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


Iliano Cervesato*
Carnegie Mellon University — Qatar
Doha, Qatar
iliano@cmu.edu

Andre Scedrov†
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 by Petri nets and multiset rewriting, and the prolific process-based models such as the $\pi$-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 and observe that, possibly modulo duality, they invariably target a small semantic fragment of linear logic that we call $\mathrm{LV}^{\mathrm{obs}}$.

In the second part of the paper, we propose a 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 and specifically $\mathrm{LV}^{\mathrm{obs}}$. The resulting interpretation is extended with a majority of linear connectives into the language of *$\omega$-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 $\pi$-calculus to rewrite operators. The language of $\omega$-multisets forms the basis for the security protocol specification language MSR 3. With relations to both multiset rewriting and process algebra, it supports specifications that are process-based, state-based, or of a mixed nature, with the potential of combining verification techniques from both worlds. Additionally, its logical underpinning makes it an ideal common ground for systematically comparing protocol specification languages.



**Keywords:** Linear logic, concurrency, multiset rewriting, process algebra, security protocols.


---


*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. He was also supported by the Qatar Foundation under grant number 930107.

†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.

# 1 Introduction

In his seminal paper [38], 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 [22, 49, 56, 67, 73], 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 [8] and Gunter and Gehlot [39, 40] 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 [45, 46, 47] followed a similar path to study the complexity of sublanguages of linear logic. Instead, Martí-Oliet and Meseguer [52, 53] and Brown and Gurr [17] approached the issue from a categorical perspective, motivating the use of additional linear connectives as net operators. Engberg and Winskel [28] reached a similar conclusion using quantales, an early model of linear logic. A few years later, Cervesato [20] compiled a comparison of a number of encodings of linear logic. In the state-transition paradigm, concurrent computation takes place on a global state shared by all agents. Each agent can act on portions of this state by applying local 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 [25, 56] (which actually mechanizes a broader form of rewriting), Colored Petri Nets [44], and the programming language GAMMA [49]. The interpretation of the state transition model of concurrency in linear logic relies on two observations: 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 [21] to support a first-order notion of multiset rewriting with existentials which we have extensively used to model cryptographic protocols [19, 22, 27], 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 [42], CCS and the $\pi$-calculus [63, 74], the join calculus [35], 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 [3] and Bellin and Scott [11] rely on classical linear logic for this purpose. Miller et al. have performed a similar investigation using intuitionistic linear logic [54], and more recently using a refinement of linear logic with a new quantifier that resembles name generation [60, 62, 77]. Abramsky has recently suggested extracting processes from proofs [4]. 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 [7], has been applied to the $\pi$-calculus by several authors [24, 54, 57] and to the study of security protocols [21]. A few researchers have compared the process-as-term and process-as-formulas approaches [54] or used them together [24]. 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 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 is 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. As a by-product of this review, we observe that, once normalized with respect to duality, these interpretations of concurrency target a well-defined semantic fragment of linear logic, which we call $\mathrm{LV}^{\mathrm{obs}}$. This language makes a prominent use of tensorial formulas and relies on a nominal interpretation of the existential quantifier akin to Miller and Tiu's $\nabla$ [62], while constraining the use of other constructs of linear logic in a uniform way.

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 [54] 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, objects 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 [3, 11] 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 a majority of the 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, to the extent of our knowledge, is 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 [38] in Pfenning's LV sequent presentation [69], which we reinterpret in a non-standard 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 and a few of LV's right rules are consolidated into a single rule that becomes a means of observing the rewriting process. The remaining right rules are discarded. It is shown that LV's cut rules are admissible.

The resulting system, which we call $\mathrm{LV}^{\mathrm{obs}}$, is much weaker than LV (because of the absence of right rules), but is the foundation of a powerful form of rewriting which we call $\omega$. We show that a tiny syntactic fragment of $\omega$ corresponds exactly to traditional multiset rewriting (or place/transition Petri nets). This constitutes an interpretation of multiset rewriting *as* (a fragment of) logic [3, 11], which we like to contrast to most previous interpretations *into* (a fragment of) logic [8, 17, 20, 28, 39, 47, 52]. The system $\omega$ similarly provides a new logical foundation to more sophisticated forms of multiset rewriting and Petri nets.

Considered in its entirety, $\omega$ can be seen 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 derived from the rules of logic. Under this interpretation, we call formulas $\omega$-multisets.

The system $\omega$ has also close ties to process algebra, in particular to the join calculus [35] and the asynchronous $\pi$-calculus [63, 74]. A simple execution-preserving translation maps process constructors of the latter to rewrite operators.

With relations to the two major paradigms for distributed and concurrent computing, $\omega$ 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. We test this proposition in the arena of cryptographic protocol analysis, in which both approaches are prominently used, and only ad-hoc mappings exist to bridge them. We outlined the development of $\omega$ into the protocol specification language MSR 3 and scrutinize various ways of expressing a protocol. Another field where the dichotomy between state-based and

$$
\begin{tikzcd}
\mathrm{LV} \arrow[rr, dash, "\supset"] \arrow[dr, dash, "\supset"] & & \mathrm{LV}_{\mathbf{1}\otimes} \arrow[r, dash, "\equiv"] \arrow[d, dash, "\cap"] & \mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes} \arrow[d, dash, "\cap"] & & \\
& \mathrm{LV}^{\mathrm{n}} \arrow[r, dash, "\supset"] & \mathrm{LV}_{\mathbf{1}\otimes\exists} \arrow[r, dash, "\equiv"] & \mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes\exists} \arrow[r, dash, "\equiv"] & \mathrm{LV}^{\mathrm{obs}} \arrow[r, dotted] & \omega
\end{tikzcd}
$$

Figure 1: Linear Languages Discussed in Section 2 and the Road to $\omega$

process-based specifications has been identified as a hindrance is model checking; we postulate that a language derived from $\omega$ could beneficially bridge this gap, although we do not explore this prospect here.

The review portion of this paper starts with a quick refresher of key elements of linear logic in Section 2, which also lays the logical foundations for the development in the rest of the paper. We then describe in some detail the traditional correspondence between multiset rewriting and linear logic in Section 3 and proceed 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 $\omega$ out of LV. Section 6 exposes $\omega$ as a form of multiset rewriting. Section 7 relates it to the process algebraic world. Section 8 brings the two together in the applied domain of security protocols.[1] Additional remarks and ideas for future developments are given in Section 9.

## 2 Linear Logic

Linear logic was defined in [38] 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 a number of fields. Girard's original paper [38] already foresees the benefits of the expressiveness of linear logic as a tool for describing concurrent systems.

We give a general review of linear logic, mainly of its intuitionistic fragment, in Section 2.1. Section 2.2 explores the relationship between the linear context of a sequent and the class of tensorial formulas. Section 2.3 extends this correspondence to signatures and formulas introduced by a restricted form of existential quantification. By then, we will have identified a useful semantic fragment of linear logic for the purpose of expressing concurrent systems, which we further massage in Section 2.4. We prove a form of cut-elimination for it in Section 2.5. Additional comments can be found in Section 2.6. The discussion in Sections 2.1–2.3 underlies the traditional interpretations of concurrent systems, reviewed in Sections 3 and 4. The remainder of the present discussion will mostly be relevant to the developments in Sections 5–8.

We will examine a number of languages based on linear logic in this section. While all will share the same syntax and sequent structure, they will differ in the rules describing their semantics and in the sequent instances that are derivable in them. This is summarized in Figure 1, to which we will often refer as a roadmap: an edge of the form $\mathcal{L}_1 \overset{\supset}{\text{———}} \mathcal{L}_2$ indicates that the set of sequents derivable in $\mathcal{L}_1$ is a (strict) superset of the sequents derivable in $\mathcal{L}_2$, *i.e.*, that $\mathcal{L}_2$ is a derivationally weaker $\mathcal{L}_1$; edges of the form $\mathcal{L}_1 \overset{\equiv}{\text{———}} \mathcal{L}_2$ mean that $\mathcal{L}_1$ and $\mathcal{L}_2$ are equally expressive in the sense that they derive the exact same sequent instances. The most expressive language we will consider is LV, a mainstream presentation of intuitionistic linear logic [69]. All others will be strictly less expressive as they will omit or significantly restrict some of the standard rules of linear logic. Because of these restrictions, it can be debated whether they can be considered logics at all. We will not take a position on this issue. Similarly to the language of Horn clauses, which underlies early logic programming and is still prominent, these formalisms have a strong connection to logic, which we will investigate in this section. Most of them have

[1] For the chronicle, this research developed almost opposite to this narration: while relating multiset rewriting and process algebraic languages for security protocol specification, we considered an extension to the former with embedded rewrite rules. This led to noticing the relation to the treatment of contexts in the sequent calculus presentation of linear logic. Formalizing this aspect yielded the structural properties, and the observation that they correspond almost exactly to the structural equivalences of the $\pi$-calculus.

been the natural target of linear encodings of specific classes of concurrent languages, as we will see, and we will develop one of them, $\mathrm{LV}^{\mathrm{obs}}$, into a powerful computational paradigm in Section 5 as the rewrite system $\omega$.

## 2.1 A Very Brief Review of Linear Logic

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 requires 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 [38]:

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

Here, $a$ and $x$ range over atomic formulas and term-level variables, respectively. We do not distinguish formulas that differ only by the name of their bound variables, and rely on implicit $\alpha$-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 (extended with sorts in Section 8). 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, $\invamp$ 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 generally 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 [69]. It relies on sequents of the form

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

Similarly to Barber's DILL [10] and Hodas and Miller's $\mathcal{L}$ [43], LV isolates reusable assumptions in the *unrestricted context* $\Gamma$ (subject to exchange, weakening and contraction), while assumptions to be used exactly once are contained in the *linear context* $\Delta$ (subject only to exchange). The combination corresponds to the single context $(!\Gamma, \Delta)$ of Girard [38], where $!\Gamma$ is the linear context obtained by prefixing each formula in $\Gamma$ with the ! modality. The *signature* $\Sigma$ lists the term-level symbols in use. We call $C$ the *goal formula*. We will deemphasize its traditional importance in the second part of this paper.

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:

$$\begin{array}{lccl} \textit{Linear contexts} & \Delta & ::= & \cdot \ \mid \ \Delta, A \\ \textit{Unrestricted contexts} & \Gamma & ::= & \circ \ \mid \ \Gamma ,\, A \\ \textit{Signatures} & \Sigma & ::= & \text{-} \ \mid \ \Sigma ,,\, x \end{array}$$

For each of these collections, the comma ("," or "," or ",,") stands for the extension operator while the bullet ("·" or "◦" or "-") represents the empty collection. The former will be overloaded into a union operator. From an algebraic perspective, unrestricted contexts behave like sets, while signatures and linear contexts are commutative monoids. Additionally, signatures shall not contain duplicate symbols (we will extend them only with eigenvariables and rely on implicit $\alpha$-renaming to ensure this constraint). A signature $\Sigma$ is *legal* for a sequent $\Gamma; \Delta \ \longrightarrow_\Sigma \ C$ if

**S: Structural rules**

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

**C: Cut rules**

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

**L: Left rules**

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

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

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

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

**R: Selected right rules**

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

**X: Other right rules**

$$\frac{}{\Gamma ; \Delta \longrightarrow_{\Sigma} \top}\ \top_{\mathbf{r}} \qquad \frac{\Gamma ; \Delta \longrightarrow_{\Sigma} C_1 \quad \Gamma ; \Delta \longrightarrow_{\Sigma} C_2}{\Gamma ; \Delta \longrightarrow_{\Sigma} C_1 \,\&\, C_2}\ \&_{\mathbf{r}} \qquad \frac{\Gamma ; \Delta \longrightarrow_{\Sigma, x} C}{\Gamma ; \Delta \longrightarrow_{\Sigma} \forall x.\, C}\ \forall_{\mathbf{r}}$$

$$\frac{\Gamma ; \Delta, A \longrightarrow_{\Sigma} B}{\Gamma ; \Delta \longrightarrow_{\Sigma} A \multimap B}\ \multimap_{\mathbf{r}} \qquad \frac{\Gamma ; \cdot \longrightarrow_{\Sigma} C}{\Gamma ; \cdot \longrightarrow_{\Sigma} !C}\ !_{\mathbf{r}}$$

Figure 2: LV Sequent Presentation of Intuitionistic Linear Logic

$\mathrm{FV}(\Gamma, \Delta, C) \subseteq \Sigma$ (slightly abusing notation). All sequents in this paper will be assumed legal, and we will use this term explicitly only for emphasis.

Given these conventions, Figure 2 displays the sequent rules for intuitionistic linear logic in its LV presentation [69]. We divide them into five segments and refer to a rule defined in the segment labeled “s” as an “s”-rule. The first segment (labeled *S*) contains the axiom rule (**id**) and rule **clone** that allows repeatedly using an unrestricted assumption in a derivation. The second segment (*C*) lists the two applicable cut rules of LV.

The left sequent rules for the fragment considered above are listed next (*L*). Observe how !’ed (pronounced *banged*) linear assumption are made available in the unrestricted context in rule $!_{\mathbf{l}}$. In rule $\forall_{\mathbf{l}}$, we rely on the auxiliary judgment $\Sigma \vdash t$ to ascertain that the term $t$ is valid with respect to signature $\Sigma$ (but do not define this notion further since we are leaving the term language unspecified). 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 $\multimap_{\mathbf{l}}$ 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 part of Figure 2 labeled *R* 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 the remaining right rules (listed in part *X*) can be useful query tools, as demonstrated for example in [28, 39] 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 $\Gamma$ and legal $\Sigma$ containing at least one term-level object, there are derivations for both $\Gamma; A_1 \longrightarrow_\Sigma A_2$ and $\Gamma; A_2 \longrightarrow_\Sigma A_1$. The non-emptiness requirement for $\Sigma$ avoids a singularity. It is easily shown that $\equiv$ is indeed an equivalence relation. It is also relatively straightforward to show that it is actually a congruence by application of the $\mathbf{cut}$ rule (although we will not need to rely on this property). Finally, replacing a formula in the goal or linear context of a derivable sequent with a logically equivalent formula retains derivability. In symbols,

- if $C_1 \equiv C_2$ and $\Gamma; \Delta \longrightarrow_\Sigma C_1$ is derivable, so is $\Gamma; \Delta \longrightarrow_\Sigma C_2$,
- if $A_1 \equiv A_2$ and $\Gamma; \Delta, A_1 \longrightarrow_\Sigma C$ is derivable, so is $\Gamma; \Delta, A_2 \longrightarrow_\Sigma C$.

Both are easily obtained using the $\mathbf{cut}$ rule.

## 2.2 Observations in the Tensorial Fragment

In this section and in the next, we will focus our attention on a derivational system based on the syntax of intuitionistic linear logic and defined by restricting the applicability of the LV rules in Figure 2. As it turns out, nearly all encodings of concurrent languages in linear logic are based on this restricted system (or its dual), although this has rarely been made explicit in the literature. In this section, we begin by recalling the well-known relationship between linear contexts and tensorial formulas, which underlies the interpretation of all propositional concurrent languages (possibly modulo duality). We then leverage it to obtain a first restricted variant of LV.

In Section 2.1, we defined the linear context $\Delta$ of an LV sequent as a commutative monoid with operation "," and unit "$\cdot$". As already observed in [38], the notion of derivability also endows an Abelian monoidal structure on the set of linear logic formulas with respect to the tensor ($\otimes$) as the operation and $\mathbf{1}$ as its unit. We call the members of this set *tensorial formulas*. This is captured in the following straightforward lemma:

**Lemma 2.1** *For any formulas $A$, $B$ and $C$, the following logical equivalences hold in LV:*

- *Associativity* : $A \otimes (B \otimes C) \equiv (A \otimes B) \otimes C$
- *Identity* : $A \otimes \mathbf{1} \equiv A$
- *Commutativity* : $A \otimes B \equiv B \otimes A$

**Proof.** The derivations for each direction of the definition of logical equivalence in Section 2.1 are obtained by simple applications of rules $\otimes_\mathbf{l}$, $\mathbf{1}_\mathbf{l}$, $\otimes_\mathbf{r}$, $\mathbf{1}_\mathbf{r}$ and $\mathbf{id}$, with all the left rules applied before any right rule. $\square$

We write $\equiv_\otimes$ for the equivalence relation based on these three properties. Clearly $\equiv_\otimes \subseteq \equiv$. Note that it is not a congruence as there is no provision for it to apply within subformulas of any other operator but $\otimes$.

The fact that linear contexts and tensorial formulas share the same algebraic structure will allow us to blur the distinction between these two notions. At the top level, this idea is familiar from categorical interpretations of logic, where a linear context $\Delta$ 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 [12, 75]. Formally, given a linear context $\Delta$, we define $\bigotimes \Delta$ as

$$\begin{cases} \bigotimes(\cdot) & = \mathbf{1} \\ \bigotimes(A, \Delta) & = A \otimes \bigotimes \Delta \end{cases}$$

By lemma 2.1, this notion is well defined since the tensor $\otimes$ is a monoidal operator with unit $\mathbf{1}$, which matches the fact that linear contexts are understood as monoids with operator "," and unit "$\cdot$". Both are commutative.

The proof-theoretic underpinning of the categorical identification of linear contexts and tensorial formulas [12, 75] relies on two properties. The first establishes that $\bigotimes \Delta$ is always derivable from $\Delta$, as expressed by the following lemma.

**Lemma 2.2** *For any legal signature $\Sigma$ and any contexts $\Gamma$ and $\Delta$, there is a derivation of the sequent*

$$\Gamma; \Delta \longrightarrow_\Sigma \bigotimes \Delta$$

**Proof.** The desired derivation is obtained inductively on any construction of $\Delta$ by iterated applications of rule $\otimes_{\mathbf{r}}$ capped by rule $\mathbf{1_r}$. Lemma 2.1 ensures that the particular construction does not matter. □

This lemma maps a linear context on the left-hand side of an LV sequent to a tensorial goal in its right-hand side. It effectively bridges the two sides of a sequent. More importantly for our purposes, it shows that it is always possible to collect the contents of the linear context into a goal formula with the same algebraic structure. If we understand the linear context as a "state" (as we will partially do in the rest of this paper), this lemma says that we can always take a snapshot of this state and report it as a goal formula. We will interpret this formula as an observation of that state.

The second property states that replacing a context $\Delta$ with the single formula $\bigotimes \Delta$ does not impact derivability.

**Property 2.3** *For any legal signature $\Sigma$, any contexts $\Gamma$ and $\Delta$, and for any formula $C$,*

$$\Gamma; \Delta \longrightarrow_\Sigma C \quad \textit{iff} \quad \Gamma; \bigotimes \Delta \longrightarrow_\Sigma C$$

**Proof.** A proof of the forward direction of this property extends the given derivation of $\Gamma; \Delta \longrightarrow_\Sigma C$ downward with uses of $\otimes_{\mathbf{l}}$ and possibly of $\mathbf{1_l}$. The reverse direction relies on $\mathbf{cut}$ applied to the sequent $\Gamma; \Delta \longrightarrow_\Sigma \bigotimes\Delta$, which is derivable by Lemma 2.2. □

This result allows us to effectively treat the linear context $\Delta$ of an LV sequent as if it were the tensorial formula $\bigotimes \Delta$. Indeed, applying this transformation on any of the rules in Figure 2 yields an admissible rule relative to LV. Moreover, applying this transformation to all rules in this figure and taking the equivalences in Lemma 2.1 as primitive would produce a formalism that is equivalent to LV in terms of derivability. More on this in Section 5.

All results presented so far hold in LV. In the rest of this section, we will focus on a semantically restricted fragment of this logic which we call $\mathrm{LV}_{\mathbf{1}\otimes}$ (and which we will further develop in Sections 2.3 — see Figure 1). The syntax of $\mathrm{LV}_{\mathbf{1}\otimes}$ is the same as LV's and is displayed in Section 2.1. Its notion of derivability differs from LV's semantics by leaving out all right rules except for $\mathbf{1_r}$ and $\otimes_{\mathbf{r}}$.[2] Therefore, the semantics of $\mathrm{LV}_{\mathbf{1}\otimes}$ is given by the rules in segments *S*, *C*, *L* of Figure 2 as well as rules $\mathbf{1_r}$ and $\otimes_{\mathbf{r}}$. Because most right rules have been omitted, $\mathrm{LV}_{\mathbf{1}\otimes}$ is a strict fragment of LV with respect to derivability: every derivable $\mathrm{LV}_{\mathbf{1}\otimes}$ sequent is derivable in LV, but not vice versa. For example, linear implication is not transitive in $\mathrm{LV}_{\mathbf{1}\otimes}$ since the sequent "$\circ; A \multimap B, B \multimap C \longrightarrow_\Sigma A \multimap C$" is not derivable in this formalism, nor is it any more the left adjunct of $\otimes$ as "$\circ; A \multimap B \multimap C \longrightarrow_\Sigma B \multimap A \multimap C$" is not derivable either in $\mathrm{LV}_{\mathbf{1}\otimes}$. This restriction is useful when modeling concurrent systems, especially process algebras, as we will see in Section 4.

As noted above, Lemma 2.2 (which holds in $\mathrm{LV}_{\mathbf{1}\otimes}$) allows collecting the contents of the linear context into a goal formula. This can be done at any point during the bottom-up process of building a derivation. It is therefore natural to view it as a form of *observation*. The semantics of $\mathrm{LV}_{\mathbf{1}\otimes}$ is such that whenever the sequent $\Gamma; \Delta \longrightarrow_\Sigma C$ is derivable, the goal formula $C$ can be construed as such an observation of some linear context appearing in some derivation.

