solutions are interpretation of a concurrent language *into* linear logic rather than *as* linear logic (with [2,9] being exceptions). In those proposals, the logic is subordinate to the concurrent language: the interleaving of connectives and quantifiers is frozen by the translation procedure, and there is often little interest in extending these interpretations with additional linear logic constructs. By contrast, we propose a methodology that interprets most connectives and all quantifiers in intuitionistic linear logic as the operators of a freely generated concurrent language. This language embeds the targeted translations mentioned above (and several others) and may be the first formalism that makes both the state-transition and the process-based models of concurrency and distributed computing available in the same language.

We develop this idea with respect to a fragment of intuitionistic linear logic [26] in Pfenning's LV sequent presentation [53], which we reinterpret in a highly unusual way to provide a new understanding of concurrent and distributed programming. We turn LV's left rules into a form of rewriting over logical contexts. It transforms a rule's conclusion into its major premise, with minor premises corresponding to finite auxiliary rewriting chains (they can be in-lined using the cut rules). The axiom rule becomes a means of observing the rewriting process. A few of LV's right rules indirectly contribute to a notion of equivalence, while the rest is discarded. It is shown that LV's cut rules are admissible.

The resulting system, which we call ω, is much weaker than LV (because of the absence of right rules), but constitute a powerful form of rewriting. We show that a tiny syntactic fragment of ω corresponds exactly to traditional multiset rewriting (or place/transition Petri nets). This constitutes an interpretation of multiset rewriting *as* (a fragment of) logic [2,9], which we like to contrast to most previous interpretations *into* (a fragment of) logic [6,13,14,21,27,32,38]. The system ω similarly provides a new logical foundation to more sophisticated forms of multiset rewriting and Petri nets.

Pushing this methodology further, we view ω as an extreme form of multiset

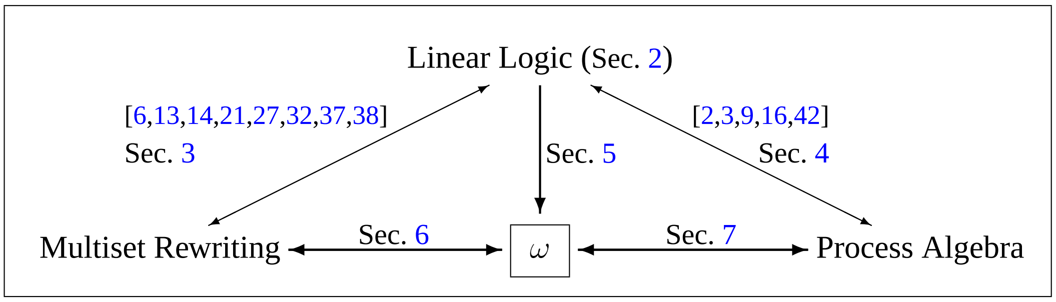

Fig. 1. Overview

rewriting: it drops the distinction between multiset elements and rewrite rules, and considerably enriches the expressive power of standard multiset rewriting with embedded rules, parametricity, choice, replication and more. Yet, its semantics is given by the rules of logic. Under this interpretation, we call formulas ω-multisets.

The system ω has also close ties to process algebra, in particular to the join calculus [25] and the asynchronous π-calculus [49,55]. A simple execution-preserving translation maps process constructors of the latter to rewrite operators, while its structural equivalence corresponds directly to logically motivated properties of ω (with one exception).

With relations to the two major paradigms for distributed and concurrent computing, ω is a promising middle ground where both state-based and process-based specifications can coexist. This prospect is particularly appealing because each paradigm has developed its own theories, tools and verification methodologies, which are often complementary and overlap only partially. Mappings of one model to the other have for the most part failed however to carry the benefits of each over to the other. The integrated language we propose has the potential of fostering new ways to use these theories, tools and methodologies cooperatively.

The tutorial portion of this paper starts with a quick refresher of key elements of linear logic in Section 2. We then describe in some detail the traditional correspondence between multiset rewriting and linear logic in Section 3 and conclude with a description of some embeddings of process algebra into this logic in Section 4.

The research portion of the paper starts with Section 5 which distills ω out of LV. Section 6 exposes ω as a new form of multiset rewriting. Section 7 relates it to the process algebraic world. Additional remarks and ideas for future developments are given in Section 8. Figure 1 summarizes the functional relations between the various languages touched in this paper, as found in the literature (along the thin edges) and in the present work (along the thick edges).

2 A Very Brief Review of Linear Logic

Linear logic was defined in [26] with the aim of overcoming some representational shortcomings of traditional logic. It quickly reached a wide audience and the new possibilities offered by this formalism were soon exploited in number of fields. Girard's original paper [26] already foresees the benefits of the expressiveness of linear logic as a tool for describing concurrent systems.

Linear logic is a refinement of traditional logic based on the idea of providing explicit control over the number of times an assumption can be used in a proof. While the set of assumptions, or context, grows monotonically in a traditional derivation, the controlled-use option of linear logic allows contexts to grow and shrink as logical rules are applied. This property is crucial in order to model concurrent systems, hence the popularity of linear logic for this purpose. Control over context formulas is obtained by replacing the connectives of traditional logic with a new set of operators. For example, conjunction ($A \wedge B$) gives way to a multiplicative tensor ($A \otimes B$) which forces its subformulas to compete for assumptions, and to an additive conjunction ($A \mathbin{\&} B$) which instead require that they use the exact same assumptions. The expressiveness of traditional logic is recovered by flagging some assumptions as reusable and promoting this concept to a first-class status as new modal operators (*e.g.*, $!A$ allows A to be used arbitrarily many times).

Linear logic comes in as many variants as traditional logic: classical, intuitionistic, minimal, propositional, first-order, higher-order, etc. In this paper, we will base our investigation on the following fragment of intuitionistic linear logic [26]:

$$A, B, C \ ::= \ a \ \mid \ \mathbf{1} \ \mid \ A \otimes B \ \mid \ A \multimap B \ \mid \ !A \ \mid \ \top \ \mid \ A \mathbin{\&} B \ \mid \ \forall x.\, A \ \mid \ \exists x.\, A$$

Here, a and x range over atomic formulas and variables, respectively. We do not distinguish formulas that differ only by the name of their bound variables, and rely on implicit α-renaming whenever convenient. We write $[t/x]A$ for the capture avoiding substitution of term t for x in A, and $\mathrm{FV}(A)$ for the set of free variables occurring in A. We shall not place any restriction on the embedded term language except for predicativity (term substitution cannot alter the outer structure of a formula). However, the applications in this paper will only require a first-order term language. In addition to the operators mentioned at the beginning of this section, we make use of the multiplicative and additive versions of truth, $\mathbf{1}$ and $\top$ respectively, of multiplicative implication $\multimap$, and of the usual quantifiers. Other operators of linear logic (for example the multiplicative and additive notions of disjunction, $\parr$ and $\oplus$, and falsehood, $\bot$ and $\mathbf{0}$) will not be of primary importance in this paper: although some authors have used them to express concurrency, these ideas can often be recast in the fragment examined here by exploiting duality. We will however briefly comment on them in appropriate sections of the paper.

Our definition of provability is based on an intuitionistic version of Pfenning's LV sequent calculus [53]. It relies on sequents of the form

$$\Gamma; \Delta \ \longrightarrow_{\Sigma} \ C.$$

Similarly to Barber's DILL [8] and Hodas and Miller's $\mathcal{L}$ [30], LV isolates reusable assumptions in the *unrestricted context* Γ (subject to exchange, weakening and contraction), while assumptions to be used exactly once are contained in the *linear context* Δ (subject only to exchange). The combination corresponds to the single context $(!\Gamma, \Delta)$ of Girard [26]. The *signature* Σ lists the term-level symbols in use.

Structural rules

$$\frac{}{\Gamma; A \longrightarrow_{\Sigma} A}\,\mathbf{id} \qquad \frac{\Gamma \text{⁏} A; \Delta, A \longrightarrow_{\Sigma} C}{\Gamma \text{⁏} A; \Delta \longrightarrow_{\Sigma} C}\,\mathbf{clone}$$

Cut rules

$$\frac{\Gamma; \Delta_1 \longrightarrow_{\Sigma} A \quad \Gamma; \Delta_2, A \longrightarrow_{\Sigma} C}{\Gamma; \Delta_1, \Delta_2 \longrightarrow_{\Sigma} C}\,\mathbf{cut} \qquad \frac{\Gamma; \cdot \longrightarrow_{\Sigma} A \quad \Gamma \text{⁏} A; \Delta \longrightarrow_{\Sigma} C}{\Gamma; \Delta \longrightarrow_{\Sigma} C}\,\mathbf{cut!}$$

Left rules

$$\frac{\Gamma; \Delta \longrightarrow_{\Sigma} C}{\Gamma; \Delta, \mathbf{1} \longrightarrow_{\Sigma} C}\,\mathbf{1}_{\mathbf{l}} \qquad \frac{\Gamma; \Delta, A_1, A_2 \longrightarrow_{\Sigma} C}{\Gamma; \Delta, A_1 \otimes A_2 \longrightarrow_{\Sigma} C}\,\otimes_{\mathbf{l}}$$

$$\frac{\Gamma; \Delta_1 \longrightarrow_{\Sigma} A \quad \Gamma; \Delta_2, B \longrightarrow_{\Sigma} C}{\Gamma; \Delta_1, \Delta_2, A \multimap B \longrightarrow_{\Sigma} C}\,\multimap_{\mathbf{l}}$$

$$(\textit{No}\ \top_{\mathbf{l}}) \qquad \frac{\Gamma; \Delta, A_i \longrightarrow_{\Sigma} C}{\Gamma; \Delta, A_1 \,\&\, A_2 \longrightarrow_{\Sigma} C}\,\&_{\mathbf{l}i}$$

$$\frac{\Gamma \text{⁏} A; \Delta \longrightarrow_{\Sigma} C}{\Gamma; \Delta, !A \longrightarrow_{\Sigma} C}\,!_{\mathbf{l}} \qquad \frac{\Sigma \vdash t \quad \Gamma; \Delta, [t/x]A \longrightarrow_{\Sigma} C}{\Gamma; \Delta, \forall x.\, A \longrightarrow_{\Sigma} C}\,\forall_{\mathbf{l}} \qquad \frac{\Gamma; \Delta, A \longrightarrow_{\Sigma \text{„} x} C}{\Gamma; \Delta, \exists x.\, A \longrightarrow_{\Sigma} C}\,\exists_{\mathbf{l}}$$

Selected right rules

$$\frac{}{\Gamma; \cdot \longrightarrow_{\Sigma} \mathbf{1}}\,\mathbf{1}_{\mathbf{r}} \qquad \frac{\Gamma; \Delta_1 \longrightarrow_{\Sigma} C_1 \quad \Gamma; \Delta_2 \longrightarrow_{\Sigma} C_2}{\Gamma; \Delta_1, \Delta_2 \longrightarrow_{\Sigma} C_1 \otimes C_2}\,\otimes_{\mathbf{r}} \qquad \frac{\Sigma \vdash t \quad \Gamma; \Delta \longrightarrow_{\Sigma} [t/x]C}{\Gamma; \Delta \longrightarrow_{\Sigma} \exists x.\, C}\,\exists_{\mathbf{r}}$$

Fig. 2. LV Sequent Presentation of Intuitionistic Linear Logic

We call C the *goal formula*.

We shall be very precise when discussing the structure of contexts and signatures. Therefore, we will use different symbols for their constructors, as given by the following grammar:

$$\Delta ::= \cdot \mid \Delta, A$$
$$\Gamma ::= \circ \mid \Gamma \text{⁏} A$$
$$\Sigma ::= \text{-} \mid \Sigma \text{„} x$$

For each of these collections, the comma (“,”, “⁏”, “„”) stands for the extension operator while the bullet (“·”, “∘”, “-”) represents the empty collection. The former will be overloaded into a union operator. From an algebraic perspective, signatures, linear and unrestricted contexts will be commutative monoids. Additionally, signatures shall not contain duplicate symbols (we will extend them only with eigenvariables and rely on implicit α-renaming to ensure this constraint).

Given these conventions, Figure 2 presents an intuitionistic subset of the sequent rules for LV [53]. The first segment contains the axiom rule (**id**) and rule

clone that allows repeatedly using an unrestricted assumption in a derivation. The second segment lists the two applicable cut rules of LV. The left sequent rules for the fragment considered above are listed next. Observe how !'ed linear assumption are made available in the unrestricted context in rule $!_l$. In rule $\forall_l$, we rely on the auxiliary judgment $\Sigma \vdash t$ to ascertain that the term t is valid with respect to signature Σ (but do not define this notion further).

