An approach to innocent strategies as graphs

Pierre-Louis Curien Claudia Faggian

Abstract

This paper proposes an approach for extending to graphs the close relation between proofs and innocent strategies. We work in the setting of L-nets, introduced by Faggian and Maurel as a game model of concurrent interaction. We show how L-nets satisfying an additional condition, which we call $\mathrm{L_S}$-nets, can be sequentialized into traditional tree-like strategies. Conversely, sequential strategies can be relaxed into more asynchonous ones.

We develop an algebra of constructors and destructors that serve to build and decompose graph strategies, and to describe a class of minimally sequential graph strategies, which can be seen as an abstract kind of multiplicative-additive proof nets.

1. Introduction

The attempt to go beyond sequential computation, to capture a parallel or asynchronous notion of computation, appears currently an active direction in game semantics. Starting with the pioneering paper by Abramsky and Mellies [1], several proposals have emerged - with different motivations - towards a notions of strategy where sequentiality is relaxed to capture a more asynchronous form of interaction [2, 3, 4, 5, 6, 7, 8, 9] (and most recently [10]). Such strategies are often defined as *graphs* with certain properties, in contrast to more traditional (sequential) strategies, such as Hyland-Ong innocent strategies [11], which are *trees*. In this sense, we will talk of graph strategies, as opposed to tree strategies.

Our specific goal and contribution here, is to relate parallel strategies and sequential strategies, by showing how strategies represented by graphs, with partial ordering information, can be sequentialized into tree-like strategies, and how conversely, sequential strategies can be relaxed into more asynchronous ones. We work in the setting of L-nets that were introduced by Faggian and Maurel [4]. As we discuss at the end of this section, there are a number of closely related settings, to which our techniques should extend. The present paper builds on a preliminary extended abstract [12].

An innocent strategy describes in an abstract way the operational behaviour of a proof (or program). An interaction between "standard" tree strategies produces a sequence of actions (called play) which describes the trace of the computation. The idea underlying L-nets (as well as other of the approaches cited above) is that the order in which the actions should be performed is not completely specified, while still remaining able to express constraints. Certain tasks may have to be performed before other tasks; other actions can be performed in parallel, or scheduled in any order. A strategy

as L-net is a *directed acyclic graph*. The interaction results in a partial order, allowing for parallelism.

L-nets are based on *designs*, which form the first brick of Girard's Ludics [13]. The tree strategies and the graph strategies that we will consider are designs and L-nets, respectively:

- Designs are a linear variant of Hyland-Ong innocent strategies on a universal arena, as discussed in [14] (see also [15]). They are also particular sorts of abstract Böhm trees [16, 17].

  To be precise, we actually work with forests (which we call L-forests, rather than trees), relaxing the connectedness of Girard's original designs.

- L-nets are (potentially infinite) graph strategies on the same universal arena.

Note that sequential strategies are a special case of graph strategies: a tree is, in particular, a graph. On the other hand, it is possible to define a class of L-nets of minimal sequentiality, which we call *parallel* L-nets. As a result, we have a homogeneous space inside which we can move, adding or relaxing sequentiality (i.e., dependency between the actions). Between completely sequential and completely parallel strategies, we get a full range of intermediate strategies with decreasing sequentiality level.

Two flavours of views. It is known that (innocent) tree strategies can be presented as sets of views with certain properties. A view is a totally ordered sequence of moves (again with certain properties), and the set of views forms a tree. Any interaction results into a totally ordered set of moves.

An L-net is a set of partially ordered views, each of which is a partially ordered set of moves, where the partial order expresses a (partial) scheduling among moves. The set of such partially ordered views forms a directed acyclic graph. Any interaction results into a partially ordered set of moves.

The proof net experience. Tree strategies can be seen as abstract sequent calculus proofs. Specifically, designs arose as abstract (untyped) versions of (focalized) sequent calculus proofs of multiplicative-additive linear logic. By contrast, parallel L-nets can be seen as abstract multiplicative-additive proof nets. Indeed, there are two standard ways to handle proofs in linear logic: either as sequent calculus proofs, or as proof nets, which are graph-like structures satisfying a so-called correctness criterion. Sequent calculus proofs can be mapped onto proof nets, by forgetting some of the order between the rules, and conversely proof nets can be sequentialized into proofs. The correctness criterion is precisely the key property that makes sequentialization possible. Here we are looking for an abstract counterpart of this correspondence.

While the origins of game semantics are closely connected to the analysis of correct proof structures [18], this paper, to the best of our knowledge, is the first one to transfer – so to say in the other direction – the use of proof net techniques to the semantic setting of (innocent) games. In this respect, our contribution fits into a general research direction aiming at bringing closer together syntax and semantics.

Relating sequential and parallel strategies. As we have anticipated, L-nets are a conservative extension of innocent strategies (in the form used by ludics). This makes it possible to relate the two approaches (graphs versus trees). We are able to associate *a set of tree-strategies* to a parallel L-net (actually, to any "correct" L-net, see below) $\mathfrak{D}$, by saturating the order, i.e., we add sequentiality to the point that all choices of scheduling of the moves during an execution are determined. Conversely, given a tree strategy Π, we have a desequentialization procedure, which returns a parallel strategy, forgetting some "inessential" information on the scheduling.

Sequentialization is not possible for an arbitrary L-net, as L-nets can be intrinsically parallel, in the sense that actions depend on each other in an essential way (see Figure 5 for an example based on a well-known non-sequential function[1]). For this reason, we introduce $\mathrm{L_S}$-nets (S for "sequentializable"), which are L-nets satisfying an additional condition called Cycles. This condition upgrades the acyclicity condition of [4], which is sufficient for computation purposes (i.e., to guarantee that strategies compose), but not for our goals here. Condition Cycles can be considered as an abstract correctness criterion, and as a matter of fact, it is the adaptation to our setting of Hughes and Van Glabbeek's toggling condition [20].

A correctness criterion for proof nets has two roles: it guarantees that (i) normalization is possible (we are not stuck with cycles during normalization), and (ii) it is possible to associate a sequent calculus proof to the graph. The acyclicity condition is a minimal criterion which takes care of (i); the new condition Cycles guarantees also (ii).

We present an algebra of constructors and destructors allowing us to build and decompose graph strategies. This in particular allows us to (co)inductively define the classes of strategies of maximal and minimal sequentiality (i.e. of sequential and parallel strategies, respectively). The destructors and constructors are also used to define the sequentialization procedure. This procedure works as a stream-like, bottom-up process (coinductively) acting on potentially infinite $\mathrm{L_S}$-nets. Dually, the same operations serve us to define a desequentialization procedure that transforms L-forests into parallel L-nets. We will show that sequentialization and desequentialization can be performed so as to be inverse to each other.

A novelty of our framework is also that it allows us to define (and sequentialize) intermediate strategies that are neither "maximally sequential" (tree strategies) nor "minimally sequential", hence allowing for a whole range of sequentiality.

Some related works (see also Section 13.1). In this paper we are interested in the question of "sequentialization". Such a notion, together with the notion of proof net, is central to proof theory, in particular that based on Linear Logic. For this reason, we carried this first investigation quite naturally in a setting that is close enough to proof theory, namely Ludics, which maintains a close and direct connection with proofs, of which strategies are an abstraction. This in particular enabled us to take profit of the important work of Hughes and Van Glabbeek in our very definition of $\mathrm{L_S}$-nets.

We expect that methods similar to those proposed in this paper could be applied to

[1] Some background on sequentiality in denotational semantics may be found in [19].

more general frameworks and to larger classes of strategies. As a matter of fact, several of the approaches quoted at the beginning of this section are quite close to L-nets:

- Mimram has shown in his PhD thesis [21][Section 2.5.8] how to characterize L-nets (but not $\mathrm{L_S}$-nets) in the terminology of asynchronous games. Via his translation, it may be possible to import our techniques in the setting of [9]. Even though the focus of their work and our work is different, this might lead to insights on how to accommodate $\mathrm{L_S}$-nets rather than just L-nets, which would amount to accommodate additives in their framework (in [9], the study of a criterion relating sequential and asynchronous games is limited to the multiplicative fragment).

- The recent work by Rideau and Winskel [10], that builds on [9], and also on some follow-up of our work [22, 23] (see Section 13.1), provides an attractive, well-structured generalization of asynchronous strategies in the language of spans of event structures. Further reformulations and extensions of our approach to sequentialization and desenquentialization could be explored in that setting.

Our work is also close in spirit to the work of Abramsky and Melliès [1] on concurrent games:

- In their approach of strategies as closure operators, there are implicit identifications of more or less sequential strategies.

- They sketch a proof of full completeness that involves an interactive / realizability style criterion. Hence they investigate the relation between sequential and parallel, if we note that definability can be seen as a kind of sequentializability.

But direct technical comparisons with our work would be difficult, partly because the strategies under study are typed (ours are not) and, more importantly, because their strategies are closure operators, and not directely comparable with HO strategies. However, Melliès and Mimram's work [9] (building itself on [8]) provides an explicit link between concurrent strategies as closure operators and concurrent strategies as strategies played on graphs or more precisely on asynchronous transition systems.

Asynchronous games and concurrent games are therefore a quite natural direction in which to look for extending the results of this paper.

$\mathrm{L_S}$-nets versus Girard's and Hughes and Van Glabbeek's proof nets. The problem of making explicit the (typed) proof nets which underly (untyped) $\mathrm{L_S}$-nets and their connection with other notions of proof nets have been studied by Di Giamberardino [24] (which moreover provides a "local" sequentialization in the typed setting – see Section 13.1). It turns out that there is an injection from Girard's proof nets with weights [25] into (a typed version of) $\mathrm{L_S}$-nets and from these into Hughes and Van Glabbeek's proof nets. All of these inclusions are strict. This is because our strategies have an "operational semantics" flavour which carries somehow a certain amount of sequentiality, given by additive jumps.

Plan of the paper. Section 2 provides some rather informal introduction to the basic language of ludics, and to the close connection between proofs and designs (i.e., strategies), and between logical rules and actions (moves). We hope that having a grasp on such intuitions can help the reader. The reader may however skip this section, or consult it later. The interested reader can of course learn much more on ludics in [13], but note that none of the structures built by Girard on top of designs is used here.

Appendix Appendix A is instead a quite formal account of the L-forests as abstract sequent calculus proofs, and will be needed in connection with Section 8.

Sections 3 and 5 introduce our graph strategies: we recall the definition of L-net [4], and we define our subclass of sequentializable L-nets, the $\mathrm{L_S}$-nets[2].

The main technical result about $\mathrm{L_S}$-nets is the Splitting Lemma (the key to sequentialization) which we prove in Appendix Appendix B. The core of the paper lies in Sections 6 through 11: we present our basic (co)algebra of elementary operations on L-nets (Sections 6 and 10), we describe, illustrate, and relate our sequentialization and desequentialization procedures (Sections 7, 8, 9, 11).

In Section 12, we impose connectedness restrictions on both the source and target of the procedures, so as to adjust the picture to Girard's original designs. Section 4 provides a more precise road-map for the Sections 5 to 12. Section 13 is a concluding section.

Notation. We use $\biguplus$ to denote disjoint union, and $\sqsubseteq$ for the prefix ordering on words. We denote concatenation of words or of a word and a letter by juxtaposition, and if I is a set of letters, we let $\xi * I$ denote $\{\xi i : \ i \in I\}$.

2. Tree strategies and sequent calculus proofs

Designs, introduced in [13], have a twofold nature: they are at the same time semantic structures (an innocent strategy, presented as a set of views) and syntactic structures, which can be understood as abstract sequent calculus proofs (in a focusing calculus, which we introduce next).

In the following, we review in which (intuitive) sense a tree strategy can be associated with a sequent calculus proof, and vice versa. In Appendix Appendix A , we provide formal procedures.

2.1. Focalization and synthetic connectives

Multiplicative and additive connectives of linear logic separate into two families: synchronous (also called positive) connectives: $\otimes, \oplus, 1, 0$, and asynchronous (or negative) ones: $\parr, \&, \bot, \top$. A formula is positive (negative) if its outermost connective is positive (negative).

A cluster of connectives with the same polarity can be seen as a single connective (called a *synthetic* connective), and a "cascade" of decompositions with the same polarity as a single rule. This corresponds to a property known as focalization, discovered by Andreoli (see [26]), and which provides a (complete) so-called *focusing* strategy in

[2] $\mathrm{L_S}$-nets were called logical L-nets in [12].

proof-search: (i) negative connectives, if any, are given priority for persistent decomposition, (ii) when a subgoal containing only positive formulas is reached, choose a positive focus, and persistently decompose it up to its negative sub-formulas.

The division of connectives into positive and negative ones is not only fundamental to proof-search in linear logic, but also corresponds to the Player/Opponent duality in a strategy, and the organisation in clusters / synthetic connectives corresponds to the strict Opponent/Player alternation.

Shift. To these standard connectives, it is natural to add two new (dual) connectives, called *shift* (first introduced in [27]) $\downarrow$ (positive) and $\uparrow$ (negative). The role of the shift operators is to change the polarity of a formula: if n is negative, $\downarrow n$ is positive, and if p is positive, $\uparrow p$ is negative. When decomposing a positive connective into its negative subformulas (or viceversa), the shift marks the polarity change. As an example, the formula $(a \& b) \oplus (c \otimes d)$ should now be written $(\downarrow(a' \& b')) \oplus (c' \otimes d')$, where, say, a' is the result of recursively decorating a with shift operators. The shift is the connective which *captures "time"* (or sequentiality): it marks a step in computation.

Focusing calculus (HS). Focalization is captured by the following sequent calculus, called HS, originally introduced by Girard in [28], and closely related to Andreoli's focusing calculus (see [26]). A nice reference where to find more details on HS and its motivations is [29].

Axioms: $\vdash x^\perp, x$
We assume by convention that all atoms x are positive (hence $x^\perp$ is negative).

Any positive (resp. negative) cluster of connectives can be written as a $\oplus$ of $\otimes$ (resp. a $\&$ of $\parr$), modulo distributivity and associativity. The rules for synthetic connectives are as follows. Notice that each rule has labels; rather than more usual labels such as $\otimes_{Left}$, $\otimes_{Right}$, etc., we use formulas in the labels, as described below.

Positive connectives: Let $p(n_1, \ldots, n_n) = \oplus_{I \in \mathcal{N}}(\otimes_{i \in I}(\downarrow n_i))$, where $\mathcal{N}$ is a set of sets I of indices (with each I subset of $\{1, \ldots, n\}$). Each $\otimes_{i \in I}(\downarrow n_i)$ is called an additive component. In the calculus, there is an introduction rule for each additive component:

$$\frac{\ldots \quad \vdash n_i, \Delta_i \quad \ldots \quad \vdash n_j, \Delta_j \quad \ldots}{\vdash p, \ldots, \Delta_i, \ldots, \Delta_j, \ldots} \ (p, n_I)$$

where $i, \ldots, j$ range over I. Each positive rule is labelled with a pair of (i) the active formula (or focus) p of the conclusion, and (ii) the set $n_I = \{n_i \ : \ i \in I\}$ of the subformulas of the additive component to which the rule corresponds.

Note that we should rather speak of a rule scheme, because even when p and n_I have been fixed, there remains freedom in the way of splitting the rest of the sequent between the premises.

Negative connectives: Let $n(p_1, \ldots, p_n) = \&_{I \in \mathcal{N}}(\parr_{i \in I}(\uparrow p_i))$. Again, we call each $\parr_{i \in I}(\uparrow p_i)$ an additive component. There is only one introduction rule, which has a premise for each additive component :

$$\frac{\ldots \quad \vdash p_I, \Delta \quad \ldots \quad \vdash p_J, \Delta \ldots}{\vdash n, \Delta} \ \{\ldots, (n, p_I), \ldots, (n, p_J), \ldots\}$$

where $p_I = \{p_i : i \in I\}$. A negative rule is labelled by a set of pairs, each of the form (focus, set of subformulas), for each premise.

We call each of the pairs we used in the labels an *action*. We call an action positive (resp. negative) if it appears in the label of a positive (resp. negative) rule. In a negative rule, there is an action for each additive component.

In the purely multiplicative case (no connectives $\oplus, \&$), all negative rules have a single premise, and hence are labelled by a single action, while only one rule can be applied to each positive connective.

It is important to notice the *duality* between positive and negative rules: each premise (encoded by the action) (n, p_I) of a negative rule corresponds to one positive rule $(n^\perp, p_I^\perp)$ (where $p_I^\perp = \{n_i^\perp : i \in I\}$).

Another observation is that, starting with a proof of a sequent $\vdash p$ or $\vdash n$ consisting of one formula only, the rules maintain the invariant that all sequents contain at most one negative formula, a fact that can be stressed by writing $n^\perp \vdash \Delta$ (resp. $n_i^\perp \vdash \Delta_i$, $n_j^\perp \vdash \Delta_j$,...) instead of $\vdash n, \Delta$ (resp. $\vdash n_i, \Delta_i, \vdash n_j, \Delta_j, \ldots$).

Finally we note the following two special cases of the positive and negative rules (when $\mathcal{N} = \{I\}$ is a singleton and I is a singleton):

$$\frac{n^\perp \vdash \Delta}{\vdash \downarrow n, \Delta}\ (\downarrow n, n) \qquad \frac{\vdash p, \Delta}{(\uparrow p)^\perp \vdash \Delta}\ (\uparrow p, p)$$

In the sequel, we will keep the shift operators mostly implicit (they can easily be reconstructed).

2.2. Designs as (untyped) focusing proofs

Designs are an abstract version of focusing proofs. They are obtained in two steps:

1. One transforms a sequent calculus proof into a tree whose nodes are labelled by actions.
2. One replaces all the formula occurrences by addresses.

Conversely, given a design, we can build the "skeleton" of a sequent calculus proof. Such a skeleton becomes a concrete (typed) proof as soon as we are able to decorate it with formulas. Let us sketch this using an example.

First example. Consider the (purely multiplicative) proof on the left-hand side of Figure 1, where $a = \uparrow a_0$ and $b = \uparrow b_0$ are negative formulas and where c, d are positive formulas.

By forgetting everything in the sequent calculus proof but the labels of the rules, we obtain the tree depicted in the top right corner of Figure 1. This representation is more concise than the original sequent proof, but it still carries all relevant information, i.e., the sequents can be reconstructed. For example, when we apply the $\otimes$ rule, we know that the context of $a \otimes b$ is c, d, because they are used afterwards (above). After the decomposition of $a \otimes b$, we know that c (resp. d) is in the context of a (resp. b) because it is used after a (resp. b).

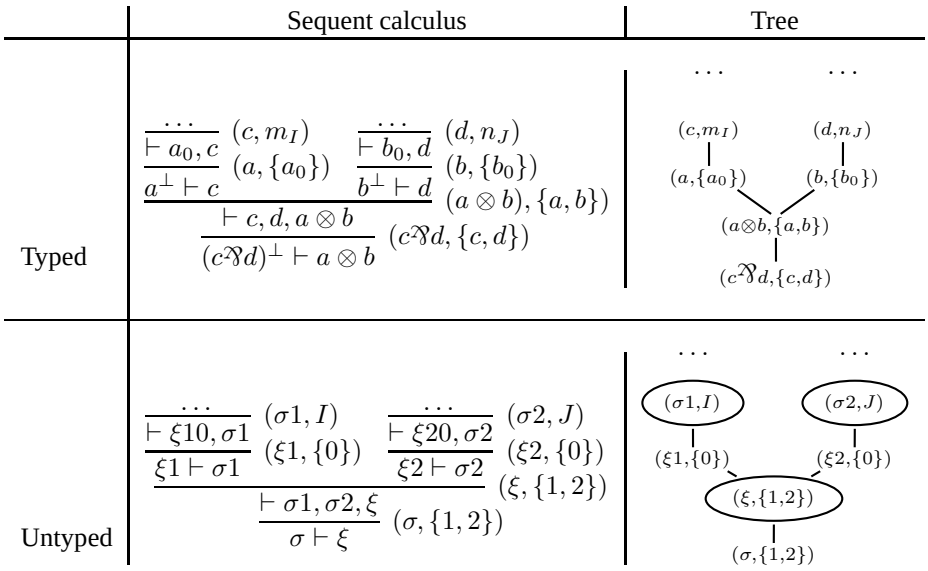


Figure 1: From focalized proofs to designs

Addresses (loci). One of the essential features of ludics is that proofs do not manipulate formulas, but *addresses*. An address is a sequence of natural numbers, which could be thought of as a channel, or as the address of a memory cell where an *occurrence of a formula* is stored. If we give address ξ to an occurrence of a formula, its (immediate) subformulas will receive addresses $\xi i, \xi j$, etc. Let $a = ((p_1 \parr p_2) \otimes m) \oplus n$. If we locate a at the address ξ, we can locate $p_1 \parr p_2, m, n$ respectively at $\xi 1, \xi 2, \xi 3$ (the choice of addresses is arbitrary, as long as each occurrence receives a distinct immediate extension of ξ). Hence what remains of a formula is a positional notation that retains the subformula information.