To demonstrate this fact, we will consider another language, which we call $\mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes}$, which differs from $\mathrm{LV}_{\mathbf{1}\otimes}$ by the fact that it discards rules $\mathbf{id}$, $\otimes_{\mathbf{r}}$ and $\mathbf{1_r}$ in favor of the following rule, distilled from Lemma 2.2,

$$\frac{}{\Gamma; \Delta \longrightarrow_\Sigma \bigotimes \Delta}\ \mathbf{obs}'$$

which will be generalized in Sections 2.3. Therefore, $\mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes}$ does not feature any of the right rules of LV. Notice also that rule $\mathbf{obs}'$ subsumes $\mathbf{id}$ as a special case. Clearly, all $\mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes}$ can do is make observations of the contents of the linear context in some sequent in a derivation and report them as goal formulas.

The following theorem states that $\mathrm{LV}_{\mathbf{1}\otimes}$ and $\mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes}$ are equivalent in the sense that their sequents are equi-derivable, as also indicated in Figure 1.

**Theorem 2.4** *The sequent $\Gamma; \Delta \longrightarrow_\Sigma C$ has a derivation $\mathcal{D}$ in $\mathrm{LV}_{\mathbf{1}\otimes}$ iff it has a derivation $\mathcal{E}$ in $\mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes}$.*

[2] We could actually limit the discussion in this section to the propositional fragment of LV, but allowing the quantifiers has no impact as long as their right rules are left out.

**Proof.** The reverse direction of this proof is easily obtained by replacing every use of rule $\mathbf{obs}'$ in $\mathcal{E}$ with the prooflet guaranteed by Lemma 2.2 (which, again, holds in $\mathrm{LV}_{\mathbf{1}\otimes}$). Therefore, $\mathcal{D}$ is structurally identical to $\mathcal{E}$ except for the fact that all occurrences of $\mathbf{obs}'$ have been expanded in place into subderivations that use only rule $\mathbf{1_r}$, $\otimes_\mathbf{r}$ and $\mathbf{id}$ (in particular, no *C*, *L* or $\mathbf{clone}$ rules).

The forward direction of this proof is slightly more involved as rule $\otimes_\mathbf{r}$ can occur in any position in the derivation $\mathcal{D}$, not just near the leaves, where Lemma 2.2 can factor out occurrences into rule $\mathbf{obs}'$. In particular, *C*- and *R*-rules, as well as $\mathbf{clone}$, can appear above rule $\otimes_\mathbf{r}$. The intuition behind the proof is to permute uses of rule $\otimes_\mathbf{r}$ upward until they are only preceded by occurrences of $\mathbf{1_r}$, $\mathbf{id}$ or other occurrences of $\otimes_\mathbf{r}$. This technique is justified by early permutability results systematically studied by Galmiche and Perrier [37, 66] and independently applied by other authors [41, 43, 61]. It is also fairly straightforward to give a direct proof by showing that if two sequents $\Gamma; \Delta_1 \longrightarrow_\Sigma C_1$ and $\Gamma; \Delta_2 \longrightarrow_\Sigma C_2$ are derivable in $\mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes}$, then the sequent $\Gamma; \Delta_1, \Delta_2 \longrightarrow_\Sigma C_1 \otimes C_2$ is also derivable in $\mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes}$. □

This theorem states that, just like $\mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes}$, the deductive power of $\mathrm{LV}_{\mathbf{1}\otimes}$ is also limited to reporting observations of the contents of some linear context.

## 2.3 Observations in the Tensorial-Existential Fragment

We will now repeat the exercise just performed in Section 2.2 but this time consider not only the interplay between tensors and linear contexts, but also the relationship between signatures and an appropriate notion of existentially quantified formulas. The outcome will be similar: we will be able to reify the operation of extending a signature as a limited form of the existential quantifier, which will be the basis for defining a restricted variant of LV centered around a notion of observation.

We shall begin by significantly restricting the semantics of existential quantification: we will leave the left rule ($\exists_\mathbf{l}$) intact but we will limit the applicability of the right rule $\exists_\mathbf{r}$ in Figure 2 to the cases where the substitution term $t$ chosen for the variable $x$ in $\exists x.\, C$ is $x$ itself. Therefore, the right rule we will consider for the existential quantifier is

$$\frac{\Sigma \vdash x \quad \Gamma; \Delta \longrightarrow_\Sigma C}{\Gamma; \Delta \longrightarrow_\Sigma \exists x.\, C}\,\exists^\mathbf{n}_\mathbf{r} \qquad \text{or more compactly} \qquad \frac{\Gamma; \Delta \longrightarrow_{\Sigma,x} C}{\Gamma; \Delta \longrightarrow_{\Sigma,x} \exists x.\, C}\,\exists^\mathbf{n}_\mathbf{r}$$

We call this restricted form of existential quantification *nominal* and write $\mathrm{LV}^\mathrm{n}$ for the formalism that differs from LV by relying on $\exists^\mathbf{n}_\mathbf{r}$ as the right rule for $\exists$ as opposed to $\exists_\mathbf{r}$. The remaining rules are as in Figure 2. $\mathrm{LV}^\mathrm{n}$ admits cut-elimination, which is shown by a simple adaptation of the proofs in [69] or [62].

Notice that rule $\exists^\mathbf{n}_\mathbf{r}$ is *almost* dual to $\exists_\mathbf{l}$, with the important difference that the variable $x$ is *not* treated as an eigenvariable: $x$ appears in the signature of $\exists^\mathbf{n}_\mathbf{r}$'s conclusion while implicit $\alpha$-renaming strictly forbids this in $\exists_\mathbf{l}$. This entails that whenever an application of $\exists^\mathbf{n}_\mathbf{r}$ to a variable $x$ occurs in a derivation of a sequent $\Gamma; \Delta \longrightarrow_\Sigma C$, then either $x$ appears already in $\Sigma$, or it is introduced by rule $\exists_\mathbf{l}$ prior to this application of $\exists^\mathbf{n}_\mathbf{r}$. Observe also that rule $\exists^\mathbf{n}_\mathbf{r}$ does not perform any kind of substitution, not even a renaming. Therefore, not only is the sequent "$\circ; \exists x.\, x = x \longrightarrow_{+,3,5} 3 + 5 = 3 + 5$" not derivable in $\mathrm{LV}^\mathrm{n}$, but neither is the sequent "$\circ; \exists x.\, x = x \longrightarrow_\Sigma \exists x_1.\, \exists x_2.\, x_1 = x_2$". Indeed, all that $\exists^\mathbf{n}_\mathbf{r}$ does is to look for all the occurrences of a variable in the goal that share a common name (say "$x$") and to bind them together using the existential quantifier: it reifies the sequent-level fact that a symbol appears in the signature into a formula-level operator in the goal of the sequent.

Quantifiers in the vain of the restricted semantics for existentials stemming from rule $\exists^\mathbf{n}_\mathbf{r}$ have been studied by several authors in recent years. One of the first proposals was Gabbay and Pitts's И-quantification aimed at investigating the meta-theory of formalisms featuring binders [36], and was later developed into the programming language FreshML [71, 76]. Cardelli and Gordon devised two complementary constructs, "revelation" and "hiding", to study the logical properties of name operators in the $\pi$-calculus [18]. More recently, Miller and Tiu introduced the $\nabla$ quantifier to capture the behavior of both $\forall$ and $\exists$ in managing names through eigenvariables but away from their handling of substitution [62].

As we will see starting from Section 3, eigenvariables introduced through quantifiers are a natural device to model objects generated during the execution of a first- or higher-order concurrent language, or to study its meta-theory. Authors such as [15, 21, 24, 57], who have remained within the traditional boundaries of logic (as opposed to those who have explored the above constructs, *e.g.* [18, 62, 36]), have relied on either existential or universal quantification (depending on which side of duality they stood) to model this phenomenon. In all cases, they implicitly assigned a nominal behavior to the chosen quantifier, along the lines of what we are doing with rule $\exists_{\mathbf{r}}^{\mathbf{n}}$. Indeed, all encodings we review in this paper will rely on such a behavior, and would forsake completeness if they adopted rule $\exists_{\mathbf{r}}$ in its full generality.

Similarly to the case of the tensor product, we begin our scrutiny of the existential quantifier by listing a few logical equivalences. They hold both in LV and in $\mathrm{LV}^{\mathrm{n}}$. We give them names analogous to similar tensorial relations, although the correspondence is not perfect.

**Lemma 2.5** *For any formulas $A$ and $B$ and term variables $x$ and $y$, the following logical equivalences hold:*

- *Nominal Associativity* : $\exists x.\,(A \otimes B) \;\equiv\; (\exists x.\,A) \otimes B$ *if* $x \notin \mathrm{FV}(B)$
- *Nominal Identity* : $\exists x.\,\mathbf{1} \;\equiv\; \mathbf{1}$
- *Nominal Commutativity*: $\exists x.\,\exists y.\,A \;\equiv\; \exists y.\,\exists x.\,A$

**Proof.** This proof follows the pattern already seen in Lemma 2.1. Each of the two derivations underlying the definition of logical equivalence are obtained by applying rules $\exists_{\mathbf{l}}$, $\otimes_{\mathbf{l}}$ and $\mathbf{1}_{\mathbf{l}}$, followed by rules $\exists_{\mathbf{r}}$ (in the restricted form of rule $\exists_{\mathbf{r}}^{\mathbf{n}}$), $\otimes_{\mathbf{r}}$, $\mathbf{1}_{\mathbf{r}}$ and $\mathbf{id}$. Once more, all the left rules are applied before any right rule in the bottom up construction of each derivation. $\square$

The last two equivalences in Lemma 2.5 have clear relations with standard properties of signatures. Nominal identity ultimately corresponds to a form of weakening on signature symbols: if $\Gamma; \Delta \longrightarrow_{\Sigma, x} C$ is derivable but $x \notin \mathrm{FV}(\Gamma, \Delta, C)$, then $\Gamma; \Delta \longrightarrow_{\Sigma} C$ is also derivable. Nominal commutativity is related to the fact that signatures are commutative monoids.

We indicate the equivalence relation on logical formulas based on the three properties in Lemma 2.5 as $\equiv_{\exists}$. Furthermore, we write $\equiv_{\otimes\exists}$ for the equivalence relation based on them and the properties specified in Lemma 2.1. Both are subrelations of $\equiv$ and neither is a congruence since they operate only at the top level.

In preparation to extending the notion of observation introduced in Section 2.1, we define the *existential closure* of a formula $C$ with respect to a signature $\Sigma$, written $\exists\Sigma.\,C$, as the formula obtained by existentially prefixing $C$ with each element in $\Sigma$. Formally,

$$\begin{cases} \exists(\cdot).\,C & = & C \\ \exists(x,\Sigma).\,C & = & \exists x.\,\exists\Sigma.\,C \end{cases}$$

Nominal commutativity (Lemma 2.5) ensures that existential closures started from different orderings of the same signature $\Sigma$ are logically equivalent. If $C$ is the tensorial formula $\bigotimes\Delta$ obtained from a linear context $\Delta$, we abbreviate $\exists\Sigma.\,\bigotimes\Delta$ as $\exists\Sigma.\,\Delta$. We shall observe that this type of formulas can be taken as a canonical form relative to the equivalence relation $\equiv_{\otimes\exists}$. Indeed, whenever $A \equiv_{\otimes\exists} B$, then $A \equiv_{\otimes\exists} \exists\Sigma.\,\Delta \equiv_{\otimes\exists} B$, for some $\Sigma$ and $\Delta$, as stated by the following lemma.

**Lemma 2.6** *For any formulas $A$ and $B$ such that $A \equiv_{\otimes\exists} B$, there exist a signature $\Sigma$ and a context $\Delta$ such that $A \equiv_{\otimes\exists} \exists\Sigma.\,\Delta$ and $B \equiv_{\otimes\exists} \exists\Sigma.\,\Delta$.*

**Proof.** By iterated applications of nominal associativity from right to left, it is possible to transform $A$ into an $\equiv_{\otimes\exists}$-equivalent formula of the form $\exists\Sigma_A.\,\Delta_A$, and similarly for $B$. Now, it must be the case that $\bigotimes\Delta_A \equiv_{\otimes\exists} \bigotimes\Delta_B$ modulo $\alpha$-conversion. Freezing such an $\alpha$-conversion, take $\Delta$ to be $\Delta_A$ for example and take $\Sigma$ to be any signature that contains all the common elements of $\Sigma_A$ and $\Sigma_B$. $\square$

If we think of the mention of a variable $x$ in the signature $\Sigma$ of a sequent $\Gamma; \Delta \longrightarrow_{\Sigma} C$ as a meta-logical binding occurrence for this variable relative to the whole sequent, then rule $\exists_{\mathbf{r}}^{\mathbf{n}}$ defines the existential quantifiers as the corresponding syntactic binder for $x$ in the goal $C$. We will shortly extend this interpretation to some situations involving the left-hand side. Its generalization to the entire sequent essentially amounts to defining a notion akin to the "telescopes" of the AUTOMATH languages [80], which is also featured in recent work on concurrent constraint programming [31]. The existential quantifier is then the formula-level reification of what can be interpreted as a sequent-level binder. Note that the main difference between rule $\exists_{\mathbf{r}}$ and $\exists_{\mathbf{r}}^{\mathbf{n}}$ is that the latter forces this narrow interpretation of existential quantification, while the former also provides support for arbitrary substitutions.

The presence of the existential quantifier in our language allows extending the statement of Lemma 2.2 to reify more of the sequent structure into a derivable goal formula. Indeed, not only is the formula $\bigotimes\Delta$ always derivable from a linear context $\Delta$, but so is its existential closure with any fragment of a legal signature for this sequent. Lemma 2.2 is upgraded as follows and is provable both in LV and in $\mathrm{LV}^{\mathrm{n}}$:

**Lemma 2.7** *For any contexts* $\Gamma$ *and* $\Delta$ *and legal disjoint signatures* $\Sigma$ *and* $\Sigma'$ *(*i.e.*, such that* $\mathrm{FV}(\Gamma, \Delta) \subseteq (\Sigma,\!, \Sigma')$*), there is a derivation of the sequent*

$$\Gamma; \Delta \ \longrightarrow_{\Sigma,\!,\Sigma'} \ \exists\Sigma'.\, \Delta$$

**Proof.** By Lemma 2.2, there is a derivation of the sequent $\Gamma; \Delta \ \longrightarrow_{\Sigma,\!,\Sigma'} \ \bigotimes \Delta$. This derivation is then extended downward by successive applications of rule $\exists_{\mathbf{r}}$ (actually $\exists^{\mathbf{n}}_{\mathbf{r}}$) on each item in $\Sigma'$. By nominal commutativity in Lemma 2.5, the actual order of $\Sigma'$ is irrelevant. $\Box$

A careful scrutiny of this proof reveals that rule $\exists_{\mathbf{r}}$ is always used in the restricted form given by $\exists^{\mathbf{n}}_{\mathbf{r}}$. Note that Lemma 2.2 is the special case of Lemma 2.7 where $\Sigma' = \cdot$.

If we interpret the “state” of a sequent to consist not only of its linear context, as in Section 2.2, but also of the symbols defined in its signature, this result allows us to construct derivable goals that observe this form of state. Indeed, Lemma 2.7 entails that the sequent $\Gamma; \Delta \ \longrightarrow_{\Sigma} \ \exists\Sigma.\, \Delta$ is always derivable. We will generally be interested in reifying not all the symbols appearing in a derivation, but only those introduced after a certain point in its construction. Hence the more general form given as Lemma 2.7.

Given this intuition, we define the *observation* of a signature $\Sigma$ and a linear context $\Delta$ as the formula $\exists\Sigma.\, \Delta$ or any formula that is equivalent to it via the relations in Lemmas 2.1 and 2.5. Note that, in the spirit of Lemma 2.7, this definition does not require $\Sigma$ to be a legal signature for $\Delta$: it may not list all the free symbols in this context. Clearly, this definition subsumes the notion of observation given in Section 2.2 as the special case where $\Sigma$ is empty. We will discuss further generalizations in Section 2.6.

Similarly to Property 2.3, the linear context and a fragment of the signature can be reified into a single existential-tensorial formula on the left-hand side of an LV sequent. As we do so, we must make sure that such quantification does not apply to any variable free in the unrestricted context or in the goal formula. Property 2.3 is upgraded as follows:

**Property 2.8** *For any contexts* $\Gamma$ *and* $\Delta$*, any formula* $C$ *and for any legal signatures* $\Sigma$ *and* $\Sigma'$ *such that* $\Sigma'$ *is disjoint from* $\mathrm{FV}(\Gamma, C)$*,*

$$\Gamma; \Delta \ \longrightarrow_{\Sigma,\!,\Sigma'} \ C \quad \textit{iff} \quad \Gamma; \exists\Sigma'.\, \Delta \ \longrightarrow_{\Sigma} \ C$$

**Proof.** The forward direction of this proof leverages the construction in the forward direction of Property 2.3, obtaining a derivation of $\Gamma; \bigotimes \Delta \ \longrightarrow_{\Sigma,\!,\Sigma'} \ C$ and then extends it downward by means of rule $\exists_{\mathbf{l}}$. The backward direction relies on $\mathbf{cut}$ and Lemma 2.7. $\Box$

This result extends the interpretation of Property 2.3 by allowing us to treat the linear context $\Delta$ of an LV sequent together with a portion $\Sigma'$ of its signature as a tensorial formula prefixed by a string of existential quantifiers over the variables in $\Sigma'$. The restriction to $\Sigma'$ not to mention any variable free in the goal $C$ is easily circumvented by first abstracting such variable away using rule $\exists^{\mathbf{n}}_{\mathbf{r}}$. Lifting the restriction on the unrestricted context $\Gamma$ requires a generalization of this result that is discussed in Section 2.6. Observe that Property 2.3 is the special instance of this result in which $\Sigma' = \cdot$.

Similarly to what we did for the tensorial language in Section 2.2, we will now carve out a sublanguage of LV (or more precisely $\mathrm{LV}^{\mathrm{n}}$) whose only derivable goal formulas are observations in the extended sense just introduced, modulo the equivalence $\equiv_{\otimes\exists}$ introduced with Lemma 2.5. This fragment, which we call $\mathrm{LV}_{\mathbf{1}\otimes\exists}$, simply extends $\mathrm{LV}_{\mathbf{1}\otimes}$ with rule $\exists^{\mathbf{n}}_{\mathbf{r}}$, so that its right rules are just $\mathbf{1}_{\mathbf{r}}$, $\otimes_{\mathbf{r}}$ and $\exists^{\mathbf{n}}_{\mathbf{r}}$ and its remaining rules are given by segments *S*, *C* and *L* in Figure 2. Note that once more $\mathrm{LV}_{\mathbf{1}\otimes\exists}$ leaves out all the right rules in block *X*. See Figure 1 for how these various languages are related.

To prove that only observations are derivable, we define another language where this is obviously the case. The semantics of this language, which we call $\mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes\exists}$, consists of the left rules of LV (segment *L* in Figure 2), its cut rules (segment *C*), rule $\mathbf{clone}$ and the following rule $\mathbf{obs}$, engineered from the statement of Lemma 2.7:

$$\frac{}{\Gamma; \Delta \ \longrightarrow_{\Sigma,\Sigma'} \ \exists\Sigma'.\, \Delta}\ \mathbf{obs}$$

**S: Structural rules**

$$\frac{}{\Gamma;\Delta \longrightarrow_{\Sigma,\Sigma'} \exists\Sigma'.\,\Delta}\ \mathbf{obs} \qquad \frac{\Gamma, A;\Delta, A \longrightarrow_{\Sigma} C}{\Gamma, A;\Delta \longrightarrow_{\Sigma} C}\ \mathbf{clone}$$

**C: Cut rules**

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

**L: Left rules**

$$\frac{\Gamma;\Delta \longrightarrow_{\Sigma} C}{\Gamma;\Delta,\mathbf{1} \longrightarrow_{\Sigma} C}\ \mathbf{1_l} \qquad \frac{\Gamma;\Delta, A_1, A_2 \longrightarrow_{\Sigma} C}{\Gamma;\Delta, A_1\otimes A_2 \longrightarrow_{\Sigma} C}\ \otimes_{\mathbf{l}} \qquad \frac{\Gamma;\Delta, B \longrightarrow_{\Sigma,\Sigma'} C}{\Gamma;\Delta,\Delta',(\exists\Sigma'.\,\Delta')\multimap B \longrightarrow_{\Sigma,\Sigma'} C}\ \multimap_{\mathbf{l}}'$$

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

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

Figure 3: The $\mathrm{LV}^{\mathrm{obs}}$ Sequent Rules for Intuitionistic Linear Logic

In particular $\mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes\exists}$ does not contain any of the right rules of LV. Note that rule $\mathbf{obs}$ subsumes rule $\mathbf{obs}'$ introduced in Section 2.2 for a similar purpose, and also embeds rule $\mathbf{id}$ as a special case.

Languages $\mathrm{LV}_{\mathbf{1}\otimes\exists}$ and $\mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes\exists}$ have the same derivational power modulo $\equiv_{\otimes\exists}$ (actually just $\equiv_{\exists}$): this equivalence is needed to normalize the derivable goal formulas of the former language, which may interleave $\exists$ and $\otimes$ and contain vacuous existential quantifications, into the orderly goals supported by rule $\mathbf{obs}$. Other than this remark, the following result tracks Theorem 2.4 in Section 2.2.

**Theorem 2.9** *Given a signature $\Sigma$, context $\Gamma$ and $\Delta$ and a formula $C$,*

1. *if the sequent $\Gamma;\Delta \longrightarrow_{\Sigma} C$ has a derivation $\mathcal{D}$ in $\mathrm{LV}_{\mathbf{1}\otimes\exists}$, then there exists a signature $\Sigma'$ and a context $\Delta'$ such that $C \equiv_{\otimes\exists} \exists\Sigma'.\,\Delta'$ and $\Gamma;\Delta \longrightarrow_{\Sigma} \exists\Sigma'.\,\Delta'$ has a derivation $\mathcal{E}$ in $\mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes\exists}$.*

2. *if the sequent $\Gamma;\Delta \longrightarrow_{\Sigma} C$ has a derivation $\mathcal{E}$ in $\mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes\exists}$, then it also has a derivation $\mathcal{D}$ in $\mathrm{LV}_{\mathbf{1}\otimes\exists}$.*

**Proof.** Similarly to Theorem 2.4, the forward direction (1) of this proof relies on rule permutation results such as [37, 66] to push rules $\mathbf{1_r}$,$\otimes_{\mathbf{r}}$ and $\exists^{\mathbf{n}}_{\mathbf{r}}$ upward in $\mathcal{D}$, where they can be factored out into constructions for rule $\mathbf{obs}$ as specified by Lemma 2.7. A direct proof of the admissibility of $\mathbf{obs}$ is more complicated than in Theorem 2.4 because more linear logic operators are involved.