Whenever one of these rules has premises, one of them mentions the same goal formula (systematically written C) as the rule's conclusion. We will call it the *major premise* of the rule. The cut rules and $\multimap_l$ also have a *minor premise* in which the goal formula changes.

The right sequent rules of linear logic will have marginal importance in the second part of this paper. The bottom part of Figure 2 lists some of them, as they are sufficient for the first part of the paper and will play an indirect role in later developments. It is conceivable, however, that these and other right rules can be useful query tools, as demonstrated for example in [21,27] relative to Petri nets. This however goes beyond the scope of this work.

Derivations are defined as usual, and denoted $\mathcal{D}$. In the second part of this paper, we will emphasize the process of constructing a derivation starting from a given sequent. A partial derivation $\mathcal{D}[\,]$ missing justification for exactly one sequent is *incomplete*. $\mathcal{D}[\,]$ is called *open* if it is incomplete along a path from the end-sequent that only follows the major premises of the rules.

We write $\equiv$ for the notion of logical equivalence given by inter-derivability, formally, $A_1 \equiv A_2$ iff for all Σ, Γ, there are derivations for both $\Gamma; A_1 \longrightarrow_\Sigma A_2$ and $\Gamma; A_2 \longrightarrow_\Sigma A_1$.

3 Traditional Interpretation of State-Transition Languages

A large number of languages for parallel and distributed programming are based on the *state transition paradigm*, in which concurrent computation takes place on a global state shared by all participating agents. Each agent has at its disposal transitions which allow it to make changes to the current state, possibly enabling other agents to perform steps. Transitions operating on disjoint portions of the state can be applied in any order, possibly concurrently.

This paradigm was first described in abstract form by Petri [51,52] in a class of graphical models altogether known as Petri nets. One particular model, place-transition Petri nets, has become de facto canonical. Colored Petri Nets, an industrial "graphical oriented language for design, specification, simulation and verification of systems" [31] directly builds on this approach. Nowadays, more often than not, the state transition paradigm takes the form of a term rewriting system, with transitions expressed as rewrite rules. Several specification and programming languages endorse this view, for example the conditional concurrent rewriting framework Maude [19,41], the programming language GAMMA [35], and the security protocol specification language MSR [17,15]. Most model checkers also embrace this view of concurrency, for example [40] in the sphere of security. Down under, all these languages are extensions of propositional multiset rewriting, which we see as a fundamental model of the state transition paradigm. State-transition Petri nets and propositional multiset rewriting are indeed syntactic variants of each other.

Using the vocabulary of multiset rewriting, we identify a state with a multiset $\tilde{s}$ of atomic symbols. We model transitions as rewrite rules of the form $\tilde{a} \to \tilde{b}$, where $\tilde{a}$ and $\tilde{b}$ are multisets: $\tilde{a} \to \tilde{b}$ is applicable in state $\tilde{s}$ if $\tilde{a}$ is contained within $\tilde{s}$; moreover applying this rule has the effect of removing $\tilde{a}$ from $\tilde{s}$ and replacing it with $\tilde{b}$. Iterating the application of rules will produce a succession of states. This leads to the natural notion of reachability of a state $\tilde{s}'$ from $\tilde{s}$, which we denote $\tilde{s} \rhd^*_R \tilde{s}'$ where R is the set of all the rules available to the agents.

The interpretation of the state transition model of concurrency in linear logic relies on two observation: first, this formalism embeds connectives that have the same monoidal algebraic structure as multisets; second, linear logic provides a mechanism to consume some assumptions and create new ones, which is exactly what is needed to simulate rule application. Specifically, a multiset $\tilde{s}$ can be represented as the tensor product $\otimes\tilde{s}$ of its elements so that the translation of a rule $\tilde{a} \to \tilde{b}$ as the linear implication $\otimes\tilde{a} \multimap \otimes\tilde{b}$ allows simulating multiset reachability by derivability in linear logic:

$$\text{if} \quad \tilde{s} \rhd^*_R \tilde{s}', \quad \text{then} \quad \ulcorner R \urcorner; \otimes\tilde{s} \longrightarrow \otimes\tilde{s}'$$

where $\ulcorner R \urcorner$ denotes the translation of all rules in R as outlined above. The reverse statement holds for a syntactically restricted fragment of linear logic. This basic interpretation has been extended to more expressive languages based on the state transition model. In particular, we have enriched it in [16] to support a first-order notion of multiset rewriting, which is at the basis of most practical languages based on the state transition paradigm.

We formally define propositional multiset rewriting and the above intuitive interpretation in linear logic in Section 3.1. We then extend this relationship to a form of first-order multiset rewriting in Section 3.2, and comment on alternative translations in Section 3.3.

3.1 Propositional Multiset Rewriting

We start with the most basic form of multiset rewriting, which can be seen as a notational variant of place/transition Petri nets. The language of *propositional multiset rewriting* (MSR_0 hereafter) is given by the following grammar:

Multisets	$\tilde{s}, \tilde{a}, \tilde{b}, \tilde{c} \ ::= \ \cdot \ \mid \ \tilde{s}; s$
Multiset rewrite rules	$r \ ::= \ \tilde{a} \to \tilde{b}$
Rule sets	$R \ ::= \ \cdot \ \mid \ R; r$

where s refers to an element of the *support set* S. Multisets $\tilde{s}$ are elements of the monoid freely generated from S, the multiset union operator "$;$" and the empty multiset "$\cdot$". A rule set R is simply a set of rewrite rules.

A rule $r = \tilde{a} \to \tilde{b}$ is *applicable* in a *state* $\tilde{s}$, if $\tilde{s}$ contains r's antecedent $\tilde{a}$ (*i.e.*, $\tilde{s} = \tilde{c} ; \tilde{a}$ for some $\tilde{c}$). In these circumstances, the *application* of r to $\tilde{s}$ yields the state $\tilde{s}'$ obtained by replacing $\tilde{a}$ with r's consequent $\tilde{b}$ in $\tilde{s}$ (*i.e.*, $\tilde{s}' = \tilde{c} ; \tilde{b}$). This is expressed by the basic multiset rewriting judgment $\tilde{s} \rhd_R \tilde{s}'$, which is formally defined by the following transition pattern:

$$msr_0 \quad : \qquad (\tilde{c} ; \tilde{a}) \rhd_{R;(\tilde{a}\to\tilde{b})} (\tilde{c} ; \tilde{b})$$

We write $\_ \rhd^*_{\_} \_$ for its reflexive and transitive closure.

The close affinity between multiset rewriting and simple fragments of linear logic has been known for a long time [6,13,14,21,27,33,38]. Indeed tensorial formulas obey the same monoidal laws as contexts, and the semantic rule msr_0 can be emulated using $\multimap_{\mathbf{l}}$ and a few auxiliary rules. We construct an homomorphic mapping by interpreting "$\cdot$", "$;$", $\to$, "$\cdot$" and "$;$" as "$\mathbf{1}$", "$\otimes$", $\multimap$, $\circ$ and $\fatsemi$ respectively. We naturally extend this mapping to the relative syntactic categories, and write $\ulcorner X \urcorner$ for the linear logic formula corresponding to entity X. More formally:

$$\begin{array}{rcl}
\ulcorner \cdot \urcorner & = & \mathbf{1} \\
\ulcorner \tilde{s} ; s \urcorner & = & \ulcorner \tilde{s} \urcorner \otimes s \\
\ulcorner \tilde{a} \to \tilde{b} \urcorner & = & \ulcorner \tilde{a} \urcorner \multimap \ulcorner \tilde{b} \urcorner \\
\ulcorner \cdot \urcorner & = & \circ \\
\ulcorner R ; r \urcorner & = & \ulcorner R \urcorner \fatsemi \ulcorner r \urcorner
\end{array}$$

The soundness of this encoding, which states that reachability between two states can be simulated by the derivability of their representations, is formally given by the following simple property:

Property 3.1 *For every pair of states $\tilde{s}$, $\tilde{s}'$ and every rule set R, if $\tilde{s} \rhd^*_R \tilde{s}'$, then $\ulcorner R \urcorner ; \ulcorner \tilde{s} \urcorner \longrightarrow \ulcorner \tilde{s}' \urcorner$.*

Proof. The proof proceeds by induction on the length of the transition chain. The base case is trivial. The proof of the step case requires showing that for every single-rule application $\tilde{s} \rhd_{R;r} \tilde{s}'$ the sequent $\ulcorner R; r \urcorner ; \ulcorner \tilde{s} \urcorner \longrightarrow \ulcorner \tilde{s}' \urcorner$ is derivable. Such a derivation is constructed by using rule $\mathbf{clone}$ to bring the encoding of the rule r in R into the linear context, then rule $\multimap_{\mathbf{l}}$ is used to isolate the part of the context corresponding to the antecedent of r and add its consequent to the rest of the context. Applications of rules $\otimes_{\mathbf{l}}$, $\mathbf{1}_{\mathbf{l}}$, $\otimes_{\mathbf{r}}$, $\mathbf{1}_{\mathbf{r}}$ and $\mathbf{cut}$ mediate between tensorial formulas and objects in the context. □

The family of mappings $\ulcorner \_ \urcorner$ identifies a syntactic fragment LL^{MSR_0} of intuitionistic linear logic. Moreover, $\ulcorner \_ \urcorner$ is a bijection over LL^{MSR_0} (modulo the monoidal laws of each formalism), and indeed the inverse of the above property holds with respect to LL^{MSR_0}:

Property 3.2 *For every states $\tilde{s}$, $\tilde{s}'$ and every rule set R, if $\ulcorner R \urcorner ; \ulcorner \tilde{s} \urcorner \longrightarrow \ulcorner \tilde{s}' \urcorner$, then $\tilde{s} \rhd^*_R \tilde{s}'$.*

Proof. This proof is much more involved than that of Property 3.1 as a generic derivation of $\ulcorner R \urcorner; \ulcorner \tilde{s} \urcorner \longrightarrow \ulcorner \tilde{s}' \urcorner$ may not neatly factor into segments that correspond to individual rewrite rule applications, and even when a single rewrite step is applied the interleaving of logical inferences may be quite wild. For this reason, the bulk of the proof consists in the rather tedious task of disentangling a generic derivation of that sequent into an orderly sequence of linear inferences that essentially mimics the construction in the proof of Property 3.1. This derivation transformation is formally based on permutability results among linear inference rules. Some additional details can be found in [16]. □

3.2 First-Order Multiset Rewriting

We now extend the above results to a richer form of multiset rewriting. We consider multiset elements that can carry structured values, and are manipulated by parametric rewrite rules. Banâtre and Le Métayer have developed this basic idea into the programming language GAMMA [7], while Jensen has turned it into the flexible formalism of colored Petri nets [31]. Maude [19,41] extends this concept by supporting the concurrent rewriting of generic terms, not just multisets. This finer model has recently been extended with the possibility of creating fresh data in the security specification language MSR [17]. We take this as the language of *first-order multiset rewriting* (MSR_1 hereafter).

Abstractly, we take the support set S to consist of first-order atomic formulas over some initial signature Σ_0. Rules assume the form

$$\textit{Multiset rewrite rules} \qquad r \;::=\; \forall \vec{x}.\tilde{a} \to \exists \vec{n}.\tilde{b}$$

where $\vec{y}$ denotes a sequence of variables $(y_1, \ldots, y_n)$ for some n. The scope of the universal variables $\vec{x}$ ranges over the whole rule, while the existential variables $\vec{n}$ can appear only in its consequent. We assume implicit α-renaming for both sorts of bound variables. We write $\Sigma \vdash t$ to indicated that t is a valid term over signature Σ, and $\Sigma \vdash \vec{t}$ for the natural extension of this notion to sequences of terms $\vec{t}$. We write $[\vec{t}/\vec{x}]\tilde{a}$ for the simultaneous substitution of terms $\vec{t} = (t_1, \ldots, t_n)$ for the variable $\vec{x} = x_1, \ldots, x_n$ in multiset $\tilde{a}$.