Let us consider an action, say, (p, n_I), where n_I corresponds to $\otimes_{i \in I}(\downarrow n_i)$. Its translation is (ξ, K), where ξ is the address of p, and K is the (finite) set of natural numbers corresponding to the relative addresses i of the subformulas n_i.

First example, continued. Coming back to our example (Figure 1), let us abstract from the type annotation (the formulas), and work with addresses. We locate $a \otimes b$ at the address ξ; for its subformulas a and b we choose the subaddresses $\xi 1$ and $\xi 2$. In the same way, we locate $c \parr d, c, d, a_0, b_0$ at the addresses $\sigma, \sigma 1, \sigma 2, \xi 10, \xi 20$, respectively. The result is depicted in the bottom right corner of Figure 1.

The two successive transformations are in fact independent. One can first transform formulas into addresses in the sequent calculus proof, yielding the bottom left abstract sequent calculus proof, and then keep only the tree of abstract labels. In Girard's terminology, the bottom left proof and the bottom right tree are called *dessin* and *dessein*, respectively: they are the syntactic face and the semantic face of the same objects, which are called designs.

To indicate the polarity, in our pictures of designs and L-nets, we circle positive

actions (as a reminder of the fact that they come from clusters of $\otimes$ and $\oplus$).

Understanding the additives. A $\&$-rule must be thought of as the superposition of two unary rules on the same formula, corresponding to the two actions $(a\&b, a)$ and $(a\&b, b)$. Given a sequent calculus proof in Multiplicative Additive Linear Logic (MALL), if for each $\&$-rule we select one of the premises, we obtain a proof where all $\&$-rules are unary. This is called a *slice* [30]. For example, below, the proof on the left can be decomposed into the two slices on the right.:

$$\dfrac{\dfrac{\overline{\vdash a, c} \quad \overline{\vdash b, c}}{\vdash a\&b, c}}{\vdash (a\&b) \oplus d, c} \quad \rightsquigarrow \quad \dfrac{\dfrac{\overline{\vdash a, c}}{\vdash a\&b, c}\,(a\&b, a)}{\vdash (a\&b) \oplus d, c} \quad \text{and} \quad \dfrac{\dfrac{\overline{\vdash b, c}}{\vdash a\&b, c}\,(a\&b, b)}{\vdash (a\&b) \oplus d, c}$$

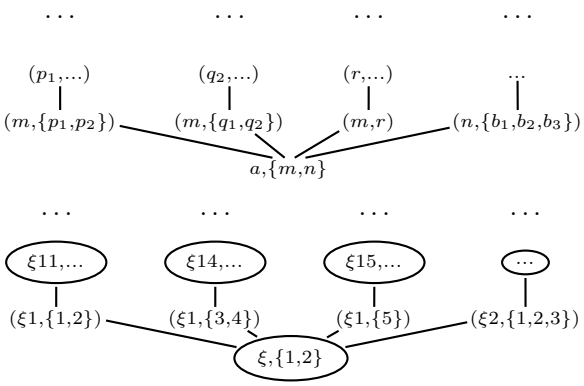
A more structured example. Let

$$a = (m \otimes n) \oplus c\ ,\ m = (p_1 \parr p_2)\&(q_1 \parr q_2)\&r\ ,\ n = b_1 \parr b_2 \parr b_3\ ,$$

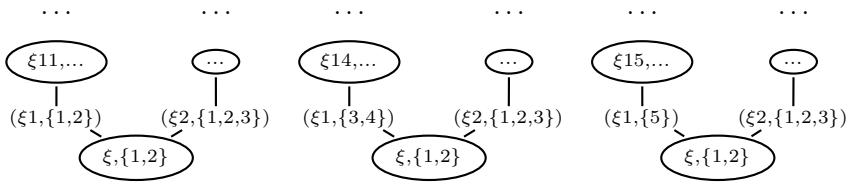
with r, p_i, q_i $(i = 1, 2)$, b_i $(i = 1, 2, 3)$ positive formulas. Consider the following proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dots}{\vdash p_1, p_2}\,(p_1, \dots) \quad \dfrac{\dots}{\vdash q_1, q_2}\,(q_2, \dots) \quad \dfrac{\dots}{\vdash r}\,(r, \dots)}{m^\perp \vdash}\,R_1 \quad \dfrac{\dfrac{\dots}{\vdash b_1, b_2, b_3}\,\dots}{n^\perp \vdash}\,R_2}{\vdash (m \otimes n) \oplus c}\,a, \{m, n\}$$

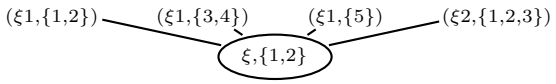
where $R_1 = \{(m, \{p_1, p_2\}), (m, \{q_1, q_2\}), (m, r)\}$ and $R_2 = \{(n, \{b_1, b_2, b_3\})\}$. The associated design is obtained as above in two steps:



It has three slices:



Bipoles. It is very natural to read a design (or an L-net) as built out of *bipoles*, which are the groups formed by a positive action (say, on address ξ) – the *root* of the bipole –, and all the negative actions which follow it (all being at immediate subaddresses ξi of ξ). The positive action corresponds to a positive connective. The negative actions are partitioned according to the addresses: each address corresponds to a formula occurrence, and each action on that address corresponds to an additive component of that formula. Each set of the partition is called an *additive rule*. For example,



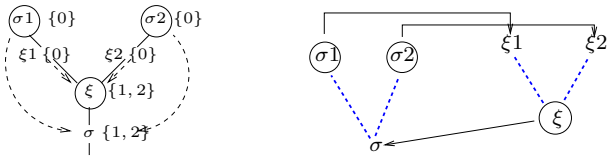
is a bipole, with the following two additive rules:

$$\{(\xi 2, \{1,2,3\})\} \quad \text{and} \quad \{(\xi 1, \{1,2\}), (\xi 1, \{3,4\}), (\xi 1, \{5\})\}\,.$$

2.3. Towards L-nets

Relating two orders. Let us consider a multiplicative design (or a slice). We are given *two partial orders*, which correspond to two kinds of information on each (occurrence of) action (σ, I): (i) a time relation (*sequential order*), specified by the tree structure of the design; (ii) a space relation (*prefix order*), corresponding to the relation of being subaddress (the arena dependency in game semantics).

Let us look again at our first example of design. We make the relation of being a subaddress explicit, by means of a dashed arrow, as follows:



If we emphasize the prefix order rather than the sequential order, we recognize something similar to a proof net (see [31]), with some additional information on sequentialization. Taking forward this idea of proof nets leads us to L-nets.

Additives: alternative choices and sharing. A strategy can be seen as representing the abstraction of a program, or the evolution of a system. An additive rule marks a choice in the possible evolution of the system.

This is immediate to see in the case of a tree. Each additive rule X can be interpreted as giving rise to different possible evolutions. For example, if $X = \{x_1, x_2\}$, to select a possible evolution, we choose one of the additive components x_1 or x_2.

Assume that some actions are performed in both evolutions. When working with trees, this part has to be duplicated. When working with graphs, only the actions that are specific to a certain evolution are made to depend on the choice, while the parts that are common are shared.

This is illustrated in Figure 2 (with $x_1 = (\xi 0, I), x_2 = (\xi 0, J)$). We will discuss this issue in more detail in Sections 6.6, 6.7, and 9.2 (where the example of Figure 2 is revisited).

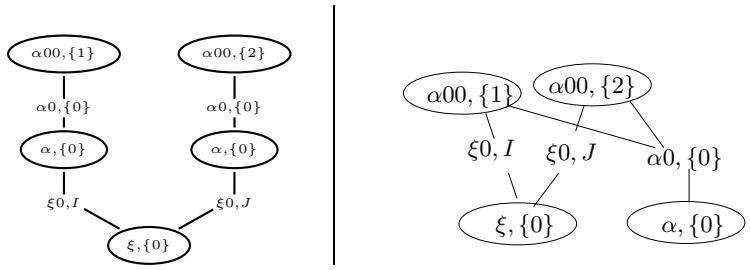


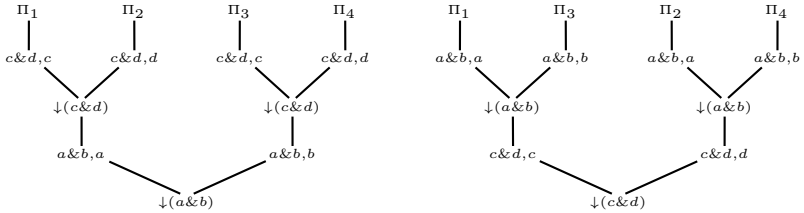
Figure 2: Duplication and sharing

Additives: a typed example. The following (typical) example with additives illustrates the relation between tree strategies and (parallel) L-nets (which will be defined shortly).

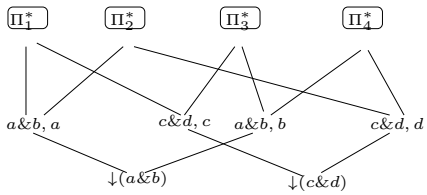
Assume that we have proofs $\Pi_1, \ldots, \Pi_4$ of $\vdash a, c$, $\vdash a, d$, $\vdash b, c$, $\vdash b, d$, respectively. In the sequent calculus (and in proof nets with additive boxes [30]) there are two distinct ways to derive $\vdash a\&b, c\&d$, and the two proofs differ only by commutations of the rules.

$$\dfrac{\dfrac{\dfrac{\Pi_1}{\vdash a, c}\quad\dfrac{\Pi_2}{\vdash a, d}}{\vdash a, c\&d}\,c\&d\quad\dfrac{\dfrac{\Pi_3}{\vdash b, c}\quad\dfrac{\Pi_4}{\vdash b, d}}{\vdash b, c\&d}\,c\&d}{\vdash a\&b, c\&d}\,a\&b \qquad \dfrac{\dfrac{\dfrac{\Pi_1}{\vdash a, c}\quad\dfrac{\Pi_3}{\vdash b, c}}{\vdash a\&b, c}\,a\&b\quad\dfrac{\dfrac{\Pi_2}{\vdash a, d}\quad\dfrac{\Pi_4}{\vdash b, d}}{\vdash a\&b, d}\,a\&b}{\vdash a\&b, c\&d}\,c\&d$$

The same phenomenon can be reproduced in the setting of designs; the above proofs closely correspond to the following two (typed) designs $\mathfrak{D}_1$ and $\mathfrak{D}_2$ (we write formulas instead of addresses, to make the example easier to grasp):



The following graph (which is a typed version of an L-net) is our intended common desequentialization of $\mathfrak{D}_1$ and $\mathfrak{D}_2$ (more on this example in Section 9.3).



3. L-nets

In this section, we introduce L-nets, which were first presented in [4]. Our definition is simpler than, but equivalent to the original definition, forgetting for the moment about one condition (the acyclicity condition), to which we shall return in Section 5.

L-nets are given by an interface, providing the names on which the L-net can communicate with other L-nets, and an internal structure, described by a *directed acyclic graph* whose nodes are labelled by *actions*. Before giving the definition of L-net, we recall some preliminary notions on directed acyclic graphs.

3.1. Directed acyclic graphs and terminology

We recall that a directed acyclic graph (dag) G is an oriented graph without (oriented) cycles. We write $a \leftarrow b$ for an edge from b to a. In all our pictures, the edges are oriented downwards. We denote by $\overset{+}{\leftarrow}$ the transitive closure of $\leftarrow$, which defines a strict partial order on the nodes of G.

We recall that the *transitive reduction* of a dag G is the graph that has the same vertices as G and whose edges are the edges $a \leftarrow b$ of G such that G does not contain another path from b to a of length > 1. (In terms of the underlying partial order, the transitive reduction retains only the covering relation, where y covers x when $x < y$ and there is no z such that $x < z < y$).) If an edge $a \leftarrow b$ of G is also in its transitive reduction, we say that a is a **predecessor** of b. A dag is called **reduced** if it coincides with its transitive reduction.

A node n of G is called a *root* (resp. a *leaf*) if there is no node a such that $a \leftarrow n$ (resp. $n \leftarrow a$).

Downward closure. Given a node $n \in G$, we denote by $n^{\downarrow}$ the downward closure of n, i.e., the sub-graph induced by restriction of G on $\{n\} \cup \{n' : \ n' \overset{+}{\leftarrow} n\}$.

3.2. L-nets

Addresses and interfaces. An **address** (called *locus* in [13]) is a string of natural numbers. We use the variables $\xi, \sigma, \alpha, \ldots$ to range over addresses . Two addresses are *disjoint* if neither is a prefix of the other.

An **interface** (called *base* in [13]) is a finite set of pairwise disjoint addresses, together with a **polarity** (positive or negative) for each address, such that *at most one* is negative. We write an interface as a sequent $\Xi \vdash \Lambda$, where Ξ is the set of the addresses with negative polarity, and Λ those with positive polarity.

An interface is *negative* if it contains a negative name, *positive* otherwise. In particular, the empty interface is positive.

An interface $\Xi \vdash \Lambda$ induces the definition of a polarity for each address of the form σ' such that there is $\sigma \sqsubseteq \sigma'$ for $\sigma \in \Xi \cup \Lambda$: the polarity of σ' is the same as the polarity of σ if the length of τ (where $\sigma' = \sigma\tau$) is even, opposite otherwise.

Actions (moves). An **action** is either the special symbol † (called daimon) or a pair $k = (\xi, I)$ where ξ is an address and I a *finite* set of natural numbers. We will say that the action k *uses* the address ξ.

Given an action $k = (\xi, I)$, we say that k generates ξi, for each $i \in I$, and also that k is the **parent** of b, if b is an action of the form $(\xi i, J)$.

The definition of polarity for the addresses induces a definition of polarity for all actions of the form $k = (\sigma', I)$: the polarity of k is the same as the polarity of σ'. The polarity of † is always defined positive.

Given an interface $\Xi \vdash \Lambda$, we denote by $A(\Xi \vdash \Lambda)$ the set of all actions for which a polarity is defined.

In the terminology of game semantics, the positive and negative actions are the Player and Opponent moves of our universal arena, respectively, while the parent relation expresses enabling constraints between the moves.

Nodes labelled by actions. We will work with dag's whose *nodes are labelled by actions*. We extend to nodes the terminology that we have introduced for the actions. We will say that a node is positive or negative, and that a node uses or generates an address, if it is the case for the labelling action.

As a matter of fact, we shall quite freely confuse a node with its labelling action, so that in the sequel $k, a, b, c, \ldots$ may denote either nodes and actions: $k = (\xi, I)$ will read as either "k is a node labelled by (ξ, I)", or "k is an action equal to (ξ, I)" (and what is meant in each instance should be clear from the context).

Now we can give the definition of L-net, as a dag whose transitive reduction satisfies six conditions. Conditions 1-4 are enough if all addresses are distinct (i.e., if the structure is purely multiplicative). Conditions 5-6 allow us to deal with the multiple use of addresses induced by the additive structure.

Definition 3.1 (L-nets). *An* L-net $\mathfrak{D}$ *is given by:*

- *An interface* $\Xi \vdash \Lambda$.
- *A possibly infinite set* A *of nodes which are labelled by actions of* $A(\Xi \vdash \Lambda)$ *(hence nodes are* occurrences *of actions).*
- *A structure on* A *of* directed acyclic, reduced, and bipartite graph *(if* $k \leftarrow k'$*, then their labelling actions have opposite polarity), whose transitive reduction satisfies conditions 1-4 and 5-6 below. We say that a node is positive (resp. negative) according to the polarity of the labelling action.*

  1. Views. *For each node* k*, all the addresses used in* $k^{\downarrow}$ *are distinct .*
  2. Parents.
     - *For each node* a*, using address* σ*, either* σ *belongs to the interface, or* σ *is generated by an action which labels a preceding node* $c \overset{+}{\leftarrow} a$*.*
     - *If* $a \leftarrow b$ *and* a *is positive, then* b *must use an address generated by* a[3]*.*
     - *If* a *is negative, it has at most one predecessor.*

     (It follows that if a negative node is not a root then its parent is the label of the unique predecessor.)
  3. Negativity. *If* $\Xi = \{\xi\}$ *is not empty (i.e. the interface is negative), then either the set of nodes* A *is empty, or at least one node uses* ξ*.*
  4. Positivity. *If* a *is a leaf, then it is positive.*

[3]This is the innocence condition, cf. [11, 32].

Two distinct nodes are called conflicting *if they use the same address. We call* additive pair *a pair of conflicting* negative *nodes which have the same predecessor.*

5. Siblings. *Any two conflicting nodes which have the same predecessor have distinct labels, i.e., of the form (σ, I_1), (σ, I_2), with $I_1 \neq I_2$.*
6. Additives. *Given two* positive *conflicting nodes k_1, k_2, there exists an additive pair w_1, w_2 such that $w_1 \overset{+}{\leftarrow} k_1$, and $w_2 \overset{+}{\leftarrow} k_2$.*

Remark 3.2. *Note that by condition* Parents *every root uses an address in the interface. Conversely, the same condition also imposes that an action using the negative address of the interface (if any) is a root, but* actions using a positive address of the interface need not be roots.

In order to obtain a good computational behaviour of L-nets as strategies, and to be able to relate them to sequential innocent strategies, we still need a correctness condition. This will lead us to $\mathrm{L_S}$-nets in Section 5.

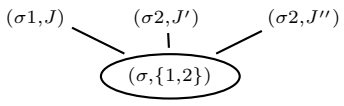
Rules and conclusions. We call *rule* of an L-net a maximal set of nodes that are pairwise conflicting and have the same or no predecessor. A rule is positive or negative according to the polarity of the nodes.

We say that a rule is *unary* if it is a singleton (a positive rule is always unary). When a rule is not unary, we call it an *additive rule* (think of each action as an additive component). Note that an additive rule is necessarily a negative rule, but negative rules can be unary (see Section 3.3). Note also that if w_1, w_2 form an additive pair, then w_1, w_2 belong to the same negative rule.

By analogy with proof nets, we call *conclusion* a rule whose nodes are all roots. By conditions Parents and Negativity, we have:

- An L-net is positive if and only if it has only positive conclusions (which are all on distinct addresses by condition Additives).
- A (non-empty) L-net is negative if and only if it has a negative conclusion. (Note that an L-net can have at most one negative conclusion, by definition of a rule, and by the assumption that an interface contains at most one negative address.)

We note that the positive nodes induce a partition of the dag into bipoles (cf. Section 2.2) (plus possibly a negative conclusion), where a bipole consists of a positive rule (its root) and a set of negative rules. For example, the following bipole has two negative rules ($R_1 = \{(\sigma 1, J)\}$ and $R_2 = \{(\sigma 2, J'), (\sigma 2, J'')\}$) and one positive rule ($R = \{(\sigma, \{1, 2\})\}$).



Enabling sets. The key role of condition Additives is to ensure a one-to-one correspondence between the nodes of an L-net and the sets of actions in their downward closure, that represent their history, or their preconditions.

Lemma 3.3. *For each pair of distinct nodes k, k' of an L-net $\mathfrak{D}$, the sets of actions of $k^{\downarrow}$ and $k'^{\downarrow}$ are different.*

Proof. Suppose that k, k' are distinct, but that the sets of actions of $k^{\downarrow}$ and $k'^{\downarrow}$ are the same. Then in particular there exists some $k_1 \in k'^{\downarrow}$ such that k and k_1 have the same label. We distinguish two cases:

- If $k_1 \neq k'$, then $k_1^{\downarrow}$ is strictly contained in $k'^{\downarrow}$, and hence by our assumption k_1 and k must be distinct.
- If $k_1 = k'$, then $k_1 \neq k$ by assumption.

Hence in both cases we have proved that k_1 and k are distinct. By condition Additives, there exists an additive pair w_1, w_2 such that $w_1 \in k^{\downarrow}$ and $w_2 \in k_1^{\downarrow}$ (and hence $w_2 \in k'^{\downarrow}$). Then, by our assumption, there is a node $w \in k^{\downarrow}$ that has the same label as w_2. But this is impossible, as it would violate condition Views applied to k. □

3.3. Slices

A *slice* $\mathfrak{S}$ of an L-net $\mathfrak{D}$ is a *downward closed* subgraph of $\mathfrak{D}$ in which no two nodes are conflicting, and which is an L-net (i.e., satisfies condition Positivity). In this paper, we also insist that slices are always *maximal* such subgraphs.