The proof in the reverse direction (2) simply amounts to expanding every use of rule $\mathbf{obs}$ into the proof fragment constructed by Lemma 2.5. □

The review portion of this paper (Sections 3–4) will rely on $\mathrm{LV}_{\mathbf{1}\otimes\exists}$ to recall the traditional translations of various concurrent languages into linear logic (actually $\mathrm{LV}_{\mathbf{1}\otimes}$ for propositional languages). Because it is a strict subset of LV, this will not alter the encodings found in the literature, just focus them by observing that they do not make full use of the constructions of linear logic.

The research part of this paper, in Sections 5–8, will build on the characterization of $\mathrm{LV}_{\mathbf{1}\otimes\exists}$ as the equivalent system $\mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes\exists}$, which we just introduced. We will spend the next two subsections massaging it for this purpose. Readers who are only interested in the review part of this paper may skip to Section 3.

## 2.4 Rewriting Implication

Our first observation will be that, because $\mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes\exists}$ is so much weaker than LV, the left rule for implication, $\multimap_l$, can be advantageously simplified without altering derivability. Its replacement will be the following rule:

$$\frac{\Gamma; \Delta_2, B \ \longrightarrow_{\Sigma,\Sigma'} \ C}{\Gamma; \Delta_1, \Delta_2, (\exists\Sigma'.\, \Delta_1) \multimap B \ \longrightarrow_{\Sigma,\Sigma'} \ C} \multimap_l'$$

which essentially requires that the antecedent $A$ of the implication in rule $\multimap_l$ in Figure 2 be the existential-tensorial formula $\exists\Sigma'.\, \Delta_1$ corresponding to the context fragment $\Delta_1$ and existentially quantified over some subset $\Sigma'$ of the sequent's signature. Notice that this formula matches exactly the goal structure in rule $\mathbf{obs}$. Note that $\multimap_l'$ is a derivable rule in LV: it is emulated by simply using the sequent $\Gamma; \Delta_1 \ \longrightarrow_{\Sigma,\Sigma'} \ \exists\Sigma'.\, \Delta_1$, which has a derivation by Lemma 2.2, as the minor premise of $\multimap_l$. One critical property of $\multimap_l'$, actually the main reason for preferring it to $\multimap_l$, is that it does not have a minor premise. We will make use of this property in Section 5.

We call $\mathrm{LV}^{\mathrm{obs}}$ the language that differs from $\mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes\exists}$ by replacing $\multimap_l$ with $\multimap_l'$. The semantics of $\mathrm{LV}^{\mathrm{obs}}$ is given by all the rules displayed in Figure 3, which embeds all the changes made to LV since Figure 2 (we have renamed some of the entities in rule $\multimap_l'$ for uniformity). See also Figure 1 for how it relates to the other languages introduced in this section. We will gray out the cut rules in Section 2.5.

We will now prove that $\mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes\exists}$ and $\mathrm{LV}^{\mathrm{obs}}$ allow deriving the same sequents. In order to do so, we need the following lemma which essentially states that $\multimap_l$ is an admissible rule in $\mathrm{LV}^{\mathrm{obs}}$.

**Lemma 2.10** *For any legal signature $\Sigma$, contexts $\Gamma$, $\Delta_1$ and $\Delta_2$, and formulas $A$, $B$ and $C$, if $\Gamma; \Delta_1 \ \longrightarrow_\Sigma \ A$ and $\Gamma; \Delta_2, B \ \longrightarrow_\Sigma \ C$ are both derivable in $\mathrm{LV}^{\mathrm{obs}}$, then $\Gamma; \Delta_1, \Delta_2, A \multimap B \ \longrightarrow_\Sigma \ C$ has a derivation in $\mathrm{LV}^{\mathrm{obs}}$.*

**Proof.** The proof proceeds by an easy induction on the given $\mathrm{LV}^{\mathrm{obs}}$ derivation of $\Gamma; \Delta_1 \ \longrightarrow_\Sigma \ A$. □

At this point, the equivalence of $\mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes\exists}$ and $\mathrm{LV}^{\mathrm{obs}}$ is easily assessed in the following corollary.

**Corollary 2.11** *The sequent $\Gamma; \Delta \ \longrightarrow_\Sigma \ C$ has a derivation $\mathcal{D}$ in $\mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes\exists}$ iff it has a derivation $\mathcal{E}$ in $\mathrm{LV}^{\mathrm{obs}}$.*

**Proof.** The forward direction proceeds by induction on $\mathcal{D}$, relying on Lemma 2.10 whenever encountering rule $\multimap_l$. The backward direction proceeds by induction on $\mathcal{E}$ and expands occurrences of $\multimap_l'$ into an application of $\multimap_l$ with $\mathbf{obs}$ as its minor premise. □

The result we just obtained also holds of sublanguages of $\mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes\exists}$. In particular, $\multimap_l$ can be replaced with $\multimap_l'$ without consequences for derivability in $\mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes}$.

## 2.5 Cut-Elimination

Another interesting property of $\mathrm{LV}^{\mathrm{obs}}$ (as well as $\mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes}$) is that the two cut rules it inherited from LV are admissible: any derivation can be transformed into an equivalent *cut-free* derivation that does not make use of them. We will now prove this property.

The first proof-theoretic proof of cut-elimination for linear logic was given in [69], and it is indeed for this purpose that LV was designed. As in traditional logic, it implements a normalization procedure that highlights the computational contents of the logic. The proof of cut-elimination for $\mathrm{LV}^{\mathrm{obs}}$ will follow the lines of [69], but it will not be as involved because $\mathrm{LV}^{\mathrm{obs}}$ is much simpler than LV. In particular, it has no right rules, which means that the normally quadratic number of cases to consider is now linear in the number of rules. This also implies that there are no cross-cuts, which give the computational meaning to the functional notion of reduction. Cut-elimination in $\mathrm{LV}^{\mathrm{obs}}$ is nonetheless important from a computational point of view because it removes the last rules featuring a minor premise, which will open the door to giving it a rewriting interpretation in Section 5.

We begin with the following auxiliary lemma, which describes some of the consequences of adding an item in the signature or contexts of a derivable $\mathrm{LV}^{\mathrm{obs}}$ sequent. The cases for the signature and the unrestricted context are just standard weakening properties.

**Lemma 2.12** *Given any legal signature $\Sigma$, contexts $\Gamma$, and $\Delta$, variable $x$ and formulas $A$ and $C$, if $\Gamma; \Delta \longrightarrow_\Sigma C$ is derivable in $\mathrm{LV}^{\mathrm{obs}}$, then*

1. *(Signature Extension)* $\Gamma; \Delta \longrightarrow_{\Sigma, x} C$ *is derivable in* $\mathrm{LV}^{\mathrm{obs}}$*;*
2. *(Linear Extension)* $\Gamma; \Delta, A \longrightarrow_\Sigma C \otimes A$ *is derivable in* $\mathrm{LV}^{\mathrm{obs}}$*;*
3. *(Unrestricted Extension)* $\Gamma, A; \Delta \longrightarrow_\Sigma C$ *is derivable in* $\mathrm{LV}^{\mathrm{obs}}$.

**Proof.** Each statement is proved by an independent induction on the given derivation for $\Gamma; \Delta \longrightarrow_\Sigma C$. □

At this point, we are ready to prove the admissibility of the $\mathbf{cut}$ rule. Notice in particular that, differently from LV [69], it does not need to be proved simultaneously with the admissibility of $\mathbf{cut}!$. This is another instance of the greater simplicity of $\mathrm{LV}^{\mathrm{obs}}$, resulting from being a much weaker language.

**Lemma 2.13 (Admissibility of $\mathbf{cut}$)** *For any legal signature $\Sigma$, contexts $\Gamma$, $\Delta_1$ and $\Delta_2$, and formulas $A$ and $C$, for every cut-free $\mathrm{LV}^{\mathrm{obs}}$ derivations of $\Gamma; \Delta_1 \longrightarrow_\Sigma A$ and $\Gamma; \Delta_2, A \longrightarrow_\Sigma C$, there is a cut-free $\mathrm{LV}^{\mathrm{obs}}$ derivation of $\Gamma; \Delta_1, \Delta_2 \longrightarrow_\Sigma C$.*

**Proof.** This proof proceeds by induction on the structure of the given derivation for $\Gamma; \Delta_1 \longrightarrow_\Sigma A$. It relies on Lemma 2.12(2) in the case of rule $\mathbf{obs}$, on Lemma 2.12(3) in the case of rule $!_\mathrm{l}$, and on Lemma 2.12(1) in the case of rule $\exists_\mathrm{l}$. □

Intuitively, the proof simply stacks the derivation of $\Gamma; \Delta_2, A \longrightarrow_\Sigma C$ on top of that of $\Gamma; \Delta_1 \longrightarrow_\Sigma A$, with minor bookkeeping to contexts and signature.

Next, we prove that rule $\mathbf{cut}!$ is also admissible. Note that this proof does depend on the admissibility $\mathbf{cut}$ in the previous lemma.

**Lemma 2.14 (Admissibility of $\mathbf{cut}!$)** *For any legal signature $\Sigma$, contexts $\Gamma$ and $\Delta$, and formulas $A$ and $C$, for every cut-free $\mathrm{LV}^{\mathrm{obs}}$ derivations $\Gamma; \cdot \longrightarrow_\Sigma A$ and $\Gamma, A; \Delta \longrightarrow_\Sigma C$, there is a cut-free $\mathrm{LV}^{\mathrm{obs}}$ derivation $\Gamma; \Delta \longrightarrow_\Sigma C$.*

**Proof.** Differently from Lemma 2.13, this proof proceeds by induction on the structure of the given derivation for $\Gamma; \Delta \longrightarrow_\Sigma C$. It uses Lemma 2.12(3) in the case of rule $!_\mathrm{l}$, and Lemma 2.12(1) in the case of rule $\exists_\mathrm{l}$. The subcase of rule $\mathbf{clone}$ where the principal formula is precisely $A$ is handled by an invocation to Lemma 2.13. □

Here, the construction is slightly more complex as the derivation of $\Gamma; \cdot \longrightarrow_\Sigma A$ can be sandwiched between that of $\Gamma, A; \Delta \longrightarrow_\Sigma C$ and an auxiliary reduction.

With both rules being admissible, cut-elimination is a standard corollary of the above lemmas.

**Theorem 2.15 (Cut elimination)** *Every derivable $\mathrm{LV}^{\mathrm{obs}}$ sequent $\Gamma; \Delta \longrightarrow_\Sigma C$ has a cut-free derivation in $\mathrm{LV}^{\mathrm{obs}}$.*

**Proof.** As usual, this proof proceeds by induction on the structure of the given derivation of $\Gamma; \Delta \longrightarrow_\Sigma C$. It relies on Lemmas 2.13 and 2.13 when encountering rules $\mathbf{cut}$ and $\mathbf{cut}!$, respectively. □

In the sequel, we will generally write $\mathrm{LV}^{\mathrm{obs}}$ to refer to the cut-free presentation of the language in Figure 3, although we may occasionally take advantage of the (admissible) cut rules. Notice again that without $\mathbf{cut}$ and $\mathbf{cut}!$, all rules in $\mathrm{LV}^{\mathrm{obs}}$ have exactly one premise (with the obvious exception of $\mathbf{obs}$). Therefore, an $\mathrm{LV}^{\mathrm{obs}}$ derivation has a very simple structure: a tower of left rules (or $\mathbf{clone}$) capped by one instance of rule $\mathbf{obs}$. There is no branching. This property and the way we engineered rule $\mathbf{obs}$ will be the foundation for the rewriting language we will extract from $\mathrm{LV}^{\mathrm{obs}}$ in Section 5.

## 2.6 Discussion

Following the trajectory initiated in Sections 2.2 and 2.3, it is natural to wonder whether it is possible to reify within the language of formulas not just the linear context $\Delta$ and the signature $\Sigma$ of an LV sequent, but also its unrestricted context $\Gamma$. We will now briefly show that this is indeed feasible and that some of the key properties we encountered in those sections are naturally generalized. The resulting observational language is however rather weak and does not permit eliminating rule $\mathbf{cut}!$, which we attribute to the specific presentation of linear logic we started from, LV.

Given an unrestricted context $\Gamma$, we write $!\Gamma$ for the linear context obtained by prefixing every formula in $\Gamma$ with !. Then, given also a linear context $\Delta$ and a signature $\Sigma$, the observation of the triple $(\Sigma, \Gamma, \Delta)$ is defined as the formula $\exists\Sigma.\ \bigotimes !\Gamma \otimes \bigotimes \Delta$, which we abbreviate as $\exists\Sigma.\ (!\Gamma, \Delta)$. Note that the relations in Lemmas 2.1 and 2.5 can be used to rearrange various parts of this formula. This augmented notion of observation reifies even more of the sequent structure. Indeed, it supports the expected extension of Lemma 2.7:

*For any contexts $\Gamma$, $\Gamma'$ and $\Delta$, and legal disjoint signatures $\Sigma$ and $\Sigma'$, there is a derivation of the sequent*

$$\Gamma, \Gamma'; \Delta \longrightarrow_{\Sigma, \Sigma'} \exists\Sigma'.\ (!\Gamma', \Delta)$$

and is proved in essentially the same way. Note that this allows us to take observations the whole "state" since the sequent $\Gamma; \Delta \longrightarrow_{\Sigma} \exists\Sigma.\ (!\Gamma, \Delta)$ is derivable. It also supports partial observations.

This very same formula can also be used to replace the entire left-hand side of a derivable LV sequent, and still maintain derivability. The following strong generalization of Property 2.8 is indeed provable by means of a simple extension of the technique used then.

*For any contexts $\Gamma$ and $\Delta$, any formula $C$ and any legal signature $\Sigma$,*

$$\Gamma; \Delta \longrightarrow_{\Sigma} C \quad \textit{iff} \quad \circ; \exists\Sigma.\ (!\Gamma, \Delta) \longrightarrow_{\cdot} \exists\Sigma.\ C$$

This result reifies the entire left-hand side of a sequent (including the signature) into a logical formula, This technique is reminiscent of the notion of "telescope" in the AUTOMATH languages [80]. It also appears in recent work on concurrent constraint programming [31]. Notice also that it is not subject to the scope limitations of Property 2.8, which it extends.

As done in Sections 2.2 and 2.3, we can define a language, $LV^{obs}_{\mathbf{1}\otimes\exists!}$, whose only provable goals are observations in the sense just defined. It consists of the left rules of LV, its cut rules, $\mathbf{clone}$ and an observation rule of the form

$$\frac{}{\Gamma, \Gamma'; \Delta \longrightarrow_{\Sigma, \Sigma'} \exists\Sigma'.\ (!\Gamma', \Delta)}$$

This language is however weaker than the extension of $LV_{\mathbf{1}\otimes\exists}$ with rule $!_{\mathbf{r}}$, call it $LV_{\mathbf{1}\otimes\exists!}$. For example, the sequent $a; \cdot \longrightarrow_{\Sigma} !!a$ is derivable in $LV_{\mathbf{1}\otimes\exists!}$ but not in $LV^{obs}_{\mathbf{1}\otimes\exists!}$. Furthermore, cut elimination does not hold in the observational language (but it does in $LV_{\mathbf{1}\otimes\exists!}$). More precisely, rule $\mathbf{cut}$ is admissible, like in $LV^{obs}$, but $\mathbf{cut}!$ is not. For example, $a; \cdot \longrightarrow_{\Sigma} !!a$ is derivable using this rule because $a; \cdot \longrightarrow_{\Sigma} !a$ and $a, !a; \cdot \longrightarrow_{\Sigma} !!a$ are derivable in this language. However the former sequent has no cut-free derivation in $LV_{\mathbf{1}\otimes\exists!}$.

The results obtained in this section will act as a foundation for the developments in the rest of the paper. A dual foundation is possible, however, and some authors have explored it, as we will see. Specifically, our uses of multiplicative conjunction ($\otimes$) and unit ($\mathbf{1}$) on the left-hand side of an LV sequent can be transformed into uses of multiplicative disjunction ($\parr$) and its unit, multiplicative falsehood, $\perp$, on the right of a multiple conclusion sequent of the form $\Gamma; \Delta \longrightarrow_{\Sigma} \Theta$ [16, 38]. The right-hand side, $\Theta$, becomes where the bulk of the action takes place, and it gets reified into the formula $\parr\Theta$. In this setting, the quantifiers are dualized as well, with $\exists$ responsible for substitution and a nominal restriction of $\forall$ managing eigenvariables. Occasionally, the unrestricted context is moved to the right as well, and every formula $A$ in it is understood as being prefixed by the $?$ modality, which is dual to ! [38].

# 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. Pratt [72, 73] has recently generalized this idea to account for transitions in progress and canceled transitions, hence obtaining a very detailed, categorically-motivated, model of concurrency.

This paradigm was first described in abstract form by Petri [67, 68] 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" [44] 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 [25, 56], the programming language GAMMA [49], and the security protocol specification language MSR [19, 22]. Most model checkers also embrace this view of concurrency, for example [55] 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. Place-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 into 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 whose formulas directly correspond to the encoding of rules and multisets. This basic interpretation has been extended to more expressive languages based on the state transition model. In particular, we have enriched it in [21] 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* ($\text{MSR}_0$ hereafter) is given by the following grammar:

$$\begin{array}{lrcl}
\textit{Multisets} & \tilde{s}, \tilde{a}, \tilde{b}, \tilde{c} & ::= & \tilde{\cdot} \mid \tilde{s} \,\tilde{,}\, s \\
\textit{Multiset rewrite rules} & r & ::= & \tilde{a} \to \tilde{b} \\
\textit{Rule sets} & R & ::= & \hat{\cdot} \mid R \,\hat{,}\, r
\end{array}$$

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 "$\tilde{,}$" and the empty multiset "$\tilde{\cdot}$". A rule set $R$ is simply a set of rewrite rules: we write $\hat{\cdot}$ and $\hat{,}$ for the empty set and the extension of a set ($R$) with an element ($r$).

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.*, $\tilde{s} = \tilde{c} \,\tilde{,}\, \tilde{a}$ 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.*, $\tilde{s}' = \tilde{c} \,\tilde{,}\, \tilde{b}$). This is expressed by the basic multiset rewriting judgment $\tilde{s} \rhd_R \tilde{s}'$, which is formally defined by the following transition pattern:

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

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 [8, 17, 20, 28, 39, 47, 52]. 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 a homomorphic mapping by interpreting "$\tilde{\cdot}$", "$\tilde{,}$", $\rightarrow$, "$\hat{\cdot}$" and "$\hat{,}$" as "$\mathbf{1}$", "$\otimes$", $\multimap$, $\circ$ and $,$ 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:

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

Note that rule sets are mapped to unrestricted contexts, which share the same algebraic structure as sets.

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 the sequent $\ulcorner R \urcorner ; \ulcorner \tilde{s} \urcorner \longrightarrow_S \ulcorner \tilde{s}' \urcorner$ is derivable in* $\mathrm{LV^n}$.

**Proof.** The proof proceeds by induction on the length of the transition chain. The base case is a trivial application of rule $\mathbf{id}$. The proof of the step case requires showing that for every single-rule application $\tilde{s} \rhd_{R;r} \tilde{s}'$ the sequent $\ulcorner R \,\hat{,}\, r \urcorner ; \ulcorner \tilde{s} \urcorner \longrightarrow_S \ulcorner \tilde{s}' \urcorner$ 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 $\otimes_\mathbf{l}$, $\mathbf{1_l}$, $\otimes_\mathbf{r}$, $\mathbf{1_r}$ and $\mathbf{cut}$ mediate between tensorial formulas and objects in the context. □

It should be noted that the derivation $\ulcorner R \urcorner ; \ulcorner \tilde{s} \urcorner \longrightarrow_S \ulcorner \tilde{s}' \urcorner$ constructed in this proof is actually valid in $\mathrm{LV}_{\mathbf{1}\otimes}$ since it does not use any right rule besides $\mathbf{1_r}$ and $\otimes_\mathbf{r}$. In fact, the interpretation of $\mathrm{MSR}_0$ into linear logic makes a very limited use of the expressive power of $\mathrm{LV^n}$.

The family of mappings $\ulcorner \_ \urcorner$ identifies a syntactic fragment $LL^{MSR_0}$ of intuitionistic linear logic, that is the linear logic formulas that are in the image of $\ulcorner \_ \urcorner$. Clearly, $\ulcorner \_ \urcorner$ 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 pair of states $\tilde{s}$, $\tilde{s}'$ and every rule set $R$, if $\ulcorner R \urcorner ; \ulcorner \tilde{s} \urcorner \longrightarrow_S \ulcorner \tilde{s}' \urcorner$ is derivable, 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 $\ulcorner R \urcorner ; \ulcorner \tilde{s} \urcorner \longrightarrow_S \ulcorner \tilde{s}' \urcorner$ 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 [37, 41, 43, 61, 66]. Some additional details can be found in [21]. □

Again, this proof lies fully in the $\mathrm{LV}_{\mathbf{1}\otimes}$ semantic sublanguage of $\mathrm{LV^n}$. Since $\mathrm{LV}_{\mathbf{1}\otimes}$ is equivalent to $\mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes}$, which is a sublanguage of $\mathrm{LV^{obs}}$, the last two properties imply that reachability in propositional multiset rewriting is mapped to derivability in this fragment of linear logic.