The basic judgment of MSR_1 has the form $\Sigma; \tilde{s} \rhd_R \Sigma'; \tilde{s}'$, where both the initial and final state consist of a signature and a multiset. A rule $r = \forall \vec{x}.\tilde{a} \to \exists \vec{n}.\tilde{b}$ in R is applicable in $\Sigma; \tilde{s}$ if its universal variables can be instantiated to Σ-valid terms $\vec{t}$ so that the antecedent matches $\tilde{s}$ (*i.e.*, $\tilde{s} = \tilde{c}; [\vec{t}/\vec{x}]\tilde{a}$). In this case, applying r results in a state $\Sigma'; \tilde{s}'$ whose signature is obtained by extending Σ with $\vec{n}$ (modulo α-renaming), and $\tilde{s}'$ is given by replacing the discovered instance of $\tilde{a}$ with the corresponding instance of $\tilde{b}$ (*i.e.*, $\tilde{s}' = \tilde{c}; [\vec{t}/\vec{x}]\tilde{b}$). This is summarized by the following schematic transition:

$$msr_1\colon\ \Sigma; (\tilde{c}; [\vec{t}/\vec{x}]\tilde{a}) \rhd_{R;(\forall \vec{x}.\tilde{a} \to \exists \vec{n}.\tilde{b})} (\Sigma, \vec{n}); (\tilde{c}; [\vec{t}/\vec{x}]\tilde{b}) \qquad \text{if } \Sigma \vdash \vec{t}.$$

Again, we write $\_ \rhd^*_{\_} \_$ for the finite iteration of $\_ \rhd_{\_} \_$.

The propositional embedding in Section 3.2 is easily extended to account for the first-order infrastructure just discussed: we shall simply map the rule binders $\forall$ and $\exists$ to the homonymous quantifiers $\forall$ and $\exists$ of linear logic. Then the semantic rule msr_1 compounds a derivation sequence consisting of rule **clone**, zero or more uses of $\forall_\mathbf{l}$, one application of $\multimap_\mathbf{l}$, and zero or more of $\exists_\mathbf{l}$. Formally, this mapping, which we still call $\ulcorner\_\urcorner$, is defined as in the propositional case, except for the translation of rewrite rules:

$$\ulcorner\forall\vec{x}.\tilde{a} \to \exists\vec{n}.\tilde{b}\urcorner = \forall\vec{x}.\ulcorner\tilde{a}\urcorner \multimap \exists\vec{n}.\ulcorner\tilde{b}\urcorner$$

This mapping identifies another fragment LL^{MSR_1} of linear logic, and is again bijective over this fragment. The formal correspondence between MSR$_1$ and LL^{MSR_1} enjoys the following soundness property [16]:

Property 3.3 *For every signatures Σ, Σ', states $\tilde{s}$, $\tilde{s}'$, and rule set R, we have that if $\Sigma; \tilde{s} \rhd^*_R (\Sigma, \Sigma'); \tilde{s}'$, then the sequent $\ulcorner R\urcorner; \ulcorner\tilde{s}\urcorner \longrightarrow_\Sigma \exists\Sigma'.\ulcorner\tilde{s}'\urcorner$ is derivable.*

Proof. This proof proceeds as in the propositional case, with the minor complication of handling the quantifiers. The one aspect worth noting is that every application of rule $\exists_\mathbf{r}$ uses a variable as its substitution term. $\square$

As noted [42], the reverse completeness argument does not hold if we allow rule $\exists_\mathbf{r}$ to be used in its full generality. In fact, the possibility of substituting composite terms t yields derivations that may not correspond to any rewrite sequence. For this reason, we shall restrict our attention to derivations that only use a variable as the substitution term of this rule. We have the following property.

Property 3.4 *For every signatures Σ, Σ', states $\tilde{s}$, $\tilde{s}'$, and rule set R, whenever the sequent $\ulcorner R\urcorner; \ulcorner\tilde{s}\urcorner \longrightarrow_\Sigma \exists\Sigma'.\ulcorner\tilde{s}'\urcorner$ has a derivation where the substitution term in occurrences of rule $\exists_\mathbf{r}$ is always a variable, then $\Sigma; \tilde{s} \rhd^*_R (\Sigma, \Sigma'); \tilde{s}'$.*

Proof. This proof relies on the derivation-transformation technique outlined in the propositional setting. The need to consider the quantifier rules nearly doubles the number of permutation that shall be considered. $\square$

3.3 Discussion

The representation of multiset rewriting in linear logic illustrated above is known as the *conjunctive encoding* because it maps the monoidal structure of multisets to multiplicative conjunction ($\otimes$) and its unit ($\mathbf{1}$). Several authors, for example [44], use the alternative *disjunctive encoding*, which relies on the observation that linear logic endows also multiplicative disjunction $\invamp$ and its unit $\bot$ the algebraic structure of a commutative monoid. Then $\tilde{s}$ is interpreted as $\invamp\tilde{s}$ and the rule $\tilde{a} \to \tilde{b}$ as the implication $\invamp\tilde{a} \multimap \invamp\tilde{b}$. Some authors [44] also dualize the use of the quantifiers $\forall$ anf $\exists$, which yields to using the reverse implication $\invamp\tilde{b} \multimap \invamp\tilde{a}$ to encode the rule $\tilde{a} \to \tilde{b}$.

These two sets of connectives are dual to each other and therefore whenever a sequent is provable, the sequent obtained by exchanging $\otimes$ and $\invamp$, and $\mathbf{1}$ and $\bot$ is also derivable. Thus, the results obtained by these authors are essentially syntactic variants of the properties reported above. The inference rules for $\wp$ and $\perp$ are given in terms of multiple conclusion sequents, of the form $\Gamma; \Delta \longrightarrow_\Sigma \Theta$, where Θ is a multiset of formulas rather than a single formula. For this reason, they make use of the derivation structure of classical linear logic [26], or at least full intuitionistic linear logic [12].

4 Some Logical Interpretations of Process-Based Languages

The *process-based paradigm* is a more recent, alternative, model of concurrency which has attracted a lot of attention, especially because it supports refined mathematical concepts closely related to concrete analysis problems. See [25,29,49,55] for an overview. This paradigm identifies each agent with a process and communications between agents replace the global state as the vehicle of computation. Beyond this common characterization, language vary greatly in the primitives they provide, which often translate in subtle semantic differences. Differently from the transition-based paradigm, there is no abstract language, or even a set of feature, that is universally accepted as the archetypal process algebra. Within the scope of this paper, this necessarily leads to fragmented interpretations into linear logic, which cannot always be readily reconciled. For this reason, the focus of this section will be a specific language, the asynchronous π-calculus [55] which we interpret in linear logic in Section 4.2. For presentation purpose, we first consider a propositional variant in Section 4.1. Other process-based languages and translations are summarily discussed in Section 4.3.

4.1 Propositional Process Algebra

We begin by studying the translation in linear logic of a minimally expressive variant of the π-calculus [55], an instructive exercise before examining the more general case in Section 4.2. Processes in this calculus can synchronize on actions, but without exchanging any value. They can also be replicated and composed in parallel. It is defined by the following grammar:

$$P, Q, R \ ::= \ \mathbf{0} \ \mid \ P \parallel Q \ \mid \ !P \ \mid \ xP \ \mid \ \overline{x}$$

where x and $\overline{x}$ are a *name* and the corresponding *co-name*, respectively. In anticipation of our study of the asynchronous π-calculus in Section 4.2, we do not allow a co-name to be followed by further activities. In Section 4.3, we will comment on the complications of allowing a process continuation, which leads to the synchronous version of the π-calculus.

Processes are endowed with a notion of structural equivalence, written $P \overset{\pi}{\equiv} Q$, given as follows:

$$P \parallel Q \overset{\pi}{\equiv} Q \parallel P \qquad P \parallel \mathbf{0} \overset{\pi}{\equiv} P \qquad P \parallel (Q \parallel R) \overset{\pi}{\equiv} (P \parallel Q) \parallel R$$

$$!P \overset{\pi}{\equiv} P \parallel !P \tag{!}$$

It makes parallel composition ($\|$) a monoidal operator with the null process $\mathbf{0}$ its unit (top line), and also interprets process replication (!P) as the parallel composition of arbitrarily many copies of P (bottom line, marked with "(!)").

Processes evolve through synchronization. In its basic form, such computation is modeled by the judgment $P \rightharpoonup Q$, and defined by the following inference patterns:

$$\frac{}{\overline{x} \parallel xP \rightharpoonup P}\,\mathbf{i/o} \qquad\qquad \frac{P \rightharpoonup P'}{P \parallel Q \rightharpoonup P' \parallel Q}\,\mathbf{cgr}\|$$

The first rule formalizes synchronization with respect to action x. The second entails that parallel composition is permeable to synchronization, but that replication and names block it. The structural equivalence $\overset{\pi}{\equiv}$ can implicitly massage processes before and after synchronization.[3] Let $\_ \rightharpoonup^* \_$ be the reflexive and transitive closure of $\_ \rightharpoonup \_$.

We define an encoding $\ulcorner\_\urcorner$ of this propositional process algebra into linear logic by homomorphically mapping $\mathbf{0}$, $\|$, and ! to $\mathbf{1}$ $\otimes$, and !, respectively. Actions are represented as the corresponding name, with xP mapped as a linear implication with antecedent x and consequent the encoding of P. More formally, $\ulcorner\_\urcorner$ is defined as follows:

$$\begin{array}{rcl}
\ulcorner\mathbf{0}\urcorner & = & \mathbf{1} \\
\ulcorner P \parallel Q\urcorner & = & \ulcorner P\urcorner \otimes \ulcorner Q\urcorner \\
\ulcorner !P\urcorner & = & !\ulcorner P\urcorner \\
\ulcorner xP\urcorner & = & x \multimap \ulcorner P\urcorner \\
\ulcorner \overline{x}\urcorner & = & x
\end{array}$$

The formal correspondence between this process algebra and linear logic is more involved than in the case of multiset rewriting as we must take into consideration structural equivalence ($\overset{\pi}{\equiv}$) in addition to computation ($\rightharpoonup^*$). We will first examine the former as it is defined independently from computation. We would expect that structural equivalence directly maps onto inter-derivability (denoted $\equiv$ earlier). This is not the case however, as there are structurally equivalent processes whose representation in linear logic has nothing to do with each other: $P \overset{\pi}{\equiv} Q$ does not entail $\ulcorner P\urcorner \equiv \ulcorner Q\urcorner$ in general. A close examination of the proof attempt points to the structural equivalence we labeled $(!)$ as the reason of this failure: $\Gamma; A \otimes !A \longrightarrow_\Sigma !A$ is not derivable in linear logic (although the reverse entailment does hold). If $(!)$ did have a counterpart, the expected soundness result would hold, as expressed by the following hypothetical result:

Property 4.1 *Given processes P and Q, if $P \overset{\pi}{\equiv} Q$, then $\ulcorner P\urcorner \equiv \ulcorner Q\urcorner$ <u>modulo the equivalence $!A \equiv A \otimes !A$</u>.*

[3] Alternatively, we could make the dependency of $\rightharpoonup$ on $\overset{\pi}{\equiv}$ explicit by introducing the following rule:

$$\frac{P \overset{\pi}{\equiv} P' \quad P' \rightharpoonup Q' \quad Q' \overset{\pi}{\equiv} Q}{P \rightharpoonup Q}\,\mathbf{cgr}\overset{\pi}{\equiv}$$

Proof. Assuming $!A \equiv A \otimes !A$ as an extra-logical axiom, the proof proceeds by structural induction on a construction of $P \overset{\pi}{\equiv} Q$. $\square$

The corresponding completeness result holds in its full generality:

Property 4.2 *Given processes P and Q, if $\ulcorner P \urcorner \equiv \ulcorner Q \urcorner$, then $P \overset{\pi}{\equiv} Q$.*

Proof. For the reason outlined in the proof of Property 3.2, the derivation underlying the equivalence $\ulcorner P \urcorner \equiv \ulcorner Q \urcorner$ need to be tidied up before the relation can be established by a simple induction. $\square$

Because structural equivalence plays a part in the computation of our process algebra, the soundness result for $\rightarrowtriangle^*$ is subject to the proviso already noted in Property 4.1:

Property 4.3 *Given processes P and Q, if $P \rightarrowtriangle^* Q$, then $\circ; \ulcorner P \urcorner \longrightarrow \ulcorner Q \urcorner$ modulo the equivalence $!A \equiv A \otimes !A$.*

Proof. This proof is again a straightforward induction once this spurious equivalence is assumed as an added axiom. $\square$

Finally, completeness for $\ulcorner \_ \urcorner$ with respect to $\rightarrowtriangle^*$ holds in its full generality, except that we may need to account for replicated processes that may have been discarded in the derivation. The statement is as follows:

Property 4.4 *Let P be a process*

If $\circ; \ulcorner P \urcorner \longrightarrow C$, then there is Γ such that $\bigotimes !\Gamma \otimes C \equiv \ulcorner Q \urcorner$ and $P \rightarrowtriangle^* Q$.