3.4. L-nets as sets of views

Just as innocent strategies (and designs), an L-net can be presented as a set of views, with some properties. In this setting, a view is not a sequence of moves, but a partial order (with a top element).

Definition 3.4 (View). *A* view *on the interface $\Xi \vdash \Lambda$ is a set $\mathfrak{c}$ of polarized actions equipped with a partial order with a maximal element, which, when considered as a dag where the nodes are labelled by themselves, is an L-net on the same interface. A view is called positive or negative, not according to the interface, but according to the polarity of its top element. We define a partial order on views as follows: $\mathfrak{c} \sqsubseteq \mathfrak{c}'$ if $\mathfrak{c}$ is the restriction of $\mathfrak{c}'$ to $\{x : x \leq a\}$, for a certain $a \in \mathfrak{c}'$. A set S of views is* closed under restriction *if $\mathfrak{c}' \in S$ and $\mathfrak{c} \sqsubseteq \mathfrak{c}'$ implies $\mathfrak{c} \in S$.*

We note that the conditions Siblings and Additives are vacuous for a view since by definition actions are not repeated.

When the order is total, views coincide with Girard's *chronicles* [13] (and conform with the notion of view in Hyland-Ong's framework), whence our choice of notation $\mathfrak{c}, \mathfrak{c}'$ for views.

From L-nets to sets of views. Any node k of an L-net $\mathfrak{D}$ defines a view: indeed, $k^{\downarrow}$ induces a partial order on its nodes (cf. Section 3.1) and by the condition Views, in $k^{\downarrow}$ there is a one-to-one correspondence between the nodes and their labelling actions. Let $\overline{n}$ be the action labelling the node n. We set:

$$\ulcorner k \urcorner = \{\overline{n} : \ n \in k^{\downarrow}\}, \text{ with the order induced by } \leftarrow .$$

Hence we can associate to each L-net $\mathfrak{D}$ a set *Views*$(\mathfrak{D})$ of views, as follows:

$$\textit{Views}(\mathfrak{D}) = \{\ulcorner n \urcorner : \ n \text{ is a node of } \mathfrak{D}\} .$$

The set *Views*$(\mathfrak{D})$ is closed under restriction.

From sets of views to L-nets. Conversely, given a set Δ of views which is closed under restriction, we define a directed graph *Graph*(Δ) as follows: the nodes are the elements of Δ, and $\mathfrak{c} \leftarrow \mathfrak{c}'$ iff $\mathfrak{c} \sqsubset_1 \mathfrak{c}'$.

Lemma 3.5. *Let Δ be a (possibly infinite) set of views closed under restriction. Then* Graph(Δ) *is an L-net iff it satisfies conditions* Positivity *and* Additives.

Proof. The conditions Parents and Views hold obviously. Condition Siblings also holds: two negative views with the same parent $\mathfrak{c}$ have the form $\mathfrak{c}_1 = \mathfrak{c} \cup \{a\}$ and $\mathfrak{c}_2 = \mathfrak{c} \cup \{b\}$. If $\mathfrak{c}_1 \neq \mathfrak{c}_2$, necessarily $a \neq b$. □

It is rather easy to express both Positivity and Additives in terms of views. Hence we can also define an-L-net on a given interface as a set of views closed under restriction, which satisfies (the analogue of) Positivity and Additives.

Relating the presentations. It is immediate that *Views*$(\textit{Graph}(\Delta)) = \Delta$, if Δ is a set of views closed under restriction. Conversely, given (the transitive reduction of) an L-net $\mathfrak{D}$, we have that *Graph*$(\textit{Views}(\mathfrak{D}))$ is isomorphic to $\mathfrak{D}$ (easy consequence of Lemma 3.3).

Summarizing, we have shown that *Views* and *Graph* are inverse bijections.

We will use both presentations for L-nets. The presentation of L-nets as sets of views, on which we will largely rely, allows us to compare nodes in different graphs, by comparing the corresponding views. This will be particularly useful in Sections 6.3 and 6.6.

Sometimes, the graph presentation is more intuitive. However, it is obvious that all notions and conditions can be expressed in either term. Observe in particular that

$$k_1 \stackrel{+}{\leftarrow} k_2 \quad \text{iff} \quad \ulcorner k_1 \urcorner \sqsubset \ulcorner k_2 \urcorner \quad \text{iff} \quad \overline{k_1} < \overline{k_2} \text{ in } \ulcorner k_2 \urcorner .$$

Conventions. We will often not distinguish between isomorphic notions, such as a view $\mathfrak{c}$ and the induced node, or a node k and the view $\ulcorner k \urcorner$. Moreover, to keep notation simple, we will sometimes write $k \in \mathfrak{c}$ (for example, $k \in \ulcorner k \urcorner$) instead of $\overline{k} \in \mathfrak{c}$, and we will decorate actions, nodes, views and L-nets with their polarity (for example, $k^+, \mathfrak{c}^+, \mathfrak{D}^+$) only if we want to stress it.

3.5. L-forests and designs

If the views are totally ordered, the above definitions produce a forest, corresponding to a "standard" innocent strategy.

Definition 3.6. *An L-forest is an L-net* Π *which is a forest; we require that if* Π *is negative, then it has only one conclusion.*

In Appendix Appendix A, we will show that L-forests arise from adding a MIX rule to the sequent calculus underlying designs.

By further restricting the notion of L-forest, we arrive at *designs*.

Proposition 3.7. *An L-forest which has a unique conclusion, and which* branches only on positive nodes. *is a design (as defined in [13]).*

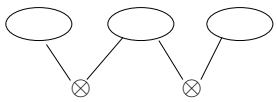
4. Sequential versus parallel strategies: an overview

Let us stop a moment to reflect on the notions we have seen so far, before entering the more technical part of the paper.

Part of the process of abstraction leading from concrete proofs to the abstract proofs of ludics is that an action (a move) can be seen as a cluster of operations that can be performed together (thanks to focalization). However, in a tree strategy (L-forests, designs, innocent strategies...), there remains a lot of artificial sequentiality, like in sequent calculus proofs for linear logic. In the case of proofs, the solution has been to develop proof nets, a theory which gave rise to many successful developments. The advantage of proof nets is that information which is irrelevant to the "essence" of the proof is forgotten. More precisely, proof nets allow us to identify sequent calculus proofs that only differ by some *permutations of rules*. Consider, for example, the two standard proofs

$$\dfrac{\dfrac{\vdash a, a^\perp \quad \vdash b, b^\perp \quad \vdash c, c^\perp}{\cdots}}{\vdash a^\perp \otimes b, b^\perp \otimes c, \ldots} \begin{matrix}(a^\perp \otimes b)\\ (b^\perp \otimes c)\end{matrix} \qquad \text{and} \qquad \dfrac{\dfrac{\vdash a, a^\perp \quad \vdash b, b^\perp \quad \vdash c, c^\perp}{\cdots}}{\vdash a^\perp \otimes b, b^\perp \otimes c, \ldots} \begin{matrix}(b^\perp \otimes c)\\ (a^\perp \otimes b)\end{matrix}$$

and compare them with the (unique) corresponding proof net, which has the following shape:



The *different permutations* of the rules correspond to *different sequentializations* of the proof net, that is, in our view, to *different schedulings* of the rules.

Similarly, sequential strategies (hence designs, in particular) distinguish proofs (or programs) which only differ by the order in which the operations are performed.

Proof nets and sequentialization. Proof nets were introduced by Girard along with linear Logic as a graph representation of proofs. To each sequent calculus proof, one associates a proof net (several sequent calculus derivations can become the same proof net). Conversely, given a proof net, we can associate to it some (usually more than one) sequent calculus derivations; this procedure is called *sequentialization*. Proof nets are defined in two steps. One first defines "logically correct" typed graphs, which are called proof structures. A proof net is a proof structure which is the image of a sequent calculus derivation. Proof nets are characterized by geometrical properties, called *correctness criterion*. A very useful one is AC, for Acyclic and Connected (also called Danos-Regnier criterion). Acyclicity (of certain paths) is the fundamental property which guarantees sequentialization. Connectness (of the paths) is instead related to the refusal in the logic of the MIX rule (see Section 12). We will apply similar techniques to L-nets.

The dynamics. In this paper, we focus on sequentialization. The dynamics of L-nets is described in[4] (more details -in a more general setting- are in [22]). Normalization (i.e., composition) of L-nets is reduced to normalization (composition) of slices: (i) decompose each L-net in its slices, (ii) normalize the slices, and (iii) put them together (superimpose), where the superposition of the slices is simply the union of the views.

Composition of slices, which is the core of L-nets normalization, is as straightforward as normalization on MLL (multiplicative linear logic) proof-nets, as slices are sort of purely multiplicative proof-nets. There are several possible ways to present it. One can use rewriting rules in the style of MLL proof-nets, or an abstract machine (as in[4]). The most elegant way is however based on Girard's "merging of orders", defined in [13] (the generalization to L-nets is in [22]): each slice can be seen as a partial order on occurrences of actions; the merging of two orders is then the transitive closure of their set-theoretical union; the acyclicity conditions (which are true for designs, but also for L-nets) insures that the result is a partial order, and is in fact a slice.

4.1. Sequentialization (and desequentialization) of L-nets

In Section 5.1, we will define a *correctness criterion*, inspired by [20], which guarantees that an L-net can be "sequentialized". An L-net which satisfies the criterion is called $\mathrm{L_S}$-net. If we continue the analogy with proof nets, we can see L-nets as proof structures, and $\mathrm{L_S}$-nets as proof nets.

We will then define (Sections 7 and 8) two procedures which we call *desequentialization* and *sequentialization*, that associate

- an $\mathrm{L_S}$-net $\mathit{deseq}(\Pi)$ to an L-forest Π (Section 8), and
- a set $\{\mathit{seq}(\mathfrak{D})\}$ of L-forests to an $\mathrm{L_S}$-net $\mathfrak{D}$ (Section 7), respectively.

We will show (Theorem 11.2) that *all dependency which is taken away by desequentialization can be (non-deterministically) restored through sequentialization*. The nondeterminism corresponds to the fact that several L-forests Π_i can be associated to the same $\mathrm{L_S}$-net $\mathfrak{D}$, each of which can be recovered by sequentialization of $\mathfrak{D}$. In Section 10, we make precise in which sense $\mathit{deseq}(\Pi)$ has less sequentiality than Π, and give a description of the $\mathrm{L_S}$-nets of "minimal sequentiality" (which we call parallel L-nets), based on the operations presented in Section 6.

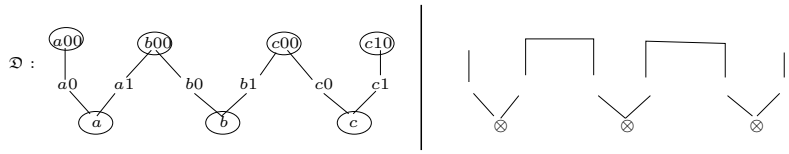


Figure 3: Maximally parallel L-net

Targetting L-forests (instead of design). As we said, we will first prove a sequentialization (resp. desequentialization) result that holds for all $\mathrm{L_S}$-nets, and that has the class of *L-forests as target* (resp. source). Only later, in Section 12, we will restrict this procedure so as to have *designs as target*. More precisely, we can characterize the class of $\mathrm{L_S}$-nets which sequentialize into a design as those which beside acyclicity satisfy also a *connectness criterion*.

We make this choice for a number of reasons. First, it is more general, and more natural, to target L-forests instead of designs. An L-net does not need to be connected (in the ordinary graph-theoretic sense), and its natural target is an L-forest.

Also, non-connectedness appears as a natural and desirable feature if we want parallelism. In further work joint work of the second author [33], a full abstraction result for the linear π-calculus [34] which involves L-forests rather than designs is established: this is because the *parallel composition* in process calculi corresponds to juxtaposition in the semantics; the natural result is not a design, but a forest. (See Section 13 for more discussion.)

Non-connectedness may also arise in proof development. It is an ingredient of Andreoli's concurrent proof construction [35]. In such a setting, there may be disconnected partial proofs that will be connected later in the proof development.

4.2. Graduating sequentiality

Consider the L-net $\mathfrak{D}$ in Figure 3. It is maximally parallel, in the sense that the only sequentiality that it expresses is relative to the axioms (see Section 8). The three actions a, b, c can be performed in parallel, or in any order. This L-net corresponds to the multiplicative proof net depicted on the right-hand side (we can think of a, b, c as tensors).

The L-net $\mathfrak{D}'$ in Figure 4, is more sequential. Indeed, the order has been increased: the action b has now to be performed after the action a_1. The actions a and c can still be performed in parallel.

The L-net $\mathfrak{D}''$ is completely sequential, as there are no choices in the scheduling: the action c has to be performed first.

This last L-net is in fact a tree, and corresponds to the following sequent calculus proof (which we only sketch). Once again, rules are labelled by the active formula, i.e. the formula which is decomposed in the rule.

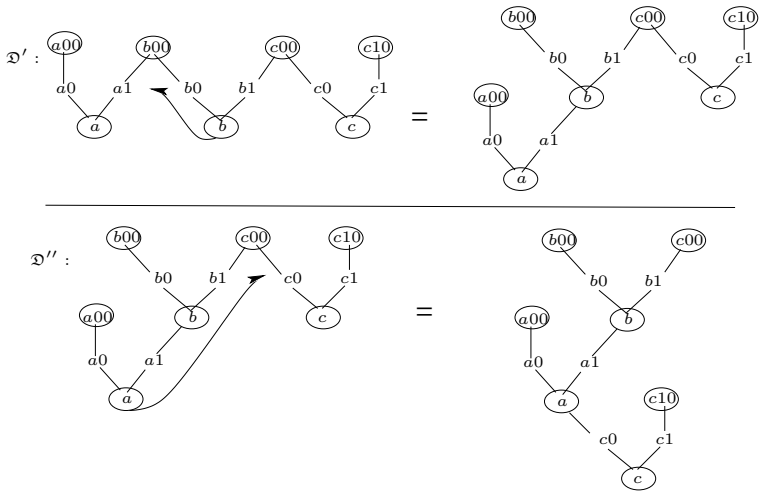


Figure 4: Partially and totally sequential L-net

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \ldots}\quad\dfrac{}{\vdash \ldots}}{\vdash \ldots.}\,b}{a1 \vdash b, c00}\,a1}{\vdash c00, a, b}\,a \qquad \dfrac{\dfrac{}{\vdash \ldots}}{\vdash c10}\,c10}{c0 \vdash a, b}\,c0 \qquad \dfrac{}{c1 \vdash}\,c1}{\vdash a, b, c}\,c$$

5. $\mathbf{L_S}$-nets

In this section we refine the notion of *L-net* of Faggian and Maurel [4]. Instead of the acyclicity condition in the original definition, we have here a stronger one, resulting in the (new) notion of $\mathrm{L_S}$-nets.

We recall that we are only interested in the properties of the underlying transitive reduction of an L-net. All conditions in this section are therefore on the transitive reduction of the graph.

Paths. The following notions are relative to some L-net $\mathfrak{D}$. An edge is an *entering edge* of the node a if it has a as target, as an oriented edge of $\mathfrak{D}$. If R is a negative rule and e an entering edge of an action $a \in R$, we call e a *switching edge* of R.

A *rule path* is a sequence of nodes $k_1, ...k_n$ belonging to distinct rules, and such that for each $i < n$ either $k_i \to k_{i+1}$ (the path is going down) or $k_i \leftarrow k_{i+1}$ (the path is going up). A rule cycle is defined similarly as a sequence of nodes $k_1, ...k_n, k_{n+1}$, where the k_i's ($i \leq n$) are distinct, where $k_1 = k_{n+1}$, and for each $i < n + 1$ either $k_i \to k_{i+1}$ or $k_i \leftarrow k_{i+1}$.

A *switching path* is a rule path which uses at most one switching edge for each negative rule, i.e., the path does not contain three successive nodes k_{i-1}, k_i, k_{i+1} such that k_i is negative, $k_i \leftarrow k_{i-1}$, and $k_i \leftarrow k_{i+1}$.

A *switching cycle* is a rule cycle which uses at most one switching edge for each negative rule.

Correctness criterion. Now we can complete the definition of $\mathrm{L_S}$-net. We want to be able to sequentialize our graphs. The following condition (which can be seen as a *correctness criterion*) guarantees that it is always possible to find a rule which does not depend on any other rule.

Definition 5.1 ($\mathrm{L_S}$-nets). *An* $\mathrm{L_S}$-net *is an L-net $\mathfrak{D}$ such that the following condition holds for its transitive reduction:*

- Cycles. *Given a non-empty union C of switching cycles of $\mathfrak{D}$, there is an additive rule W not intersecting C, and a pair $w_1, w_2 \in W$ such that for some nodes $c_1, c_2 \in C$, $w_1 \overset{+}{\leftarrow} c_1$, and $w_2 \overset{+}{\leftarrow} c_2$.*

This criterion, closely inspired by the analogous criterion given in [20], is delicate; its technical meaning (and the need to consider a set of cycles and not only one) will be apparent in the proof of the Splitting Lemma (in Appendix Appendix B), which is an adaptation of the analogous proof in [20]. Below, we try however to provide some intuitions, by giving a typical example of its failure, namely an instance of the well known Gustave function[4].

L-nets and $\mathrm{L_S}$*-nets.* The condition Cycles is a strengthening of the acyclicity condition of [4]. Acyclicity asserts that there are no switching cycles in a slice[5]. It is immediate that the condition Cycles implies the acyclicity condition, and reduces to it in a purely mutliplicative framework (i.e., in the absence of any additive rule). Notice that while acyclicity is a property of a slice, the new condition speaks of cycles which traverse slices.

In Figure 5, we show an example of an L-net that satisfies the acyclicity condition but does not satisfy the condition Cycles.

There are three additive rules: $\{(\alpha 0, \{1\}), (\alpha 0, \{2\})\}$, $\{(\beta 0, \{1\}), (\beta 0, \{2\})\}$, and $\{(\gamma 0, \{1\}), (\gamma 0, \{2\})\}$.

We could type this L-net as follows:

[4]This example is inspired by the Berry-Kleene function, also known as Gustave function, which is the simplest example of a stable but non sequential function (see [19]). The Gustave function recurs often in the theory of MALL proof nets; in particular, we adapt here the example given in [20]

[5]It is the condition that (together with connectedness) characterizes mutliplicative proof nets.

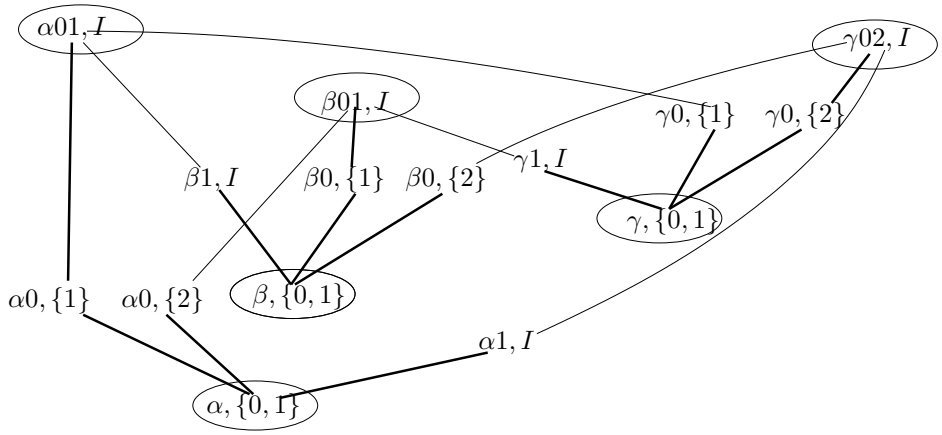
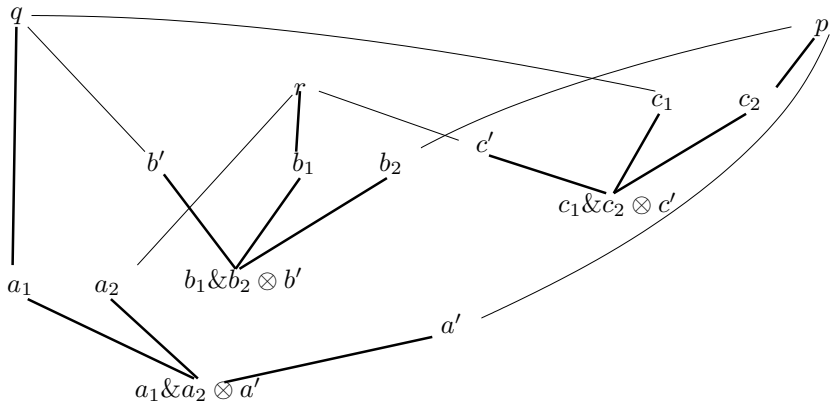


Figure 5: Gustave L-net



We think of the three conclusions as Tensor rules, and of the leaves (p, q, r) as occurrences of axioms (for example, $\vdash 1$). We see a_1, a_2 as the two components of a $\&$, and similarly for b_1, b_2 and c_1, c_2 (hence, each conclusion can be typed as $(1\&1) \otimes \uparrow 1$). We check our claims as follows.