## 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 [9], while Jensen has turned it into the flexible formalism of colored Petri nets [44]. Maude [25, 56] 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 [22]. We take this as the language of *first-order multiset rewriting* ($\mathrm{MSR}_1$ hereafter).

Abstractly, we take the support set $S$ to consist of first-order atomic formulas over some initial signature $\Sigma_0$. 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 $\alpha$-renaming for both sorts of bound variables. We write $\Sigma \vdash t$ to indicate that $t$ is a valid term over signature $\Sigma$, 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 $\mathrm{MSR}_1$ has the form $\Sigma; \tilde{s} \rhd_R \Sigma'; \tilde{s}'$, where both the initial and final states 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 $\vec{x}$ can be instantiated to $\Sigma$-valid terms $\vec{t}$ so that the antecedent matches $\tilde{s}$ (*i.e.,* $\tilde{s} = \tilde{c} ; [\vec{t}/\vec{x}]\tilde{a}$ for some $\tilde{c}$). In this case, applying $r$ results in a state $\Sigma'; \tilde{s}'$ whose signature is obtained by extending $\Sigma$ with $\vec{n}$ (modulo $\alpha$-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:

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

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 $\mathbf{clone}$, 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:

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

This mapping identifies another syntactic fragment $LL^{MSR_1}$ of linear logic, and is again bijective over this fragment according to the nominal semantics of the existential quantifier discussed in Section 2.3. The formal correspondence between $\mathrm{MSR}_1$ and $LL^{MSR_1}$ enjoys the following soundness property [21]:

**Property 3.3** *For every signatures $\Sigma$, $\Sigma'$, 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 in $\mathrm{LV^n}$ (and LV).*

**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 $\exists_\mathbf{r}$ comes in the form of its nominal variant $\exists^\mathbf{n}_\mathbf{r}$: the existential quantifier in the conclusion's goal formula is introduced exclusively as a syntactic binder for all occurrences of a free variable in the premise's goal — no substitution is performed. □

A close inspection of the construction performed in this proof reveals that the sequent $\ulcorner R \urcorner ; \ulcorner \tilde{s} \urcorner \longrightarrow_\Sigma \exists \Sigma'. \ulcorner \tilde{s}' \urcorner$ is derived in the sublanguage $\mathrm{LV}_{\mathbf{1} \otimes \exists}$ of $\mathrm{LV^n}$ (and LV), which was introduced in Section 2.3. An even closer inspection shows that the goal formulas in the conclusion of this property has the structure, $\exists \Sigma'. \Delta'$, prescribed by rule $\mathbf{obs}$, which entails that this construction actually lies result holds within $\mathrm{LV^{obs}}$.

As noted in [57], 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 arbitrary terms $\vec{t}$ yields derivations that may not correspond to any rewrite sequence. For example, given a signature $\Sigma$ containing the constants $+$, $3$ and $5$, a general use of rule $\exists_\mathbf{r}$ allows us to build a derivation for the linear logic sequent $a \multimap b(3+4); a \longrightarrow_\Sigma \exists x.\, b(x)$. However, there is no rewrite sequence for translation of this sequent according to $\ulcorner \_ \urcorner$, *i.e.,* for the the judgment $\Sigma; a \rhd^*_{a \to b(3+5)} (\Sigma, x); b(x)$. For this reason, we must restrict our attention to derivations that make use of rule $\exists^\mathbf{n}_\mathbf{r}$ rather than the more general $\exists_\mathbf{r}$. Therefore the following property is only valid in $\mathrm{LV^n}$, and not LV.

**Property 3.4** *For every signatures $\Sigma$, $\Sigma'$, 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 in $\mathrm{LV^n}$, 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. $\square$

This proof too is in the $\mathrm{LV}_{\mathbf{1}\otimes\exists}$ sublanguage of $\mathrm{LV}^{\mathrm{n}}$, and therefore in $\mathrm{LV}^{\mathrm{obs}}$.

The general analysis just performed clearly applies to first-order multiset rewriting systems which do not make use of $\exists$, *i.e.*, whose rules have the form $\forall\vec{x}.\tilde{a} \to \tilde{b}$. Such systems are at the basis of formalisms such as GAMMA [9], colored Petri nets [44], and in a sense Maude [25, 56]. For this subclass, the logical construction just discussed specializes to multi-conclusion linear Horn clauses, which Kanovich has extensively mined for complexity results [45, 46].

### 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 [59], use the alternative *disjunctive encoding*, which relies on the observation that linear logic endows also multiplicative disjunction $\parr$ and its unit $\perp$ with the algebraic structure of a commutative monoid. Then $\tilde{s}$ is interpreted as $\parr\tilde{s}$ and the rule $\tilde{a} \to \tilde{b}$ as the implication $\parr\tilde{a} \multimap \parr\tilde{b}$. Some authors [59] also dualize the use of the quantifiers $\forall$ and $\exists$, which yields the reverse implication $\parr\tilde{b} \multimap \parr\tilde{a}$ as an encoding of the rule $\tilde{a} \to \tilde{b}$. In these cases, it is $\forall$ which is given a nominal semantics via a restriction of rule $\forall_{\mathbf{l}}$ similar to $\exists^{\mathbf{n}}_{\mathbf{r}}$.

These two sets of connectives are dual to each other and therefore whenever a sequent is provable, the sequent obtained by exchanging $\otimes$ and $\parr$, and $\mathbf{1}$ and $\perp$ is also derivable. Thus, the results obtained by these authors are essentially syntactic variants of the properties reported above. The inference rules for $\parr$ and $\perp$ are given in terms of multiple conclusion sequents, of the form $\Gamma; \Delta \longrightarrow_{\Sigma} \Theta$, where $\Theta$ is a multiset of formulas rather than a single formula. For this reason, they make use of the derivation structure of classical linear logic [38], or at least full intuitionistic linear logic [16].

## 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 [35, 42, 63, 74] 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, languages vary greatly in the primitives they provide, which often translates 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 on a specific language, the asynchronous $\pi$-calculus [74] which we interpret in linear logic in Section 4.2. For presentation purposes, we first consider a propositional variant in Section 4.1. Other process-based languages and translations are summarily discussed in Section 4.3. We will later provide a detailed encoding of one of them, the join calculus [35], in Section 7.3.

### 4.1 Propositional Process Algebra

We begin by studying the translation in linear logic of a minimally expressive variant of the $\pi$-calculus [74], 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:

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

where $x$ and $\overline{x}$ are a *name* and the corresponding *co-name*, respectively. Furthermore, $P \parallel Q$ is the parallel composition of $P$ and $Q$, and $!P$ is process replication. In anticipation of our study of the asynchronous $\pi$-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 $\pi$-calculus. We call the present language the *propositional asynchronous $\pi$-calculus* and refer to it as $a\pi_0$.

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

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

It makes parallel composition ($\parallel$) a monoidal operator with the null process $\mathbf{0}$ its unit. Traditionally, many authors have considered an additional structural equivalence, $!P \;\overset{\pi}{\equiv}\; P \parallel !P$, which interprets process replication as the parallel composition of arbitrarily many copies of a process. Following a number of authors, *e.g.*, [24, 29, 57, 70], we will turn it into a one-sided reduction. A detailed discussion of this issue can be found in Section 4.3.

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

$$\frac{}{\overline{x} \parallel xP \rightarrowtriangle P}\,\textbf{red\_i/o} \qquad\qquad \frac{P \rightarrowtriangle P'}{P \parallel Q \rightarrowtriangle P' \parallel Q}\,\textbf{red}\parallel \qquad\qquad \frac{}{!P \rightarrowtriangle !P \parallel P}\,\textbf{red!}$$

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 third extracts a copy from a replicated process. The structural equivalence $\overset{\pi}{\equiv}$ can implicitly massage processes before and after synchronization.[3] Let $\_ \rightarrowtriangle^* \_$ be the reflexive and transitive closure of $\_ \rightarrowtriangle \_$.

We define an encoding $\ulcorner \_ \urcorner$ of this propositional process algebra into linear logic by homomorphically mapping $\mathbf{0}$, $\parallel$, 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:

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

This encoding identifies a syntactically restricted fragment of propositional linear logic which we call $LL^{a\pi_0}$.

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 ($\rightarrowtriangle^*$). We first examine the former as it is defined independently from computation: intuitively, $\overset{\pi}{\equiv}$ and $\equiv_\otimes$ are homomorphic modulo the encoding $\ulcorner \_ \urcorner$. Indeed, the following soundness result states that structural equivalence maps to tensorial equivalence.

**Property 4.1** *Given processes $P$ and $Q$, if $P \overset{\pi}{\equiv} Q$, then $\ulcorner P \urcorner \equiv_\otimes \ulcorner Q \urcorner$.*

**Proof.** The proof proceeds by structural induction on a construction of $P \overset{\pi}{\equiv} Q$. □

Since $\equiv_\otimes \subseteq \equiv$, this property entails that $\ulcorner P \urcorner$ and $\ulcorner Q \urcorner$ are logically equivalent in $\mathrm{LV}^{\mathrm{n}}$.

The corresponding completeness result holds also, for the formulas constructed as encoding of well-formed processes:

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

**Proof.** Because $\ulcorner \_ \urcorner$ is an isomorphism over $LL^{a\pi_0}$ with respect to the two equivalence relations, the proof proceeds by a simple induction on the construction of $\ulcorner P \urcorner \equiv_\otimes \ulcorner Q \urcorner$. □

Clearly this property does not hold if we were to replace tensorial equivalence $\equiv_\otimes$ with logical equivalence $\equiv$ since the latter relation is much larger, even relative to $LL^{a\pi_0}$: for example $\ulcorner !\overline{x} \urcorner \equiv \ulcorner !!\overline{x} \urcorner$, but $!\overline{x} \overset{\pi}{\not\equiv} !!\overline{x}$ (and also $\ulcorner !\overline{x} \urcorner \not\equiv_\otimes \ulcorner !!\overline{x} \urcorner$).

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

$$\frac{P \overset{\pi}{\equiv} P' \quad P' \rightarrowtriangle Q' \quad Q' \overset{\pi}{\equiv} Q}{P \rightarrowtriangle Q}\,\textbf{red}\overset{\pi}{\equiv}$$

Given Lemma 4.1, process reduction is directly captured by derivability of its linear logic representation. This establishes the soundness of the encoding.

**Property 4.3** *Given processes $P$ and $Q$, let $\Sigma_P$ be the set of all names in $P$. If $P \rightarrowtriangle^* Q$, then $\circ;\ulcorner P\urcorner \longrightarrow_{\Sigma_P} \ulcorner Q\urcorner$ is derivable in $\mathrm{LV}^{\mathrm{n}}$.*

**Proof.** This proof is again a straightforward induction. $\Box$

Note that this property holds in $\mathrm{LV}^{\mathrm{n}}$, but not in the weaker systems discussed in Section 5. The issue is that they do not allow observing formulas stored in the unrestricted context, so that for example $!\overline{x} \rightarrowtriangle^* !\overline{x} \parallel \overline{x}$ holds in $a\pi_0$ but $\circ;\ulcorner !\overline{x}\urcorner \longrightarrow_x \ulcorner !\overline{x} \parallel \overline{x}\urcorner$ has no derivation in $\mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes}$ (although it has one in $\mathrm{LV}^{\mathrm{n}}$). To recover soundness, this property needs to be weakened as follows:

**Property 4.4** *Given processes $P$ and $Q$, let $\Sigma_P$ be the set of all names in $P$. If $P \rightarrowtriangle^* Q$, then there exist contexts $\Gamma$ and $\Delta$ and a process $Q'$ such that $\circ;\ulcorner P\urcorner \longrightarrow_{\Sigma_P} \bigotimes\Delta$ is derivable in $\mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes}$ where $\ulcorner Q'\urcorner = \bigotimes(!\Gamma,\Delta)$ and $Q \stackrel{\pi}{\equiv} Q'$. Moreover, each formula $!A$ in $!\Gamma$ is a subformula of $\ulcorner P\urcorner$.*

**Proof.** This proof is again a straightforward induction on $P \rightarrowtriangle^* Q$, simulating each reduction with the sequence of rule applications as dictated by the encoding. Then $\Gamma$ is simply the unrestricted context appearing in the last sequent prior to making the final observation. $\Box$

Completeness is a much more complicated affair, even in $\mathrm{LV}^{\mathrm{n}}$. The reverse of Property 4.3 does not hold: for example, $\circ;\ulcorner !\overline{x}\urcorner \longrightarrow_x \ulcorner \overline{x}\urcorner$ is derivable in every linear language examined in Section 2, but there is no sequence of reductions that yields $!\overline{x} \rightarrowtriangle^* \overline{x}$ in $a\pi_0$. Here, the logical derivation has discarded the representation of the replicated process $!\overline{x}$ on the right-hand side, which is allowed since it is stored as a formula $(x)$ in the unrestricted context: the process we would expect is $\overline{x} \parallel !\overline{x}$. This suggests complementing the right-hand side of the sequent with some appropriate process, as in Property 4.4. This again does not work: $\circ;\ulcorner !\overline{x}\urcorner \longrightarrow_x\ !!x$ is derivable in $\mathrm{LV}^{\mathrm{n}}$, but the semantics of $a\pi_0$ is unable to extract the doubly replicated process $!!\overline{x}$ out of $!\overline{x}$. However, deriving $!!A$ from $!A$ is not possible in any of our observational languages either. This suggests using for example $\mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes}$ instead of $\mathrm{LV}^{\mathrm{n}}$, which leads to the following weak completeness result:

**Property 4.5** *Let $P$ be a process, $\Sigma_P$ be the set of all names in $P$, and $\Delta$ be a context. If $\circ;\ulcorner P\urcorner \longrightarrow_{\Sigma_P} \bigotimes\Delta$ is derivable in $\mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes}$, then there is an unrestricted context $\Gamma$ such that $\ulcorner Q\urcorner = \bigotimes(!\Gamma,\Delta)$ and $P \rightarrowtriangle^* Q$. Moreover, each formula $!A$ in $!\Gamma$ is a subformula of $\ulcorner P\urcorner$.*

**Proof.** This proof proceeds in the now usual fashion: inferences need to be reordered according to correspond to the permutability laws to parallel process inferences. The context $\Gamma$ is constructed as follows: whenever rule $\mathbf{obs}'$ is used on a sequent of the form $\Gamma';\Delta \longrightarrow_{\Sigma_P} \bigotimes\Delta$ we extend the derivation so that it yields $\Gamma';\Delta \longrightarrow_{\Sigma_P} \bigotimes !\Gamma' \otimes \bigotimes\Delta$, and whenever combining subderivations of this form, we trim common banged formulas using the cut rule and rule $!_1$. $\Box$

This result pigeonholes the interpretation of this particular process algebra in an observational language, here $\mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes}$ since $a\pi_0$ is propositional, rather than in the larger space of (propositional) linear logic derivability. The fact that we need to reconstruct an unrestricted context $\Gamma$ to achieve completeness suggests that a stronger notion of observation, along the lines of the language $\mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes\exists!}$ briefly discussed in Section 3.3, would be an even better target than $\mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes}$. As we observed, a logically well-behaved language with these characteristics has not yet been isolated.

## 4.2 First-Order Process Algebra: the Asynchronous $\pi$-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 the 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 (first-order) asynchronous $\pi$-calculus (hereafter $a\pi_1$). It is formally defined by the following grammar [74]:

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

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 $\alpha$-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 **0**), which makes communication asynchronous. This core calculus can easily be generalized to support polyadic channels, complex terms, and pattern matching [74].

We generalize the notion of structural equivalence, still written $P \stackrel{\pi}{\equiv} Q$, 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:

| | | |
|---|---|---|
| $P \parallel Q \stackrel{\pi}{\equiv} Q \parallel P$ | $\nu x.\nu y.P \stackrel{\pi}{\equiv} \nu y.\nu x.P$ | |
| $P \parallel \mathbf{0} \stackrel{\pi}{\equiv} P$ | $\nu x.\mathbf{0} \stackrel{\pi}{\equiv} \mathbf{0}$ | |
| $P \parallel (Q \parallel R) \stackrel{\pi}{\equiv} (P \parallel Q) \parallel R$ | $\nu x.(P \parallel Q) \stackrel{\pi}{\equiv} (\nu x.P) \parallel Q$ | if $x \notin \mathrm{FN}(Q)$ |

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:

$$\frac{}{\overline{x}\langle y\rangle \parallel x(z)P \rightarrowtriangle [y/z]P}\,\mathbf{red\_i/o} \qquad \frac{P \rightarrowtriangle P'}{P \parallel Q \rightarrowtriangle P' \parallel Q}\,\mathbf{red}\|\qquad \frac{P \rightarrowtriangle P'}{\nu x.P \rightarrowtriangle \nu x.P'}\,\mathbf{red}\nu \qquad \frac{}{!P \rightarrowtriangle !P \parallel P}\,\mathbf{red!}$$

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

The encoding of $a\pi_1$ in linear logic extends the propositional representation given in Section 4.1 with a case for the hiding operator (modeled as a nominal existential quantifier) and revised definitions for input and output. We reserve a binary predicate symbol $\mathsf{c}$ 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:

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

Let $LL^{a\pi_1}$ be the syntactic fragment of linear logic in the image of this encoding. Note that $\equiv_{\otimes\exists}$ and $\stackrel{\pi}{\equiv}$ are homomorphic over $LL^{a\pi_1}$.

The soundness and completeness results reported in Section 4.1 for the propositional variant of this calculus extend naturally to the first-order setting. The structural equivalence of $a\pi_1$ maps to the tensorial-existential equivalence $\equiv_{\otimes\exists}$ of $\mathrm{LV}^{\mathrm{obs}}$ over $LL^{a\pi_1}$:

**Property 4.6** *For any processes $P$ and $Q$, $P \stackrel{\pi}{\equiv} Q$ iff $\ulcorner P\urcorner \equiv_{\otimes\exists} \ulcorner Q\urcorner$.*

**Proof.** This proof is similar to the propositional versions seen in Section 4.1. For both directions, we proceed by induction on the construction of the appropriate equivalence. □

As in the propositional case, since $\equiv_{\otimes\exists} \subseteq \equiv$, the forward direction can be further strengthened: if $P \stackrel{\pi}{\equiv} Q$, then $\ulcorner P\urcorner \equiv \ulcorner Q\urcorner$ in $\mathrm{LV}^{\mathrm{n}}$. However, no similar generalization applies for the reverse direction.

Reduction chains correspond to derivability in $\mathrm{LV}^{\mathrm{obs}}$, but as for $a\pi_0$, we shall be very careful (and rather verbose) about how to state the correctness of the encoding. The soundness and completeness results are summarized in the following property.

**Property 4.7** *Let $P$ be a process and $\Sigma_P = \mathsf{c}, \mathrm{FN}(P)$.*

1. *For any process $Q$, if $P \rightarrow^* Q$, then there exist a signature $\Sigma$, contexts $\Gamma$ and $\Delta$, and a process $Q'$ such that $\circ; \ulcorner P \urcorner \longrightarrow_{\Sigma_P} \exists \Sigma. \Delta$ is derivable in $\mathrm{LV}^{\mathrm{obs}}$ where $Q \overset{\pi}{\equiv} Q'$ and $\ulcorner Q' \urcorner = \exists \Sigma. !\Gamma, \Delta$. Moreover, each formula $!A$ in $!\Gamma$ is a subformula of $\ulcorner P \urcorner$.*

2. *For any signature $\Sigma$ and context $\Delta$, if $\circ; \ulcorner P \urcorner \longrightarrow_{\Sigma_P} \exists \Sigma. \Delta$ has a derivation in $\mathrm{LV}^{\mathrm{obs}}$, then there are an unrestricted context $\Gamma$ and a process $Q$ such that $\ulcorner Q \urcorner = \exists \Sigma. (!\Gamma, \Delta)$ and $P \rightarrow^* Q$. Moreover, each formula $!A$ in $!\Gamma$ is a subformula of $\ulcorner P \urcorner$.*

**Proof.** This proof extends the techniques used in Section 4.1 to handle the propositional infrastructure with the treatment of quantifiers, especially (nominal) existential quantification, discussed in Section 3.2. The proof of the first statement proceeds by induction over the given reduction chain, eventually picking $\Gamma$ as the unrestricted context discarded by rule $\mathbf{obs}$. The proof of the second part is similar to that of property 4.4. Here, $\Sigma$ represents the signature symbols added to $\Sigma_P$ by rule $\exists_1$, which we collect by means of rule $\mathbf{obs}$ and make explicit in the goal formula. □

The second part of this property (completeness) admits a stronger statement in $\mathrm{LV}^{\mathrm{n}}$, namely, if $P \rightarrow^* Q$, then $\circ; \ulcorner P \urcorner \longrightarrow_{\Sigma_P} \ulcorner Q \urcorner$ is derivable in $\mathrm{LV}^{\mathrm{n}}$. As in the propositional case, no such strengthened soundness property (part 1) holds in $\mathrm{LV}^{\mathrm{n}}$.

Similarly to the propositional case, this result forces the interpretation of the first-order asynchronous $\pi$-calculus in the observational language $\mathrm{LV}^{\mathrm{obs}}$ rather than in the more general $\mathrm{LV}^{\mathrm{n}}$. The fact that we need to reconstruct lost replicated processes indicates that this language is not a perfect target, but as indicated earlier, the quest for such a perfect target is still on-going.

## 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 distinct meanings of "reuse" in linear logic (where ! is idempotent, so that $!A \equiv !!A$) in contrast to "replication" in process algebra (which is not idempotent, so that $!P \overset{\pi}{\not\equiv} !!P$), and their role with respect to the notion of structural equivalence (further discussed below). 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 $⅋$ and $\bot$ where we used $\otimes$ and $\mathbf{1}$, is an equally valid option that several authors have explored (*e.g.* [57]).