Proof. This proof proceeds in the now usual fashion: inferences need to be reordered according to the permutability laws to parallel process inferences. The context Γ is constructed as follows: whenever rule $\mathbf{id}$ is used on a sequent of the form $\Gamma'; A \longrightarrow A$ we extend the derivation so that it yields $\Gamma'; A \longrightarrow \otimes !\Gamma' \otimes A$, and whenever combining subderivations of this form, we trim common banged formulas using the cut rule and the right rule for ! (not shown in Figure 2). $\square$

Since $!A \equiv A \otimes !A$ interpreted as mutual derivability does not hold in linear logic, it is clear that our encoding, or maybe linear logic itself (as the same issue is cited in [18,42,54]), does not accurately capture execution in the π-calculus, as traditionally defined. It has however been observed that the right-to-left reading of this equivalence is of difficult implementability, which suggest an alternative execution model in which only half of $(!)$ is kept, in the form of an added case in the definition of $\_ \rightarrowtriangle \_$:

$$\frac{}{!P \rightarrowtriangle !P \parallel P}$$

This, which corresponds exactly to rule $!_1$ in ω, turns the above property into an exact correspondence. Therefore, this amended language can be seen as fragment of linear logic in the same sense as MSR$_0$ was identified with LL^{MSR_0} in the previous section, but the language we started with cannot.

4.2 First-Order Process Algebra: the Asynchronous π-Calculus

We now extend the propositional language defined above by allowing actions to carry arguments, so that a co-name process, now of the form $\overline{x}\langle y\rangle$, implements the output of y over the *channel* x, and a name-prefixed process, now $x(y)P$, dually inputs a value from channel x, binds it to variable y, and then passes it to process P. We additionally introduce the hiding operator, $\nu x.P$, which creates a new channel or variable name. Because an output process does not have a continuation, the resulting language corresponds to a minimal form of the asynchronous π-calculus (hereafter $a\pi$). It is formally defined by the following grammar [55]:

$$P, Q, R \ ::= \ \mathbf{0} \ \mid \ P \parallel Q \ \mid \ !P \ \mid \ \nu x.P \ \mid \ x(y)P \ \mid \ \overline{x}\langle y\rangle$$

where x and y are *names* (or *channels*). Hiding ($\nu x.P$) and input over a channel x ($x(y)P$) bind the names x and y respectively, up to α-renaming. We write $\mathrm{FN}(P)$ for the set of names free in process P and $[x/y]P$ for the substitution (renaming) of x for y in P. Input and output ($\overline{x}\langle y\rangle$) are monadic, and the latter can only be the last action of a process (together with $\mathbf{0}$), which makes communication asynchronous. This core calculus can easily be generalized to support polyadic channels, complex terms, and pattern matching.

We generalize the notion of structural equivalence, still written $P \overset{\pi}{\equiv} Q$, to partially allow hiding to commute with parallel composition and other hiding operators. The overall definition of this relation is reported in the following table, where the right side has been added to the clauses in the previous section:

$P \parallel Q \overset{\pi}{\equiv} Q \parallel P$	$\nu x.\nu y.P \overset{\pi}{\equiv} \nu y.\nu x.P$
$P \parallel \mathbf{0} \overset{\pi}{\equiv} P$	$\nu x.\mathbf{0} \overset{\pi}{\equiv} \mathbf{0}$
$P \parallel (Q \parallel R) \overset{\pi}{\equiv} (P \parallel Q) \parallel R$	$\nu x.(P \parallel Q) \overset{\pi}{\equiv} P \parallel \nu x.Q$ if $x \notin \mathrm{FN}(P)$
$!P \overset{\pi}{\equiv} P \parallel !P \quad (!)$	

The computation semantics extends the rules seen in the propositional case to account for the argument of input and output actions, and for hiding. Altogether, they take the following form:

$$\frac{}{\overline{x}\langle y\rangle \parallel x(z)P \multimap [y/z]P}\ \mathbf{i/o} \qquad \frac{P \multimap P'}{P \parallel Q \multimap P' \parallel Q}\ \mathbf{cgr}\parallel \qquad \frac{P \multimap P'}{\nu x.P \multimap \nu x.P'}\ \mathbf{cgr}\nu$$

The first rule formalizes the transmission of a name y over a channel x (*reaction*). The remaining two entail that parallel composition and hiding are permeable to communication, but that replication and input block it. Again, structural equivalence $\overset{\pi}{\equiv}$ can implicitly act on processes during computation.

The encoding of $a\pi$ in linear logic extends the propositional representation given in Section 4.1 with a case for the hiding operator (modeled as an existential quantifier) and revised definitions for input and output. We reserve a binary predicate symbol c and use it as a universal channel when representing input and output:

$\ulcorner \overline{x}\langle y\rangle \urcorner = \mathsf{c}(x,y)$ and $\ulcorner x(y)P \urcorner = \forall y.\, \mathsf{c}(x,y) \multimap \ulcorner P \urcorner$, where $\ulcorner P \urcorner$ is the encoding of the embedded process P. The resulting mapping is therefore as follows:

$$\begin{array}{rcl}
\ulcorner \mathbf{0} \urcorner & = & \mathbf{1} \\
\ulcorner P \parallel Q \urcorner & = & \ulcorner P \urcorner \otimes \ulcorner Q \urcorner \\
\ulcorner !P \urcorner & = & !\ulcorner P \urcorner \\
\ulcorner \nu x. P \urcorner & = & \exists x.\, \ulcorner P \urcorner \\
\ulcorner x(y)P \urcorner & = & \forall y.\, \mathsf{c}(x,y) \multimap \ulcorner P \urcorner \\
\ulcorner \overline{x}\langle y\rangle \urcorner & = & \mathsf{c}(x,y)
\end{array}$$

The soundness and completeness results reported in Section 4.1 for the propositional variant of this calculus extend naturally to the first-order setting. All the provisos discussed there, especially about structural congruence (!), still apply. Furthermore, for the reason already noted when discussing the interpretation of first-order multiset rewriting in linear logic in Section 3.2, the main completeness result shall be restricted to sequents derivable with instances of rule $\exists_{\mathbf{r}}$ that never use non-variable terms as the substitution term. These various results are summarized in the following property, whose proof relies on the techniques discussed earlier.

Property 4.5 *Let P and Q be processes and $\Sigma_P = \mathsf{c}, \mathrm{FN}(P)$.*

- *If $P \overset{\pi}{\equiv} Q$, then $\ulcorner P \urcorner \equiv \ulcorner Q \urcorner$ $\underline{\text{modulo } !A \equiv A \otimes !A}$.*
- *If $\ulcorner P \urcorner \equiv \ulcorner Q \urcorner$, then $P \overset{\pi}{\equiv} Q$.*
- *If $P \rightarrowtriangle^{*} Q$, then $\circ; \ulcorner P \urcorner \longrightarrow_{\Sigma_P} \ulcorner Q \urcorner$ $\underline{\text{modulo } !A \equiv A \otimes !A}$.*
- *If $\circ; \ulcorner P \urcorner \longrightarrow_{\Sigma_P} C$ has a derivation where the substitution term in occurrences of rule $\exists_{\mathbf{r}}$ is always a variable, then there are Σ and Γ such that $\exists \Sigma.\, \bigotimes !\Gamma \otimes C \equiv \ulcorner Q \urcorner$ and $P \rightarrowtriangle^{*} Q$.*

4.3 Discussion

The calculi we examined in the previous two sections are very simple, and so is their interpretation in linear logic, yet it identifies points of friction between the two formalisms, notably about the encoding of structural equivalence. It should also be noted that the semantics we captured is purely operational as it models the evolution of a system as its processes communicate with each other. This is the very simplest, and least interesting, notion of behavior. We will now briefly discuss alternative translations, competing process algebras, and other semantics.

As in the case of multiset rewriting, we used a conjunctive encoding. The dual disjunctive representation, which relies on $\parr$ and $\perp$ where we used $\otimes$ and $\mathbf{1}$, is an equally valid option that several authors have explored (*e.g.* [42]).

As noted earlier, process-based languages come in many variants which have not yet been reduced to a common denominator. The *synchronous π-calculus* [55] differs from the formalism studied in Section 4.2 by allowing outputs processes of the form $\overline{x}\langle y\rangle P$: this process is blocked until some other process synchronizes with it by performing an input on channel x. Such synchronization on output complicates the translation in linear logic, as indirectly pointed out in [11] and [16] because we need to simulate the blocking/unblocking of computation with dedicated tokens: the simple-minded translation of $\overline{x}\langle y\rangle P$ as $\mathsf{c}(x, y) \otimes \ulcorner P \urcorner$ does not work and shall instead be replaced by $\mathsf{w}_x \multimap (\mathsf{c}(x, y) \otimes \ulcorner P \urcorner)$ where the constant w_x needs to be consumed before $\mathsf{c}(x, y)$ can be released — a process available to execute an input on x will provide w_x. The synchronous π-calculus often provides a non-deterministic choice operator, $P + Q$, which allows synchronization with either P or Q. While it is tempting to interpret $+$ as the linear connective $\&$, whose left rule non-deterministically chooses one of the disjuncts to continue the computation, this mapping is inadequate as it ignore the synchronization requirement. While we are unaware of a general solution within linear logic, a correct encoding has been given in the closely related CLF logical framework [18]. Further behavioral variations of the process algebras have been proposed in the literature, see for example [55] for additional variants of the π-calculus. We are not aware of a systematic attempt at interpreting them in linear logic, although we believe such a translation could be beneficial.

The translations given in this section have focused on the operational semantics of process algebras as reduction calculi, which may be used in a programming language [54] or for model checking purposes. Other semantic notions, such as may- and must-testing, or bisimulation, are particularly useful for verification purposes as they can scrutinize fine properties of process expressions. Limited work, mostly relative to the process-as-term interpretation, has aimed at reinterpreting these notions in linear logic, with [42,44] providing an interesting perspective on this little investigated problem.

A number of other interpretations of process algebras in linear logic have been proposed. Abramsky's "proofs-as-processes" relates classical linear logic with the synchronous π-calculus [2,3,9]. Here concurrent computation corresponds to proof normalization (cut elimination), giving the system a functional flavor, with [3] stressing the notion of realizability. Proofs are expressed as proof nets rather than derivations, as done here. Closer to the encodings in this paper are approaches in which logical formulas are identified with processes and proofs with concurrent computations. For example, Miller outlines a translation from the π-calculus into linear logic: processes become formulas and π-calculus reduction becomes entailment [42]. These ideas are generalized and reformulated as a logical framework in Miller's proposal for the specification logic Forum [43].

5 A Rewriting View of Linear Logic

The semantics of a logic is generally given as a set of inference rules that can be composed to build derivations. Traditionally, derivations are used to support judgments such as the entailment of a formula from given assumptions. To this end, a derivation shall be finite and closed, in the sense that the premises of every rule in it are themselves justified by (sub-)derivations.

In sequel, we emphasize a radically different view of rules, derivations, and ultimately logic. We will be primarily interested in the vertical process of extending open derivations upwards, with little concern for finiteness. The horizontal process of closing a derivation (and proving something, in the traditional sense) will be of secondary importance, mostly as a form of observation.

We develop this idea with respect to a fragment of intuitionistic linear logic [26] in Pfenning's LV sequent presentation [53]. We turn LV's left rules into a form of rewriting over logical contexts. It transforms a rule's conclusion into its major premise, with minor premises corresponding to finite auxiliary rewriting chains (they can be in-lined using the cut rules). The axiom rule becomes a means of observing the rewriting process. A few of LV's right rules indirectly contribute to a notion of equivalence, while the rest is discarded. It is shown that LV's cut rules are admissible.

The resulting system, which we call ω, is much weaker than LV (because of the absence of right rules), but constitute a powerful form of rewriting. We show that a tiny syntactic fragment of ω corresponds exactly to traditional multiset rewriting (or place/transition Petri nets). This constitutes an interpretation of multiset rewriting *as* (a fragment of) logic, which we like to contrast to the previous interpretations *into* (a fragment of) logic [6,13,14,21,27,32,38]. The system ω similarly provides a new logical foundation to more sophisticated forms of multiset rewriting and Petri nets.