1. *The Gustave L-net satisfies acyclicity*, because there are several switching cycles, but none appears inside a single slice. This property is an immediate consequence of the following two observations:
   (a) The set of slices of the Gustave L-net is $\{p^{\downarrow}, q^{\downarrow}, r^{\downarrow}\}$. Indeed, any two axioms are separated by an additive pair, in the sense that each of the two axioms depends on a different component of the additive pair. For example, if we consider p, q we have $c_1 \leftarrow q$ and $c_2 \leftarrow p$. Similarly, the pair p, r (resp. q, r) is separated by the rule $\{b_1, b_2\}$ (resp. $\{a_1, a_2\}$).
   (b) Each switching cycle has to use at least two axioms.
2. *The Gustave L-net does not satisfy the condition* Cycles, because there are switching cycles that use (intersect) *all* the additive rules, which a fortiori does not leave

any space for an additive rule outside the cycle to "break" it. Here is one: start from $(a_1 \& a_2) \otimes a'$, go up through a_1 to q, down through b' to $(b_1 \& b_2) \otimes b'$, up through b_1 to r, down through c' to $(c_1 \& c_2) \otimes c'$, up through c_2 to p, and finally down through a'.

We will see in Section 6.5 that the Gustave L-net, does not admit sequentialization.

6. Operations on $\mathrm{L_S}$-nets

In this section, we introduce operations that allow us to construct and decompose L-nets and $\mathrm{L_S}$-nets.

- *Constructors:* rooting, boxing, superposition, and additive union.
- *Destructors:* root removal, splitting, and scoping.

In the sequel, we will make an extensive use of these operations.

We will first (Sections 6.2 through 6.4) treat the operations that deal with positive rules and unary negative rules (which is enough for the purely multiplicative case). We will then introduce the additive structure, where *sharing* plays a crucial role (Sections 6.6 and 6.7).

Warning. Throughout the section, we assume that the operations we define take *reduced* L-nets (resp. $\mathrm{L_S}$-nets) (in the sense that their dag is reduced) as input. We will see that these operations return (possibly partial) L-nets (resp. $\mathrm{L_S}$-nets) that are not always reduced (see Remark 6.6). This is actually the reason why we allowed non reduced dags in the definition of an L-net.

6.1. Preliminary properties

A convenient notion is that of partial L-net. We say that an L-net is *partial* if it possibly does not satisfy condition Positivity.

We will make a repeated use of the following result.

Lemma 6.1 (Downward closure). *Let $\mathfrak{D}$ be an L-net. Any downward closed subset G of $\mathfrak{D}$ is a (possibly partial) L-net. If $\mathfrak{D}$ satisfies condition* Cycles*, so does G.*

Proof. All properties are inherited from $\mathfrak{D}$. Any view of G is also a view of $\mathfrak{D}$. Let us check the preservation of conditions Additives and Cycles. If k_1, k_2 are two distinct nodes in G on the same address, then condition Additives for $\mathfrak{D}$ provides a pair of negative nodes w_1, w_2 such that $w_i \overset{+}{\leftarrow} k_i$, which (by downward closure) belong to G.

Observe that any cycle in G is also a cycle in $\mathfrak{D}$. If we have a collection of switching cycles inside G, the condition Cycles for $\mathfrak{D}$ gives us an additive rule W that is not traversed by any of the cycles, and a pair $w_1, w_2 \in W \cap G$. Then, taking $W \cap G$, w_1, and w_2, the condition holds in G. □

Corollary 6.2. *If $\mathfrak{D}$ is an $\mathrm{L_S}$-net and K is a set of positive nodes of $\mathfrak{D}$, then the subgraph induced on $\bigcup\{k^{\downarrow} : k \in K\}$ is an $\mathrm{L_S}$-net.*

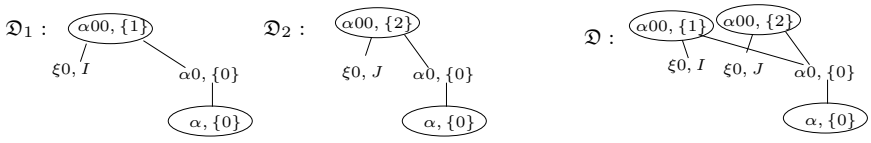


Figure 6: Superposition

Superposition of L-nets. Given a collection of L-nets, let us consider their union *as sets of views* (the union is not disjoint in general). We call this operation *superposition*. Under which conditions is a superposition of L-nets an L-net? By Lemma 3.5, if $\mathfrak{D}_1, \mathfrak{D}_2$ are L-nets (resp. $\mathrm{L_S}$-nets), $\mathfrak{D}_1 \cup \mathfrak{D}_2$ is an L-net (resp. an $\mathrm{L_S}$-net) iff it satisfies condition Additives (resp. conditions Additives and Cycles).

Remark 6.3. *In defining the superposition of graphs (and additive union, in Section 6.6), it is crucial that we work not just with nodes, but with their view. This allows us to* compare nodes belonging to different graphs.

Example 1: sharing of context. Consider the two L-nets $\mathfrak{D}_1, \mathfrak{D}_2$ in Figure 6. The superposition of $\mathfrak{D}_1$ and $\mathfrak{D}_2$ produces the L-net $\mathfrak{D} = \mathfrak{D}_1 \bigcup \mathfrak{D}_2$.
In fact, the set of views of $\mathfrak{D}_1$ is the set of views defined by each of its nodes k, that is:

$$\{\overset{\alpha 0,0}{\underset{\alpha,0}{\bigcirc}}, \alpha,0, (\xi 0, I), \ulcorner(\alpha 00, \{1\})\urcorner = \mathfrak{D}_1\}.$$

The set of views of $\mathfrak{D}_2$ is:

$$\{\overset{\alpha 0,0}{\underset{\alpha,0}{\bigcirc}}, \alpha,0, (\xi 0, J), \ulcorner(\alpha 00, \{2\})\urcorner = \mathfrak{D}_2\}.$$

The resulting union is:

$$\{\overset{\alpha 0,0}{\underset{\alpha,0}{\bigcirc}}, \alpha,0, (\xi 0, I), (\xi 0, J), \mathfrak{D}_1, \mathfrak{D}_2\},$$

which corresponds to $\mathfrak{D}$.

Example 2: positive n-ary rules. Superposition allows us to construct positive rules k which have more then one premiss, as illustrated in Figure 7.

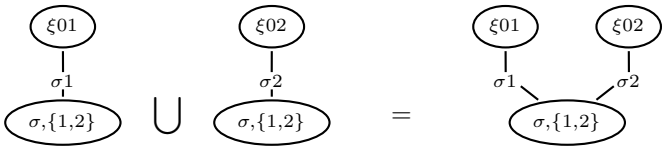


Figure 7: Positive binary rule

6.2. Constructing: rooting and boxing

The following constructions allow us to add a new *unary* conclusion to an L-net.

Definition 6.4 (Rooting). *Let $\mathfrak{D}$ be a positive (resp. negative) L-net, of interface $\vdash \xi i, \xi j, \ldots, \Delta$ (resp. $\xi i \vdash \Delta$). Let (ξ, I) be a negative (resp. positive) action. We indicate by $x \circ \mathfrak{D}$ the graph obtained as follows:*

1. *add a node $x = (\xi, I)$ to $\mathfrak{D}$;*
2. *add an edge $x \leftarrow k$ for each node k which uses an address ξi (for some $i \in I$).*

If (ξ, I) is positive, the result is always an L-net (on the interface $\vdash \xi, \Delta$). If (ξ, I) is negative, the result is a possibly partial L-net (on the interface $\xi \vdash \Delta$). The condition Positivity is satisfied only if at least one of the addresses ξi is used in $\mathfrak{D}$.

Definition 6.5 (Boxing). *Let $\mathfrak{D}$ be a positive L-net (of interface $\vdash \xi i, \xi j, \ldots, \Delta$) and let (ξ, I) be a negative action, with $i, j, \ldots \in I$. We indicate by $x.\mathfrak{D}$ the graph obtained as follows:*

1. *add a node $x = (\xi, I)$ to $\mathfrak{D}$;*
2. *add an edge $x \leftarrow k$ for each node k which belongs to a conclusion of $\mathfrak{D}$.*

The result is clearly an L-net of interface $\xi \vdash \Delta$.

Remark 6.6. *Note that the edges by the boxing construction survive in the transitive reduction, while some of the edges added by the rooting construction may be just "transitivity" edges that do not add anything to the underlying transitive reduction.*

On positive L-nets, rooting and boxing give us two choices for adding a new negative node:

- *Rooting is a parallel operation,* in the sense that it only adds the minimum amount of sequentiality that is needed for condition Parents to hold.
- *Boxing instead is a serial (sequential) operation*, which adds a maximal amount of sequentiality. If we think in terms of proof nets, *boxing corresponds to enclosing $\mathfrak{D}$ in a box, which has x as principal port.*

As we will see in Section 10, repetitive and consistent use of rooting and boxing will lead to (abstract versions of) proof nets and sequent calculus proofs, respectively.

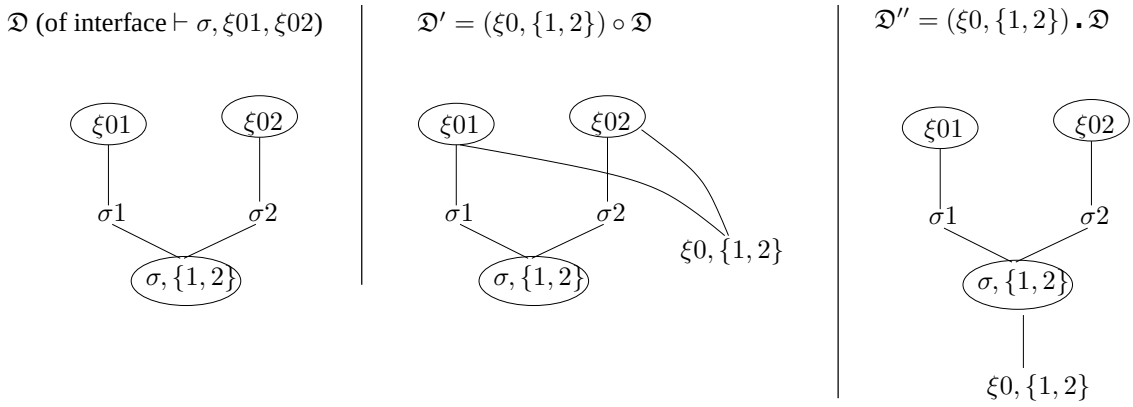


Figure 8: Rooting and boxing

Examples. In Figure 8 we give an example of both constructions. Given the L-net $\mathfrak{D}$ of interface $\vdash \sigma, \xi 01, \xi 02$, we add a negative action $(\xi, \{1,2\})$. The L-net $\mathfrak{D}'$ is obtained by rooting, while $\mathfrak{D}''$ results from boxing. Again, we can see the positive action $(\sigma, \{1,2\})$ as a Tensor, and the negative action $(\xi 0, \{1,2\})$ as a Par. We can see $\mathfrak{D}'$ as a kind of proof net, while $\mathfrak{D}''$ is a tree, corresponding to a sequent calculus proof.

Remark 6.7. *Rooting and boxing are two extremes. In between, we can define intermediate operators which add, on top of rooting, as much sequentiality as we wish: after rooting, we add any number of edges from positive nodes to x. Let us indicate this (generically) by $x \triangleleft \mathfrak{D}$. Hence, considering the respective designs as partial orders, we have:*

$$(x \circ \mathfrak{D}) \quad \subseteq \quad (x \triangleleft \mathfrak{D}) \quad \subseteq \quad (x \centerdot \mathfrak{D}) \,.$$

The differences between the three L-nets are only the amount of order between x and the nodes of $\mathfrak{D}$.

6.3. Constructing a positive rule

Let us call $\mathfrak{D}_1$ the L-net $\sigma_1 \centerdot \xi 01^+$, and $\mathfrak{D}_2$ the L-net $\sigma_2 \centerdot \xi 02^+$. Let $k = (\sigma, \{1,2\})$. Then the design $\mathfrak{D}$ of Figure 8 can be assembled as follows: $\mathfrak{D} = (k \circ \mathfrak{D}_1) \bigcup (k \circ \mathfrak{D}_2)$ (note the superposition of the two occurrences of k). Observe on the other hand that $\mathfrak{D}_1 \bigcup \mathfrak{D}_2$ is not an L-net (there are two negative conclusions).

The correctness of the construction relies on the following property.

Proposition 6.8. *If two positive* $\mathrm{L_S}$*-nets are of the form $k^+ \circ \mathfrak{D}_1, k^+ \circ \mathfrak{D}_2$ and are such that the sets of addresses used by $\mathfrak{D}_1$ and $\mathfrak{D}_2$ are disjoint, then $(k^+ \circ \mathfrak{D}_1) \cup (k^+ \circ \mathfrak{D}_2)$ is an* $\mathrm{L_S}$*-net.*

Proof. Condition Additives is obviously inherited by the disjointness assumption. We show that this also true of condition Cycles. Suppose that there is a switching cycle traversing both $\mathfrak{D}_1$ and $\mathfrak{D}_2$, and consider two minimal portions of the cycle going from

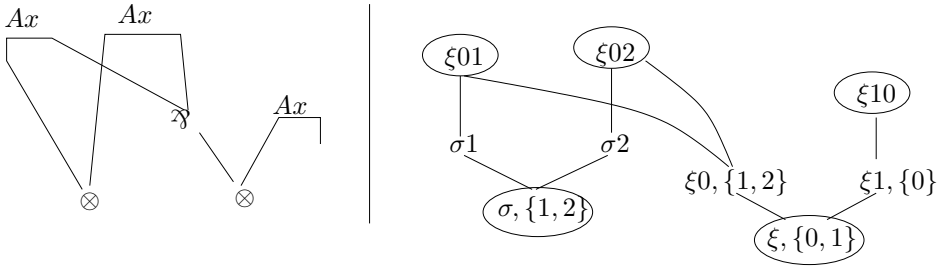


Figure 9: Splitting

$\mathfrak{D}_1$ to $\mathfrak{D}_2$ and from $\mathfrak{D}_2$ to $\mathfrak{D}_1$, respectively. At most one of these portions can go through k. Thus, the other portion consists of two consecutive nodes $\mathfrak{c}_1$ and $\mathfrak{c}_2$, with, say, $\mathfrak{c}_1 \in \mathfrak{D}_1$, $\mathfrak{c}_2 \in \mathfrak{D}_2$, and $\mathfrak{c}_1 \sqsubset_1 \mathfrak{c}_2$, contradicting the disjointness assumption. □

Remark 6.9. *We write* $(k^+ \circ \mathfrak{D}_1) \cup (k^+ \circ \mathfrak{D}_2)$ *instead of* $k \circ (\mathfrak{D}_1 \uplus \mathfrak{D}_2)$*, because* $\mathfrak{D}_1 \uplus \mathfrak{D}_2$ is not *an L-net, according to our definition.*

In fact, only the superpositions of this form (which are hence morally disjoint unions) will be needed in the purely multiplicative case. Note that the "real" superposition as we have seen in the Example 1 (Figure 6) involves an additive rule (to get an intuition, the reader can take a look also at Figure 11).

6.4. Removing a negative unary rule: root removal

The following operation allows us to decompose a *negative L-net*, whose negative conclusion is *unary* (this is the only negative destructor we need in the case of a purely multiplicative L-net).

Definition 6.10 (Root removal). *Given an L-net* $\mathfrak{D}$ *(of interface* $\xi \vdash \Delta$*) with a negative unary conclusion* $\{x\}$ *with* $x = (\xi, I)$*, we indicate by* $\mathfrak{D} \setminus x$ *the graph obtained from* $\mathfrak{D}$ *by removing* x*.*

It is immediate that the result is an L-net on the interface $\vdash \ldots, \xi i, \ldots, \Delta$ ($i \in I$).

6.5. Removing a positive rule: splitting

We next state a key lemma, relying on the notion of splitting rule, that allows us to decompose a *positive* $\mathrm{L_S}$*-net* into disjoint components, where each component is itself an $\mathrm{L_S}$-net.

The notion of splitting. The notion of splitting comes from the theory of proof nets: given a proof net $\mathfrak{R}$ whose conclusions are all positive (Tensor), a conclusion is splitting if by deleting it, $\mathfrak{R}$ splits into two connected components $\mathfrak{R}_1, \mathfrak{R}_2$, which are themselves proof nets. This allows us to (inductively) decompose $\mathfrak{R}$ into proof nets of smaller size. Observe that not any positive conclusion is splitting. Let us consider the proof net in Figure 9 (left). The right-most Tensor is splitting, while the left-most is not.

The same notion immediately translates in our setting. Let us consider the $\mathrm{L_S}$-net given in Figure 9 (right). The node $(\xi, \{0,1\})$ is splitting, while the node $(\sigma, \{1,2\})$ is not, because by deleting it we do not have an $\mathrm{L_S}$-net. If we delete the node $(\xi, \{0,1\})$, we obtain two $\mathrm{L_S}$-nets: $\mathfrak{D}'$ (as in Figure 8) and $\mathfrak{C} = (\xi 1, \{0\}) \centerdot (\xi 10)^+$. The $\mathrm{L_S}$-net illustrated in the picture can indeed be written as

$$(\xi, \{0,1\}) \circ \mathfrak{D}' \bigcup (\xi, \{0,1\}) \circ \mathfrak{C}$$

Let us formulate this more formally.

Definition 6.11 (Splitting rules). 1. *A negative rule $W = \{\ldots, w_I, \ldots\}$ of an L-net $\mathfrak{D}$ is called splitting if either it is conclusion of $\mathfrak{D}$ (each w_I is a root), or if after deleting the edges $w \leftarrow w_I$ to the common predecessor (for all $w_I \in W$), there is no more connection (i.e., no sequence of consecutive edges) between any of the w_I's and w.*

2. *A positive rule of $\mathfrak{D}$ is called splitting if it is a conclusion and all negative rules just above it are splitting.*

Lemma 6.12 (Splitting Lemma). *Every $\mathrm{L_S}$-net $\mathfrak{D}$ has a splitting conclusion. In particular, if all the conclusions are positive (i.e., if $\mathfrak{D}$ is positive), there is at least one positive splitting rule.*

The proof is given in Appendix Appendix B.

As a consequence of the Splitting Lemma, we have the following property.

Proposition 6.13 (Splitting). *Let $\mathfrak{D}$ be an $\mathrm{L_S}$-net. If $\mathfrak{D}$ is positive, then there exists a positive conclusion $k = (\xi, I)$, which we call a splitting conclusion of $\mathfrak{D}$, such that*

$$\mathfrak{D} = (\bigcup_{i \in I} (k \circ \mathfrak{D}_i)) \biguplus \mathfrak{C} \,.$$

where all $\mathfrak{D}_i$'s and $\mathfrak{C}$ are $\mathrm{L_S}$-nets and do not share addresses.

Proof. Let $k = (\xi, I)$ be a splitting positive conclusion. By deleting k, the graph splits into several connected components. Let us indicate by $\mathfrak{D}_i$ the part of the graph which is connected to some nodes of address ξi, and let us indicate by $\mathfrak{C}$ the rest of the graph.

1. It is immediate that both $\mathfrak{C}$ and each $k \circ \mathfrak{D}_i$'s are downward closed, and hence are $\mathrm{L_S}$-nets by Lemma 6.1. It follows readily that $\mathfrak{D}_i = (k \circ \mathfrak{D}_i) \setminus k$ is an $\mathrm{L_S}$-net, for all i.