As noted earlier, process-based languages come in many variants which have not yet been reduced to a common denominator. The *synchronous $\pi$-calculus* [74] 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 [15] and [21], 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 $\mathsf{w}_x$ needs to be consumed before $\mathsf{c}(x, y)$ can be released — a process available to execute an input on $x$ will provide $\mathsf{w}_x$. The synchronous $\pi$-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 [24, 73]. While we are unaware of a general solution within linear logic, a correct encoding has been given in the closely related CLF logical framework [24]. Further behavioral variations of the process algebras have been proposed in the literature, see for example [74] for additional variants of the $\pi$-calculus. We are not aware of a systematic attempt at interpreting them in linear logic, although we believe such a translation could be beneficial. We will examine the join calculus [35] in a later section of this paper.

Many traditional accounts of the $\pi$-calculus, starting with [63], replace the one-way reduction $!P \rightarrow !P \parallel P$, implemented as rule $\mathbf{red}!$ above, with the two-way structural equivalence $!P \overset{\pi}{\equiv} !P \parallel P$. It would be natural to map it to the logical equivalence $!A \equiv !A \otimes A$, except that this is not an equivalence at all in LV (or in any presentation of linear logic): the sequent $\circ; A \otimes !A \ \longrightarrow_\Sigma \ !A$ is not derivable (although the reverse entailment does hold). Siding with several other authors (*e.g.*, [24, 29, 57, 70]), we opt instead for the more computational interpretation given by rule $\mathbf{red}!$. Alternatively, we could have accommodated $!P \overset{\pi}{\equiv} !P \parallel P$ as a structural equivalence by adopting $!A \equiv !A \otimes A$ as an extra-logical axiom in the completeness results in this section, so that each of them would postulate the existence of a linear logic derivation modulo $!A \equiv !A \otimes A$.

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 [70] 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 [57, 59] providing an interesting perspective on this little investigated problem. The treatment of these notions, although extremely interesting, is outside the scope of the present paper.

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 $\pi$-calculus [3, 4, 11]. Here concurrent computation corresponds to proof normalization (cut elimination), giving the system a functional flavor, with [4] 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 $\pi$-calculus into linear logic: processes become formulas and $\pi$-calculus reduction becomes entailment [57]. These ideas are generalized and reformulated as a logical framework in Miller's proposal for the specification logic Forum [58].

# 5 A Rewriting View of Linear Logic

In Section 2, we started from a traditional presentation of intuitionistic linear logic, Pfenning's LV [69], and isolated a semantic fragment that we massaged into the deductive system $\mathrm{LV}^{\mathrm{obs}}$. In the last two sections, we showed that the mainstream interpretations of various models of concurrency into linear logic target derivational behaviors that fit squarely within $\mathrm{LV}^{\mathrm{obs}}$.

In this section, we propose an alternative reading of $\mathrm{LV}^{\mathrm{obs}}$ as a rewrite system. In a way, we have done all the work in Section 2 already: with the exception of $\mathbf{obs}$, all rules in $\mathrm{LV}^{\mathrm{obs}}$'s cut-free semantics in Figure 3 have a single premise, which permits viewing them as a description of how to rewrite their conclusion into this one premise (the judgment $\Sigma \vdash t$ in rule $\forall_1$ is a simple side-condition, not a second premise). This process is guided by focusing on a single context formula (generally in the linear context, except for $\mathbf{clone}$). The goal formula never changes: it is instantiated by rule $\mathbf{obs}$, which is always applicable. If we take the signature and the two contexts to be the "state" that is rewritten by applying the rules of $\mathrm{LV}^{\mathrm{obs}}$ upward, then $\mathbf{obs}$ allows observing any state reachable during the rewrite process.

Reading the rules of $\mathrm{LV}^{\mathrm{obs}}$ in this way forces us to reflect on two strongly ingrained tenets of computational logic: the finiteness of derivations and the importance of the goal formula. By definition, a derivation is a finite object and the chaining of rules during proof search has the objective of finding such a finite derivation. In $\mathrm{LV}^{\mathrm{obs}}$, a derivation stump can almost always be grown indefinitely (the only exceptions involve purely additive-multiplicative theories [51], whose derivations are necessarily finite) and rule $\mathbf{obs}$ can stop this process at any point to observe what is being achieved. This endorses a view of derivations as infinite objects which can be approximated by a series of finite observations (the derivations in the traditional sense). This view is a perfect fit for concurrent systems, which are generally intended to model infinite behaviors.

In the computational view of the logical scaffolding, a sequent's goal is often interpreted as something we are interested in proving and the contents of the context as assumptions to be used during the derivation. The notion of goal-oriented proof search [61] strongly embraces this idea. The rewriting reading of $\mathrm{LV}^{\mathrm{obs}}$ reverses the focus of proof-building as it is the contexts that drive the construction of the derivation while the goal is used to make observations. Indeed, the goal is often just an output variable.

This reading of $\mathrm{LV}^{\mathrm{obs}}$ provides the foundations for a powerful form of rewriting, which we refer to as $\omega$. We will show in Section 6 that a tiny syntactic fragment of $\omega$ corresponds exactly to traditional multiset rewriting (or place/transition Petri nets). In Section 7, we similarly demonstrate that a small subset of $\omega$ naturally captures the

$$
\begin{array}{lcll}
\mathbf{1}_{\mathbf{l}} & : & \Sigma; \Gamma; (\Delta, \mathbf{1}) \;\Rightarrow_{\bar{\omega}}\; \Sigma; \Gamma; \Delta & \\
\otimes_{\mathbf{l}} & : & \Sigma; \Gamma; (\Delta, A_1 \otimes A_2) \;\Rightarrow_{\bar{\omega}}\; \Sigma; \Gamma; (\Delta, A_1, A_2) & \\
\multimap'_{\mathbf{l}} & : & (\Sigma ,, \Sigma'); \Gamma; (\Delta, \Delta', (\exists \Sigma'.\, \Delta') \multimap B) \;\Rightarrow_{\bar{\omega}}\; (\Sigma ,, \Sigma'); \Gamma; (\Delta, B) & \\
(\top_{\mathbf{l}}) & : & (\textit{No rule for } \top) & \\
\&_{\mathbf{l}i} & : & \Sigma; \Gamma; (\Delta, A_1 \mathbin{\&} A_2) \;\Rightarrow_{\bar{\omega}}\; \Sigma; \Gamma; (\Delta, A_i) & \\
\forall_{\mathbf{l}} & : & \Sigma; \Gamma; (\Delta, \forall x.\, A) \;\Rightarrow_{\bar{\omega}}\; \Sigma; \Gamma; (\Delta, [t/x]A) & \text{if} \quad \Sigma \vdash t \\
\exists_{\mathbf{l}} & : & \Sigma; \Gamma; (\Delta, \exists x.\, A) \;\Rightarrow_{\bar{\omega}}\; (\Sigma ,, x); \Gamma; (\Delta, A) & \\
!_{\mathbf{l}} & : & \Sigma; \Gamma; (\Delta, !A) \;\Rightarrow_{\bar{\omega}}\; \Sigma; (\Gamma \mathbin{\text{'}} A); \Delta & \\
\mathbf{clone} & : & \Sigma; (\Gamma \mathbin{\text{'}} A); \Delta \;\Rightarrow_{\bar{\omega}}\; \Sigma; (\Gamma \mathbin{\text{'}} A); (\Delta, A) &
\end{array}
$$

Figure 4: A Rewriting Interpretation of $\mathrm{LV}^{\mathrm{obs}}$

operational semantic of various process algebras, and it does so in a simpler way than our logical interpretation in Section 4.

Taken in its entirety, $\omega$ can be viewed 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. It can also be viewed as a sophisticated process algebra which supports the atomic execution of complex communication patterns, a rich set of process operators and a primitive notion of state. Furthermore, $\omega$ has deep logical roots since its semantics was obtained rather directly from the rules of intuitionistic linear logic. Of course, $\omega$ is much weaker than logic since it discards nearly all right rules of the LV sequent presentation, yet what is retained constitutes a powerful form of rewriting, as we will see. This development is indeed reminiscent of (and somewhat dual to) the synthesis of abstract logic programming from the proof-theory of intuitionistic logic [61].

With relations to the two major paradigms for distributed and concurrent computing, $\omega$ is a promising middle ground where both state-based and process-based specifications can coexist. We test this proposition in Section 8 in the arena of cryptographic protocol specification, in which both approaches are prominently used, and only ad-hoc mappings exist to bridge them. There, we hint at the development of $\omega$ into the protocol specification language MSR 3 and scrutinize various ways of expressing a protocol in it.

The present section is organized as follows. In Section 5.1, we formalize the reading of the rules of $\mathrm{LV}^{\mathrm{obs}}$ as a rewrite system. We streamline it in Section 5.2 into $\omega$ and highlight some of its properties in Section 5.3. Additional considerations are found in Section 5.4.

## 5.1 Interpreting Observational Sequent Rules to Rewrite Rules

With the exception of $\mathbf{obs}$ (and the cut rules, which we have proved admissible in $\mathrm{LV}^{\mathrm{obs}}$), each rule in Figure 3 can be interpreted as a transformation of the sequent in its conclusion to the sequent in its premise, possibly subject to side-conditions. We formalize this observation as a rewrite system whose *states* are triples $(\Sigma; \Gamma; \Delta)$ consisting of the signature $\Sigma$ and the two contexts, $\Gamma$ and $\Delta$, of an LV sequent. These entities continue to have the algebraic structure assigned to them in Section 2.1: $\Sigma$ is a commutative monoid without duplicate elements, $\Gamma$ is a set, and $\Delta$ is a commutative monoid. Recall that we write “$,,$”, “$\text{'}$” and “$,$” for their respective operations, and “$\text{-}$”, “$\circ$” and “$\cdot$” for the corresponding units. We deliberately omit the goal formula ($C$) for two reasons: technically, it never changes going from the conclusion to the premise of a rule; additionally, 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

$$\Sigma; \Gamma; \Delta \;\Rightarrow_{\bar{\omega}}\; \Sigma'; \Gamma'; \Delta'$$

reserving the form $\_ \Rightarrow^*_{\bar{\omega}} \_$ for its reflexive and transitive closure. The mapping from the left-rules in Figure 3 to the single-step rewrite judgment on the one hand, and from open derivations to the multi-step judgment on the other are schematically described as follows:

$$\frac{\Gamma'; \Delta' \longrightarrow_{\Sigma'} C}{\Gamma; \Delta \longrightarrow_{\Sigma} C} \quad \rightsquigarrow \quad \Sigma; \Gamma; \Delta \Rrightarrow_{\bar{\omega}} \Sigma'; \Gamma'; \Delta' \qquad\qquad \dfrac{\dfrac{\Gamma''; \Delta'' \longrightarrow_{\Sigma''} C}{\vdots}}{\Gamma; \Delta \longrightarrow_{\Sigma} C} \quad \rightsquigarrow \quad \Sigma; \Gamma; \Delta \Rrightarrow^*_{\bar{\omega}} \Sigma''; \Gamma''; \Delta''$$

The resulting single-step rewrite rules are displayed in Figure 4 — we retained the name of the corresponding inference rule from Figure 3. The judgment $\Sigma \vdash t$ in rule $\forall_1$ is a simple side-condition. We call this system $\bar{\omega}$. It is an intermediate step in the definition of $\omega$, indeed rules $\mathbf{1}_1$ and $\otimes_1$ have been grayed out because we will dispense with them in Section 5.2 by identifying linear contexts and tensored formulas.

Before we further massage the rewrite system just obtained into $\omega$ in the next section, we will formally prove that $\bar{\omega}$ is sound and complete with respect to $\mathrm{LV}^{\mathrm{obs}}$. Intuitively, this holds because they are just two different presentations of the same formal system. Before doing so, the following lemma will come handy: it essentially states that the signature and the unrestricted context grow monotonically as the rewrite process unfolds.

**Lemma 5.1** *For any signatures $\Sigma$ and $\Sigma''$ and contexts $\Gamma$, $\Gamma''$, $\Delta$ and $\Delta'$, whenever $\Sigma; \Gamma; \Delta \Rrightarrow^*_{\bar{\omega}} \Sigma''; \Gamma''; \Delta'$, there exist a signature $\Sigma'$ and a context $\Gamma'$ such that $\Sigma'' = \Sigma „ \Sigma'$ and $\Gamma'' = \Gamma ⨾ \Gamma'$.*

**Proof.** A simple inspection of the rules in Figure 4 shows that the signature and the unrestricted context grow monotonically. This is formalized by an easy induction. □

This lemma allows us to display any rewrite chain as $\Sigma; \Gamma; \Delta \Rrightarrow^*_{\bar{\omega}} (\Sigma „ \Sigma'); (\Gamma ⨾ \Gamma'); \Delta'$ without loss of generality.

We next turn to the completeness result: any cut-free derivation in $\mathrm{LV}^{\mathrm{obs}}$ can be read as a rewriting sequence in $\bar{\omega}$.

**Property 5.2** *For any signature $\Sigma$, contexts $\Gamma$ and $\Delta$, and formula $C$, if $\Gamma; \Delta \longrightarrow_{\Sigma} C$ is derivable in $\mathrm{LV}^{\mathrm{obs}}$, then there exist a signature $\Sigma'$ and contexts $\Gamma'$ and $\Delta'$ such that $\Sigma; \Gamma; \Delta \Rrightarrow^*_{\bar{\omega}} (\Sigma „ \Sigma'); (\Gamma ⨾ \Gamma'); \Delta'$ and $C = \exists \Sigma'.\, \Delta'$.*

**Proof.** By construction, every rule in Figure 4 is obtained from one of the left rules of $\mathrm{LV}^{\mathrm{obs}}$. By rule $\mathbf{obs}$, the formula $C$ has the prescribed shape. Formally, the proof proceeds by induction on the given derivation of $\Gamma; \Delta \longrightarrow_{\Sigma} C$. □

The converse soundness result states that any rewrite sequence built in $\bar{\omega}$ corresponds to a derivation in $\mathrm{LV}^{\mathrm{obs}}$. This is expressed by the following property:

**Property 5.3** *For any signatures $\Sigma$ and $\Sigma'$ and any contexts $\Gamma$, $\Gamma'$, $\Delta$ and $\Delta'$, if $\Sigma; \Gamma; \Delta \Rrightarrow^*_{\bar{\omega}} (\Sigma „ \Sigma'); (\Gamma ⨾ \Gamma'); \Delta'$, then there is an $\mathrm{LV}^{\mathrm{obs}}$ derivation $\mathcal{D}$ of $\Gamma; \Delta \longrightarrow_{\Sigma} \exists \Sigma'.\, \Delta'$.*

**Proof.** This proof is essentially the reverse of the proof of Property 5.2. □

By virtue of the discussion outlined in Section 2.6, this result can be strengthened to mention the unrestricted context extension in the goal formula. Of course, this forces us to step outside of $\mathrm{LV}^{\mathrm{obs}}$. Indeed, the following result holds in $\mathrm{LV}^{\mathrm{n}}$, although it could be specialized to the language that we tentatively called $\mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes\exists!}$ in Section 2.6.

**Lemma 5.4** *For any signatures $\Sigma$ and $\Sigma'$ and any contexts $\Gamma$, $\Gamma'$, $\Delta$ and $\Delta'$, if $\Sigma; \Gamma; \Delta \Rrightarrow^*_{\bar{\omega}} (\Sigma „ \Sigma'); (\Gamma ⨾ \Gamma'); \Delta'$, then there is an $\mathrm{LV}^{\mathrm{n}}$ derivation $\mathcal{D}$ of $\Gamma; \Delta \longrightarrow_{\Sigma} \exists \Sigma'.\, (!\Gamma', \Delta')$.*

**Proof.** As indicated in Section 2.6, this proof makes use of the fact that for any signatures $\Sigma$ and $\Sigma'$ and any contexts $\Gamma$, $\Gamma'$ and $\Delta$, the sequent $\Gamma ⨾ \Gamma'; \Delta \longrightarrow_{\Sigma „ \Sigma'} \exists \Sigma'.\, (!\Gamma', \Delta)$ is always derivable in $\mathrm{LV}^{\mathrm{n}}$. □

Before we move on to defining $\omega$, we shall briefly reflect on the impact that altering $\mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes\exists}$ into $\mathrm{LV}^{\mathrm{obs}}$ had in preparation to extracting the rewrite system shown in Figure 4. See Sections 2.3–2.5 for details. $\mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes\exists}$ features three rules with two premises ($\multimap_1$, $\mathbf{cut}$ and $\mathbf{cut!}$): one of them is a major premise that carries over the goal formula in the conclusion, the other is a minor premise that mentions a totally different goal formula. Directly

$$\begin{array}{|lll|}
\hline
\multimap_{\mathbf{l}}' & : & (\Sigma ,, \Sigma'); \Gamma; (\Delta, \Delta', (\exists \Sigma'.\, \Delta') \multimap B) \;\Rightarrow\; (\Sigma ,, \Sigma'); \Gamma; (\Delta, B) \\
\&_{\mathbf{l}i} & : & \Sigma; \Gamma; (\Delta, A_1 \,\&\, A_2) \;\Rightarrow\; \Sigma; \Gamma; (\Delta, A_i) \\
\forall_{\mathbf{l}} & : & \Sigma; \Gamma; (\Delta, \forall x.\, A) \;\Rightarrow\; \Sigma; \Gamma; (\Delta, [t/x]A) \quad \text{if} \quad \Sigma \vdash t \\
\exists_{\mathbf{l}} & : & \Sigma; \Gamma; (\Delta, \exists x.\, A) \;\Rightarrow\; (\Sigma ,, x); \Gamma; (\Delta, A) \\
!_{\mathbf{l}} & : & \Sigma; \Gamma; (\Delta, !A) \;\Rightarrow\; \Sigma; (\Gamma ,, A); \Delta \\
\mathbf{clone} & : & \Sigma; (\Gamma ,, A); \Delta \;\Rightarrow\; \Sigma; (\Gamma ,, A); (\Delta, A) \\
\hline
\end{array}$$

Figure 5: The Rules of $\boldsymbol{\omega}$-Rewriting

giving $\mathrm{LV}^{\mathrm{obs}}_{\mathbf{1}\otimes\exists}$ a rewriting interpretation would have yielded an unusual rewrite system. Indeed, we would have been forced to regard the minor premise in rules $\mathbf{cut}$, $\mathbf{cut}!$ and $\multimap_{\mathbf{l}}$ as prescribing the existence of an auxiliary finite rewriting chain that enables the step corresponding to the major premise to each of these rules. This auxiliary chain must be finite, and therefore capped by rule $\mathbf{obs}$. This would make the single step relation $\Rightarrow_{\bar{\omega}}$ dependent on the multi-step observation relation $\Rightarrow^*_{\bar{\omega}}$ in these rules. Replacing rule $\multimap_{\mathbf{l}}$ with the single premise rule $\multimap_{\mathbf{l}}'$ essentially amounts to in-lining the auxiliary rewriting chain corresponding to the minor premise. The elimination of rules $\mathbf{cut}$ and $\mathbf{cut}!$ achieves the same effect, although in a slightly more complicated way. See Section 2.5 for details.

## 5.2 The Rewriting System $\boldsymbol{\omega}$

We will now define $\omega$ by simply identifying contexts and formulas in $\bar{\omega}$, an idea familiar from categorical interpretations of logic [12, 75], just like we did at the logical level in Section 2.2. More precisely, we identify the tensor $\otimes$ and its unit $\mathbf{1}$ with the union "," and unit "$\cdot$" constructors of linear contexts, respectively. Since rule $\otimes_{\mathbf{l}}$ in Figure 4 states that $\otimes$ reduces to "," we will simply take the later as a primitive. Rule $\mathbf{1}_{\mathbf{l}}$ similarly reduces $\mathbf{1}$ to the empty multiset. Our language of formulas is then updated as follows:

$\omega$*-Multisets* $\qquad A, B, C, \Delta \;::=\; a \;\mid\; \cdot \;\mid\; A, B \;\mid\; A \multimap B \;\mid\; !A \;\mid\; \top \;\mid\; A \,\&\, B \;\mid\; \forall x.\, A \;\mid\; \exists x.\, A$

With respect to the original grammar of linear logic in Section 2.1, we have simply replaced $\mathbf{1}$ with "$\cdot$" and $\otimes$ with ",'. We will refer to formulas of this form as $\omega$*-multisets*. Note that this definition also states that a linear context $\Delta$ is now just an $\omega$-multiset. But an unrestricted context is *not* an $\omega$-multiset.

This mild redefinition of formulas allows us to streamline the rewrite machinery developed in Section 5.1 for $\mathrm{LV}^{\mathrm{obs}}$. The structure of states remains unchanged: triples of the form $(\Sigma; \Gamma; \Delta)$ for a signature $\Sigma$, an unrestricted context $\Gamma$ and a linear context/$\omega$-multiset $\Delta$. We write

$$\Sigma; \Gamma; \Delta \;\Rightarrow_{\omega}\; \Sigma'; \Gamma'; \Delta' \qquad \text{and} \qquad \Sigma; \Gamma; \Delta \;\Rightarrow^*_{\omega}\; \Sigma'; \Gamma'; \Delta'$$

for the single-step and multi-step rewrite judgments, respectively. The semantics of the former simply omits the rules for $\mathbf{1}$ and $\otimes$ from Figure 4 while the latter is defined as its reflexive and transitive closure. For future reference, we report the rules of $\_ \Rightarrow_{\omega} \_$ in Figure 5.

Having defined $\omega$, we will dedicate the rest of this section to showing that it is equivalent to $\bar{\omega}$, the rewrite system obtained in the last section from $\mathrm{LV}^{\mathrm{obs}}$, and then to porting some of its properties to $\omega$. We shall begin by defining a transformation $(\_)^{\varnothing}$ on LV formulas that replaces each occurrence of the tensor or its unit with "," or "$\cdot$" respectively. Therefore, it maps any linear logic formula $A$ into the corresponding $\omega$-multiset $(A)^{\varnothing}$ according to the grammar shown at the beginning of this section. We omit the straightforward inductive definition. We extend this transformation to the linear and intuitionistic contexts of LV by applying $(\_)^{\varnothing}$ to each of their constituent formulas: given $\Gamma$ and $\Delta$, we obtain $(\Gamma)^{\varnothing}$ and $(\Delta)^{\varnothing}$. Recall that the linear context $\Delta$ of a state in $\omega$ is itself an $\omega$-multiset (but of course this is not true of its unrestricted context $\Gamma$ — avoiding the confusion was indeed our main reason for choosing different notations for their constructors).