Pushing this methodology further, we view ω as an extreme form of multiset rewriting: it drops the distinction between multiset elements and rewrite rules, and considerably enriches the expressive power of standard multiset rewriting with embedded rules, parametricity, choice, replication and more. Yet, its semantics is given by the rules of logic. Under this interpretation, we call formulas ω-multisets.

The system ω has also close ties to process algebra, in particular to the join calculus [25] and the asynchronous π-calculus [49,55]. A simple execution-preserving translation maps process constructors of the latter to rewrite operators, while its structural equivalence corresponds directly to logically motivated properties of ω (with one exception).

With relations to the two major paradigms for distributed and concurrent computing, ω is a promising middle ground where both state-based and process-based specifications can coexist.

In this section, we will give a rewriting interpretation to a fragment of linear logic in its LV sequent presentation. We then refine it by cut-elimination into a system that we call ω.

5.1 A Rewriting Interpretation of LV

With the exception of $\mathbf{id}$, the rules in the three upper segments of Figure 2 can be interpreted as a transformation of the sequent in their conclusion to the sequent in their major premise, possibly subject to side-conditions given by a minor premise.

id	:	$\Sigma;\Gamma;\Delta \Rrightarrow^* \Sigma;\Delta$
Trans.	:	$\Sigma;\Gamma;\Delta \Rrightarrow^* \Sigma'';\Delta''$ if $\Sigma;\Gamma;\Delta \Rightarrow \Sigma';\Gamma';\Delta'$
		and $\Sigma';\Gamma';\Delta' \Rrightarrow^* \Sigma'';\Delta''$
clone	:	$\Sigma;(\Gamma, A);\Delta \Rightarrow \Sigma;(\Gamma, A);(\Delta, A)$
cut	:	$\Sigma;\Gamma;(\Delta_1,\Delta_2) \Rightarrow \Sigma;\Gamma;(\Delta_2, A)$
		if $\Sigma;\Gamma;\Delta_1 \Rrightarrow^* \Sigma; A$
cut!	:	$\Sigma;\Gamma;\Delta \Rightarrow \Sigma;(\Gamma, A);\Delta$ if $\Sigma;\Gamma;\cdot \Rrightarrow^* \Sigma; A$
$\mathbf{1}_l$	:	$\Sigma;\Gamma;(\Delta,\mathbf{1}) \Rightarrow \Sigma;\Gamma;\Delta$
$\otimes_l$	:	$\Sigma;\Gamma;(\Delta, A_1 \otimes A_2) \Rightarrow \Sigma;\Gamma;(\Delta, A_1, A_2)$
$\multimap_l$	:	$\Sigma;\Gamma;(\Delta_1,\Delta_2, A \multimap B) \Rightarrow \Sigma;\Gamma;(\Delta_2, B)$
		if $\Sigma;\Gamma;\Delta_1 \Rrightarrow^* \Sigma; A$
$\forall_l$	:	$\Sigma;\Gamma;(\Delta,\forall x.\, A) \Rightarrow \Sigma;\Gamma;(\Delta,[t/x]A)$ if $\Sigma \vdash t$
$\exists_l$	:	$\Sigma;\Gamma;(\Delta,\exists x.\, A) \Rightarrow (\Sigma,, x);\Gamma;(\Delta, A)$
$(\top_l)$	:	(*No rule for* $\top$)
$\&_{li}$	:	$\Sigma;\Gamma;(\Delta, A_1 \& A_2) \Rightarrow \Sigma;\Gamma;(\Delta, A_i)$
$!_l$	:	$\Sigma;\Gamma;(\Delta, !A) \Rightarrow \Sigma;(\Gamma, A);\Delta$

Fig. 3. A Rewriting Interpretation of LV

We formalize this observation as a rewrite system whose *states* are triples $(\Sigma;\Gamma;\Delta)$ consisting of the signature and the two contexts of an LV sequent. We deliberately omit the goal formula (C) for two reasons: technically, it never changes going from the conclusion to the major premise of a rule; strategically, we embrace this as an opportunity to explore logical derivations as open-ended processes rather than finite justifications of the provability of a goal given a priori. We denote this form of upward step in a derivation by means of the rewrite judgment

$$\Sigma;\Gamma;\Delta \Rightarrow \Sigma';\Gamma';\Delta'$$

reserving the form $\_ \Rrightarrow^* \_$ for its reflexive and transitive closure. Our progresses can be tracked on Figure 3.

Given this interpretation, we can regard the minor premise in rules **cut**, **cut**! and $\multimap_l$ as prescribing the existence of an auxiliary finite rewriting chain that enables the step associated to each of these rules (the judgment $\Sigma \vdash t$ in rule $\forall_l$ is instead a simple side-condition). Consolidating this intuition requires introducing some extra machinery. First, note that the subderivation corresponding to this auxiliary chain must be finite, and therefore is capped by a rule without premises, often **id**. This implements a shift of focus from the left-hand side of a sequent to its right-hand side. We interpret this as an observation.

We will be interested in observing the contents of the linear context Δ of an arbitrary state $(\Sigma;\Gamma;\Delta)$. In order to maintain a precise accounting of the symbols in use, we define the *observation* of $(\Sigma;\Gamma;\Delta)$ as the pair $(\Sigma;\Delta)$.[4] Making an

[4] The investigation of a notion of observation that includes the unrestricted context Γ is left for

observation can then be expressed by the judgment

$$\Sigma; \Gamma; \Delta \ \Rrightarrow \Sigma; \Delta.$$

Notice that it differs from the axiom rule $\mathbf{id}$ only because the linear context Δ can be arbitrary rather than a single formula A. We produce an exact correspondence by identifying contexts and formulas, an idea familiar from categorical interpretations of logic [56,10]. More precisely, we identify the tensor $\otimes$ and its unit $\mathbf{1}$ with the union "," and unit "$\cdot$" constructors of linear contexts, respectively. Therefore, a linear context Δ is interpreted as the formula $\bigotimes\Delta$ obtained by tensoring together all its constituent formulas. This is the essence of the symmetric monoidal (closed) structure that underlies most categorical models of linear logic [56,10]. From a sequent calculus point of view, this is acceptable since $\Gamma; \Delta \ \longrightarrow_\Sigma \ C$ has a derivation if and only if $\Gamma; \bigotimes\Delta \ \longrightarrow_\Sigma \ C$ has one. [5]

From now on, we will use $\otimes$ and "," interchangeably (and similarly for $\mathbf{1}$ and "$\cdot$"). For the ease of the reader, we will tend to prefer $\otimes$ and $\mathbf{1}$ within the scope of other logical operators and in observation states, while "," and "$\cdot$" will appear at the top level of a regular state. We shall stress, however, that they are now only notational variants for the same concept.

The algebraic properties of linear contexts as commutative monoids can then be written as explicit *structural laws* under the logical interpretation:

$$\begin{array}{lrcl}
\text{Assoc.} & : A \otimes (B \otimes C) & \equiv & (A \otimes B) \otimes C \\
\text{Unit} & : \qquad A \otimes \mathbf{1} & \equiv & A \\
\text{Comm.} & : \qquad A \otimes B & \equiv & B \otimes A
\end{array}$$

These identities over linear contexts correspond to the notion of logical equivalence given by inter-derivability defined in Section 2, *i.e.*, $A_1 \equiv A_2$ iff for all Σ, Γ, there are derivations for both $\Gamma; A_1 \ \longrightarrow_\Sigma \ A_2$ and $\Gamma; A_2 \ \longrightarrow_\Sigma \ A_1$.

Observe that it would be incorrect to similarly fold the unrestricted context constructors $_{9}$ and $\circ$ into $\otimes$ and $\mathbf{1}$ since $!(A \otimes B)$ is not equivalent to $!A \otimes !B$ in linear logic. This is our main reason for choosing different notations for their constructors.

Going back to our goal of interpreting the subderivations originating on a minor premise as auxiliary rewrite chains, we define the multi-step observation judgment

$$\Sigma; \Gamma; \Delta \ \Rrightarrow^* \Sigma^\star; \Delta^\star$$

future work.

[5] A proof of the forward direction only uses $\otimes_\mathbf{l}$ and possibly $\mathbf{1}_\mathbf{l}$. The reverse direction relies on $\mathbf{cut}$ and the sequent $\Gamma; \Delta \ \longrightarrow_\Sigma \ \bigotimes\Delta$ whose simple derivation uses rules $\mathbf{id}$, $\otimes_\mathbf{r}$ and $\mathbf{1}_\mathbf{r}$; $\mathbf{cut}$ can later be eliminated.

as the composition of $\_ \Rrightarrow^{*} \_$ and $\_ \Rrightarrow \_$, or more directly:

$$
\begin{array}{lll}
\textbf{id} & : \ \Sigma; \Gamma; \Delta \ \Rrightarrow^{*} \ \Sigma; \Delta & \\
\text{Trans.} & : \ \Sigma; \Gamma; \Delta \ \Rrightarrow^{*} \ \Sigma^{\star}; \Delta^{\star} & \text{if } \ \Sigma; \Gamma; \Delta \ \Rrightarrow \ \Sigma'; \Gamma'; \Delta' \\
 & & \text{and } \ \Sigma'; \Gamma'; \Delta' \ \Rrightarrow^{*} \ \Sigma^{\star}; \Delta^{\star}
\end{array}
$$

Were it not for $\exists$, this would constitute an adequate rewriting interpretation of LV. To visualize the remaining issue, consider for example a derivation $\mathcal{D}$ of the minor premise $\Gamma; \Delta_1 \ \longrightarrow_{\Sigma} \ A$ of rule **cut**

$$
\begin{array}{ccc}
\multicolumn{3}{c}{\overline{\Gamma ,\, \Gamma'; A' \ \longrightarrow_{\Sigma ,\, \Sigma'} \ A'}} \\
\nwarrow & \vdots & \swarrow \\
\multicolumn{3}{c}{\Gamma; \Delta_1 \ \longrightarrow_{\Sigma} \ A}
\end{array}
$$

By the time a branch of $\mathcal{D}$ is closed, for example by rule **id** in this sketch, uses of rule $\exists_{\mathbf{l}}$ will have extended the original signature Σ with new symbols Σ' (rule $!_{\mathbf{l}}$ will have similarly extended Γ, but this is of little concern to us). The formula A' in this instance of **id** may mention symbols x_i in Σ', and may also contribute to the overall goal formula A. Now, since A is defined over Σ, the noted uses of x_i in A' must occur bound in A. With Figure 2 as our definition of provability, this binder is $\exists$ and rule $\exists_{\mathbf{r}}$ has introduced it. [6]

With this understanding of derivations as a guideline, we identify observation states $\Sigma; \Delta$ and existential formulas $\exists \Sigma.\, \Delta$, seen as an abbreviation for $\exists x_1. \ldots \exists x_n . \bigotimes \Delta$ where $\Sigma = (x_1, \ldots, x_n)$. This is logically justified by the fact that, if $\Gamma; \Delta \ \longrightarrow_{\Sigma} \ C$ is derivable, so is $\circ; \exists \Sigma.\, (!\Gamma \otimes \Delta) \ \longrightarrow_{\cdot} \ \exists \Sigma.\, C$ where $!\Gamma \otimes \Delta = \bigotimes_{A \text{ in } \Gamma} !A \otimes \bigotimes \Delta$. [7] This technique is reminiscent of the notion of "telescope" in the AUTOMATH languages [60]. It also appears in recent work on concurrent constraint programming [22].

Having further blurred the distinction between the brick and mortar of sequents (or states) and the logical operators, we will use the notations $\Sigma; \Delta$ and $\exists \Sigma.\, \Delta$ interchangeably, often mixing them as in the following sketch of the rewrite chain corresponding to a derivation of the minor premise $\Gamma; \Delta_1 \ \longrightarrow_{\Sigma} \ A$ of the hypothet-

[6] Of course, $\forall$ is as likely a candidate in a complete proof system for linear logic. Our objective is not completeness, however, and this discussion should be taken as motivation only.