2. Suppose for a contradiction that there are two nodes k_1, k_2, say in $\mathfrak{D}_i$ and in $\mathfrak{D}_j$ using the same address. By condition Additives applied to $\mathfrak{D}$, there exists an additive pair w_1, w_2, with w_1, w_2 below k_1, k_2, respectively, which by downward closedness implies $w_1 \in \mathfrak{D}_i$ and $w_2 \in \mathfrak{D}_j$. This is impossible because all the nodes in any negative rule W of $\mathfrak{D}$ belong to the same connected component. □

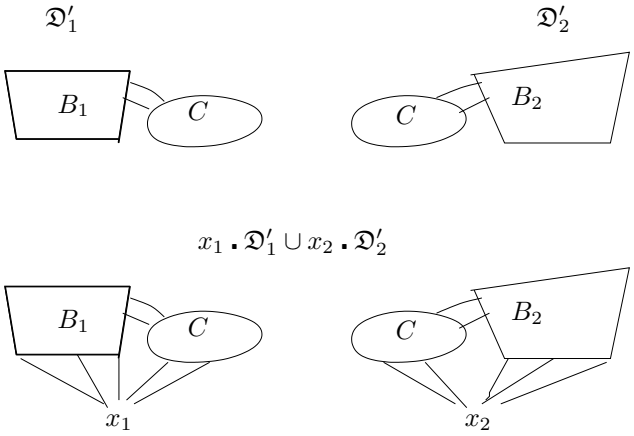


Figure 10: Marking choices without sharing

An example of a non-splitting L-net. The L-net given in Figure 5 does not satisfy condition Cycles. Observe in fact that all conclusions are positive, but none of them is splitting. As a consequence, we have no way to decompose this L-net into L-nets of smaller size.

6.6. Constructing: additive union

In this section and the following one, we introduce the operations which deal with additive structure. We first give an informal intuition for the constructions.

As we have already mentioned in Section 2.3, the additives convey the notion of choice between possible different evolutions of the system. Each choice is marked by a *different* action on the same name, such as for example $x_1 = (\xi, I_1)$ and $x_2 = (\xi, I_2)$ (with $I_1 \neq I_2$).

For example, consider the two possible evolutions described by the (generic) L-nets $\mathfrak{D}'_1$ and $\mathfrak{D}'_2$ given at the top of Figure 10 (see also Section 9.2 for a concrete example). Using the operations introduced so far (boxing and superposition), we could obtain $(x_1 \centerdot \mathfrak{D}'_1) \cup (x_2 \centerdot \mathfrak{D}'_2)$ (bottom of the figure). The additive rule $X = \{x_1, x_2\}$ marks the branching. Given the resulting L-net, we can select a possible evolution by selecting one of the components of X, and consider the sub-net.

Now, as suggested by the drawings, let us assume that a part of the evolution is common to the two possible choices: each $\mathfrak{D}'_i$ can be partitioned into B_i and C, where the part B_i is specific to $\mathfrak{D}'_i$, while the part C is common to the two possible evolutions (i.e. $\mathfrak{D}'_1 \cap \mathfrak{D}'_2 = C$) Instead of duplicating C as we have just done by boxing x_1 (resp. x_2) below $\mathfrak{D}'_1$ (resp. $\mathfrak{D}'_2$), we want to *share* the common part C. This is the purpose of the "additive union" (see figure 11).

Specifically, we start with a collection of L-nets $\mathfrak{D}_i$, where each one has a unary negative conclusion of the form $x_i = (\xi, I_i)$ (for example, we could have $\mathfrak{D}_i = x_i \circ \mathfrak{D}'_i$). We then obtain their additive union by proceeding in two steps:

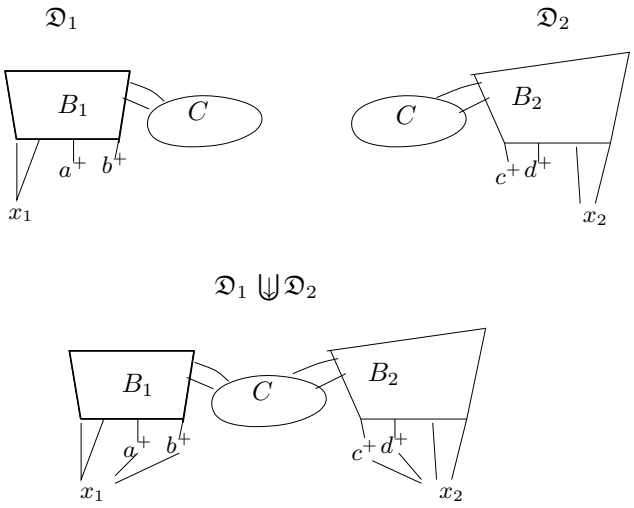


Figure 11: Additive union

1. We make the dependency from a choice explicit, by adding edges towards the additive component x_i for all actions that are not common to all evolutions (i.e. all actions in B_i).
2. We *share* the part C that is common by superposing the common views (cf. Remark 6.3).

The operation is illustrated in figure 11. We can now give the formal definition

Definition 6.14 (Additive union). *Let $\mathfrak{D}_I, \mathfrak{D}_J, \ldots$ be a collection of L-nets which have respective negative* unary *conclusions $x_I = (\xi, I)$, $x_J = (\xi, J), \ldots$ on the same address ξ (with distinct $I, J, \ldots$). Their additive union $\Downarrow_I \mathfrak{D}_I$ is defined as the following superposition:*

$$\Downarrow_I \mathfrak{D}_I = \bigcup_I \Phi(\mathfrak{D}_I) \ ,$$

where each $\Phi(\mathfrak{D}_I)$ is obtained from $\mathfrak{D}_I$ by adding edges in such a way that $x_I \leftarrow k$ (in $\Phi(\mathfrak{D}_I)$ for each positive node $k \in \mathfrak{D}_I$ such that $\ulcorner k \urcorner \notin \mathfrak{D}_J$ (for some $J \neq I$).

Intuitively, Φ is a function on views which *marks* with an edge towards x_I the views of $\mathfrak{D}_I$ which are specific to it, or more precisely those views which are *not shared* by all $\mathfrak{D}_J$'s.

Remark 6.15. 1. *Notice that if the set of $\mathfrak{D}_I$'s is a singleton, then $\Phi(\mathfrak{D}_I) = \mathfrak{D}_I$. This observation allows us to treat both unary and non unary conclusions as a single case.*

2. *If $\mathfrak{D}_I = x_I \centerdot \mathfrak{C}_I$, $\mathfrak{D}_J = x_J \centerdot \mathfrak{C}_J, \ldots$, then $\Phi(\mathfrak{D}_I) = \mathfrak{D}_I$ for all I, and hence additive union boils down to superposition (in this case, just disjoint union): $\Downarrow_I \mathfrak{D}_I = \biguplus_I \mathfrak{D}_I$.*

The following two lemmas will play a crucial role in the decomposition of L-nets.

Lemma 6.16. *With the notations of Definition 6.14, and assuming that there are at least two designs $\mathfrak{D}_I, \mathfrak{D}_J$ in the collection, we have the following partitions:*

1. *The views of $\Phi(\mathfrak{D}_I)$ can be split into two disjoint sets:*

$$\Phi(\mathfrak{D}_I) = C \uplus B_I \ ,$$

where

$$C = \bigcap_J \mathfrak{D}_J = \{\mathfrak{c} : \ \mathfrak{c} \in \mathfrak{D}_J, \text{ for all } J\}$$
$$B_I = \{\ulcorner k \urcorner \in \Phi(\mathfrak{D}_I) : \ x_I \overset{+}{\leftarrow} k\} = \{\mathfrak{c} \in \Phi(\mathfrak{D}_I) : \ (\xi, I) \in \mathfrak{c}\}$$

2. *If $\mathfrak{D} = \biguplus_I \ \mathfrak{D}_I$ then $\mathfrak{D} = C \uplus (\biguplus_I \ B_I)$.*

Proof.

1. If a view $\mathfrak{c}$ of $\Phi(\mathfrak{D}_I)$ does not belong to B_I, then, by construction, no edge has been added, which implies both that $\mathfrak{c}$ is a view of $\mathfrak{D}_I$ and that it belongs to all $\mathfrak{D}_J$'s ($J \neq I$). Thus $\Phi(\mathfrak{D}_I) = C \cup B_I$. Moreover, if $(\xi, I) \in \mathfrak{c}$, then $\mathfrak{c}$ cannot belong to any $\mathfrak{D}_J$ ($J \neq I$), as this would entail that $\mathfrak{D}_j$ would have a node labelled by (ξ, I), contradicting the assumption that $\mathfrak{D}_J$ has a *unary* conclusion. Hence the union is disjoint.
2. If $\mathfrak{D} = \biguplus_I \ \mathfrak{D}_I = \bigcup_I \ \Phi(\mathfrak{D}_I)$, we can write $\mathfrak{D} = \bigcup_I \ (C \uplus B_I) = C \uplus (\bigcup_I \ B_I)$. Finally, the B_I's are all pairwise disjoint since a view cannot contain (ξ, I) and (ξ, J) with I, J distinct.

□

Proposition 6.17. *$\biguplus_I \ \mathfrak{D}_I$ is an L-net. Moreover the construction preserves condition* Cycles*.*

Proof. It is immediate that each $\Phi(\mathfrak{D}_I)$ is an L-net. All properties are inherited from $\mathfrak{D}$. As for condition Cycles, notice that all the newly added edges enter x_I, and no switching path which uses the new edges to x_I can continue to form a cycle. Hence all $\Phi(\mathfrak{D}_I)$'s are $\mathrm{L_S}$-nets.

We are left to show (cf. Section 6.3) that $\biguplus_I \ \mathfrak{D}_I = \bigcup_J \ \Phi(\mathfrak{D}_J)$ satisfies conditions Additives and Cycles. We just check condition Cycles. It is convenient to partition the nodes of $\biguplus_I \ \mathfrak{D}_I$ as in Lemma 6.16.

Assume that a collection of switching cycles is contained inside one of the $\Phi(\mathfrak{D}_J)$'s: in such a case the additive pair is given by the condition applied to $\Phi(\mathfrak{D}_J)$.

Otherwise, we have at least a node $k_1 \in B_I$ and a node $k_2 \in B_J$ (with $I \neq J$) traversed by the cycles. By construction, $x_I \overset{+}{\leftarrow} k_1$ and $x_J \overset{+}{\leftarrow} k_2$, and condition Cycles is satisfied. □

Remark 6.18. *Notice that $\bigcap \mathfrak{D}_J$ is not an L-net in general , because its maximal views do not need to be positive.*

6.7. Removing an additive rule: scoping

Root removal is all we need to decompose a negative slice. But in the general setting where additive rules are not all unary, we need to define a more complex operation. Given an L-net whose conclusion is an additive rule $X = \{x_1, x_2\}$, we can retrieve the evolution corresponding to the choice of x_1 by deleting all actions that depend on x_2. In this way, we recover the actions that are specific to the choice marked by x_1, as well as the actions that are common to the other possible evolutions. We call this operation scoping.

Definition 6.19 (Scoping). *Let $\mathfrak{D}$ be an L-net of negative conclusion $X = \{x_I : I \in \mathcal{N}\}$, where $x_I = (\xi, I), x_J = (\xi, J), \ldots$. For all $I \in \mathcal{N}$, we define the* scope of x_I in $\mathfrak{D}$ *as follows:*

$$\mathit{Scope}(x_I, \mathfrak{D}) = \{\mathfrak{c} : \exists\, \mathfrak{c}' \ (\mathfrak{c} \sqsubseteq \mathfrak{c}', \mathfrak{c}' \in \mathfrak{D}, \mathfrak{c}' \textit{positive}, \textit{ and } (\forall\, J \in \mathcal{N} \setminus \{I\} \ x_J \notin \mathfrak{c}'))\} \ .$$

or, equivalently:

$$\mathit{Scope}(x_I, \mathfrak{D}) = \bigcup \{k^{\downarrow} : k \textit{ positive and } (\forall\, J \in \mathcal{N} \setminus \{I\} \ x_J \not\frown k)\} \ .$$

By Lemma 6.1, if $\mathfrak{D}$ is an L-net (resp. an $\mathrm{L_S}$-net), $\mathit{Scope}(x_I, \mathfrak{D}) \subseteq \mathfrak{D}$ is an L-net (resp. an $\mathrm{L_S}$-net).

Lemma 6.20. *If $\mathfrak{D} = \biguplus_I \mathfrak{D}_I$, then:*

1. *$\mathit{Scope}(x_I, \mathfrak{D}) = \Phi(\mathfrak{D}_I)$ (for all I indexing the additive union).*
2. *Assume $\mathfrak{D}_I = x_I \circ \mathfrak{C}_I$, or $\mathfrak{D}_I = x_I \,.\, \mathfrak{C}_I$. Then:*

$$\mathit{Scope}(x_I, \mathfrak{D}) \setminus x_I \ = \ \mathfrak{C}_I \ .$$

Proof.

1. By Lemma 6.16, we can write $\mathfrak{D} = C \uplus (\biguplus_J B_J)$. We show:
   - $B_I \subseteq \mathit{Scope}(x_I, \mathfrak{D})$. Indeed, if $(\xi, I) \in \mathfrak{c}$, then c can contain no other action on the same address.
   - $C \subseteq \mathit{Scope}(x_I, \mathfrak{D})$. Cf. the proof of Lemma 6.16, where we have established that a view in C cannot contain any action (ξ, J).
   - $B_J \bigcap \mathit{Scope}(x_I, \mathfrak{D}) = \emptyset$ $(J \neq I)$. This is obvious since a view in B_j contains by definition an action (ξ, J).

   It follows that $\mathit{Scope}(x_I, \mathfrak{D}) = C \uplus B_I = \Phi(\mathfrak{D}_I)$.
2. This follows immediately from 1, since all what Φ does to $\mathfrak{D}_I$ is undone when x_I is removed.

□

Note that when $X = \{X_I\}$ is unary, then scoping boils down to identity, i.e., $\mathit{Scope}(x_I, \mathfrak{D}) = \mathfrak{D}$, and the more complex operation $\mathit{Scope}(x_I, \mathfrak{D}) \setminus x_I$ boils down to simple root removal.

6.8. Using constructors and destructors

Summing up the content of this section, the operators that we have presented can be grouped into two families:

- Rooting, boxing, superposition, and additive union are *constructors*.
- Root removal, splitting, and scoping are *destructors*. The decomposition of an $\mathrm{L_S}$-net goes as follows:

  $\mathfrak{D}$ *is positive.* All conclusions are positive. If there are no negative rules, we are done: $\mathfrak{D}$ is reduced to its conclusions.
  Otherwise, by splitting, we get $\mathfrak{D} = (\bigcup_i \ (x \circ \mathfrak{D}_i)) \uplus \mathfrak{C}$. Hence

  $$\mathfrak{D} \quad \text{decomposes into} \quad \dots \ , \ \mathfrak{D}_i \ , \ \dots \ , \ \mathfrak{C} \ .$$

  $\mathfrak{D}$ *is negative* (and non empty). Let X be the unique negative conclusion of $\mathfrak{D}$.

  i. If X is unary, we decompose $\mathfrak{D}$ by root removal.

  ii. Otherwise, we reduce $\mathfrak{D}$ to the previous case by scoping.

  Altogether, if $X = \{\dots, x_I, \dots\}$,

  $$\mathfrak{D} \quad \text{decomposes into} \quad \dots \ , \ \mathit{Scope}(x_I, \mathfrak{D}) \setminus x_I \ , \ \dots \ .$$

These constructors and destructors are put to use in the following sections.

7. Sequentializing a graph strategy

An edge of an L-net states a dependency, or a precedence among actions. The aim of this section is to provide a procedure, which takes an $\mathrm{L_S}$-net $\mathfrak{D}$ and returns an L-forest, which is obtained by adding enough such dependency edges to $\mathfrak{D}$. Let us consider a very simple example: a (partially ordered) view $\mathfrak{c}$.

A sequentialization of $\mathfrak{c}$ is a linear extension of the partial order. That is, we add sequentiality (edges) to obtain a total order. A total order that extends c defines a complete scheduling of the tasks, respecting the constraint that each action is performed only after all of its original constraints are satisfied.

Dependency between the actions of a slice, and of sets of slices (L-nets) is more subtle, as there are also global constraints. The key point in the sequentialization is to select a rule that does not depend on others. This is exactly the role of the Splitting Lemma, and the reason for the condition Cycles.

The process of sequentialization is non-deterministic, as one can expect, i.e., there are several tree strategies which can be associated to the same $\mathrm{L_S}$-net.

As we have both multiplicative and additive structure, when sequentializing we will perform two tasks:

1. add sequentiality (sequential links) until the order in each view is completely determined;
2. separate slices which are shared through additive superposition.

Sequentialization procedure. The following procedure *progressively* transforms a *potentially infinite* $\mathrm{L_S}$-net $\mathfrak{D}$ into an L-forest on the same interface as $\mathfrak{D}$. It works bottom-up and follows the paradigm of lazy, stream-like computation.

The procedure is non-deterministic. In what follows, $\mathfrak{D}' = \mathit{seq}(\mathfrak{D})$ should be read as: "$\mathfrak{D}'$ is a possible sequentialization of $\mathfrak{D}$".

A) $\mathfrak{D}$ is negative. Let $X = \{\dots, x_I, \dots\}$ be the unique negative conclusion of $\mathfrak{D}$. Let $\mathfrak{D}_I = \mathit{Scope}(x_I, \mathfrak{D}) \setminus x_I$, for all I. Then:

- $\mathit{seq}(\mathfrak{D}) = \bigcup_I \ (x_I \centerdot \mathit{seq}(\mathfrak{D}_I))$.

Here, boxing and scoping take care of the two tasks mentioned above, respectively. Note that nodes lying in some $\mathit{Scope}(x_I, \mathfrak{D}) \bigcap \mathit{Scope}(x_J, \mathfrak{D})$ are duplicated.

B) $\mathfrak{D}$ is positive.

1. Assume that $\mathfrak{D}$ is connected (in the ordinary graph-theoretic sense).

If $\mathfrak{D}$ consists of a single positive node, we are done.

Otherwise we select a positive splitting rule $x = (\xi, I)$ and proceed as follows. By Proposition 6.13, each of the components $\mathfrak{D}_i$ obtained by splitting is an $\mathrm{L_S}$-net whith a negative conclusion on an address ξi. Then:

- $\mathit{seq}(\mathfrak{D}) = \bigcup_i \ (x \circ \mathit{seq}(\mathfrak{D}_i))$.

2. Assume $\mathfrak{D} = \biguplus_i \ \mathfrak{C}_i$, where the $\mathfrak{C}_i$'s are the connected components of $\mathfrak{D}$. Then:

- $\mathit{seq}(\mathfrak{D}) = \biguplus_i \ (\mathit{seq}(\mathfrak{C}_i))$.

Proposition 7.1. *If $\mathfrak{D}$ is an $\mathrm{L_S}$-net on the interface $\Xi \vdash \Delta$, $\mathit{seq}(\mathfrak{D})$ is an L-forest on the same interface.*

Proof. We have already established all partial results needed to prove this. □

This procedure applies to infinite L-nets, by coinduction. Indeed, one can formally show that L-forests form a final coalgebra and the $\mathrm{L_S}$-nets form a coalgebra for a functor F on sets, and that *seq* is the associated unique coalgebra morphism. We only sketch the construction below. The reader unfamiliar with final coalgebra semantics can get the necessary background from, say [36].

- One considers the functor F in the category of sets and functions that takes a set X to the disjoint union of the set of all finite sets whose elements are of the form $((\xi, I), \{\dots, a_i, \dots\})$, where the a_i's form a collection of elements of X indexed by a subset of I, and of the set of all $\{\dots, ((\zeta, J), a_J), \dots\}$, where the a_J's form a collection of elements of X indexed by some $\mathcal{N} \subseteq \mathcal{P}_f(\omega)$.

- One proves that the collection of all L-forests forms a final coalgebra for this functor. The situation is similar to that of, say, Böhm trees. The coalgebra structure takes a positive (respectively negative) L-forest and decomposes it into its root(s) and its immediate subforests.

- Thanks to the Splitting Lemma, one can choose a decomposition for each positive $\mathrm{L_S}$-net, and codify this "oracle" in the form of a coalgebra structure on the collection of all $\mathrm{L_S}$-nets.
- Then *seq* (along the oracle) is the unique coalgebra morphism from this coalgebra to the final coalgebra. That it is a coalgebra morphism amounts to the equations given above to define $\mathit{seq}(\mathfrak{D})$.

More concretely, the progressive construction of $\mathit{seq}(\mathfrak{D})$ yields at any step an actual finite part Π – the part of the forest that has been already recognized –, and a collection of $\mathrm{L_S}$-nets to sequentialize, each associated with a leaf of Π.

8. Desequentializing a tree strategy

In order to define the desequentialization procedure, we need to introduce a new notion, that of decoration.

Making the axioms explicit: decorations. In Section 4, we have illustrated the purpose of desequentialization by taking as example the relation between proof nets and sequent calculus proofs. Our aim is to remove some artificial sequentialization, while preserving essential information:

1. *axioms* (multiplicative proof net = formula tree + axioms [30]);
2. *dependency due to additive rules*: some nodes must not be shared.