Now, $\omega$ is sound with respect to $\bar{\omega}$: for every rewrite sequence in $\bar{\omega}$, a corresponding rewrite sequence exists in $\omega$. This is given by the following lemma.

**Lemma 5.5** *For any signatures* $\Sigma$ *and* $\Sigma'$ *and contexts* $\Gamma$, $\Gamma'$, $\Delta$ *and* $\Delta'$, *if* $\Sigma; \Gamma; \Delta \;\Rightarrow^*_{\bar{\omega}}\; \Sigma'; \Gamma'; \Delta'$, *then* $\Sigma; (\Gamma)^{\oslash}; (\Delta)^{\oslash} \;\Rightarrow^*_{\omega}\; \Sigma'; (\Gamma')^{\oslash}; (\Delta')^{\oslash}$.

**Proof.** The proof proceeds by induction on the given rewriting sequence of $\Sigma; \Gamma; \Delta \;\Rightarrow^*_{\bar{\omega}}\; \Sigma'; \Gamma'; \Delta'$. It essentially elides all applications of rules $\mathbf{1}_1$ and $\otimes_1$. In particular, the rewriting sequence in $\omega$ will typically be shorter. □

The converse completeness results holds also: given a rewrite sequence in $\omega$, it is always possible to turn some "," into $\otimes$ and "$\cdot$" into $\mathbf{1}$ and insert appropriate applications of rules $\otimes_1$ and $\mathbf{1}_1$ to reconstruct a valid rewrite sequence in $\bar{\omega}$.

**Lemma 5.6** *For any signatures* $\Sigma$ *and* $\Sigma'$ *and contexts* $\Gamma$, $\Gamma'$, $\Delta$ *and* $\Delta'$, *if* $\Sigma; (\Gamma)^{\oslash}; (\Delta)^{\oslash} \;\Rightarrow^*_{\omega}\; \Sigma'; \Gamma'; \Delta'$, *then there exists contexts* $\Gamma''$ *and* $\Delta''$ *such that* $\Sigma; \Gamma; \Delta \;\Rightarrow^*_{\bar{\omega}}\; \Sigma'; \Gamma''; \Delta''$ *where* $\Gamma'' = (\Gamma')^{\oslash}$ *and* $\Delta'' = (\Delta')^{\oslash}$.

**Proof.** Through an induction on the construction of $\Sigma; (\Gamma)^{\oslash}; (\Delta)^{\oslash} \;\Rightarrow^*_{\omega}\; \Sigma'; \Gamma'; \Delta'$, this proof essentially reinstates applications of rules $\mathbf{1}_1$ and $\otimes_1$ whenever needed to apply a subsequent (non-tensorial) rule. □

Having just shown that $\bar{\omega}$ and $\omega$ are equivalent, we will take some notational liberties from now on for the convenience of the reader. First, we will omit the subscript $_\omega$ from the rewrite judgments of $\omega$, obtaining

$$\Sigma; \Gamma; \Delta \;\Rightarrow\; \Sigma'; \Gamma'; \Delta' \qquad\text{and}\qquad \Sigma; \Gamma; \Delta \;\Rightarrow^*\; \Sigma'; \Gamma'; \Delta'$$

(we have already adopted this convention in Figure 5). Second, we resurrect $\otimes$ and $\mathbf{1}$ as syntactic sugar: we will indeed use $\otimes$ and "," interchangeably from now on (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. Third, we will keep the transformation $(\_)^{\oslash}$ implicit.

All the properties we proved in Section 5.1 hold also in $\omega$. In particular, the following results adapts Lemma 5.1 by stating that as the rewriting unfolds, the signature and unrestricted context grow monotonically.

**Lemma 5.7** *For any signatures* $\Sigma$ *and* $\Sigma''$ *and contexts* $\Gamma$, $\Gamma''$, $\Delta$ *and* $\Delta'$, *whenever* $\Sigma; \Gamma; \Delta \;\Rightarrow^*\; \Sigma''; \Gamma''; \Delta'$, *there exist a signature* $\Sigma'$ *and a context* $\Gamma'$ *such that* $\Sigma'' = \Sigma ,, \Sigma'$ *and* $\Gamma'' = \Gamma ; \Gamma'$.

**Proof.** This easy proof follows the steps used to prove Lemma 5.1. The equivalence of $\bar{\omega}$ and $\omega$ cannot be directly leveraged here. □

Next, since $\bar{\omega}$'s rewriting semantics is sound and complete with respect to $\mathrm{LV}^{\mathrm{obs}}$'s notion of derivability, so is $\omega$. (Recall that we are keeping $(\_)^{\oslash}$ implicit.)

**Corollary 5.8**

- *For any signature* $\Sigma$, *contexts* $\Gamma$ *and* $\Delta$, *and formula* $C$, *if* $\Gamma; \Delta \longrightarrow_{\Sigma} C$ *is derivable in* $\mathrm{LV}^{\mathrm{obs}}$, *then there exist a signature* $\Sigma'$ *and contexts* $\Gamma'$ *and* $\Delta'$ *such that* $\Sigma; \Gamma; \Delta \;\Rightarrow^*\; (\Sigma ,, \Sigma'); (\Gamma ; \Gamma'); \Delta'$ *and* $C = \exists \Sigma'.\, \Delta$.
- *For any signatures* $\Sigma$ *and* $\Sigma'$ *and any contexts* $\Gamma$, $\Gamma'$ *and* $\Delta$, *if* $\Sigma; \Gamma; \Delta \;\Rightarrow^*\; (\Sigma ,, \Sigma'); (\Gamma ; \Gamma'); \Delta'$, *then there is an* $\mathrm{LV}^{\mathrm{obs}}$ *derivation* $\mathcal{D}$ *of* $\Gamma; \Delta \longrightarrow_{\Sigma} \exists \Sigma'.\, \Delta'$.

**Proof.** By Properties 5.2 and 5.3, each results holds with respect to $\bar{\omega}$. Lemmas 5.5 and 5.6 map them to $\omega$. □

Just like in the case of $\bar{\omega}$, a stronger soundness result that refers to $\mathrm{LV}^{\mathrm{n}}$ is obtainable for $\omega$. We will not have a need for it in this paper.

## 5.3 Some Properties of $\omega$

In this section, we will briefly examine some properties of $\omega$ that will be useful in the sequel. We start with the following simple weakening lemma, which states that a rewrite sequence remains valid if we augment its starting and ending states with the identical objects.

**Lemma 5.9** *For any signatures* $\Sigma$, $\Sigma'$ *and* $\Sigma''$ *and contexts* $\Gamma$, $\Gamma'$, $\Gamma''$, $\Delta$, $\Delta'$ *and* $\Delta''$, *if* $\Sigma; \Gamma; \Delta \;\Rightarrow^*\; \Sigma'; \Gamma'; \Delta'$, *then* $(\Sigma \mathbin{,\!,} \Sigma''); (\Gamma \mathbin{\mathring{,}} \Gamma''); (\Delta, \Delta'') \;\Rightarrow^*\; (\Sigma' \mathbin{,\!,} \Sigma''); (\Gamma' \mathbin{\mathring{,}} \Gamma''); (\Delta', \Delta'')$.

**Proof.** By induction on $\Sigma; \Gamma; \Delta \;\Rightarrow^*\; \Sigma'; \Gamma'; \Delta'$. □

Next, if a variable does not appear in the context parts of the initial and final states of a rewrite sequence, then there is an equivalent rewrite chain that does not make use of it at all.

**Lemma 5.10** *If* $(\Sigma \mathbin{,\!,} x); \Gamma; \Delta \;\Rightarrow^*\; (\Sigma' \mathbin{,\!,} x); \Gamma'; \Delta'$ *and* $x \notin \mathrm{FV}(\Gamma, \Gamma', \Delta, \Delta')$ *and* $\Sigma$ *contains at least one term-level object, then* $\Sigma; \Gamma; \Delta \;\Rightarrow^*\; \Sigma'; \Gamma'; \Delta'$.

**Proof.** This proof proceeds by induction on the given rewrite sequence. Note that rule $\forall_1$ may draw on $x$ for an intermediate substitution. We then show that such uses of $x$ can be replaced with any other term (which exists by assumption). □

Note that $x$ must occur in the initial signature: if $\Sigma; \Gamma; \Delta \;\Rightarrow^*\; (\Sigma' \mathbin{,\!,} x); \Gamma'; \Delta'$ and $x \notin \mathrm{FV}(\Gamma', \Delta')$, there may be no rewrite sequence for $\Sigma; \Gamma; \Delta \;\Rightarrow^*\; \Sigma'; \Gamma'; \Delta'$. For example $\text{-}; \circ; \exists x.\, \mathbf{1} \;\Rightarrow^*\; x; \circ; \cdot$ but $\text{-}; \circ; \exists x.\, \mathbf{1} \;\Rightarrow^*\; \text{-}; \circ; \cdot$ is not achievable.

We next turn to the equivalence $\equiv_{\otimes\exists}$ we derived from Lemmas 2.1 and 2.5 in Section 2. Whenever two formulas are related through $\equiv_{\otimes\exists}$, the system $\omega$ will rewrite them to states that differ at most by the contents of their signature. In its bare-bones form, this has the following statement.

**Lemma 5.11** *Let* $A$ *and* $B$ *be formulas such that* $A \equiv_{\otimes\exists} B$ *and let* $\Sigma = \mathrm{FV}(A)$. *Then there exist signatures* $\Sigma_A$ *and* $\Sigma_B$ *and a linear context* $\Delta$ *such that* $\Sigma; \circ; A \;\Rightarrow^*\; \Sigma_A; \circ; \Delta$ *and* $\Sigma; \circ; B \;\Rightarrow^*\; \Sigma_B; \circ; \Delta$.

**Proof.** After observing that if $A \equiv_{\otimes\exists} B$, then $\mathrm{FV}(A) = \mathrm{FV}(B)$, the proof proceeds by induction on the evidence that $A \equiv_{\otimes\exists} B$, applying rule $\exists_1$ as needed to move existentially bound variables to the signature. $\Sigma_A$ and $\Sigma_B$ could be different because processing the equivalence $\exists x \mathbf{1} \equiv_{\otimes\exists} \mathbf{1}$ in this way introduces the variable $x$ in one of the signatures but not in the other. □

It should be noted that the same result holds for $\equiv_{\otimes}$ and $\equiv_{\exists}$ since they are subrelations of $\equiv_{\otimes\exists}$. Moreover, if $A \equiv_{\otimes} B$, then $A$ and $B$ are the same $\omega$-multiset: more precisely, if $A \equiv_{\otimes} B$ then $(A)^{\otimes} = (B)^{\otimes}$.

Finally, swapping $\equiv_{\otimes\exists}$-equivalent terms in a rewrite sequence will eventually produce states that differ at most by their signature.

**Lemma 5.12** *For any formulas* $A$ *and* $B$, *signatures* $\Sigma$ *and* $\Sigma'$, *and contexts* $\Gamma$, $\Gamma'$, $\Delta$ *and* $\Delta'$, *if* $A \equiv_{\otimes\exists} B$ *and* $\Sigma; \Gamma; (\Delta, A) \;\Rightarrow^*\; \Sigma'; \Gamma'; \Delta'$, *then there exist signatures* $\Sigma''_A$ *and* $\Sigma''_B$, *and context* $\Gamma''$ *and* $\Delta''$ *such that* $\Sigma'; \Gamma'; \Delta' \;\Rightarrow^*\; \Sigma''_A; \Gamma''; \Delta''$ *and* $\Sigma; \Gamma; (\Delta, B) \;\Rightarrow^*\; \Sigma''_B; \Gamma''; \Delta''$.

**Proof.** By Lemmas 5.11 and 5.9, there exist signatures $\Sigma_A$ and $\Sigma_B$ and a context $\Delta^*$ such that $\Sigma; \Gamma; (\Delta, A) \;\Rightarrow^*\; \Sigma_A; \Gamma; (\Delta, \Delta^*)$ and $\Sigma; \Gamma; (\Delta, B) \;\Rightarrow^*\; \Sigma_B; \Gamma; (\Delta, \Delta^*)$. The proof then proceeds by induction on the rewrite sequence $\Sigma; \Gamma; (\Delta, A) \;\Rightarrow^*\; \Sigma'; \Gamma'; \Delta'$: any time a rule applies to a formula that is not a subcomponent of $A$, then these rewrite sequences are extended accordingly, while any application of a rule operating on $A$ is already captured within $\Sigma; \Gamma; (\Delta, A) \;\Rightarrow^*\; \Sigma_A; \Gamma; (\Delta, \Delta^*)$ and $\Sigma; \Gamma; (\Delta, B) \;\Rightarrow^*\; \Sigma_B; \Gamma; (\Delta, \Delta^*)$. □

## 5.4 Discussion

So far, we have extracted a rewriting system from a substantial fragment of linear logic. Before assessing the rewriting merits of $\omega$ in sections to come, we shall conclude this part with reflections on our methodology and comparisons with related ideas from the literature. Many of the issues raised below are challenges that will be interesting to explore in future work.

Although our presentation of LV in Section 2 encompasses a majority of the constructs of linear logic, Girard's original formalism makes a few more operators available [38]. It is natural to wonder whether $\omega$ could be enriched with some of them. The remaining operators of the minimal intuitionistic fragment of linear logic are $\oplus$ and its unit $\mathbf{0}$. An $\omega$-style reading of the left rule of $\oplus$,

$$\frac{\Gamma; \Delta, A_1 \longrightarrow_\Sigma C \quad \Gamma; \Delta, A_2 \longrightarrow_\Sigma C}{\Gamma; \Delta, A_1 \oplus A_2 \longrightarrow_\Sigma C} \oplus_l$$

seems to require that the two branches shall share a common observation, which is vaguely reminiscent of bisimulation or may-testing. We do not understand this rule as a general rewriting operation at this stage. Its nullary form, $\mathbf{0}_l$, 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 [16], the left rule for $\otimes$,

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

seems to endow multiplicative disjunction with a rewriting semantics that splits the state and starts two totally independent computations, each with its own observations. 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 $\omega$ coincide with the fragment of linear logic at the core of the type-theoretic logical framework for concurrency CLF [24, 79] (and indeed, the semantics of $\omega$ is closely related to the "synchronous" fragment of CLF). The fact that two independent investigations led to similar languages suggests that the constructs comprising $\omega$ (and CLF) have some intrinsic "good" properties, although we do not fully understand them yet.

Although $\omega$ can be summarized fairly accurately as the result of giving a computational interpretation to the left sequent rules of linear logic, this paper has shown that formalizing this intuition is a rather involved process, as attested by the numerous intermediate languages in Figure 1 (admittedly, some of them had mainly expository value). This raises fascinating questions about the relation between the starting point of an investigation such as this one (here intuitionistic linear logic presented as LV sequents) and the ease of the formal development.

We started this investigation from LV because it elegantly captures the structural characteristics of linear logic, especially as far as reusability is concerned. It also permitted relatively simple proofs of our various results. Our attempts at using other expositions of linear logic were not as successful: the traditional single-context sequent rules for ! proved difficult to tame, which is in contrast with the ability to segregate reusable assumptions in an algebraically confined unrestricted context. Moreover, relying on two-sided sequents made the goal formula available to observe the state of the computation. The fact that LV made this work possible raises the question of whether some other presentations of linear logic could have made it even easier, and more broadly of the exact role of the structure of judgments in a meta-theoretic investigation.

Our starting point was also a specific fragment of linear logic. Linearity was clearly key to achieving a rewrite system because it supports a view of context formulas as consumable resources which is in line with the destructive nature of rewriting. It is however conceivable that a similar development can be carried out starting from other sub-structural logics, and possibly even from specific presentations of, say, traditional intuitionistic logic.

The methodology proposed here places a strong emphasis on the left rules of (linear) logic, with the right rules reduced to an observational rule. It is worth contrasting this characteristic with the tenets of logic programming as uniform provability [61], 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 [7, 43, 58]. In a partial departure from this short tradition, Kobayashi and Yonezawa's ACL [48] derives its semantics from specialized versions of left rules of linear logic (when examined through the lens of duality). This, together with its acceptance of open derivations and support for concurrency, makes ACL a close relative to $\omega$. 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) [50], interestingly stirring it in the direction of constraint programming (see also [31]). To the extent of our knowledge, ACL and HLcc are the proposals closest to $\omega$ in the literature.

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. The deductive system LV$^{\mathrm{obs}}$ in Section 2.3 supports a different view of rules, derivations, and in a sense logic. It is 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) assumes secondary importance, essentially as a form of observation. This endows $\omega$ with a semantics based on transition-sequences, which is commonplace in rewriting theory. In $\omega$, it is a small conceptual step to distill minimal partial orders (traces) by forcing sequentiality only when steps actually depend on each other. This observation can be transported back to the logical side by considering a notion of derivation based not on trees but on partial orders of dependencies (essentially DAGs). Andreoli's "desequentialized proofs" [6] 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 [68], possibly the earliest model of concurrency, and a number of specification approaches including automata for model checking [55] and inductive definitions [65].

We show that a tiny syntactic fragment of $\omega$ corresponds exactly to traditional multiset rewriting, with its usual semantics given by a few of the rules in Figure 5. Given the way we developed $\omega$ through Section 5, this constitutes an interpretation of multiset rewriting *as* (a fragment of) logic, which we like to contrast to a number of earlier interpretations *into* (a fragment of) logic [8, 17, 20, 28, 39, 47, 52]. The system $\omega$ similarly provides a logical interpretation of more sophisticated forms of multiset rewriting and Petri nets.

## 6.1 Propositional Multiset Rewriting

We recall from Section 3.1 that propositional multiset rewriting (MSR$_0$) applies rewrite rules of the form $r = \tilde{a} \to \tilde{b}$ to states $\tilde{s}$ where multisets $\tilde{a}$, $\tilde{b}$ and $\tilde{s}$ are commutative monoids with operation "$\tilde{,}$" and unit "$\tilde{\cdot}$". Its rewriting semantics is given by the meta-rule $msr_0$ of the form $(\tilde{c} \tilde{,} \tilde{a}) \rhd_{R \hat{,} (\tilde{a} \to \tilde{b})} (\tilde{c} \tilde{,} \tilde{b})$. Here $R$ is a set of such rewrite rules, and we wrote "$\hat{,}$" and "$\hat{\cdot}$" for set union and the empty set respectively. See Section 3.1 for details.

In Section 3.1 we defined a homomorphic embedding of the entities of MSR$_0$ into linear logic by interpreting "$\tilde{\cdot}$", "$\tilde{,}$", $\to$, "$\hat{\cdot}$" and "$\hat{,}$" as "$\otimes$", "$\mathbf{1}$", $\multimap$, $\circ$ and $,$ respectively. We denoted this family of encodings as $\ulcorner \_ \urcorner$. Then, reachability in MSR$_0$ corresponded to derivability in LV$^{\mathrm{obs}}$. This correspondence was complete in the fragment of linear logic in the image of this encoding, which we called $LL^{MSR_0}$.

The same encodings support a sound and complete interpretation of MSR$_0$ into $\omega$. Since tensorial formulas are identified with linear contexts in this language, "$\tilde{\cdot}$" and "$\tilde{,}$" are mapped to "$\cdot$" and ",". Then, propositional multiset rewriting is immediately recognized as a form of $\omega$-rewriting by interpreting multisets as linear contexts and rule sets as unrestricted contexts. Indeed multisets obey the same monoidal laws as contexts, and the semantic rule $msr_0$ introduced in Section 3.1 can be seen as an application of rule **clone** immediately followed by $\multimap_1'$. The soundness of this encoding is formally stated by the following simple property:

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

**Proof.** This easy proof can be approached in two ways: we can either proceed by a simple induction on the given rewrite chain, or we can invoke Property 3.1 to obtain the LV$^{\mathrm{obs}}$ sequent $\ulcorner R \urcorner; \ulcorner \tilde{s} \urcorner \longrightarrow_S \ulcorner \tilde{s}' \urcorner$, from which Corollary 5.8 (part 1) yields the desired result once we observe that neither the signature nor the unrestricted context can be extended since $\ulcorner R \urcorner$ and $\ulcorner \tilde{s} \urcorner$ do not make use of either $\exists$ or !. □

Just like $\ulcorner \_ \urcorner$ identified a syntactic fragment $LL^{MSR_0}$ of linear logic over which completeness holds, it now identifies a related fragment $\omega^{MSR_0}$ of $\omega$ on which reachability in the two languages coincide. Indeed, the inverse of the above property holds:

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

**Proof.** By Corollary 5.8 (part 2), the LV$^{\mathrm{obs}}$ sequent $\ulcorner R \urcorner; \ulcorner \tilde{s} \urcorner \longrightarrow_S \ulcorner \tilde{s}' \urcorner$ is derivable since the representation does not make use of $\exists$ or !. Now, the desired result follows by Property 3.2. A direct proof requires permuting applications of the $\omega$-rules in Figure 5 just as done in the proof of Property 3.2. □

Together, these properties and the simple mapping underlying them allow us to view propositional multiset rewriting as a fragment of $\omega$-rewriting, and therefore of linear logic. In particular, it permits redefining the semantics of $\mathrm{MSR}_0$ on a purely logical basis very directly.