[7] This proof extends the technique seen in the forward direction of Footnote 5 with uses of $\exists_{\mathbf{l}}$, $\exists_{\mathbf{r}}$ and $!_{\mathbf{l}}$, in addition to $\otimes_{\mathbf{l}}$ and $\mathbf{1}_{\mathbf{l}}$. Notice that $\exists_{\mathbf{r}}$ invariably uses a variable x in Σ as the substitution term t, giving $\exists$ a flavor very close to Miller and Tiu's ∇ [47,48]. The reverse of this property does not hold in general. With some surprise, we could not find a categorical counterpart of this technique.

ical use of **cut** above:

$$\begin{tikzcd}
(\Sigma,, \Sigma'); (\Gamma, \Gamma'); \Delta^\star \arrow[r, phantom, "\Rrightarrow"] & (\Sigma,, \Sigma'); \Delta^\star \\
\Sigma; \Gamma; \Delta_1 \arrow[u, Rightarrow, "*"] \arrow[r, phantom, "\Rrightarrow^*"] & \Sigma; \underbrace{\exists\Sigma'.\, \Delta^\star}_{A} \arrow[u, phantom, "|||"]
\end{tikzcd}$$

Here, we fold the added symbols Σ' into the observed linear context $\Delta^\star$ in order to construct the formula used in the major premise. Identical considerations apply to any LV rule with a minor premise (here **cut**! and $\multimap_l$).

With the rewrite steps induced by rules **cut**, **cut**! and $\multimap_l$ ending in states of the form $\Sigma; \Gamma; (\Delta, A)$ with $A = \exists\Sigma'.\, \Delta'$, it is natural to allow individual binders $\exists x$ among $\exists\Sigma'$ to move around, either to hug more closely formulas in Δ' or to extend their scope to include elements of Δ, as long as this does not cause either bound symbols to become free or variable capture. We formalize this possibility by means of the following *mobility laws*, which extend the monoidal equivalence $\equiv$ introduced earlier:

$$\begin{array}{lrcl}
\text{assoc. :} & \exists x.\, (A \otimes B) & \equiv & A \otimes \exists x.\, B \text{ if } x \notin \mathrm{FV}(A) \\
\text{unit} \quad : & \exists x.\, \mathbf{1} & \equiv & \mathbf{1} \\
\text{comm.:} & \exists x.\, \exists y.\, A & \equiv & \exists y.\, \exists x.\, A
\end{array}$$

The first pushes binders inside a formula (or state) by skipping objects where it does not occur, the second eliminates unused binders, and the third allows binders to commute. As for the monoidal laws, the formulas on each side of $\equiv$ are interderivable in linear logic. Notice the resemblance between the monoidal and mobility laws (highlighted through related labels), that type theory explains by pointing out that an existential quantifier can be interpreted as a form of dependent conjunction.

This completes our rewriting interpretation of LV. The resulting rewrite rules are summarized in Figure 3. With the exception of the added rule Tran, each maintains the name of the LV inference it was obtained from. Rules $\mathbf{1}_l$ and $\otimes_l$ have been grayed out as redundant since they are subsumed by the identification of linear contexts and tensored formulas. Rules **cut**, **cut**! and $\multimap_l$ make implicit use of the identification of observation states and existential formulas, just described.

The rewriting interpretation displayed in Figure 3 is sound with respect to the rules of linear logic, even when extended with the right rules not considered in Figure 2. This is expressed by the following property:

Property 5.1 (Soundness)

(i) *If* $\Sigma; \Gamma; \Delta \;\; \Rrightarrow^* \;\; \Sigma'; \Gamma;' \Delta'$, *then there exist LV formulas* C *and* C' *and a derivation* $\mathcal{D}[]$ *of* $\Gamma; \Delta \longrightarrow_\Sigma C$ *open at* $\Gamma'; \Delta' \longrightarrow_{\Sigma'} C'$.

id	:	$\Sigma; \Gamma; \Delta \ \Rrightarrow^* \ \Sigma; \Delta$
Trans.	:	$\Sigma; \Gamma; \Delta \ \Rrightarrow^* \ \Sigma''; \Delta''$ if $\Sigma; \Gamma; \Delta \ \Rightarrow \ \Sigma'; \Gamma'; \Delta'$ and $\Sigma'; \Gamma'; \Delta' \ \Rrightarrow^* \ \Sigma''; \Delta''$
clone	:	$\Sigma; (\Gamma , A); \Delta \ \Rightarrow \ \Sigma; (\Gamma , A); (\Delta, A)$
$\multimap_l'$	:	$\Sigma; \Gamma; (\Delta, A, A \multimap B) \ \Rightarrow \ \Sigma; \Gamma; (\Delta, B)$
$\forall_l$	:	$\Sigma; \Gamma; (\Delta, \forall x. \, A) \ \Rightarrow \ \Sigma; \Gamma; (\Delta, [t/x]A)$ if $\Sigma \vdash t$
$\exists_l$	:	$\Sigma; \Gamma; (\Delta, \exists x. \, A) \ \Rightarrow \ (\Sigma ,, x); \Gamma; (\Delta, A)$
$\&_{l i}$	:	$\Sigma; \Gamma; (\Delta, A_1 \,\&\, A_2) \ \Rightarrow \ \Sigma; \Gamma; (\Delta, A_i)$
$!_l$	:	$\Sigma; \Gamma; (\Delta, !A) \ \Rightarrow \ \Sigma; (\Gamma , A); \Delta$

Fig. 4. The Rules of ω-Rewriting

(ii) *If* $\Sigma; \Gamma; \Delta \ \Rrightarrow^* \ (\Sigma, \Sigma'); \Delta^\star$, *then there is an LV derivation* $\mathcal{D}$ *of* $\Gamma; \Delta \ \longrightarrow_\Sigma \ \exists \Sigma'. \bigotimes \Delta^\star$.

Clearly, no completeness result holds as we have forsaken most right rules of LV: for example, no rewrite chain can validate $\cdot; \circ; (a \multimap b, b \multimap c) \ \Rrightarrow^* \ \cdot; (a \multimap c)$, although linear implication *is* transitive in linear.

5.2 *Cut-Elimination and ω-Rewriting*

The interpretation of LV as a rewrite system in Figure 3 is unusual in the sense that the single step relation $\Rightarrow$ depends on the multi-step observation relation $\Rrightarrow^*$ in rules $\multimap_l$, **cut** and **cut**!. In this section, we refine it into a presentation that is immune from this oddity. We will call it system ω, and refer to its use as ω-rewriting.

First observe that rule $\multimap_l$ admits the simplified form

$$\multimap_l{}' \quad : \qquad \Sigma; \Gamma; (\Delta, A, A \multimap B) \ \Rightarrow \ \Sigma; \Gamma; (\Delta, B).$$

Indeed, every use of $\multimap_l$ in a rewrite sequence can be replaced with an instance of **cut** followed by one of $\multimap_l{}'$.

More interestingly, like their logical counterparts [53], both cut rules are admissible in our rewrite system, *i.e.*, any rewriting sequence can be transformed into an observationally equivalent *cut-free* sequence that does not make use of them. Intuitively, this will amount to in-lining the auxiliary rewriting chain whenever one of the cut rules is used. A formal account follows the lines of a standard proof of cut-elimination, *e.g.* [53], but is not as involved.

We first prove the following weakening lemma by a simple induction on the given rewriting sequence.

Lemma 5.2 (Weakening)
For any Σ', Γ' *and* Δ', *if* $\Sigma; \Gamma; \Delta \ \Rrightarrow^* \ \Sigma^\star; \Delta^\star$, *then* $(\Sigma ,, \Sigma'); (\Gamma , \Gamma'); (\Delta, \Delta') \ \Rrightarrow^* \ (\Sigma^\star ,, \Sigma'); (\Delta^\star, \Delta')$.

The most delicate aspect of the work in this section is the proper accounting for signature symbols. This can be summarized in the following lemma, also proved by induction.

Lemma 5.3
If $\Sigma; \Gamma; \exists\Sigma'.\,\Delta \;\Rrightarrow^{*}\; \Sigma^{\star}; \Delta^{\star}$, *then* $(\Sigma, \Sigma'); \Gamma; \Delta \;\Rrightarrow^{*}\; \Sigma^{\star}; \Delta^{\star}$.

We can now tackle the rewriting equivalent of the admissibility of the cut rules that is, every chain that could be produced from two cut-free chains by means of $\mathbf{cut}$ (or $\mathbf{cut}!$), can also be obtained without.

Lemma 5.4 (Admissibility of $\mathbf{cut}$ and $\mathbf{cut}!$)

(i) *For any cut-free rewriting chains* $\Sigma; \Gamma; \Delta_1 \;\Rrightarrow^{*}\; \Sigma; A$ *and* $\Sigma; \Gamma; (\Delta_2, A) \;\Rrightarrow^{*}\; \Sigma^{\star}; \Delta^{\star}$, *there is a cut-free sequence* $\Sigma; \Gamma; (\Delta_1, \Delta_2) \;\Rrightarrow^{*}\; \Sigma^{\star}; \Delta^{\star}$.

(ii) *For any cut-free rewriting chains* $\Sigma; \Gamma; \cdot \;\Rrightarrow^{*}\; \Sigma; A$ *and* $\Sigma; (\Gamma ⨾ A); \Delta \;\Rrightarrow^{*}\; \Sigma^{\star}; \Delta^{\star}$, *there is a cut-free sequence* $\Sigma; \Gamma; \Delta \;\Rrightarrow^{*}\; \Sigma^{\star}; \Delta^{\star}$.

The proof of (1) simply prefixes the first rewriting chain to the second, using the above lemmas to align signatures and contexts. As for (2), we shall replace every application of rule $\mathbf{clone}$ on the formula A with a similar in-lining of the first rewriting chain.

On the basis of this lemma, we can eliminate every occurrence of $\mathbf{cut}$ and $\mathbf{cut}!$ in a rewriting chain.

Theorem 5.5 (Cut elimination)
For every rewrite sequence $\Sigma; \Gamma; \Delta \;\Rrightarrow^{*}\; \Sigma^{\star}; \Delta^{\star}$, *there exist a cut-free rewrite sequence* $\Sigma; \Gamma; \Delta \;\Rrightarrow^{*}\; \Sigma^{\star}; \Delta^{\star}$.

With $\multimap_{\mathrm{l}}$ replaced with $\multimap_{\mathrm{l}}'$ and the cut rules shown to be redundant, the rewriting interpretation of LV is succinctly described by the rules in Figure 4. Notational conventions and structural properties are as in Figure 3. We will refer to these rules as system ω, and to their use as ω-rewriting.

5.3 Discussion

So far, we have extracted a rewriting system from a large fragment of linear logic. Before assessing the rewriting merits of ω in sections to come, we shall conclude this part with reflections on our methodology and comparisons with related ideas from the literature. We start by remarking on a few natural questions, although proper answers will be sought in future work.

First: *What about the other connectives of linear logic?* The remaining operators of minimal intuitionistic logic are $\oplus$ and its unit $\mathbf{0}$. An ω-style reading of the left rule of $\oplus$:

$$\frac{\Gamma; \Delta, A_1 \longrightarrow_{\Sigma} C \qquad \Gamma; \Delta, A_2 \longrightarrow_{\Sigma} C}{\Gamma; \Delta, A_1 \oplus A_2 \longrightarrow_{\Sigma} C}\;\oplus_{\mathrm{l}}$$

seems to require a form of synchronization between two possibly infinite rewrite chains. We do not understand this rule as a rewriting operation at this stage. Its nullary form, $\mathbf{0}_{\mathrm{l}}$, suggests instead a reading of $\mathbf{0}$ as a “mirage” operator, as anything can be observed in its presence. Moving to a multiple conclusion sequent form in the style of FILL [12], the left rule for $\parr$:

$$\frac{\Gamma; \Delta_1, A_1 \longrightarrow_\Sigma \Theta_1 \quad \Gamma; \Delta_2, A_2 \longrightarrow_\Sigma \Theta_2}{\Gamma; \Delta_1, \Delta_2, A_1 \parr A_2 \longrightarrow_\Sigma \Theta_1, \Theta_2} \parr_l$$

seems to endow multiplicative disjunction with a rewriting semantics that splits the state and starts two completely independent computations. However, further research is required to validate this reading and extend the current work to multiple conclusion sequents. We did not venture in the realm of classical linear logic.

Interestingly, the connectives currently comprising ω coincides with the fragment of linear logic at the core of the type-theoretic logical framework for concurrency CLF [18,59]. We do not know at this stage if there is more to this than a mere coincidence.

Second: *How sensitive is the definition of ω to the specific presentation of linear logic?* We chose LV because it elegantly capture the structural characteristics of the logic, especially as far as reusability is concerned. Its rules were amenable to a sensible rewriting interpretation, and it permitted relatively simple proofs of our various results. It is however our untested conjecture that the methodology used to derive ω can be applied to other presentations, probably with different degrees of ease. It will be interesting to compare the resulting rewrite systems.