The second issue is addressed by our definition of additive union (cf. Section 6.6). As for axioms, such information is present in the source L-forest (or design), but is *implicit* (and not uniquely determined). To make the information on the axioms explicit, we introduce an auxiliary notion: we decorate each leaf k with a set of addresses, which we denote by $\mathit{link}(k)$.

The notion of decoration is closely related to the sequent calculus presentation of an L-forest. In Section 2.2, we already sketched how to move from an L-forest to an explicit sequent calculus style presentation. Given an L-forest Π, one has to associate to each node k of Π a sequent of addresses. In particular, each leaf $k = (\xi, I)$ in the forest should correspond to a *generalized axiom* in the sequent calculus proof, of either of the two forms

$$\frac{}{\vdash \xi, \Gamma}\ k = (\xi, I)^+ \qquad \frac{}{\vdash \Gamma}\ k = \dagger$$

(see Appendix Appendix A for details). This is where decoration helps: the sequent associated with a leaf k can be inferred from $\mathit{link}(k) = \{\xi_1, \ldots, \xi_n\}$: if k has label (ξ, I) (resp. $\dagger$), the sequent is $\vdash \xi, \xi_1, \ldots, \xi_n$ (resp.$\vdash \xi_1, \ldots, \xi_n$). The other way around, starting from a proof, we transfer the sequent information of the generalised axioms to the link sets.

Decorated leaves as boxes. The idea of decoration is also associated with that of *truncation*. Suppose that we truncate the tree Π after the node k, leaving out the subtrees Π_i, above k. The sequent associated to the node k, which is now a leaf, is the interface of the subtree $\bigcup\ (k \circ \Pi_i)$. Hence, the addresses in *link*(k) are meant as the addresses which are used in the Π_i's. In this sense, a decorated leaf acts as a sort of (black) box: we hide the content of the box (i.e., $\bigcup\ (k \circ \Pi_i)$) and only keep memory of the interface (the conclusion of the box).

Link sets versus infinitary expansions. In ludics, identity axioms are interpreted by *infinitary* strategies, called faxes in [13]. These strategies are an instance of the copy-cat strategies of game semantics. In such infinitary strategies, every generated action is eventually used. Faxes are the typical example of what we want to enclose in a box. Actually, even if we are establishing general results, the kind of strategies we are really interested in are those corresponding to proofs. Morally, the (real) use of link sets is to deal with finite truncations of these strategies. Seen from that point of view, link sets are a way to express (in a finitary way) the axioms.

Definition 8.1. *A* decorated $\mathrm{L_S}$*-net is an* $\mathrm{L_S}$*-net* $\mathfrak{D}$ *in which all leaves* k *are equipped with a finite set* $link(k)$ *of addresses (called the link set of* k*), in such a way that the conditions on* $\mathrm{L_S}$*-nets hold with respect to all addresses (thus, including those in the link sets). The label of a node is now a decorated action, i.e., an action possibly together with a link set . Two labels* $((\xi, I), \Lambda_1)$ *and* $((\xi, I), \Lambda_2)$*, with* $\Lambda_1 \neq \Lambda_2$*, are considered different*[6].

We still use $\mathfrak{D}$ *to denote a decorated* $\mathrm{L_S}$*-net.*

Observe that if $\mathfrak{D}$ is an L-net, and given an assignment of link sets to the leaves of $\mathfrak{D}$, all what we have to check for it to yield a decorated $\mathrm{L_S}$-net are conditions Parents and Additives.

From now on, we extend the definition of "a node uses an address" as follows:

Definition 8.2 (used addresses). *Let* k *be a node labelled by an action and possibly a link set. We say that the node* k *uses an address* ξ *if either* ξ *is the address of the action, or* appears in the link set *of* k.

The extension of the operators introduced in Section 6 to decorated L-nets is immediate. The extended definition plays a role only when rooting an L-net on a negative action: now (with the notation of Definition 6.4), we add[7] an edge $(\xi, I) \leftarrow k$ for each node k such that k is generated by (ξ, I), or k is a leaf such that $\xi i \in link(k)$ for some i. We maintain the same notations as in Section 6.

The desequentialization procedure takes as input finite decorated L-forests. More precisely we choose a special decoration discipline. In Appendix Appendix A, we prove that it is always possible to choose a decoration which satisfies the following property.

[6] This is natural if we consider that they correspond to different axioms.

[7] These new edges are close in spirit to the μ-pointers introduced by Laurent in his investigations on game semantics for first-order (classical) logic [37].

Definition 8.3 (well-decorated). *A well decorated* $\mathrm{L_S}$*-net is a decorated* $\mathrm{L_S}$*-net* $\mathfrak{D}$ *such that all addresses of the interface, and all addresses generated by a negative action of* $\mathfrak{D}$ *are used in* $\mathfrak{D}$ *(in the sense of Definition 8.2).*

Lemma 8.4. *Every L-forest can be well decorated.*

Proof. See Corollary Appendix A.5. □

Desequentialization procedure. Let Π be a *finite* well-decorated L-forest. Its desequentialization *deseq*(Π) is defined by induction as follows:

Π *is negative.* Let $X = \{x_I, x_J, \ldots\}$ be the conclusion of Π. Let us call Π_I the subforest above x_I (i.e., $\Pi = \bigcup_I \ (x_I \centerdot \Pi_I)$). Then:

- *deseq*$(\Pi) = \biguplus_I \ (x_I \circ$ *deseq*$(\Pi_I))$.

Π *is positive.*

1. Assume that Π is a tree of conclusion x, using address ξ. If the tree is reduced to a single node, then we are done (base case). Otherwise, it has the form $\Pi = \bigcup_i \ (x \circ \Pi_i)$, where each Π_i is the subforest of all the trees on the address ξi ($i \in I$). Then:

   - *deseq*$(\Pi) = \bigcup_i \ (x \circ$ *deseq*$(\Pi_i))$.

2. Assume $\Pi = \biguplus_i \ \Pi_i$. Then:

   - *deseq*$(\Pi) = \biguplus_i \ ($*deseq*$(\Pi_i))$.

Remark 8.5. *One checks easily that the sets of labels of* Π *and deseq*(Π) *are the same (intuitively, no node is deleted), and that if* l *is the label of a leaf in* Π*, it also labels a leaf in deseq*(Π).

Proposition 8.6. *If* Π *is a finite well decorated L-forest on the interface* $\Xi \vdash \Delta$*, then deseq*(Π) *is an* $\mathrm{L_S}$*-net on the same interface.*

Proof. That *deseq*(Π) is a partial $\mathrm{L_S}$-net is a consequence of Propositions 6.17 and 6.8 (that the hypotheses of the latter are met is a consequence of Remark 8.5).

Condition Positivity follows from Lemma 8.7 below, and by (the upgraded) definition of rooting. □

Notice that decorations play a role only to prove that *deseq*$(\mathfrak{D})$ satisfies condition Positivity: the *well-decorated assumption* guarantees that *no rooting involved in the construction is partial*.

Lemma 8.7. *Let* Π *be a well decorated L-forest on the interface* $\Xi \vdash \Delta$*. All the addresses of the interface are used in deseq*(Π).

Proof. Similar to the proof of Lemma 12.6. □

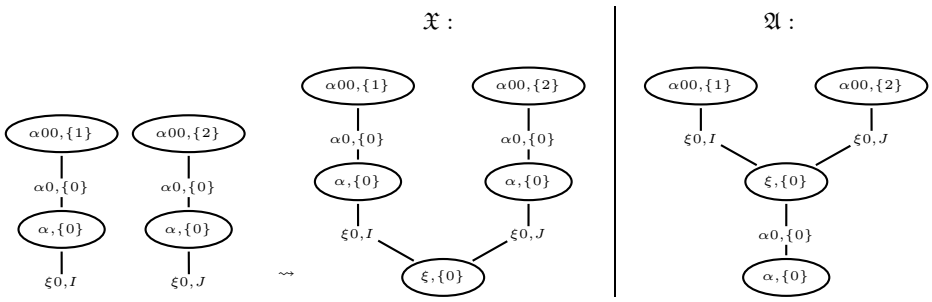


Figure 12: Two possible sequentializations

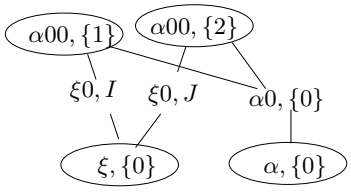
Remark 8.8. *Note that we have used* induction *rather than coinduction in this section. This is because the operations involved (additive union, rooting, decoration) do not lend themselves to coinduction, unlike (disjoint) union and boxing: say, boxing on one hand is a "black box" operation while rooting on the other hand requires visiting the structure of the L-net (to look for where to add new edges).*

We can think of the desequentialization of an infinite L-forest Π *as follows: take an (arbitrarily large) finite truncation* Π' *of* Π*, and graft appropriately on deseq*(Π') *the subforests that have been taken away by the truncation. The result depends on the choice of the truncation, and is not a parallel L-net (since the grafted trees are untouched), but it is the best one can hope.*

9. Examples of sequentialization and desequentialization

9.1. Sequentialization

Consider the following $\mathrm{L_S}$-net $\mathfrak{R}$:



(cf. Figure 2). We have two negative rules ($\{(\xi 0,I),(\xi 0,J)\}$ and $\{(\alpha 0,\{0\})\}$), and two positive conclusions, that are both splitting. To sequentialize, we choose one of them. If we choose $(\xi,\{0\})$, we obtain the two trees on the left-hand side of Figure 12, and then the design $\mathfrak{X}$. Instead, by choosing $(\alpha,\{0\})$ we obtain the design $\mathfrak{A}$ (on the right).

9.2. Desequentialization

Example 1. Desequentializing either $\mathfrak{A}$ or $\mathfrak{X}$ in the example of Section 9.1, equipped with the only possible uniform decoration

$$link(\alpha 00,1)=\{\xi 0*I\} \quad , \quad link(\alpha 00,2)=\{\xi 0*J\}\ ,$$

yields the original L-net $\mathfrak{R}$.

Example 2. Our next example is a variant of the previous one and illustrates the process of adding edges, on one hand because of rooting plus decoration, and of the other hand because of additive union. Let us consider the design in Figure 13, where we just omit an obvious negative action at the place of .... The only uniform decoration is:

$$link(b) = \{\alpha 001, \xi 0 * I\} \quad , \quad link(c) = \{\alpha 002, \xi 0 * J\} \ .$$

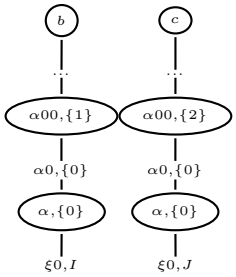


Figure 13: An L-forest ...

Following the desequentialization procedure, a few easy steps produce the two L-nets $\mathfrak{D}_1, \mathfrak{D}_2$, represented in Figure 14. Note that by the decoration, rooting has produced edges from b, c to $(\xi 0, I)$, $(\xi 0, J)$, respectively.

We then obtain $\mathfrak{D}'_1 = \Phi(\mathfrak{D}_1)$ by adding the relation $(\xi 0, I) \leftarrow (\alpha 00, \{1\})$, and $\mathfrak{D}'_2$ in a similar way. (Note that, say the edge from c to $(\xi 0, I)$ need not be shown anymore, since it "holds" by transitivity.)

Finally, the superposition $\mathfrak{D}'_1 \bigcup \mathfrak{D}'_2$ produces the L-net on the right-hand side of Figure 14.

9.3. Additives

As our last illustration, we resume the last example of Section 2.3. With the notation of that section, we have that desequentialization applied to either $\mathfrak{D}_1$ or $\mathfrak{D}_2$ yields $\mathfrak{R}$, and that we get either $\mathfrak{D}_1$ or $\mathfrak{D}_2$ back when sequentializing $\mathfrak{R}$, depending on whether we choose to start from $a\&b$ or from $c\&d$ (both $a\&b$ and $c\&d$ are splitting).

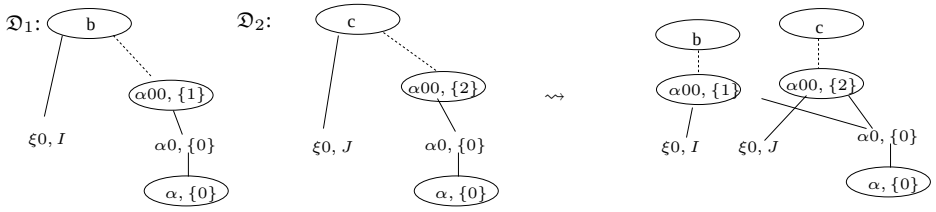


Figure 14: ... and its desequentialization

10. An algebraic presentation

In this section, we focus on the $\mathrm{L_S}$-nets generated by the constructors (rooting, boxing, superposition, and additive union), and we single out two important classes of $\mathrm{L_S}$-nets, obtained by consistently using rooting, or consistently using boxing, respectively (cf. Section 6.2). In the first case, we speak of *parallel* L-nets, which we regard as *abstract proof nets*. In the second case, we get the *L-forests*, which correspond to *abstract sequent calculus proofs*.

In the following, we denote with $\mathfrak{D}^+$ a positive L-net, and with $\mathfrak{D}^-_\sigma$ a negative L-net whose negative conclusion uses address σ. We denote by k^+ a (possibly decorated) positive action.

Abstract proof nets. A *parallel L-net* is an $\mathrm{L_S}$-net generated by the following grammar:

$$\begin{array}{lcl} \mathfrak{D} & ::= & \mathfrak{D}^+ \mid \mathfrak{D}^-_\sigma \\ \mathfrak{D}^+ & ::= & \mathfrak{E}^+ \uplus \ldots \uplus \mathfrak{E}^+ \\ \mathfrak{E}^+ & ::= & k^+ \mid \bigcup_{i\in I} ((\xi, I)^+ \circ \mathfrak{D}^-_{\xi_i}) \\ \mathfrak{D}^-_\sigma & ::= & \Biguplus_J (\sigma, J)^- \circ \mathfrak{D}^+ \end{array}$$

Such an $\mathrm{L_S}$-net has *minimal sequentiality*, in the sense that the we use constructors of minimal sequentiality.

Abstract sequent calculus proofs. The *sequential* L-nets are the L-nets generated by the following grammar:

$$\begin{array}{lcl} \mathfrak{D} & ::= & \mathfrak{D}^+ \mid \mathfrak{D}^-_\sigma \\ \mathfrak{D}^+ & ::= & \mathfrak{E}^+ \uplus \ldots \uplus \mathfrak{E}^+ \\ \mathfrak{E}^+ & ::= & k^+ \mid \bigcup_{i\in I} ((\xi, I)^+ \circ \mathfrak{D}^-_{\xi_i}) \\ \mathfrak{D}^-_\sigma & ::= & \bigcup_J ((\sigma, J)^- \centerdot \mathfrak{D}^+) \end{array}$$

It is clear that sequential L-nets and L-forests are one and the same thing.

In both syntaxes, the production rule $\mathfrak{D}^+ ::= \mathfrak{E}^+ \uplus \ldots \uplus \mathfrak{E}^+$ takes care of graphs that are not connected. Notice also that, by construction, both classes of L-nets hereditarily admit splitting.

Remark 10.1.
1. *In the description of* sequential *L-nets, we have chosen the form $x^+ \circ \mathfrak{D}$, but we could write it also as $x^+ \centerdot \mathfrak{D}$, the result being the same. When $\mathfrak{D}$ is an L-forest, positive rooting behaves in a "boxing-like" fashion in $x^+ \circ \mathfrak{D}$. By this observation, together with Remark 6.9, we see that the syntax of L-forests is essentially a combination of boxings and disjoint unions (as expected for a forest!).*
2. *In the production $\mathfrak{D}^-_\sigma ::= \bigcup_J ((\sigma, J)^- \centerdot \mathfrak{D}^+)$, we can replace the (disjoint) union by the (trivial) additive union symbol (cf Remark 6.15). Hence the difference between the two syntaxes indeed lies in a systematic use of rooting versus boxing.*

Last but not least, according to the discussions in Sections 7 and 8, the syntax of L-forests can be read coinductively, while the syntax of parallel L-nets is inductive, and hence defines finite graphs.

11. Relating sequential and parallel strategies

In this section, we study the relation between L-forests and parallel L-nets. We have already proved (Proposition 7.1) that for every $\mathrm{L_S}$-net, *seq*$(\mathfrak{D})$ is an L-forest. Conversely, the following is an immediate consequence of the definition of parallel L-nets.

Proposition 11.1. *For every finite L-forest* Π*,* *deseq*(Π) *is a parallel L-net.*

Every time we desequentialize an L-forest Π, there is a way to resequentialize back to Π.

Theorem 11.2. *Given a finite L-forest* Π*, there exists a strategy of sequentialization such that* $\Pi =$ *seq*(*deseq*(Π)).

Proof. We only consider the interesting cases.

Π *is negative.* Let us denote by $\{x_I, \dots\}$ the conclusion of the tree. Since $\Pi = \bigcup_I \ (x_I \,.\, \Pi_i)$, its desequentialization is *deseq*$(\Pi) = \biguplus_I \ (x_I \circ$ *deseq*$(\Pi_I))$. To sequentialize, we use scoping. By Lemma 6.20 (ii), *Scope*$(x_I, ($*deseq*$(\Pi))) \setminus x_I =$ *deseq*(Π_I). Hence

$$\textit{seq}(\textit{deseq}(\Pi)) = \bigcup_I \ (x_I \,.\, \textit{seq}(\textit{deseq}(\Pi_I))) \ .$$

Π *is positive.* If the root is x, $\Pi = \bigcup_i \ (x.\Pi_i)$. Since *deseq*$(\Pi) = \bigcup_i \ (x \circ$ *deseq*$(\Pi_i))$, to sequentialize it we select x as splitting rule. Removing x gets us back to the set of all *deseq*(Π_i)'s. Hence:

$$\textit{seq}(\textit{deseq}(\Pi)) = \bigcup_i \ (x \circ \textit{seq}(\textit{deseq}(\Pi_i))) \ .$$

□

Theorem 11.2 says that in the desequentialization there is no essential loss of information. All dependency (sequentialization) which is taken away can be restored.

Establishing a result in the opposite direction (i.e., *deseq*(*seq*$(\mathfrak{D})) = \mathfrak{D}$) only makes sense starting from a parallel L-net, because as *deseq*(Π) reduces sequentiality to a "minimal" amount, if $\mathfrak{D}$ is not parallel there is no hope that *deseq*(*seq*$(\mathfrak{D})) = \mathfrak{D}$.

Theorem 11.3. *If* $\mathfrak{R}$ *is a parallel L-net, it admits a sequentialization procedure such that* *deseq*(*seq*$(\mathfrak{R})) = \mathfrak{R}$.

Proof. Following the destructors, we are guaranteed (i) to have splitting, and (ii) that when we use scoping, we are in the situation described by Lemma 6.20. We just spell out the definitions.

- If $\mathfrak{R}$ *is negative*, we have $\mathfrak{R}^- = \biguplus_I \ (x_I \circ \mathfrak{R}_I)$. By definition of sequentialization, we have

$$\textit{seq}(\mathfrak{R}) = \bigcup_I \ (x_I \,.\, (\textit{Scope}(x_I, \mathfrak{R}) \setminus x_I)) \ .$$

But by Lemma, 6.20, we have $\mathit{Scope}(x_I, \mathfrak{R}) \setminus x_I = \mathfrak{R}_I$. Hence we have in fact $\mathit{seq}(\mathfrak{R}) = \bigcup_I \ (x_I \centerdot (\mathit{seq}(\mathfrak{R}_I))$, from which

$$\mathit{deseq}(\mathit{seq}(\mathfrak{R}^-)) = \bigdownarrow_I \ (x_I \circ \mathit{deseq}(\mathit{seq}(\mathfrak{R}_I)))$$

follows.

- If $\mathfrak{R}$ *is positive*, assume $\mathfrak{R}^+ = \bigcup_i \ (x \circ \mathfrak{R}_i)$ (all others cases are immediate). By construction, x is a splitting positive rule, and we select it. We have that $\mathit{seq}(\mathfrak{R}) = \bigcup_i \ (x \circ \mathit{seq}(\mathfrak{R}_i))$. Hence we have:

$$\mathit{deseq}(\mathit{seq}(\mathfrak{R}^+)) = \bigcup_i \ (x \circ \mathit{deseq}(\mathit{seq}(\mathfrak{R}_i))) \ .$$

□

In the result above, we follow the structure of the term describing the L-net, by substituting all occurrences of boxing with occurrences of rooting, and vice-versa. If $R = R'$ as L-nets, even if the description is different, $\mathit{deseq}(\mathit{seq}(\mathfrak{R})) = \mathit{deseq}(\mathit{seq}(\mathfrak{R}))$. In case we do not follow the structure of a term, it is possible that the operation of scoping cancel some actions; this is due to the fact that the L-net are not typed (i.e. are taken on a universal arena, without constraints).