## 6.2 First-Order Multiset Rewriting

In Section 3.2 we defined first-order multiset rewriting, $\mathrm{MSR}_1$, by allowing state elements to carry ground structured terms and extending multiset rewrite rules, which assumed the form $\forall\vec{x}.\tilde{a} \to \exists\vec{n}.\tilde{b}$. The semantics of $\mathrm{MSR}_1$ was defined by the meta-rule $msr_1$ given by $\Sigma; (\tilde{c}_{\bar{,}}\, [\vec{t}/\vec{x}]\tilde{a}) \rhd_{R;(\forall\vec{x}.\tilde{a}\to\exists\vec{n}.\tilde{b})} (\Sigma, \vec{n}); (\tilde{c}_{\bar{,}}\, [\vec{t}/\vec{x}]\tilde{b})$ where $\Sigma \vdash \vec{t}$. The encoding of $\mathrm{MSR}_0$ into $\mathrm{LV}^{\mathrm{obs}}$ was extended by simply mapping the rewrite rule binders $\forall$ and $\exists$ to the homonymous quantifiers $\forall$ and $\exists$. Then, reachability in $\mathrm{MSR}_1$ was captured by derivability in $\mathrm{LV}^{\mathrm{obs}}$ in a sound and complete way with respect to the image of the resulting encoding, a fragment of linear logic that we called $LL^{MSR_1}$. See Property 3.3 in Section 3.2.

As for $\mathrm{MSR}_0$ in Section 6.1, the extended encoding and the consequent soundness and completeness results hold in $\omega$ as well, this time with respect to a fragment of $\omega$ that we call $\omega^{MSR_1}$. Indeed, we have the following composite result:

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

**Proof.** This proof proceeds along the lines of the proofs of Properties 6.1 and 6.2. The main difference is that rule $\exists_1$ is now applicable, but notice that there is still no mention of ! so that the unrestricted context never changes. The proof can be carried out either directly or by going through $\mathrm{LV}^{\mathrm{obs}}$ by means of results proved in Sections 3.2 and 5.2. $\square$

Notice that this statement is more streamlined than the corresponding Property 3.3 in linear logic as the final state mentions an explicit unrestricted context that is not directly visible in $\mathrm{LV}^{\mathrm{obs}}$.

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

## 6.3 Discussion

From the above discussion, it is clear that $\mathrm{MSR}_1$ accounts only for a very small fragment ($\omega^{MSR_1}$) of $\omega$. We will now explore what else $\omega$ 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, $\omega$ 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 $\omega^{MSR_1}$. In this way, $\omega$ not only internalizes the rewriting operation within the state, but also makes it available for manipulation as a first-class object.

Furthermore, $\omega$ 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 "·") 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 $\mathrm{MSR}_1$ rule is an example, $(a \multimap b) \,\&\, !(c \multimap \mathbf{1})$ is another.[4]

*Embedded rewrites*, such as $(a \multimap b, (c, d \multimap e))$,[5] 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

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

[5] "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."

Hierarchical and Object Petri Nets [33, 78], fully realize this program by permitting nets to manipulate other nets, often using reflection to move between levels. Among them, Farwer's Linear Logic Petri Nets [32, 34] 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 [49], 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 $\omega$ 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.

It is instead a theoretical investigation of the notion of concurrency that led Pratt to propose a semantics that accounts not only for applicable and executed transitions, but also for transitions in progress and preempted transitions [72, 73]. This model borrows concepts from linear logic and extends them within category theory. An interesting byproduct of this interpretation is the postulation of a duality between states and events (transitions), which can be understood as a duality between information and time [73].

# 7 A Logical Bridge to Process Algebra

As we mentioned earlier, formalisms such as the $\pi$-calculus [63] 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 $\omega$ is closely related to three such process algebras: the asynchronous $\pi$-calculus [63, 74] and its propositional variant, which we introduced in Section 4, and the join calculus [35]. We will show that a simple execution-preserving translation maps process constructors to rewrite operators in $\omega$, while structural equivalence map are captured by the structure of states in the rewrite language. As we do so, we will focus on process algebras as computation rather than analysis mechanisms. In particular, we will concentrate on a trace-based semantics, leaving the investigation of finer notions, such as bisimulation, for future work.

## 7.1 Propositional Process Algebra

In Section 4.1 we introduced the language $a\pi_0$, a propositional variant of the asynchronous $\pi$-calculus [63, 74], devised a simple encoding to linear logic, and showed that its operational semantics was captured by derivability in the target fragment of the logic, which we called $LL^{a\pi_0}$. Processes $P$ in $a\pi_0$ were freely generated from $\|$ (parallel composition), $\mathbf{0}$ (the inert process), ! (process replication) as well as name prefixing $xP$ and isolated co-names $\overline{x}$. A notion of structural equivalence, written $P \overset{\pi}{\equiv} Q$, gave processes a commutative monoidal structure with respect to $\|$ and $\mathbf{0}$, and the reduction semantics allowed a co-name to expose a process prefixed by the corresponding name and let $!P$ to spawn copies of $P$. We denoted it as $P \rightharpoonup^* Q$. See Section 4.1 for the details.

The encoding $\ulcorner \_ \urcorner$ of $a\pi_0$ in linear logic mapped homomorphically $\|$, $\mathbf{0}$ and ! to $\otimes$, $\mathbf{1}$ and ! respectively, it associated a co-name $\overline{x}$ to the propositional constant $x$, and turned a named prefix $xP$ into the linear implication $x \multimap \ulcorner P \urcorner$. This encoding identified a fragment $LL^{a\pi_0}$ of linear logic on which structural equivalence coincided with logical equivalence, and iterated reduction with derivability. Once more, see Section 4.1 for details.

As we interprete $a\pi_0$ in $\omega$, we retain the encoding $\ulcorner \_ \urcorner$ unchanged, modulo the identification of tensorial formulas and linear contexts. This has an important implications: while in $\mathrm{LV}^{\mathrm{obs}}$ structural equivalence needed to be modeled by by $\equiv_{\otimes}$-equivalence (whose soundness and completeness are given by Lemmas 4.1 and 4.2), they are mapped simply to the monoidal structure of the linear context in $\omega$, as shown in Section 5.3. Therefore, whenever structural equivalence is needed in an $a\pi_0$ reduction sequence, it can be emulated implicitly in $\omega$.

The soundness of the encoding is expressed by the following property, which corresponds to Property 4.3 in Section 4.1. Note that, because the end-state of $\omega$ has more structure than the goal formula of $\mathrm{LV}^{\mathrm{obs}}$, this statement is more precise than Property 4.3.

**Property 7.1** *Given processes $P$ and $Q$, let $\Sigma_P$ be the set of all names in $P$. If $P \rightharpoonup^* Q$, then there exist contexts $\Gamma$ and $\Delta$ and a process $Q'$ such that $\Sigma_P; \circ; \ulcorner P \urcorner \Rrightarrow^* \Sigma_P; \Gamma; \Delta$ where $Q \overset{\pi}{\equiv} Q'$ and $\ulcorner Q' \urcorner = (!\Gamma, \Delta)$.*

**Proof.** This statement is proved by a straightforward induction on the given reduction sequence. Going through $\mathrm{LV}^{\mathrm{obs}}$ by first appealing to Property 4.3 and then porting the result to $\omega$ using the first part of Corollary 5.8 is ineffective because of the imprecision of Property 4.3. $\square$

A corresponding completeness result holds with respect to the syntactic fragment $\omega^{a\pi_0}$ of $\omega$ identified by the encoding $\ulcorner\_\urcorner$. This is analogous to, but again more precise than, Property 4.5 from Section 4.1.

**Property 7.2** *Let $P$ be a process, $\Sigma_P$ be the set of all names in $P$, and $\Gamma$ and $\Delta$ be contexts. If $\Sigma_P; \circ; \ulcorner P \urcorner \;\Rightarrow^*\; \Sigma_P; \Gamma; \Delta$ holds in $\omega$, then there exists a process $Q$ such that $P \rightharpoonup^* Q$ and $\ulcorner Q \urcorner = (!\Gamma, \Delta)$.*

**Proof.** Similarly to the proof of Property 6.2 in Section 6.1, we shall reorder applications of the $\omega$-rules in Figure 5 (similarly to what happened in the proof of Property 7.2) before factoring them out as reductions in $a\pi_0$. $\square$

## 7.2 The Asynchronous $\pi$-Calculus

The definition of the asynchronous $\pi$-calculus, $a\pi_1$, introduced in Section 4.2 extends the propositional case by interpreting names and co-names as communication channels and using them for processes to exchange messages. A co-name process $\overline{x}$ becomes $\overline{x}\langle y\rangle$ where $y$ represents some message sent over channel $x$, and the name-prefixed process $xP$ now assumes the form $x(y)P$, where $y$ is a variables to which a message sent over $x$ will be bound to and possibly used inside $P$. For simplicity, we identified channels and messages as names, although this idea can be considerably refined [74]. Finally, we included the hiding operator, $\nu x.P$, which binds the name $x$ within $P$. This lead to extending the notion of structural equivalence with properties of $\nu$. The reduction semantics was altered only to the point of modeling reduction and allowing them to take place in the scope of the hiding operator.

The representation of this language in linear logic extended the propositional encoding $\ulcorner\_\urcorner$ reviewed in Section 7.1 by reserving a binary predicate symbol $\mathsf{c}$ and use it as a universal channel when representing input and output: $\ulcorner \overline{x}\langle y\rangle \urcorner = \mathsf{c}(x,y)$ and $\ulcorner x(y)P \urcorner = \forall y.\, \mathsf{c}(x,y) \multimap \ulcorner P \urcorner$. Moreover, it modeled $\nu$ as $\exists$. With this encoding, structural equivalence in $a\pi_1$ corresponded to logical equivalence in $\mathrm{LV}^{\mathrm{obs}}$ and reduction to derivability. This is formalized in Properties 4.7 in Section 4.2.

The interpretation of $a\pi_1$ in $\omega$ retains this logical encoding, but once more the identification of tensorial formulas and linear contexts implies that $\|$ and $\mathbf{0}$ are effectively mapped to "," and "$\cdot$" respectively. The computational meaning of $a\pi_1$'s structural equivalence relation is captured by Lemmas 5.11 and 5.12 in Section 5.3, which say that $\equiv_{\otimes\exists}$-equivalent $\omega$-multisets eventually yield similar states, and we already know by Property 4.6 that $\equiv_{\otimes\exists}$ and $\stackrel{\pi}{\equiv}$ are isomorphic relative to $\ulcorner\_\urcorner$ over $LL^{a\pi_1}$ (and therefore $\omega^{a\pi_1}$).

We now extend the propositional soundness and completeness results for reduction obtained in Section 7.1 to $a\pi_1$ relative to the syntactic fragment $\omega_{a\pi_1}$ of $\omega$ in the image of the encoding $\ulcorner\_\urcorner$. Both results are presented together in the following property, which corresponds to Properties 4.7 in Section 4.2. Note again that the structure of states in $\omega$ allows more precise and concise statements.

**Property 7.3** *Let $P$ be a process and $\Sigma_P = \mathsf{c}„ \mathrm{FN}(P)$.*

- *For any process $Q$ such that $P \rightharpoonup^* Q$, there exist a signature $\Sigma$, contexts $\Gamma$ and $\Delta$ and a process $Q'$ such that $\Sigma_P; \circ; \ulcorner P \urcorner \;\Rightarrow^*\; (\Sigma_P„ \Sigma); \Gamma; \Delta$ where $Q \stackrel{\pi}{\equiv} Q'$ and $\ulcorner Q' \urcorner = \exists\Sigma.\,(!\Gamma, \Delta)$.*
- *For any signature $\Sigma$ and contexts $\Gamma$ and $\Delta$, if $\Sigma_P; \circ; \ulcorner P \urcorner \;\Rightarrow^*\; (\Sigma_P„ \Sigma); \Gamma; \Delta$, then there exists a process $Q$ such that $P \rightharpoonup^* Q$ and $\ulcorner Q \urcorner = \exists\Sigma.\,(!\Gamma, \Delta)$.*

**Proof.** Each of these two proofs follow a strategy that is similar to the propositional cases in Section 7.1, with the minor complication of dealing with the quantifiers. In particular, the first part proceeds by a simple induction on the given reduction sequence, while the second requires performing dependency-preserving permutations of $\omega$ rewrites to cluster them into groups that correspond to the reduction rules (mainly $\mathbf{red\_i/o}$). $\square$

In Section 8, we will briefly consider a language $a\pi_1^+$ that extends $a\pi_1$ with terms over some signature $\Sigma$, polyadic channels, and pattern matching, but does never hide names used as channels. Because of this last aspect, the easy extension of $\ulcorner\_\urcorner$ to $a\pi_1^+$ does not need to rely on the auxiliary symbol $\mathsf{c}$. A strong version of Property 7.3 is valid for this language, so that it can be seen as a fragment $\omega^{a\pi_1^+}$ of $\omega$ (and therefore of linear logic).

### 7.3 The Join Calculus

The asynchronous core of the join calculus is defined by the following grammar [35]:

$$\begin{array}{lrcl} \textit{Processes} & P, Q, R & ::= & \mathbf{0} \mid P \parallel Q \mid \mathsf{def}\ D\ \mathsf{in}\ P \mid x\langle\vec{y}\rangle \\ \textit{Definitions} & D, E & ::= & J \triangleright P \mid D \wedge E \mid \top \\ \textit{Join patterns} & J, I & ::= & J \parallel I \mid x\langle\vec{y}\rangle \end{array}$$

*Processes* $P$ consist of the parallel composition of messages over polyadic channels $x$ ($x\langle\vec{y}\rangle$) and definitions ($\mathsf{def}\ D\ \mathsf{in}\ P$). A *definition* $D$ is a collection of *rules* ($J \triangleright P$) where each *join pattern* $J$ is given by one or more messages patterns (also $x\langle\vec{y}\rangle$). The name tuples $\vec{y}_D$ in the pattern of a definition $D = J \triangleright P$ are bound in $P$, while the channel names $\vec{x}_D$ are defined. A definition $\mathsf{def}\ J_1 \triangleright P_1 \wedge \ldots \wedge J_n \triangleright P_n\ \mathsf{in}\ Q$ binds all channel names $\vec{x}_{J_i}$ in each $P_j$ and $Q$. Bound names are subject to implicit $\alpha$-conversion. We write $\mathrm{FN}(P)$ for the free names of a process $P$ (and similarly definitions), and $[\vec{z}/\vec{y}]P$ for the simultaneous capture-avoiding substitution of names $\vec{z}$ for $\vec{y}$ in process $P$.

The join calculus defines a structural congruence, written $\equiv_j$, which specifies that processes (resp. rules) form a monoid with operation $\parallel$ (resp. $\wedge$) and unit $\mathbf{0}$ (resp. $\top$). It moreover comprises the following equivalences for definitions:

$$\boxed{\begin{array}{rcl} \mathsf{def}\ \top\ \mathsf{in}\ P & \equiv_j & P \\ (\mathsf{def}\ D\ \mathsf{in}\ P) \parallel Q & \equiv_j & \mathsf{def}\ D\ \mathsf{in}\ (P \parallel Q) \\ & & \text{if } \vec{x}_D \cap \mathrm{FN}(Q) = \emptyset \\ \mathsf{def}\ D\ \mathsf{in}\ (\mathsf{def}\ E\ \mathsf{in}\ P) & \equiv_j & \mathsf{def}\ (D \wedge E)\ \mathsf{in}\ P \\ & & \text{if } \vec{x}_E \cap \mathrm{FN}(D) = \emptyset \end{array}}$$

A process can always be $\equiv_j$-converted to the canonical form $\mathsf{def}\ D\ \mathsf{in}\ P$, where $P$ does not contain definitions.

The operational semantics of the join calculus is expressed by the judgment $P \blacktriangleright Q$ given by the following rule, up to $\equiv_j$:

$$\begin{array}{rl} & \mathsf{def}\ (J \triangleright P) \wedge D\ \mathsf{in}\ ([\vec{z}/\vec{y}_J]J \parallel Q) \\ \blacktriangleright & \mathsf{def}\ (J \triangleright P) \wedge D\ \mathsf{in}\ ([\vec{z}/\vec{y}_J]P \parallel Q) \end{array}$$

That is, whenever an instance $[\vec{z}/\vec{y}_J]J$ of the join pattern $J$ of a rule $J \triangleright P$ appears the body of a canonical process, it can be replaced with the corresponding instance $[\vec{z}/\vec{y}_J]P$ of the rule's right-hand side $P$. Expectedly, we write $\_ \blacktriangleright^* \_$ for the reflexive and transitive closure of $\_ \blacktriangleright \_$.

We define a mapping of the various syntactic classes of the join calculus into $\omega$. As usual, we write it $\ulcorner\_\urcorner$, overloading this notation for processes, rules and patterns. This mapping, which is spelled out below, homomorphically maps the monoids of the join calculus to the tensorial core of $\omega$. Similarly to the $\pi$-calculus, messages and patterns are rendered with the help of a family of auxiliary symbols $\vec{\mathsf{c}}$ of increasing arity (to accommodate the names $\vec{y}$ in $x\langle\vec{y}\rangle$). We rely on $\omega$'s universal quantifier to govern the bound variables $\vec{y}_J$ of a rule $J \triangleright P$, while $\exists$ is needed to bind the variables $\vec{x}_D$ defined in a definition. The transition potential of rules is captured by means of linear implication, while their reusability is naturally expressed using !. Altogether, we have the following definition for $\ulcorner\_\urcorner$:

$$\begin{array}{lrcl} P: & \ulcorner\mathbf{0}\urcorner & = & \cdot \\ & \ulcorner P \parallel Q\urcorner & = & \ulcorner P\urcorner, \ulcorner Q\urcorner \\ & \ulcorner\mathsf{def}\ D\ \mathsf{in}\ P\urcorner & = & \exists\vec{x}_D.\,(\ulcorner D\urcorner, \ulcorner P\urcorner) \\ & \ulcorner x\langle\vec{y}\rangle\urcorner & = & \mathsf{c}(x, \vec{y}) \\ D: & \ulcorner J \triangleright P\urcorner & = & !\forall\vec{y}_J.\,(\ulcorner J\urcorner \multimap \ulcorner P\urcorner) \\ & \ulcorner D \wedge E\urcorner & = & \ulcorner D\urcorner, \ulcorner E\urcorner \\ & \ulcorner\top\urcorner & = & \cdot \\ J: & \ulcorner J \parallel I\urcorner & = & \ulcorner J\urcorner, \ulcorner I\urcorner \\ & \ulcorner x\langle\vec{y}\rangle\urcorner & = & \mathsf{c}(x, \vec{y}) \end{array}$$

As in previous cases, this encoding is invertible, so that every formula $A$ in its image identifies an object $X$ of the appropriate syntactic category in the join calculus. We write $\omega^J$ for the fragment of $\omega$ in the image of $\ulcorner\_\urcorner$.

The encoding we just defined is next shown to preserve the operational semantics of the join calculus. This is formalized in the following property.

**Property 7.4** *Let $P$ be a process and $\Sigma_P = \vec{c}_{\,,,}\mathrm{FN}(P)$. Then, $P \blacktriangleright^* Q$ if and only if there exist a signature $\Sigma$, contexts $\Gamma$ and $\Delta$ and a process $Q'$ such that $\Sigma_P; \circ; \ulcorner P \urcorner \Rightarrow^* (\Sigma_{P\,,,}\Sigma); \Gamma; \Delta$ where $\ulcorner Q' \urcorner = \exists \Sigma. (!\Gamma, \Delta)$ and $Q' \equiv_j Q$.*

**Proof.** This proof proceeds along the lines of the proof of Property 7.3 in Section 7.2. □

Once more, this result allows interpreting the language under examination as a fragment $\omega^J$ of $\omega$, and therefore of linear logic.

## 7.4 Discussion

Once more, our encoding of $a\pi_1$ makes use of a fraction of the syntax of $\omega$. In particular, $\top$ and $\&$ are not used at all and at most one object appears in the antecedent of $\multimap$. It is natural to interpret $\&$ as a form of non-deterministic choice. Its semantics in $\omega$ is different, however, from the choice operator, $+$, found in the synchronous $\pi$-calculus [63], as the reaction rule of the latter realizes both choice and communication in the same step [73]. As noted in [24], its emulation with $\&$ would be sound, but in general incomplete as intermediate stages are visible in $\omega$.

The $\pi$-calculus, like many process algebras, does not allow multiple concurrent communications to occur atomically: for example nothing prevents $xyP$ to reducing into $yP$ when run in parallel with a process that provides $\overline{x}$ but not $\overline{y}$. If we wanted $xyP$ to reduce to $P$ if both $\overline{x}$ and $\overline{y}$ are present, and stay put otherwise, something we may write, $\{x, y\}P$, we would need to introduce a complex mechanism of transactions. By contrast, multiset rewriting supports atomic transitions triggered by the presence of an arbitrary number of multiset elements. The operational behavior of the join calculus [35] natively provides a similarly adaptable notion of atomicity. The same holds true of $\omega$: the expression $x \multimap (y \multimap \ulcorner P \urcorner)$ has the same sequentializing semantics as $xyP$ in the $\pi$-calculus, while $(x, y) \multimap \ulcorner P \urcorner$ atomically rewrites $x$ and $y$ into $\ulcorner P \urcorner$, as can be achieved in multiset rewriting or in the join calculus.

Several authors have taken to the task of giving logical interpretations to process algebras, with particular focus on the $\pi$-calculus. Operationally sound and complete CLF encodings of both the synchronous and asynchronous versions of this language are given in [24]. A propositional fragment of the $\pi$-calculus is instead analyzed in [57]. That paper attempts a logical account of a form of testing equivalence. The adaptation to $\omega$ of classical notions of inter-process equivalence goes beyond the scope of the present work, but will be particularly interesting to undertake as future work.

# 8 Specifying Security Protocols

With the recent surge of interest in security protocols, numerous languages have been adapted or invented for the purpose of specifying and reasoning about these subtle distributed algorithms. With a few exceptions, these languages tend to be either process-oriented or state-based. The former include the spi-calculus [2], a security-enhanced version of the $\pi$-calculus, strand spaces [30], and others such as [26]. The latter comprises formalisms directly based on multiset rewriting [19, 22], tool-specific languages [55], inductive definitions [65], colored Petri nets [5, 14], and more.