Third: *Can this methodology be applied to other logics?* We have not tried yet, but this is a reasonable supposition. Linear logic is a good starting point because its interpretation of context formulas as consumable resources is in line with the destructive nature of rewriting. Other sub-structural logics are clearly promising candidates, but it is conceivable that interesting results could emerge from specific presentations of, say, traditional intuitionistic logic.

Fourth: *How does this compare to other proof-as-computations paradigms?* The methodology proposed here places a strong emphasis on the left rules of (linear) logic, with the right rules reduced to justifications of natural equivalences. It is worth contrasting this characteristic with the tenets of logic programming as uniform provability [46], which instead extracts the operational semantics of a logical operator from its right sequent rules. This approach has robustly been extended to linear logic programming [5,30,43]. In a partial departure from this short tradition, Kobayashi and Yonezawa's ACL [34] derives its semantics from specialized versions of left rules of linear logic (when examined through the lense of duality). This, together with its acceptance of open derivations and support for concurrency, makes ACL a close relative to ω. Differently from our proposal, however, it considers a limited fragment of logic, and falls short of endowing it with a rewriting interpretation. Saraswat and Lincoln hint at a similar interpretation for their Higher-order Linear Concurrent Constraint language (HLcc) [36], interestingly stirring it in the direction of constraint programming (see also [22]). To the extent of our knowledge, ACL and HLcc are the closest proposals to ω in the literature.

Fifth: *Can logic benefit from ω?* We will see in just a few lines that ω is intimately related to various languages for concurrent computation, and can be taken to shine a logical light onto them. It remains to be investigated whether this relation can be ridden in the reverse direction as well, *i.e.*, that results and techniques from concurrency theory can find application in logic. One candidate is the very notion of derivation. We endowed ω with a semantics based on transition-*sequences*, which is common place in rewriting theory. It is however a small conceptual step to distill minimal partial orders (*traces*) by forcing sequentiality only when steps depend on each other. System ω may then carry traces over to logic, with a sound and usable notion of derivation not based on trees but on DAGs. Andreoli's "desequentialized proofs" [4] appear closely related to this idea.

6 Multiset Rewriting

As already mentioned, multiset rewriting captures the essence of a paradigm for concurrent and distributed computation characterized by a prominent notion of state, separate from the transitions that act upon it. Other members of this family include Petri nets [52], possibly the earliest model of concurrency, and a number of specification approaches including automata for model checking [40] and inductive definitions [50]. We will now show that the various popular forms of multiset rewriting examined in Section 3 are syntactic fragments of ω-rewriting. Therefore, thanks to the logical foundations laid out in Section 5, this constitutes an interpretation of multiset rewriting *as* linear logic, which we like to contrast to the interpretations *into* linear logic traditionally found in the literature.

6.1 Propositional Multiset Rewriting

Propositional multiset rewriting is immediately recognized as a form of ω-rewriting by interpreting multisets as linear contexts (or tensored formulas) and rule sets as unrestricted contexts. Indeed multisets obey the same monoidal laws as contexts, and the semantic rule msr_0 can be seen as an application of rule $\mathbf{clone}$ immediately followed by $\multimap_1'$. Formally, we construct an homomorphic mapping by interpreting "$\tilde{:}$", "$\tilde{;}$", $\rightarrow$, "$\hat{:}$" and "$\hat{;}$" as ",", "$\cdot$", $\multimap$, $\circ$ and $\fatsemi$ respectively. We naturally extend this mapping to the relative syntactic categories, and again write $\ulcorner X \urcorner$ for the object in ω corresponding to entity X. The soundness of this encoding is formally stated by the following simple property:

Property 6.1 *For states $\tilde{s}$, $\tilde{s}'$ and every rule set R, if $\tilde{s} \rhd_R^* \tilde{s}'$, then $S; \ulcorner R \urcorner; \ulcorner \tilde{s} \urcorner \Rrightarrow^* S; \ulcorner \tilde{s}' \urcorner$.*

The family of mappings summarized as $\ulcorner \_ \urcorner$ identifies a syntactic fragment $\tilde{\omega}_0$ of ω (and linear logic). Moreover, $\ulcorner \_ \urcorner$ is a bijection over $\tilde{\omega}_0$ (modulo the monoidal laws of each formalism). It can then be shown that the inverse of the above property holds:

Property 6.2 *For every states $\tilde{s}$, $\tilde{s}'$ and every rule set R, if $S; \ulcorner R \urcorner; \ulcorner \tilde{s} \urcorner \Rrightarrow^* S; \ulcorner \tilde{s}' \urcorner$, then $\tilde{s} \rhd_R^* \tilde{s}'$.*

Together, these properties and the trivial mapping underlying them allow us to view propositional multiset rewriting as a fragment of ω-rewriting, and therefore of linear logic. In particular, it permits redefining the semantics of MSR$_0$ on a purely logical basis.

6.2 *First-Order Multiset Rewriting*

The propositional embedding in Section 6.1 is easily extended to account for the first-order infrastructure discussed in Section 3.2: we shall simply map the rule binders $\forall$ and $\exists$ to the homonymous quantifiers $\forall$ and $\exists$ of ω. Then the semantic rule msr_1 compounds an ω-rewrite sequence consisting of rule $\mathbf{clone}$, zero or more uses of $\forall_\mathbf{l}$, one application of $\multimap'_\mathbf{l}$, and zero or more of $\exists_\mathbf{l}$.

The resulting mapping, which we still call $\ulcorner\_\urcorner$, identifies another fragment $\tilde{\omega}_1$ of ω, and is again bijective over this fragment. The formal correspondence between MSR$_1$ and system ω is captured by the following property [16]:

Property 6.3 *For every signatures Σ, Σ', states $\tilde{s}$, $\tilde{s}'$, and rule set R, we have that $\Sigma; \tilde{s} \rhd^*_R \Sigma'; \tilde{s}'$ if and only if $\Sigma; \ulcorner R \urcorner; \ulcorner \tilde{s} \urcorner \Rrightarrow^* \Sigma'; \ulcorner \tilde{s}' \urcorner$.*

Again, this result not only logically justifies the semantics of MSR$_1$, but allows viewing this language as a fragment of ω, and ultimately of linear logic.

As noted in [42], a similar relation between MSR$_1$ and linear logic does not hold in the reverse direction. It holds here because of the restricted use of rule $\exists_\mathbf{r}$ embedded in ω (and the incompleteness of this system w.r.t. linear logic).

6.3 *Discussion*

From the above discussion, it is clear that MSR$_1$ accounts only for a very small fragment ($\tilde{\omega}_1$) of ω. We will now explore what else ω has to offer as a rewriting framework, and relate it to proposals in the Petri net and multiset rewriting communities.

In a major departure from traditional state-based formalisms, ω dissolves the boundary between states (usually flat collections of strictly atomic elements, even when carrying structured data) and the actuators of state change (rules). Indeed, objects of the form $A \multimap B$ can appear in the linear context, where they are responsible for the rewriting behavior in $\tilde{\omega}_1$. In this way, ω not only internalizes the rewriting operation within the state, but also makes it available for manipulation as a first-class object.

Furthermore, ω replaces the monolithic transition rules of traditional state-based languages with a toolkit of elementary state transformers drawn from the ranks of linear logic: $\otimes$ and $\mathbf{1}$ (or "," and "$\cdot$") are the basic glue, $\multimap$ expresses rewrite, ! is a reusability mark, $\forall$ introduces parameters, $\exists$ allows generating fresh data, $\&$ offers choice, and $\top$ is the unusable object. Complex transformations can easily be assembled by composing basic operator: an MSR$_1$ rule is an example,

$(a \multimap b) \mathbin{\&} !(c \multimap \mathbf{1})$ is another.[8]

Embedded rewrites, such as $(a \multimap b, (c, d \multimap e))$,[9] are a particularly important case of composition as they allow dynamically modifying the rule set available for rewriting. This will be our bridge to process algebra in the next section.

Similar ideas have been incorporated in enhanced forms of Petri nets, and to a lesser extent into multiset rewriting. Indeed, Valk argued for self-modifying nets as far back as 1978. A number of recent proposals, such as Hierarchical or Object Petri Nets [23,58], fully realize this program by permitting nets to manipulate other nets, often using reflection to move between levels. Among them, Farwer and Misra's Linear Logic Petri Nets [24] are rather interesting as they operate on embedded linear logic formulas. On the multiset rewriting side, Le Métayer outlined a higher-order extension to GAMMA [35], which blurs the distinction between state and rules.

Most of these proposals are motivated by software engineering considerations, often modularity and control, sometimes inspired by process algebra. The resulting formalisms tend to be powerful but also complex, as they build on the already heavy definitions of Petri nets. It is however conceivable that they enjoy embeddings in ω akin to those sketched in Sections 6.1 and 6.2. This would endow these extensions with a formal justification in (linear) logic, and possibly enable simpler presentations.

7 A Logical Bridge to Process Algebra

As we mentioned earlier, formalisms such as the π-calculus [49] support an alternative, *process-based*, representation of distributed and concurrent systems. It shuns the global state and static collection of transitions of multiset rewriting and other state-based models in favor of evolving communicating processes that tie together the data and the program of an agent, at the same time blurring the distinction between them.

We will show in this section that ω is closely related to one such process algebras: the asynchronous π-calculus [49], which we introduced in Section 4. As we do so, we will focus on them as computation rather than analysis mechanisms. In particular, we will concentrate a trace-based semantics, leaving the investigation of finer notions, such as bisimulation, for future work.

We will use the definition for the asynchronous π-calculus already introduced in Section 4.2. The encoding in ω closely follows the linear logic representation discussed there, which we repeat for the benefit of the reader. We define an encoding $\ulcorner \_ \urcorner$ of the asynchronous π-calculus into ω as follows: first we map homomorphically $\mathbf{0}$, $\|$, ν and $!$ to $\mathbf{1}$ (or "$\cdot$"), $\otimes$ (or ","), $\exists$ and $!$, respectively. Then, we reserve a binary predicate symbol c and use it as a universal channel when representing input

[8] "Either turn an a into a b once, or delete arbitrarily many c's."

[9] "Upon encountering an a, transform it into a b and introduce a single-use rule that will transform a c and a d into an e when these object appear in the state."

and output: $\ulcorner \overline{x}\langle y \rangle \urcorner = \mathsf{c}(x,y)$ and $\ulcorner x(y)P \urcorner = \forall y.\, \mathsf{c}(x,y) \multimap \ulcorner P \urcorner$, where $\ulcorner P \urcorner$ is the encoding of the embedded process P. Once more, $\ulcorner \_ \urcorner$ identifies a fragment $\omega_{a\pi}$ of ω so that any formula A in this fragment can unambiguously be interpreted as the representation of some process P, *i.e.*, $A = \ulcorner P \urcorner$.

Just as our interpretation in linear logic in Section 4.2, the expected soundness of $\ulcorner \_ \urcorner$ over execution does not hold in general because $\Gamma; A \otimes !A \longrightarrow_{\Sigma} !A$ is not derivable in linear logic (although the reverse entailment does hold). This yields the following hypothetical result, which is closely related to Property 4.5:

Property 7.1 *Let P be a process and $\Sigma_P = \mathsf{c}, \mathrm{FN}(P)$.*

- *If $P \overset{\pi}{\equiv} Q$, then $\ulcorner P \urcorner \equiv \ulcorner Q \urcorner$ $\underline{\text{modulo } !A \equiv A \otimes !A}$.*
- *If $\ulcorner P \urcorner \equiv \ulcorner Q \urcorner$, then $P \overset{\pi}{\equiv} Q$.*
- *If $P \rightarrowtriangle^* Q$, then there are Σ, Γ and Δ such that $\Sigma_P; \circ; \ulcorner P \urcorner \Rrightarrow^* (\Sigma_P, \Sigma); \Gamma; \Delta$ where $\exists \Sigma.\, !\Gamma \otimes \Delta \equiv \ulcorner Q \urcorner$ $\underline{\text{modulo } !A \equiv A \otimes !A}$.*
- *If $\Sigma_P; \circ; \ulcorner P \urcorner \Rrightarrow^* (\Sigma_P, \Sigma); \Gamma; \Delta$ and $\exists \Sigma.\, !\Gamma \otimes \Delta \equiv \ulcorner Q \urcorner$, then $P \rightarrowtriangle^* Q$.*

Again, an alternative to introducing this hypothesis consists in taking a more computational view of the π-calculus [18,42,54]) and replacing the structural equivalence law $!P \overset{\pi}{\equiv} P \parallel !P$ with the execution rule

$$\frac{}{!P \rightarrowtriangle !P \parallel P}$$

This, which corresponds exactly to rule $!_1$ in ω, turns the above property into an exact correspondence. Therefore, this amended language, that we shall call $a\pi'$, can be seen as fragment of linear logic in the same sense as MSR_1 was identified with $\tilde{\omega}_1$ in the previous section, but $a\pi$ itself cannot.