Corollary 11.4 (Completeness). *An* $\mathrm{L_S}$*-net* $\mathfrak{D}$ *is a parallel L-net if and only if there is an L-forest* Π *such that* $\mathfrak{D} = \mathit{deseq}(\Pi)$*. In particular, parallel L-nets are* $\mathrm{L_S}$*-nets.*

Remark 11.5. *The crucial point in the proof of Theorem 11.3 is that the following holds for a parallel L-net:*

$$\mathfrak{R}^- = \bigdownarrow_I \ (x_I \circ (\mathit{Scope}(x_I, \mathfrak{R}) \setminus x_I)) \ ,$$

i.e. we can decompose (or destruct) a negative parallel L-net (scoping) and then reconstruct it (rooting and additive union). This does not hold in general for all (finite) $\mathrm{L_S}$*-nets.*

Remark 11.6. *We have omitted decorations in this section for simplicity. To be perfectly rigorous, one should maintain decorations through all the sequentialization and desequentialization process. For example, the sequentialization of a well decorated* $\mathrm{L_S}$*-net is defined just as the sequentialization of an* $\mathrm{L_S}$*-net (the only difference concerns the base case, where the decorations are kept).*

12. Restricting the picture to designs

In this section, we show that we can get rid of the MIX rule (and hence restrict our attention to Girard's original designs) by (unsurprisingly) imposing an additional connectedness assumption on L_S-nets.

Given an L-net $\mathfrak{D}$ and a slice $\mathfrak{S} \subseteq \mathfrak{D}$, a *switching graph* of $\mathfrak{S}$ is a subgraph obtained from $\mathfrak{S}$ by choosing a single entering edge for each negative node, and deleting all the other ones. A slice is *S-connected* if all its switching graphs are connected. Finally, we call an L-net S-connected if all its slices are.

An S-connected L-forest is obviously a tree, and in fact it is a design.

Lemma 12.1. *An L-forest Π is S-connected iff it is a design (in the sense of [13]).*

Sequentialization and desequentialization preserve S-connectedness, and hence by restriction to S-connected L_{S}-nets our results specialize to designs, rather than arbitrary L-forests. We will give details only for the desequentialization. The proofs concerning the sequentialization are similar and simpler.

Lemma 12.2 (Slices). *Let $\mathfrak{D}^+ = \bigcup_i (x^+ \circ \mathfrak{D}_i)$. All slices of $\mathfrak{D}$ have the form $\mathfrak{S} = \bigcup_i (x^+ \circ \mathfrak{S}_i)$, where each $\mathfrak{S}_i$ is a slice of $\mathfrak{D}_i$.*

Let $\mathfrak{D}^- = \Downarrow_I (x_I \circ \mathfrak{D}_I) = \bigcup_I \Phi(x_I \circ \mathfrak{D}_I)$. If $\mathfrak{S}$ is a slice of $\mathfrak{D}$, then $\mathfrak{S}$ is a slice of $\Phi(x_I \circ \mathfrak{D}_I)$ (for some I), and conversely. Moreover, $\mathfrak{S}_I = \mathfrak{S} \setminus x_I$ is a slice of $\mathfrak{D}_I$. On can recover $\mathfrak{S}$ from $x_I \circ \mathfrak{S}_I$ by adding appropriate edges from some nodes of $\mathfrak{S}_I$ to x_I.

Proposition 12.3. *If the L-net $\mathfrak{D}$ is S-connected, seq(Π) is S-connected, and hence it is a design.*

In order to restrict the converse transformation, we need a strengthening of Lemma 8.7.

Definition 12.4. *A* uniformly decorated *L-forest is an L-forest that is well decorated slicewise, i.e., each slice $\mathfrak{S}$ uses all the addresses of the interface, and all the addresses generated by a negative action of $\mathfrak{S}$.*

Lemma 12.5. *Every L-forest can be uniformly decorated.*

Proof. See Corollary Appendix A.5. □

There is a bijective correspondence between uniformly decorated L-forests, and their sequent calculus representation (Proposition Appendix A.4).

Lemma 12.6 (Used addresses). *Let Π be a uniformly decorated L-forest on the interface $\Xi \vdash \Delta$. If $\mathfrak{S}$ is a slice of the decorated L-net deseq(Π), all the addresses of the interface are used in $\mathfrak{S}$.*

Proof. The claim is true if Π consists of a single decorated action on $\vdash \Gamma$ ($\underline{\Pi}$ is essentially reduced to an axiom).

Assume $\Pi = \bigcup_i ((\xi, I)^+ \circ \Pi_i)$ is positive and has interface $\vdash \xi, \Delta$. Each Π_i is an L-forest of interface $\xi i \vdash \Delta$. For each i, any slice $\mathfrak{S}_i \subseteq$ *deseq*(Π_i) uses all the addresses in $\xi i \vdash \Delta$. Hence $\bigcup_i ((\xi i) \circ$ *deseq*$(\mathfrak{S}_i))$ is a slice which uses all the addresses in $\vdash \xi, \Delta$.

Assume $\Pi = \bigcup_I \ ((\xi, I)^- \centerdot \Pi_I)$ *is negative and has interface* $\xi \vdash \Delta$. By Lemma 12.2, $\mathfrak{T}$ is a slice in $\biguplus_I \ ((\xi, I)^- \circ \mathit{deseq}(\Pi_I))$ iff $\mathfrak{T}$ is a slice of $\Phi((\xi, I) \circ \mathit{deseq}(\Pi_I))$, for some I. Moreover, $\mathfrak{T}_I = \mathfrak{T} \setminus x_I$ is a slice of $\mathit{deseq}(\Pi_I)$.

Each Π_I is an L-forest of interface $\vdash \xi * I, \Delta$ Hence $\mathfrak{T}_I$ uses all the addresses in this interface, and we can obtain the slice $\mathfrak{T}' = (\xi, I)^- \circ \mathfrak{S}_I$ which uses all the addresses in $\xi \vdash \Delta$. Finally, since $\mathfrak{T}$ is obtained from $\mathfrak{T}'$ by adding some edges, this operation does not change the nodes, and hence does not change the set of addresses that are used. □

Proposition 12.7. *If* Π *is a finite design, and if we choose a uniform decoration for* Π*, then* $\mathit{deseq}(\Pi)$ *is S-connected.*

Proof. By assumption, Π is an S-connected L-forest.

Π *is negative.* By Lemma 12.2, $\mathfrak{S}$ is a slice of $\mathit{deseq}(\Pi) = \bigcup_I \ \Phi(x_I \circ \mathit{deseq}(\Pi_I))$ iff $\mathfrak{S}$ is a slice of $\Phi(x_I \circ \mathfrak{D}_I)$, for some I. We have that $\mathfrak{S}_I = \mathfrak{S} \setminus x_I$ is a slice of $\mathit{deseq}(\Pi_I)$. By hypothesis, $\mathfrak{S}_I$ is S-connected. Let $x_I = (\xi, I)$. By Lemma 12.6, in $x_I \circ \mathfrak{S}_I$ there are some edges connecting x_I to the nodes of $\mathfrak{S}_I$, those using some ξi. We obtain $\mathfrak{S}$ by adding some more edges.

We conclude by observing that (i) any choice of an edge entering x_I leaves x_I connected to a node of $\mathfrak{S}_I$, (ii) any switching S of $\mathfrak{S}$ restricted to $\mathfrak{S}_I$ is a switching of $\mathfrak{S}_I$, and (iii) by hypothesis, any two nodes of $\mathfrak{S}_I$ are connected in S.

Π *is positive.* By Lemma 12.2, $\mathfrak{S}$ is a slice of $\mathit{deseq}(\Pi) = \bigcup_i \ (x \circ \mathit{deseq}(\Pi_i))$ iff $\mathfrak{S} = \bigcup_i \ (x \circ \mathfrak{S}_i)$, and $\mathfrak{S}_i$ is a slice of $\mathit{deseq}(\Pi_i)$, for all i. By induction, all the $\mathfrak{S}_i$'s are S-connected, and hence $\mathfrak{S}$ is S-connected. □

13. Discussion and further work

In this section, we point to some follow-up developments that have taken place after our preliminary presentation [12], and we indicate a number of open questions and directions for future work.

13.1. Some follow-up developments

Proof nets. Above, we have claimed that tree strategies can be seen as *abstract sequent calculus proofs*, and that L-nets correspond to *abstract proof nets*, and we have argued that one could move gradually between the two notions. In [38], Faggian and Di Giamberardino have explored and realized these ideas in a *typed setting*. They have introduced a new syntax for (multiplicative) proof nets, called J-nets: a J-net is a typed L-net. They give a new, remarkably simple proof of sequentialization [39], based on the following insights. By building on the semantical experience, graphs (proof nets) can be treated as orders. To add sequentiality corresponds to adding edges, in such a way that the correctness criterion is still satisfied. When a graph is saturated (no edge can be added without violating the acyclicity condition of the correctness criterion), it turns out to be a tree, and hence can be seen as a sequent calculus proof.

The extension of this result to additives is also possible, and is studied by Di Giamberardino in his PhD thesis [40, 24].

Event Structures. The goal of generalizing the notion of innocent strategy in such a way that sequentiality is relaxed is pursued further in work by Faggian and Piccolo [41, 33, 22, 23]. They observed that L-nets are in fact a class of event structures [42]. Event structures are a fundamental tool of the "true concurrency" approach to the study of parallel and concurrent programming languages: concurrency, dependency, and conflict are directly expressed. Namely, an event structure describes a concurrent system in terms of a partial order, which specifies the causality relation between actions, and a conflict relation, which specifies which actions are mutually exclusive. L-nets can be naturally presented as event structures: the order relation is the same, and the conflict relation is induced by "using the same name". More precisely, L-nets appear as a class of *confusion free* event strategies, where conflict (choice) is localized in cells. Our notion of rule corresponds directly to the notion of cell.

Linear pi-calculus. The bridge between game semantics and concurrency theory that we discussed in the previous paragraph allows also for a game semantical analysis of the linear pi-calculus, introduced by Yoshida, Honda and Berger [34]. In [33], it is shown that ludics is a model for the finitary linear pi-calculus. More precisely, the translation (as in [43]) of linear pi-calculus into event structures produces an *L-forest*. Somehow surprisingly, this seems to say that an L-forest has some degree of "parallelism", as it corresponds to a term of an asynchronous pi-calculus. The restriction to designs is obtained if we furthermore assume a "sequentiality" constraint (expressed – and named such – in [34] as "at most one output is active at each single time").

This raises a number of interesting questions. The translation from the linear pi-calculus into event structures follows [43] and closely corresponds to the sequential constructors given here (cf. Section 10). We wonder what the parallel constructors would produce, and what a "parallel" strategy captures (see also below).

Completeness. It is natural to wonder if our setting would allow for a completeness result with respect to a (focalized) version of MALL. The answer is positive, and the calculus is quite naturally the calculus underlying Ludics, i.e. HS (see Section 2). The technical development of the construction goes beyond the scope of this paper, however the details are provided in a technical report [44]. With respect to the material presented in this paper, there are only two missing ingredients: the definition of an arena, and a notion of total strategy (since in this paper we work in a general, untyped setting). Assume to take the natural definitions. By exploiting the immediate relation between HS proofs and L-forests, it is immediate to interpret an HS proof into into a parallel L-net: from the proof, one moves to the corresponding L-forest, and one then applies desequentialization. The converse can be obtained by factorizing via sequentialization: given a total parallel L-net $\mathfrak{R}$ on the suitable arena, we can sequentialize it into the L-forest $\subseteq \mathfrak{R}$, which corresponds to a HS proof.

13.2. Some directions for future work

Graduating sequentiality. The sequentialisation and desequentialisation procedures that we have defined here are globally and (co)inductively defined. We should be able to follow the lines of Section 13.1 in our untyped setting, since there is no essential difference between J-nets and L-nets. This would enhance the meaning of "maximal

sequentiality" as a saturation in the sense that adding any other edge would create cycles. However, we still miss and would like to have a more precise characterization and understanding of what it means to have minimal sequentiality (see also next paragraph).

Induced equivalence. A notion of parallel strategy is not only of interest as an "asynchronous" model of computation, but could play the same role that proof nets play in providing an equational theory for proofs. It would be nice to have an independent characterization of the equivalence relation on L-forests induced by desequentialization:

$$\Pi_1 \cong \Pi_2 \iff \mathit{deseq}(\Pi_1) = \mathit{deseq}(\Pi_2) \ .$$

Also, in reference to the linear pi-calculus interpretation mentioned above, it would be interesting to understand the induced equivalence relation on processes.

Exponentials. A direction in which we expect a rather straightforward extension of our techniques is the setting of Ludics with exponentials. In [15], building on Maurel's PhD thesis [45], Basaldella and Faggian have shown that ludics can be extended with repetitions, so as to have exponentials. We expect that L-nets can be extended also to designs with repetitions, leading to some analogue of exponential proof nets. In this perspective, it might be useful to take profit of the co-inductive constructions and methods developped by Terui and Basaldella [46, and citations therein].

From proofs to programs. As we already mentioned, in this paper we have chosen to work in a framework, Ludics, which originated from a proof-theoretical analysis, and maintains a close and direct connection with proofs, to profit from the toolkit accumulated by the proof-theory of Linear Logic. We hope that similar methods as those proposed in this paper can be extended to a larger class of strategies, thus leading to models that can handle programming languages rather than proofs.

Acknowledgments. We wish to thank Dominic Hughes and Rob van Glabbeek for fruitful exchanges on the technique of domination, and Olivier Laurent for numerous discussions on MALL proof nets.

Appendix A. Sequent calculus presentation of L-forests and decoration

In this section, we recall the sequent calculus for designs [13] (see also [17]). We add a *MIX rule* (that is *not* part of Girard's original framework in [13]), and we examine the correspondence between such extended designs and L-forests.

Notation. In this section, Π will range over sequent calculus proofs. This does not conflict with our use of Π to denote L-forests, since we show here in some detail that they are essentially one and the same thing.

Girard's original sequent calculus for designs is the following (an interface is called well-formed if it consists of pairwise disjoint addresses with respect to the prefix ordering):

Daimon: ($\vdash \Lambda$ well formed)

$$\frac{}{\vdash \Lambda}\ \dagger$$

Positive rule ($I \subseteq \omega$ finite, one premise for each $i \in I$, all Λ_i's pairwise disjoint and included in Λ, $\vdash \xi, \Lambda$ well formed):

$$\frac{\cdots \quad \xi i \vdash \Lambda_i \quad \cdots}{\vdash \xi, \Lambda}\ (\xi, I)^+$$

Negative rule ($\mathcal{N} \subseteq \mathcal{P}_f(\omega)$ possibly infinite, one premise for each $J \in \mathcal{N}$, all Λ_J's included in Λ, $\xi \vdash \Lambda$ well formed):

$$\frac{\cdots \quad \vdash \xi * J, \Lambda_J \quad \cdots}{\xi \vdash \Lambda}\ \{(\xi, J)^- :\ J \in \mathcal{N}\}$$

MIX ($\vdash \Lambda_1, \ldots, \Lambda_n$ well formed, all Λ_m's pairwise disjoint)

$$\frac{\vdash \Lambda_1 \quad \ldots \quad \vdash \Lambda_n}{\vdash \Lambda_1, \ldots, \Lambda_n}\ \mathit{MIX}$$

(Replacing the Λ_i's with concrete contexts Γ_i of formulas, this is the well-known MIX rule of linear logic.)

Remark Appendix A.1. *The negative rule conveys some inherent weakening. Each action $(\xi, J)^-$ creates simultaneously all the addresses ξj ($j \in J$), which are recorded in the sequent, regardless of whether they will be used or not.*

Applications of the rule Daimon yield positive leaves in a proof tree. We will also consider as a positive leaf any proof tree of the following form:

$$\frac{\cdots \quad \dfrac{}{\xi i \vdash \Lambda_i}\ \emptyset \quad \cdots}{\vdash \xi, \Lambda}\ (\xi, I)^+$$

where all negative rules are applied with $\mathcal{N}$ empty. We will write simply:

$$\frac{}{\vdash \xi, \Lambda}\ (\xi, I)^+$$

We now briefly review how we can translate (in this extended setting) a sequent calculus proof Π of a sequent $\Xi \vdash \Lambda$ into an L-forest $\overline{\Pi}$ on the interface $\Xi \vdash \Lambda$. We use the syntax introduced in Section 10. We omit the (easy) proof that $\overline{\Pi}$ is an L-forest.

- Daimon. Then $\overline{\Pi} = \dagger$.
- $(\xi, I)^+$. Then $\overline{\Pi} = \bigcup_{i \in I}\ ((\xi, I)^+ \circ \overline{\Pi_i})$, where the Π_i's are the proofs of the sequents $\xi i \vdash \Lambda_i$ ($i \in I$). Note that $\overline{\Pi}$ is a tree.
- $\{(\xi, J)^- :\ J \in \mathcal{N}\}$. Then $\overline{\Pi} = \bigcup_J\ ((\xi, J)^- \centerdot \overline{\Pi_J})$, where the Π_J's are the proofs of the sequents $\vdash \xi * J, \Lambda_J$.

- MIX. Then $\overline{\Pi} = \biguplus_m \overline{\Pi_m}$, where the Π_m's are the proofs of the sequents $\vdash \Lambda_m$'s.

It should be clear that the rules of $\overline{\Pi}$ (as defined in Section 3) are in one-to-one correspondence with the occurrences of rules in Π. By going from Π to $\overline{\Pi}$, we have just forgotten all sequent informations except at the root. We now examine the converse direction, from L-forests to sequent calculus proofs.

Proposition Appendix A.2. *The mapping $\Pi \mapsto \overline{\Pi}$ is onto. More precisely, for every L-forest $\mathfrak{D}$ on an interface $\Xi \vdash \Lambda$ there exists a* uniform *sequent calculus proof Π of conclusion $\Xi \vdash \Lambda$ such that $\mathfrak{D} = \overline{\Pi}$, where a uniform proof is a proof in which the positive and negative rules are constrained as follows:*

$$\frac{\cdots \quad \xi i \vdash \Lambda_i \quad \cdots}{\vdash \xi, \biguplus_i \Lambda_i} \ (\xi, I)^+$$

with the side condition that $\Lambda = \bigcup_i \Lambda_i$ (i.e., no address is lost)

$$\frac{\cdots \quad \vdash \xi * J, \Lambda \quad \cdots}{\xi \vdash \Lambda} \ \{(\xi, J)^- : \ J \in \mathcal{N}\}$$

i.e., all Λ_J's are chosen maximal (and equal to Λ).

Proof. We have to extend the setting of [13] (see also [17]) from designs to L-forests, and to make sure that the target is restricted to uniform proofs. Let $\mathfrak{D}$ be an L-forest. There are four cases. In each case, we sketch how to (coinductively) generate the final rule of the proof (we give more details for the quite similar proof of Proposition Appendix A.4 (2) below).

1. If $\mathfrak{D}$ is a leaf, then the associated proof is its interface.
2. If $\mathfrak{D}$ is negative on interface $\xi \vdash \Lambda$, then it is easily seen that each $\vdash \xi * J, \Lambda$ is an interface for the corresponding subtree of $\mathfrak{D}$, so that we can carry on the construction (cases 1, 3, 4).
3. If $\mathfrak{D}$ is a positive L-forest with more than one root, then we transform each of the trees $\mathfrak{E}_j$ of $\mathfrak{D}$ into a proof of $\vdash \Lambda'_j$, where each Λ'_j consists of the minimal addresses used in $\mathfrak{E}_j$, and then we accommodate the constraint $\Lambda = \bigcup_j \Lambda_j$ by dispatching arbitrarily any $\xi \in \Lambda \setminus (\bigcup_i \Lambda'_i)$ to exactly one of the Λ'_j's, yielding suitable Λ_j's.
4. If $\mathfrak{D}$ is positive and is a tree, then we carry on the construction on its immediate subtrees (case 2), and we assemble them essentially as in case 3.