This profusion of formalisms has triggered an intense research activity intent to comparing and bridging them [15, 21, 23, 26]. In spite of clear commonalities, these mappings are very specific to the languages they consider, and therefore somewhat ad-hoc and hardly reusable. With a foot in both the state- and process-based camp and easy embeddings, we highlight a security-conscious version of $\omega$ as a reusable and logically motivated intermediate language for carrying on these investigations. This language, that we call MSR 3, is itself a promising formalism for the specification of cryptographic protocols, precisely because it supports both representation paradigms, and can combine them when convenient. The following discussion should be taken as a taste of MSR 3, as we will discuss the details of this language in a future publication.

## 8.1 A Preview of MSR 3

In order to represent security protocols, we consider an initial signature $\Sigma_s$ that makes available function symbols $\{\_\}_{\_}$ and $[\_,\_]$ to symbolically express encryption and concatenation (for succinctness, we will often omit the brackets in the latter). Other cryptographic operations can be included as needed. We also require $\Sigma_s$ to provide predicate symbols $\mathsf{N}(\_)$ and $\mathsf{I}(\_)$ to represent network messages in transit and intruder knowledge. Other predicates, for example to hold values local for a principal, can also appear in $\Sigma_s$. Different forms of data can be distinguished through typing, although we will refrain from doing so here for the sake of brevity.

The language MSR 1 [22] adopts such a signature in a first-order multiset rewriting framework of the sort analyzed in Section 6.2. It is therefore a fragment of $\omega$. In this section, we will use $\omega$ itself as a language for specifying protocols.

As often done, we will use the Needham-Schroeder public-key protocol [64] as an example. This protocol, informally described below, has the purpose of establishing communication between an initiator $A$ and a responder $B$, and authenticating $A$ to $B$.

$$\begin{array}{ccccl} A & \rightarrow & B & : & \{A, n_A\}_{k_B} \\ B & \rightarrow & A & : & \{n_A, n_B\}_{k_A} \\ A & \rightarrow & B & : & \{n_B\}_{k_B} \end{array}$$

Here, $A$ creates a fresh value (nonce) $n_A$ and sends it together with her name to $B$, encrypted with $B$'s public key $k_B$. Upon successfully decrypting this message, $B$ creates his own nonce $n_B$ and sends it to $A$ together with $n_A$, both encrypted with $A$'s public key $k_A$. Upon recognizing $n_A$ as her original nonce, $A$ sends $n_B$ encrypted back to $B$ as an acknowledgment.

We will now express the initiator's part of this protocol in $\omega$. We are immediately faced with the choice of which representation paradigm to use. We give both a state-based and a process-based specification. For the sake of brevity, we do not explicitly represent administrative tasks such as a principal accessing his or his interlocutor's keys (see [22]): this will allow us to concentrate on the overall structure of the specification rather than these details.

The state-based representation of the initiator role of this protocol is expressed by the following two rules:

$$\boxed{\begin{array}{l} \forall A.\, \forall k_B. \\ \qquad \mathbf{1} \multimap \exists n_A.\, \mathsf{N}(\{A, n_A\}_{k_B}), \mathsf{L}(A, n_A, k_B) \end{array}}$$
$$\boxed{\begin{array}{l} \forall A.\, \forall k_B.\, \forall k_A.\, \forall n_A.\, \forall n_B. \\ \qquad \mathsf{N}(\{n_A, n_B\}_{k_A}), \mathsf{L}(A, n_A, k_B) \multimap \mathsf{N}(\{n_B\}_{k_B}) \end{array}}$$

The first captures the initial step of the protocol, while the second expresses the rest from the initiator's point of view. In order to ensure that these rules are executed in the proper order, they rely on the auxiliary predicate $\mathsf{L}$, which has also the task of communicating the parameters of the execution to the second rule (in particular the value of $n_A$). This encoding resembles very closely the specification of this protocol in MSR 1 [22] and other state-based formalisms.

The process-based representation of this role does away with the auxiliary predicate $\mathsf{L}$ altogether in favor of a nested implication:

$$\boxed{\begin{array}{l} \forall A.\, \forall k_B. \\ \qquad \mathbf{1} \multimap \exists n_A.\, \mathsf{N}(\{A, n_A\}_{k_B}), \\ \qquad\qquad\qquad \boxed{\begin{array}{l} \forall n_B.\, \forall k_A. \\ \qquad \mathsf{N}(\{n_A, n_B\}_{k_A}) \multimap \mathsf{N}(\{n_B\}_{k_B}) \end{array}} \end{array}}$$

This closely resembles the description of this role in a process-based language such as strand spaces [30] or the spi-calculus [2]. Observe the nested vs. cascaded nature of the specification. Miller has shown that, given some constraints on $\mathsf{L}$, these two specifications are logically equivalent [59] (although not in the sense of $\equiv$).

Differently from all other protocol specification languages we are aware of, $\omega$ makes both styles available when expressing a protocol. Not only can the specifier choose which one is most appropriate to the task at hand, but she can mix and match them at her leisure. Indeed, the initiator and receiver roles are not even required to use the same paradigm, so that if our first specification is used, the receiver could seamlessly be process-based. This may be useful, for example, when analyzing client-server protocols for denial-of-service vulnerabilities where one may want to use the more succinct process-oriented form for the client, but a state-based representation for the server in order to clearly account for how much data is stored (in the auxiliary predicate L) between exchanges. A mixed representation may also be beneficial when representing the intruder capabilites, as a process-based encoding tends to over-sequentialize the specification [15]. We expect these benefits to grow with the size and complexity of the protocol at hand.

MSR 1 has been extended with a powerful type system into MSR 2 [19]. We similarly define the language MSR 3 as the corresponding strongly typed version of $\omega$. A precise description of MSR 3 goes beyond the scope of this paper.

### 8.2 Discussion

The coexistence of both the state- and process-based paradigm in $\omega$ makes it a useful melting pot, not just as a specification tool, but also as an intermediate language when comparing different formalisms. Indeed, it is well known that the terrain between the two paradigms is bumpy and treacherous [13, 15, 23], and any new road shall reckon with these difficulties. System $\omega$ suggests a different approach: engineer a robust translation between the state- and the process-based fragments of this language, and use it as a fast expressway to relate them. Other languages can then be mapped to the closest fragment of $\omega$ by what would be neighborhood roads in our analogy.

To be more precise, we define an execution preserving embedding of all of $\omega$ in $\omega^{MSR_1}$, which we identified as the counterpart of first-order multiset rewriting, a quintessential state-based language. This encoding is rather simple and well-behaved. Space limitation prevent us from presenting it, and we shall refer the interested reader to [15], which gives a similar translation.

What fragment of $\omega$ (or what process algebra) best captures the process-based paradigm is open to discussion. Were we to take $\omega^{a\pi_1^+}$, we would similarly map $\omega$ to this sublanguage. See again [15] for details. This direction is not as easy, and it is not clear whether a fully satisfactory solution exists.

Now, in order to relate, say, strand spaces [30] and Paulson inductive encoding [65], it suffices to produce a shallow encoding of the former into $\omega^{a\pi_1^+}$, a similarly simple translation of the latter to $\omega^{a\pi_1}$, and then use the two internal translations we just sketched to bridge them.

## 9 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 an observation rule, and extending our focus beyond finite derivations. The resulting language, which we called system $\omega$, is a flexible specification formalism for concurrent systems: at any point the state of the execution corresponds to the left-hand side of a linear sequent, with atomic formulas representing shared state or messages in transit, and composite formulas standing for concurrent processes; the next state is obtained by applying a left rule bottom-up, and therefore guided by the connectives and quantifiers appearing in some formula of the current state (in a fashion that is dual but otherwise not dissimilar to abstract logic programming [61]); such a transition sequence is potentially unbounded, which allows modeling infinite systems and corresponds to the construction of a potentially infinite "proof"; but it can be interrupted at any point by closing the derivation with the observation rule, which implements a notion of observation as a finite approximation of a possibly infinite computation.

Specifically, $\omega$ 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 $\omega$'s strong ties to process algebra, with simple execution-preserving embeddings of the join calculus and a computational variant of asynchronous $\pi$-calculus. We suggested relying on $\omega$'s position as a logical meeting point of multiset rewriting and process algebra for the purpose of expressing and reasoning about cryptographic protocols, an application area where both types of formalisms have been used, often in complementary ways. By being able to handle state-based and process-based components in the same specification, $\omega$ has the potential of overcoming the current state versus action dichotomy, which has been identified as a major hindrance, for example, in model checking.

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 $\omega$ to evolve as more questions about its logical foundations are answered (see Section 5.4). Pursuing the relation with process algebraic languages is particularly interesting in light of the results in Section 7 and the application potential of $\omega$ in the sphere of security protocol specification (Section 8).

## Acknowledgments

We are grateful to Frank Pfenning, Dale Miller, Mark-Oliver Stehr, Valeria de Paiva, Gerald Allwein, Steve Zdancevic, Vijay Saraswat and Vaughan Pratt for their comments on various aspects of this work. We are also indebted to the anonymous reviewers for their careful reading and cunning suggestions.

## References

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

[2] M. Abadi and A. Gordon. A calculus for cryptographic protocols: the spi calculus. *Information and Computation*, 148(1):1–70, 1999.

[3] Samson Abramsky. Proofs as processes. *Theoretical Computer Science*, 135:5–9, 1994.

[4] Samson Abramsky. Process realizability. In F.L. Bauer and R. Steinbruggen, editors, *Proc. 1999 Marktoberdorf Summer School*, pages 167–180. IOS Press, 2000.

[5] Salah Aly and Khaled Mustafa. Protocol verification and analysis using Colored Petri Nets. Technical Report 04-003, DePaul University, Chicago, IL, 2004. `http://facweb.cs.depaul.edu/research/TechReports/TR04-003.pdf`.

[6] Jean-Marc Andreoli. Focussing proof-net construction as a middleware paradigm. In A. Voronkov, editor, *Proc.CADE–18*, pages 501–516, Copenhagen, Denmark, 2002. Springer Verlag LNAI 2392.

[7] Jean-Marc Andreoli and Remo Pareschi. Linear objects: Logical processes with built-in inheritance. *New Generation Computing*, 9:445–473, 1991.

[8] Andrea Asperti. A logic for concurrency. Technical report, Computer Science Department, University of Pisa, 1987.

[9] Jean-Pierre Banâtre and Daniel Le Métayer. Programming by multiset transformation. *Communications of the ACM*, 36(1):98–111, 1993.

[10] Andrew Barber. Dual intuitionistic linear logic. Technical Report ECS-LFCS-96-347, Laboratory for Foundations of Computer Sciences, University of Edinburgh, 1996.

[11] G. Bellin and P. J. Scott. On the $\pi$-calculus and linear logic. *Theoretical Computer Science*, 135:11–65, 1994.

[12] N. Benton, 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)*, pages 61–84, 1993.

[13] G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96(1):217–248, 1992.

[14] Jonathan Billington, Guy Edward Gallasch, and Bing Han. A coloured Petri net approach to protocol verification. *Lectures on Concurrency and Petri Nets: Advances in Petri Nets*.

[15] Stefano Bistarelli, Iliano Cervesato, Gabriele Lenzini, and Fabio Martinelli. Relating Multiset Rewriting and Process Algebras for Security Protocol Analysis. *Journal of Computer Security*, 13(1):3–47, 2005.

[16] Torben Braüner and Valeria de Paiva. Cut-elimination for full intuitionistic linear logic. Technical Report RS-96-10, BRICS, Denmark, 1996.

[17] C. Brown and D. Gurr. A categorical linear framework for Petri nets. In *Proc. LICS'90*, pages 208–218, Philadelphia, PA, 1990. IEEE Computer Society Press.

[18] Luca Cardelli and Andrew D. Gordon. Logical Properties of Name Restriction. In S. Abramsky, editor, *Proceedings of the 5th International Conference on Typed Lambda Calculi and applications — TCLA'01*, pages 46–60, Krakow, Poland, 2001. Springer-Verlag LNCS 2044.

[19] I. Cervesato. Typed MSR: Syntax and examples. In *1st International Workshop on Mathematical Methods, Models and Architectures for Computer Networks Security — MMM'01*, pages 159–177. Springer-Verlag LNCS 2052, 2001.

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

[21] Iliano Cervesato, Nancy Durgin, Max Kanovich, and Andre 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.

[22] Iliano Cervesato, Nancy Durgin, Patrick Lincoln, John Mitchell, and Andre Scedrov. A meta-notation for protocol analysis. In *Proc. CSFW'99*, pages 55–69, Mordano, Italy, 1999. IEEE Computer Society Press.

[23] Iliano Cervesato, Nancy A. Durgin, Patrick D. Lincoln, John C. Mitchell, and Andre Scedrov. Relating strands and multiset rewriting for security protocol analysis. In *13th IEEE Computer Security Foundations Workshop — CSFW'00*, pages 35–51, Cambrige, UK, 2000. IEEE Computer Society Press.

[24] Iliano Cervesato, Frank Pfenning, David Walker, and Kevin Watkins. A concurrent logical framework II: Examples and applications. Technical Report CMU-CS-02-102, Computer Science Department, Carnegie Mellon University, 2002.

[25] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and José F. Quesada. Maude: Specification and programming in rewriting logic. *Theoretical Computer Science*, 2001.

[26] F. Crazzolara and G. Winskel. Events in security protocols. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 96–105, Philadelphia, PA, 2001. ACM Press.

[27] Nancy Durgin, Patrick Lincoln, John Mitchell, and Andre Scedrov. Multiset rewriting and the complexity of bounded security protocols. *Journal of Computer Security*, 12(2):247–311, 2004. 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.

[28] U. Engberg and G. Winskel. Petri nets as models of linear logic. In A. Arnold, editor, *15th Colloquium on Trees in Algebra and Programming*, pages 147–161, Copenhagen, Denmark, 1990. Springer-Verlag LNCS 431.

[29] Joost Engelfriet and Tjalling Gelsema. A new natural structural congruence in the pi-calculus with replication. *Acta Informatica*, 40(6–7):385–430, 2004.

[30] Thayer Fábrega, Jonathan Herzog, and Joshua Guttman. Strand spaces: Why is a security protocol correct? In *Proceedings of the 1998 IEEE Symposium on Security and Privacy*, pages 160–171, Oakland, CA, May 1998. IEEE Computer Society Press.

[31] François Fages, Paul Ruet, and Sylvain Soliman. Phase semantics and verification of concurrent constraint programs. In *Proceedings of the 13th IEEE Colloquium on Logic in Computer Science — LICS'98*, pages 141–152, Indianapolis, 1998. IEEE Computer Society.

[32] Berndt Farwer. Towards Linear Logic Petri Nets — From P/T-Nets to Object Systems. Technical Report FBI-HH-B-211/98, Universität Hamburg, Fachbereich Informatik, 1998.

[33] Berndt Farwer. A linear logic view of object Petri nets. *Fundamenta Informaticae*, 37(3):225–246, 1999.

[34] Berndt Farwer and Kundan Misra. Dynamic modification of system structures using LLPNs. In *Proc. 5th Conference on Perspectives of System Informatics*, pages 177–190, Novosibirsk, Russia, 2003.

[35] Cédric Fournet and Georges Gonthier. The reflexive CHAM and the join-calculus. In *Proc. POPL'96*, pages 372–385. ACM Press, 1996.

[36] Murdoch J. Gabbay and Andrew M. Pitts. A new approach to abstract syntax involving binders. In *14th Annual Symposium on Logic in Computer Science*, pages 214–224. IEEE Computer Society Press, Washington, 1999.

[37] Didier Galmiche and Guy Perrier. On proof normalisation in linear logic. *Theoretical Computer Science*, 135(1):67–110, 1994. Also available as Technical Report CRIN 94-R-113 from the Centre di Recherche en Informatique de Nancy.

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

[39] Carl Gunter and Vijay Gehlot. Nets as tensor theories. In *10th International Conference on Application and Theory of Petri Nets*, pages 174–191, Bonn, Germany, 1989.

[40] Carl Gunter and Vijay Gehlot. A proof-theoretic semantics for true concurrency. Preliminary report, University of Pennsylvania, 1989.

[41] James Harland and David Pym. The uniform proof-theoretic foundation of linear logic programming (extended abstract). In V. Saraswat and K. Ueda, editors, *International Symposium on Logic Programming — ISLP'91*, pages 304–318, San Diego, California, 1991.

[42] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[43] Joshua Hodas and Dale Miller. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation*, 110(2):327–365, 1994.

[44] Kurt Jensen. Coloured Petri nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Advances in Petri Nets*, pages 248–299. Springer-Verlag LNCS 254, 1986.

[45] Max Kanovich. Horn programming in linear logic is NP-complete. In *Proceedings of the Seventh Annual IEEE Synposium on Logic in Computer Science — LICS'92*, pages 200–210. IEEE Computer Society Press, 1992.

[46] Max Kanovich. The complexity of Horn fragments of linear logic. *Annals of Pure and Applied Logic*, 69:195–241, 1994.

[47] Max I. Kanovich. Linear logic as a logic of computation. *Annals of Pure and Applied Logic*, 67(1–3):183–212, 1994.

[48] Naoki Kobayashi and Akinori Yonezawa. ACL — A concurrent linear logic programming paradigm. In D. Miller, editor, *Proc. ILPS'03*, pages 279–294, Vancouver, Canada, 1993. MIT Press.

[49] Daniel Le Métayer. Higher-order multiset programming. In *Proc. DIMACS workshop on specifications of parallel algorithms*. American Mathematical Society, DIMACS series in Discrete Mathematics, Vol. 18, 1994.

[50] Patrick Lincoln and Vijay Saraswat. Higher-order, linear, concurrent constraint programming. Manuscript, 1993.

[51] Patrick Lincoln and Andre Scedrov. First Order Linear Logic Without Modalities is NEXPTIME-Hard. *Theoretical Computer Science*, pages 139–154, 1994.

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

[53] Narciso Martí-Oliet and José Meseguer. From Petri nets to linear logic. In D.H. Pitt, D.E. Rydeheard, P. Dybier, A.M. Pitt, and A. Poigné, editors, *Category Theory and Computer Science*, pages 313–340, Manchester, UK, 1989. Springer-Verlag LNCS 389.

[54] Raymond McDowell, Dale Miller, and Catuscia Palamidessi. Encoding transition systems in sequent calculus. *Theoretical Computer Science*, 294(3):411–437, 2003.

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

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

[57] Dale Miller. The $\pi$-calculus as a theory in linear logic: Preliminary results. In E. Lamma and P. Mello, editors, *Proc. ELP*, pages 242–265. Springer-Verlag LNCS 660, 1992.

[58] Dale Miller. A multiple-conclusion specification logic. *Theoretical Computer Science*, 165(1):201–232, 1996.

[59] Dale Miller. Encryption as an abstract data-type: An extended abstract. In Iliano Cervesato, editor, *Proc. FCS*, pages 3–14, Ottawa, Canada, 2003.

[60] Dale Miller. 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.

[61] Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

[62] Dale Miller and Alwen Tiu. A proof theory for generic judgments. *ACM Transactions on Computational Logic*, 6(4):749–783, 2005.

[63] Robin Milner. *Communicating and Mobile Systems: the $\pi$-Calculus*. Cambridge University Press, 1999.

[64] Roger Needham and Michael Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[65] Lawrence C. Paulson. Proving properties of security protocols by induction. In *Proc. CSFW'97*. IEEE Computer Society Press, 1997.

[66] Guy Perrier. *De la construction de preuves à la programmation parallèle en logique linéaire*. PhD thesis, Université Henri Poincaré, Nancy, France, 1995.

[67] Carl Adam Petri. *Kommunikation mit Automaten.* Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962.

[68] Carl Adam Petri. Fundamentals of a theory of asynchronous information flow. In *Proc. IFIP*, pages 386–390, Amsterdam, 1963. North Holland Publ. Comp.

[69] Frank Pfenning. Structural cut elimination. In D. Kozen, editor, *Proc. LICS'95*, pages 156–166, San Diego, CA, 1995. IEEE Computer Society Press.

[70] Benjamin C. Pierce and David N. Turner. Pict: A programming language based on the pi-calculus. In Gordon Plotkin, Colin Stirling, and Mads Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*, pages 455–494. MIT Press, 2000.

[71] Andrew M. Pitts and Murdoch J. Gabbay. A metalanguage for programming with bound names modulo renaming. In *Proceedings of the 5th International Conference on Mathematics of Program Construction — MPC'2000*.

[72] Vaughan R. Pratt. Event-state duality: the enriched case. In *Proceedings of CONCUR'02*, pages 41–56, Brno, Czech Republic, 2002. Springer-Verlag LNCS 2421.

[73] Vaughan R. Pratt. Transition and cancellation in concurrency and branching time. *Mathematical Structures in Computer Science, special issue on the difference between sequentiality and concurrency*, 13(4):485–529, August 2003.

[74] Davide Sangiorgi and David Walker. *The $\pi$-Calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

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

[76] Mark R. Shinwell, Andrew M. Pitts, and Murdoch J. Gabbay. FreshML: Programming with binders made simple. In *Eighth ACM SIGPLAN International Conference on Functional Programming (ICFP 2003)*, pages 263–274, Uppsala, Sweden, August 2003. ACM Press.

[77] Alwen Tiu and Dale Miller. A proof search specification of the $\pi$-calculus. In *3rd Workshop on the Foundations of Global Ubiquitous Computing*, volume 138 of *ENTCS*, pages 79–101, September 2004.

[78] Rüdiger Valk. 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*, pages 1–25, Lisbon, Portugal, 1998. Springer-Verlag LNCS 1420.

[79] Kevin Watkins, Iliano Cervesato, Frank Pfenning, and David Walker. A concurrent logical framework I: Judgments and properties. Technical Report CMU-CS-02-101, Computer Science Department., Carnegie Mellon University, 2002.

[80] J. Zucker. Formalisation of classical mathematics in AUTOMATH. In *Colloques Internationaux du Centre National de Recherche Scientifique*, volume 249, pages 135–145, July 1975.