8 Conclusions and Future Work

We have endowed a large fragment of linear logic with a rewriting semantics by interpreting the left sequent rules of linear logic as rewrite transitions, folding selected right rules into a structural equivalence, and extending our focus beyond finite derivations. The resulting language, which we called system ω, has been shown to embed popular forms of multiset rewriting and Petri nets, giving a clean logical reading to their semantics. We have also demonstrated ω's strong ties to process algebra, with simple execution-preserving embeddings of a computational variant of asynchronous π-calculus.

As implied in the “Discussion” paragraphs concluding each of the above sections, this work can be extended in numerous directions. In particular, we expect the definition of ω to evolve as questions about is logical foundations are answered (see Section 5.3). Pursuing the relation with process algebraic languages is particularly interesting in light of the results in Section 7 and the application potential of ω in the sphere of security protocol specification.

Acknowledgments

We are grateful to Frank Pfenning, Dale Miller, Mark-Oliver Stehr, Valeria de Paiva, Gerald Allwein, Steve Zdancevic and Vijay Saraswat for their comments on various aspects of this work.

References

[1] *Workshop on logic programming and concurrency*, `http://www.lix.polytechnique.fr/%7elutz/orgs/lpc_geocal06.html` (2006).

[2] Abramsky, S., *Proofs as processes*, Theoretical Computer Science **135** (1994), pp. 5–9.

[3] Abramsky, S., *Process realizability*, in: F. Bauer and R. Steinbruggen, editors, *Proc. 1999 Marktoberdorf Summer School* (2000), pp. 167–180.

[4] Andreoli, J.-M., *Focussing proof-net construction as a middleware paradigm*, in: A. Voronkov, editor, *Proc.CADE–18* (2002), pp. 501–516.

[5] Andreoli, J.-M. and R. Pareschi, *Linear objects: Logical processes with built-in inheritance*, New Generation Computing **9** (1991), pp. 445–473.

[6] Asperti, A., *A logic for concurrency*, Technical report, Computer Science Department, University of Pisa (1987).

[7] Banâtre, J.-P. and D. Le Métayer, *Programming by multiset transformation*, Communications of the ACM **36** (1993), pp. 98–111.

[8] Barber, A., *Dual intuitionistic linear logic*, Technical Report ECS-LFCS-96-347, Laboratory for Foundations of Computer Sciences, University of Edinburgh (1996).

[9] Bellin, G. and P. J. Scott, *On the π-calculus and linear logic*, Theoretical Computer Science **135** (1994), pp. 11–65.

[10] Benton, N., G. Bierman, V. de Paiva and M. Hyland, *Linear lambda-calculus and categorical models revisited*, in: E. B editor, *Proc. of the 6th Workshop on Computer Science Logic (CSL'92)*, 1993, pp. 61–84.

[11] Bistarelli, S., I. Cervesato, G. Lenzini and F. Martinelli, *Relating Multiset Rewriting and Process Algebras for Security Protocol Analysis*, Journal of Computer Security **13** (2005), pp. 3–47.

[12] Braüner, T. and V. de Paiva, *Cut-elimination for full intuitionistic linear logic*, Technical Report RS-96-10, BRICS, Denmark (1996).

[13] Brown, C. and D. Gurr, *A categorical linear framework for Petri nets*, in: *Proc. LICS'90* (1990), pp. 208–218.

[14] Cervesato, I., *Petri nets and linear logic: a case study for logic programming*, in: M. Alpuente and M. I. Sessa, editors, *Proc. GULP–PRODE'95*, Marina di Vietri, Italy, 1995, pp. 313–318.

[15] Cervesato, I., *Typed MSR: Syntax and examples*, in: *1st International Workshop on Mathematical Methods, Models and Architectures for Computer Networks Security — MMM'01* (2001), pp. 159–177.

[16] Cervesato, I., N. Durgin, M. Kanovich and A. Scedrov, *Interpreting strands in linear logic*, in: H. Veith, N. Heintze and E. Clark, editors, *2000 Workshop on Formal Methods and Computer Security*, Chicago, IL, 2000.

[17] Cervesato, I., N. Durgin, P. Lincoln, J. Mitchell and A. Scedrov, *A meta-notation for protocol analysis*, in: *Proc. CSFW'99* (1999), pp. 55–69.

[18] Cervesato, I., F. Pfenning, D. Walker and K. Watkins, *A concurrent logical framework II: Examples and applications*, Technical Report CMU-CS-02-102, Computer Science Department, Carnegie Mellon University (2002).

[19] Clavel, M., F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer and J. F. Quesada, *Maude: Specification and programming in rewriting logic*, Theoretical Computer Science (2001).

[20] Durgin, N., P. Lincoln, J. Mitchell and A. Scedrov, *Multiset rewriting and the complexity of bounded security protocols*, Journal of Computer Security **12** (2004), pp. 247–311, preliminary report under the title "Undecidability of bounded security protocols" in "Workshop on Formal Methods and Security Protocols (FMSP'99", The 1999 Federated Logic Conference (FLoC'99), Trento, Italy, July, 1999.

[21] Engberg, U. and G. Winskel, *Petri nets as models of linear logic*, in: A. Arnold, editor, *15th Colloquium on Trees in Algebra and Programming* (1990), pp. 147–161.

[22] Fages, F., P. Ruet and S. Soliman, *Phase semantics and verification of concurrent constraint programs*, in: *Proceedings of the 13th IEEE Colloquium on Logic in Computer Science — LICS'98*.

[23] Farwer, B., *A linear logic view of object Petri nets*, Fundamenta Informaticae **37** (1999), pp. 225–246.

[24] Farwer, B. and K. Misra, *Dynamic modification of system structures using LLPNs*, in: *Proc. 5th Conference on Perspectives of System Informatics*, Novosibirsk, Russia, 2003, pp. 177–190.

[25] Fournet, C. and G. Gonthier, *The reflexive CHAM and the join-calculus*, in: *Proc. POPL'96* (1996), pp. 372–385.

[26] Girard, J.-Y., *Linear logic*, Theoretical Computer Science **50** (1987), pp. 1–102.

[27] Gunter, C. and V. Gehlot, *Nets as tensor theories*, in: *10th International Conference on Application and Theory of Petri Nets*, Bonn, Germany, 1989, pp. 174–191.

[28] Gunter, C. and V. Gehlot, *A proof-theoretic semantics for true concurrency*, Preliminary report, University of Pennsylvania (1989).

[29] Hoare, C. A. R., *Communicating sequential processes*, Communications of the ACM **21** (1978), pp. 666–677.

[30] Hodas, J. and D. Miller, *Logic programming in a fragment of intuitionistic linear logic*, Information and Computation **110** (1994), pp. 327–365.

[31] Jensen, K., *Coloured Petri nets*, in: W. Brauer, W. Reisig and G. Rozenberg, editors, *Advances in Petri Nets* (1986), pp. 248–299.

[32] Kanovich, M. I., *Linear logic as a logic of computation*, Annals of Pure and Applied Logic **67** (1994), pp. 183–212.

[33] Kanovich, M. I., *Linear logic as a logic of computation*, Annals of Pure and Applied Logic **67** (1994), pp. 183–212.

[34] Kobayashi, N. and A. Yonezawa, *ACL — A concurrent linear logic programming paradigm*, in: D. Miller, editor, *Proc. ILPS'03* (1993), pp. 279–294.

[35] Le Métayer, D., *Higher-order multiset programming*, in: *Proc. DIMACS workshop on specifications of parallel algorithms* (1994).

[36] Lincoln, P. and V. Saraswat, *Higher-order, linear, concurrent constraint programming* (1993), manuscript.

[37] Martí-Oliet, N. and J. Meseguer, *From Petri nets to linear logic*, in: D. Pitt, D. Rydeheard, P. Dybier, A. Pitt and A. Poigné, editors, *Category Theory and Computer Science* (1989), pp. 313–340.

[38] Martí-Oliet, N. and J. Meseguer, *From Petri nets to linear logic*, Mathematical Structures in Computer Science **1** (1991), pp. 66–101, revised version of paper in LNCS 389.

[39] McDowell, R., D. Miller and C. Palamidessi, *Encoding transition systems in sequent calculus*, Theoretical Computer Science **294** (2003), pp. 411–437.

[40] Meadows, C., *The NRL protocol analyzer: an overview*, in: *2nd International Conference on the Practical Applications of Prolog*, 1994.

[41] Meseguer, J., *Conditional rewriting logic as a unified model of concurrency*, Theoretical Computer Science **96** (1992), pp. 73–155.

[42] Miller, D., *The π-calculus as a theory in linear logic: Preliminary results*, in: E. Lamma and P. Mello, editors, *Proc. ELP* (1992), pp. 242–265.

[43] Miller, D., *A multiple-conclusion specification logic*, Theoretical Computer Science **165** (1996), pp. 201–232.

[44] Miller, D., *Encryption as an abstract data-type: An extended abstract*, in: I. Cervesato, editor, *Proc. FCS*, Ottawa, Canada, 2003, pp. 3–14.

[45] Miller, D., *A proof theoretic approach to operational semantics*, in: *Proc. of the workshop on Algebraic Process Calculi: The First Twenty Five Years and Beyond.*, Bertinoro, Italy, 2005.

[46] Miller, D., G. Nadathur, F. Pfenning and A. Scedrov, *Uniform proofs as a foundation for logic programming*, Annals of Pure and Applied Logic **51** (1991), pp. 125–157.

[47] Miller, D. and A. Tiu, *A proof theory for generic judgments: An extended abstract*, in: *Proceedings of LICS 2003*, IEEE, 2003, pp. 118–127.

[48] Miller, D. and A. Tiu, *A proof theory for generic judgments*, ACM Transactions on Computational Logic **6** (2005), pp. 749–783.

[49] Milner, R., "Communicating and Mobile Systems: the π-Calculus," Cambridge University Press, 1999.

[50] Paulson, L. C., *Proving properties of security protocols by induction*, in: *Proc. CSFW'97* (1997).

[51] Petri, C. A., "Kommunikation mit Automaten." Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962.

[52] Petri, C. A., *Fundamentals of a theory of asynchronous information flow.*, in: *Proc. IFIP* (1963), pp. 386–390.

[53] Pfenning, F., *Structural cut elimination*, in: D. Kozen, editor, *Proc. LICS'95* (1995), pp. 156–166.

[54] Pierce, B. C. and D. N. Turner, *Pict: A programming language based on the pi-calculus*, in: G. Plotkin, C. Stirling and M. Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*, MIT Press, 2000 pp. 455–494.

[55] Sangiorgi, D. and D. Walker, "The π-Calculus: a Theory of Mobile Processes," Cambridge University Press, 2001.

[56] Seely, R. A. G., *Linear logic, $\star$-autonomous categories and cofree coalgebras*, in: J. W. Gray and A. Scedrov, editors, *Categories in Computer Science and Logic* (1989), pp. 371–382, proceedings of the AMS-IMS-SIAM Joint Summer Research Conference, June 14–20, 1987, Boulder, Colorado; Contemporary Mathematics Volume 92.

[57] Tiu, A. and D. Miller, *A proof search specification of the π-calculus*, in: *3rd Workshop on the Foundations of Global Ubiquitous Computing*, ENTCS **138**, 2004, pp. 79–101.

[58] Valk, R., *Petri nets as token objects: An introduction to elementary object nets*, in: J. Desel and M. Silva, editors, *19th International Conference on Application and Theory of Petri Nets* (1998), pp. 1–25.

[59] Watkins, K., I. Cervesato, F. Pfenning and D. Walker, *A concurrent logical framework I: Judgments and properties*, Technical Report CMU-CS-02-101, Computer Science Department., Carnegie Mellon University (2002).

[60] Zucker, J., *Formalisation of classical mathematics in AUTOMATH*, , **249**, 1975, pp. 135–145.