□

Remark Appendix A.3. *The uniform proof discipline described in the statement of Proposition Appendix A.2 corresponds to pushing weakening maximally to the leaves.*

The assignment of a sequent calculus proof to an L-forest is non-deterministic, i.e., the map $\Pi \to \overline{\Pi}$ is not injective. But, with decorations (cf. Section 8), we get a bijective correspondence. We recall (cf. Definitions 8.1, 8.3, and 12.4) that:

- a decorated L-forest is an L-forest $\mathfrak{D}$ in which all leaves k are equipped with a finite set $link(k)$ of addresses (called the link set of k), in such a way that the conditions on L-nets hold with respect to all addresses (including those in the link sets);
- a well decorated L-net is a decorated L-net $\mathfrak{D}$ such that all addresses of the interface, and all addresses generated by a negative action of $\mathfrak{D}$ are used in $\mathfrak{D}$, i.e., appear as a label of the underlying L-net or in a link set;
- a uniformly decorated L-forest is a decorated L-forest $\mathfrak{D}$ such that every slice of $\mathfrak{D}$ is well decorated.

Note that the interface of a well decorated L-net is determined by the rest of the structure: it is the set of all minimal addresses appearing in the L-net (for non decorated L-nets, we have only an inclusion of the latter set in the interface).

Proposition Appendix A.4. 1. *Well decorated L-forests are in one-to-one correspondence with sequent calculus proofs (of their uniquely determined interface) subject to the restriction that in all applications of the positive rule (resp. negative rule) we have* $\biguplus_{i\in I} \Lambda_i = \Lambda$ *(resp.* $\bigcup_{J\in\mathcal{N}} \Lambda_J = \Lambda$*).*
2. *Uniformly decorated L-forests are in one-to-one correspondence with the proofs subject to the further restriction of uniformity (*$\Lambda_J = \Lambda$*, for all* J*, cf. Proposition Appendix A.2).*

Proof. 1. The correspondence in one direction is obtained by adapting $\overline{\Pi}$, as follows: for each leaf with conclusion $\vdash \Lambda$ (resp. $\vdash \xi, \Lambda$) obtained by an application of $\dagger$ (resp. $(\xi, I)^+$), the translation is now $\dagger$ (resp. $(\xi, I)^+$) with $link(\dagger) = \Lambda$ (resp. $link((\xi, I)^+) = \Lambda$). It is easily checked that the respective restrictions on the construction of proofs ensure that $\overline{\Pi}$ is a well decorated or a uniformly decorated L-forest.

Conversely, given a well decorated L-forest $\mathfrak{D}$, we associate (deterministically) a proof of its interface, as follows.

- $\mathfrak{D} = \biguplus_i \mathfrak{C}_i$ has several positive conclusions. Then we translate each of the trees $\mathfrak{C}_1, \ldots, \mathfrak{C}_n$ of $\mathfrak{D}$, yielding proofs of sequents $\vdash \Lambda_1, \ldots, \vdash \Lambda_n$. By condition Additives we are sure that the Λ_i's are distinct. Therefore we can apply the MIX rule, and we define $\underline{\mathfrak{D}}$ as
$$\frac{\underline{\mathfrak{C}}_1 \quad \cdots \quad \underline{\mathfrak{C}}_n}{\vdash \Lambda_1, \ldots, \Lambda_n}\ MIX$$
where $\vdash \biguplus \Lambda_i$ is clearly the interface of $\mathfrak{D}$.
- $\mathfrak{D}$ has conclusion $k = \dagger$ with $link(k) = \Lambda$. Then we can use the positive rule and define $\underline{\mathfrak{D}}$ as
$$\frac{}{\vdash \Lambda}\ \dagger$$
- $\mathfrak{D}$ has only one positive conclusion $(\xi, I)^+$. If $\mathfrak{D}$ is reduced to a leaf, then we proceed as in the previous case. If $\mathfrak{D} = \bigcup_{\{j\in J\}} ((\xi, I)^+ \circ \mathfrak{D}_j)$, for some $J \subseteq I$,

by the same reasoning as in the first case, we have that the $\underline{\mathfrak{D}_j}$'s are proofs of sequents $\vdash \Lambda_j$, for paiwise disjoint Λ_j's. Then we define $\underline{\mathfrak{D}}$ as

$$\frac{\ldots \quad \underline{\mathfrak{D}_j} \quad \ldots \quad \overline{\xi k \vdash}^{\,\emptyset} \quad \ldots}{\vdash \Lambda}\ (\xi, I)^+$$

where j (resp. k) ranges over J (resp. $I \setminus J$), and where Λ is the union of the Λ_i's.

- $\mathfrak{D} = \bigcup_J \ ((\xi, J) \centerdot \mathfrak{D}_J)$. By definition of well decorated L-nets, the interface of each $\mathfrak{D}_J$ is $\vdash \xi * J, \Lambda_J$. Then we can use the negative rule and define $\underline{\mathfrak{D}}$ as

$$\frac{\ldots \quad \underline{\mathfrak{D}_J} \quad \ldots}{\xi \vdash \Lambda}\ \{(\xi, J)^- : \ J \in \mathcal{N}\}$$

where $\mathcal{N} = \{J : \ (\xi, J)^-$ is a root of $\mathfrak{D}\}$ and Λ is the union of the Λ_J's.

It is straightforward to prove that this transformation is inverse to the transformation $\Pi \mapsto \overline{\Pi}$.

2. This correspondence is simply obtained by restricting the correspondence to uniformly decorated L-nets and to uniform proofs. □

Corollary Appendix A.5. *Every L-forest can be uniformly decorated, and hence a fortiori well decorated.*

Proof. To an L-forest $\mathfrak{D}$, we can associate a uniform proof by Proposition Appendix A.2, and then a uniform decoration, by Proposition Appendix A.4. □

Remark Appendix A.6. *Note that the bijective correspondences of Proposition Appendix A.4 induce a bijective correspondence between the link sets used in the decoration of an L-forest and the generalized axioms used in the corresponding sequent calculus proof.*

Appendix B. Proof of the Splitting Lemma

In this section, we prove Lemma 6.13, namely that every $\mathrm{L_S}$-net $\mathfrak{D}$ has a splitting conclusion.

We recall from section 6.5 that a negative rule $W = \{\ldots, w_I, \ldots\}$ of an L-net $\mathfrak{D}$ is called splitting if either it is conclusion of the $\mathrm{L_S}$-net (each w_I is a root), or if after deleting all the edges $w_I \to w$ there is no more connection (i.e., no sequence of consecutive edges) between any of the w_I's and w, and that a positive conclusion of $\mathfrak{D}$ is called splitting if all negative rules just above it are splitting.

If $\mathfrak{D}$ is negative, then the Splitting Lemma holds vacuously. For positive $\mathfrak{D}$'s, we first establish the following Negative Splitting Lemma.

Lemma Appendix B.1 (Negative Splitting Lemma). *Every positive* $\mathrm{L_S}$*-net* $\mathfrak{D}$ *which has a negative rule has a splitting negative rule among all the negative rules of level 1 (i.e., located just above a conclusion).*

Our proof is an adaptation to our setting of the proof of the similar lemma in [20]. A switching path $x_0 \dots x_n$ is called *strong* (and denoted $x_0 \Leftarrow x_n$) if either its last node is positive or if it ends upwards in the last node. Strong switching paths satisfy the following concatenation property: if γ_1 is a strong switching path and γ_2 is a switching path such that their concatenation $\gamma_1\gamma_2$ is a rule path (cf. Section 5), then $\gamma_1\gamma_2$ is switching, and if moreover γ_2 is strong, then $\gamma_1\gamma_2$ is strong.

Definition Appendix B.2 (Domination). *Given an* $\mathrm{L_S}$*-net* $\mathfrak{D}$*, a negative rule* X *and a finite set of nodes* G*, we say that* G *is an* X*-zone if for every* $z \in G$ *there are nodes* $x \in X$ *and* x' *such that* $x \leftarrow x' \Leftarrow z$*, where the path* $x' \Leftarrow z$ *is included in* G*. Given a node* z *of* $\mathfrak{D}$*, we say that* X dominates z*, denoted* $X \trianglelefteq_{\mathfrak{D}} z$ *(or simply* $X \trianglelefteq z$*), if there exists an* X*-zone* G *in* $\mathfrak{D}$ *such that* $z \in G$*. We say that the zone* G *and the sequence* $x \leftarrow x' \Leftarrow z$ *witness* $X \trianglelefteq z$*.*

The following statement lists some simple consequences of the definition of domination.

Lemma Appendix B.3.
1. *X-zones are closed under unions.*
2. *If $X \trianglelefteq z$ is witnessed by a sequence $x \leftarrow x' \Leftarrow z$, then X dominates every node of the path $x' \Leftarrow z$.*
3. *If $x \leftarrow y$ for some $x \in X$, then $X \trianglelefteq y$.*
4. *Given a negative rule W, if X dominates a node $w \in W$, then X dominates all $w' \in W$.*
5. *If $X \trianglelefteq y$, and if $y \leftarrow z$, or if $z \leftarrow y$ and z is not negative, then $X \trianglelefteq z$.*

Proof. The first three parts of the statement are obvious. Let $X \trianglelefteq w$ be witnessed by G and $x \leftarrow x' \Leftarrow w$. By definition of strong, the path $x' \Leftarrow w$ terminates with $k \leftarrow w$. Then we obtain a strong path to any $w' \in W$ by just replacing the last edge with $k \leftarrow w'$. It follows that $G \cup W$ is an X-zone, and therefore $(\forall\, w' \in W \;\; X \trianglelefteq w')$.

We now prove the last assertion of the statement. Let G and $x \leftarrow a \Leftarrow y$ be a witness of $X \trianglelefteq y$. If z does not belong to a rule that intersects the sequence $a \dots y$, then the sequence $a \dots yz$ is a path, that is switching by the assumptions. Hence $G \cup \{z\}$ and $x \leftarrow a \Leftarrow z$ are a witness for $X \trianglelefteq z$. If z intersects the sequence $a \dots y$, then we conclude using the second and fourth parts of the statement. □

Thanks to Lemma Appendix B.3, we will henceforth safely assume that X-zones are *rule-saturated*, i.e. are unions of rules.

The notion of domination extends to rules. Let W be a rule. We write $W_1 \trianglelefteq W_2$ if there exists $w_2 \in W_2$ such that $W_1 \trianglelefteq w_2$ (or, equivalently, if $W_1 \trianglelefteq w_2$ for all $w_2 \in W_2$, by Lemma Appendix B.3). If X is not dominated, we say that it is *free*.

Lemma Appendix B.4. *Domination is transitive.*

Proof. Assume $X \trianglelefteq Y$ and $Y \trianglelefteq Z$, witnessed by (rule-saturated) G and $x \leftarrow a \Leftarrow y$, and by G' and $y' \leftarrow b \Leftarrow z$ ($z \in Z$, $y, y' \in Y$, $x \in X$), respectively. By Lemma Appendix B.3, we can assume $y = y'$. It is enough to show that $G' \cup G$ is an X-zone, and for this we only have to consider $z' \in G' \setminus G$, if any, witnessed by $y'' \leftarrow b' \Leftarrow z'$. Let z'' be the last node in the sequence $y''b' \dots z'$ which is in G, witnessed by $x' \leftarrow a' \Leftarrow z''$. Then, concatenating with the rest of the sequence from (the successor of) z'' to z', we obtain a path (by construction, and because G is rule-saturated). This path is strong and switching because its constituents are. □

Lemma Appendix B.5. *Let W be a negative rule. If $w \in W$ is below a node of a switching cycle C, then W dominates all nodes of the cycle. If $w_1, w_2 \in W$ are such that $w_i \overset{+}{\leftarrow} z_0$ and $w_j \overset{+}{\leftarrow} z_n$, then W dominates every node in a switching path from z_0 to z_n.*

Proof. We prove the second part of the statement (the reasoning is the same for the first part). Let G_1 (resp. G_2) be the set of nodes on a path going up from w_1 to z_0 (resp. from w_2 to z_n). We will show that $G_1 \cup G_2 \cup C$ is a W-zone. That G_1 and G_2 are W-zones follows readily from Lemma Appendix B.3. Let $z \in C$. Because C is switching, we have either $z_0 \Leftarrow z$ or $z_n \Leftarrow z$. Suppose that we have, say, $z_0 \Leftarrow z$. Let z_1' be the first node on the way up from w_0 to z_0 that belongs to a rule intersecting $z_0 \Leftarrow z$ at some z_2'. Then the sequence obtained by going up from w_1 to z_2' and then to z is witnessing $W \trianglelefteq z$. □

Lemma Appendix B.6. *Let $\mathfrak{D}$ be a* finite $\mathrm{L_S}$*-net. If a rule X intersects a switching cycle, then X is dominated by an additive rule W which intersects no switching cycle.*

Proof. We construct a sequence of negative rules W_i as follows. We set $X = W_0$. If W_i intersects a switching cycle, then applying the condition Cycles gives us a rule W_{i+1}. We have $W_{i+1} \trianglelefteq W_i$, by Lemma Appendix B.5. At each iteration the union of the cycles increases strictly, and hence by finiteness of $\mathfrak{D}$ we eventually reach some negative rule $W_n = W$ which intersects no switching cycle. Moreover, we have $W \trianglelefteq X$ by Lemma Appendix B.4. □

Due to the finiteness condition in the previous lemma, we will first establish the Splitting Lemma in the finite case, and then show how to lift the result also to infinite $\mathrm{L_S}$-nets.

Lemma Appendix B.7. *If $X \trianglelefteq X$, then X is in a switching cycle.*

Proof. If $X \trianglelefteq X$, we have $x \leftarrow a \Leftarrow x$ for some $x \in X$. Then we can close the path from a to x with the edge $x \leftarrow a$. Because the path from a to x is strong, the cycle is switching. □

Proposition Appendix B.8. *Let $\mathfrak{D}$ be a finite* $\mathrm{L_S}$*-net. Every negative rule is either free or dominated by a free negative rule. As a consequence, if there are negative rules, there exists a free negative rule.*

Proof. The proof is by contradiction. Let X be a negative rule that is neither free nor dominated by a free negative rule. We will build an infinite sequence of rules X_i which are all distinct, are all dominated, and are such that

$$\dots X_{i+1} \trianglelefteq X_i \trianglelefteq \dots X_1 \trianglelefteq X_0 \trianglelefteq X \ ,$$

contradicting the finiteness of $\mathfrak{D}$. We take $X_0 = X$. By assumption, X_0 is not free. Suppose that we have constructed the sequence up to X_i. We distinguish two cases:

1. If X_i is not in a switching cycle, we choose any X_{i+1} such that $X_{i+1} \trianglelefteq X_i$ (this is possible since X_i is not free by induction hypothesis). This X_{i+1} is fresh as otherwise we would have by transitivity $X_i \trianglelefteq X_i$, contradicting our assumption on X_i, by Lemma Appendix B.7.
2. If X_i is in a switching cycle, then by Lemma Appendix B.6 we can choose a rule X_{i+1} such that X_{i+1} intersects no switching cycle. This X_{i+1} is fresh as otherwise we would have by transitivity $X_{i+1} \trianglelefteq X_{i+1}$, and this contradicts our assumption about the choice of X_{i+1}, by Lemma Appendix B.7.

In both cases, we have constructed a fresh X_{i+1} such that $X_{i+1} \trianglelefteq X_i$. Moreover, by transitivity, $X_{i+1} \trianglelefteq X$, from which it follows that X_{i+1} is not free. □

Let X, Y be distinct negative rules.

- We write $X \longleftrightarrow Y$ if there is a switching path $z_0 \dots z_n$ (called witnessing path) such that $x \leftarrow z_0$ and $z_n \rightarrow y$, for some $x \in X$ and $y \in Y$.
- We write $X \rightarrow\!\leftarrow Y$ if X, Y belong to the same bipole, i.e., $x \rightarrow k$ and $y \rightarrow k$, for some k and all $x \in X$ and $y \in Y$.

Lemma Appendix B.9. *If X, Y and Z are negative rules such that $X \neq Y$, $X \trianglelefteq Z$ and $Y \trianglelefteq Z$, then $X \longleftrightarrow Y$.*

Proof. Consider $x \leftarrow a \trianglelefteq z$ (for some $x \in X$ and $z \in Z$). Let z' be the first node on the path from a to z such that $Y \trianglelefteq z'$ (and hence $y \leftarrow b \Leftarrow z'$ for some $y \in Y$). Then we get a path witnessing $X \longleftrightarrow Y$ by going from a to z' and then from z' to b. This sequence of nodes is a rule path since if it were not, there would be z_1 in the first portion and z_2 in the second portion belonging to the same rule, but we have that $Y \trianglelefteq z'$ implies $Y \trianglelefteq z_2$ which in turn implies $Y \trianglelefteq z_1$, contradicting the minimality of z'. It is switching since the path from b to z' is strong. □

Lemma Appendix B.10. *If X is a free negative rule of $\mathfrak{D}$ and does not split, then there exist free negative rules Y, Z of $\mathfrak{D}$ such that $X \rightarrow\!\leftarrow Y$ and $X \longleftrightarrow Z$.*

Proof. Let c be the node just below X. Since X does not split, for some $x \in X$ we can form a cycle (in the ordinary sense of graph theory, i.e. without the disjoint rules assumption) $xc \dots ax$, without using any edge between c and X other than $c \leftarrow x$. Since X is free, c is a conclusion of the net, and the next node on the cycle must be some y such that $c \leftarrow y$. By construction, y belongs to a rule Y distinct from X, and thus we have $X \rightarrow\!\leftarrow Y$.

Next we observe that we cannot have $X \trianglelefteq x$, because X is free, and that $X \trianglelefteq a$ because $x \leftarrow a$ (since the only edge of $\mathfrak{D}$ out of x is already used). Let b be the first node, following the cycle in the direction $xc\ldots$, such that $X \trianglelefteq b$. The node z' before b must be negative and we must have $z' \leftarrow b$ as otherwise we would have $X \trianglelefteq z'$ by Lemma Appendix B.3. Let Z' be the rule to which z' belongs. Then we have $X \longleftrightarrow Z'$. If Z' is free, we can set $Z = Z'$. If Z' is not free, it dominates some free Z, by Proposition Appendix B.8, and we conclude by Lemma Appendix B.9 (since $X \trianglelefteq \{b\}$, and $Z \trianglelefteq \{b\}$ by transitivity). □

We are now able to prove the Negative Splitting Lemma for *finite* $\mathrm{L_S}$-nets, i.e. for $\mathrm{L_S}$-nets having finitely many nodes.

Proof (Negative Splitting Lemma, finite case). If the $\mathrm{L_S}$-net $\mathfrak{D}$ has no splitting negative rules, then all its conclusions must be positive, and starting from a free negative rule X_0 (whose existence is guaranteed by Proposition Appendix B.8), and using again and again Lemma Appendix B.10, we can build an infinite sequence $X_0 \rightarrow\!\!\leftarrow X_1 \longleftrightarrow X_2 \rightarrow\!\!\leftarrow \ldots$ where X_{i+1} is a free negative rule and $X_{i+1} \neq X_i$, for all i. Since there are only finitely many free negative rules, we have $X_i = X_j = X$ for some $i < j$. By the definition of the $\rightarrow\!\!\leftarrow$ and $\longleftrightarrow$ relations, we can form a switching sequence of nodes starting in $X_i = X$ and ending in $X_j = X$ which is nondegenerate (i.e. of length at least 2) since $X_i \neq X_{i+1}$. But this sequence is not guaranteed to be a rule path. To build a rule path, we take two nodes z_1 and z_2 at minimal distance in the sequence such that z_1 and z_2 belong to the same rule. Again, this distance is non-degenerate, as z_1 and z_2 cannot belong to the same path witnessing some $X_{2k+1} \longleftrightarrow X_{2k+2}$ ($i \leq 2k+1 < j$), and moreover, by the same reason, the path $z_1 z'_1 \ldots z'_2 z_2$ from z_1 to z_2 must cross some X_n. We distinguish two cases:

1. If $z'_1 \leftarrow z_1$ or $z'_2 \leftarrow z_2$, say, $z'_1 \leftarrow z_1$, then we also have $z'_1 \leftarrow z_2$, and adding this (reversed) edge to the path from z'_1 to z_2 yields a switching cycle. Then, by a (weakened form of) Lemma Appendix B.6, we obtain that X_n is dominated.
2. If $z_1 \leftarrow z'_1$ and $z_2 \leftarrow z'_2$, then we are in the situation of the (second part of the statement of) Lemma Appendix B.5, and we also obtain that X_n is dominated.

We have reached a contradiction, since X_n is free by construction. □

The Negative Splitting Lemma holds actually for arbitrary $\mathrm{L_S}$-nets. The following definition and lemma ensure a finiteness condition, even if our $\mathrm{L_S}$-net $\mathfrak{D}$ is infinite.

Definition Appendix B.11. *Let $\mathfrak{D}$ be an $\mathrm{L_S}$-net whose conclusions are all positive. We denote by $Neg_1(\mathfrak{D})$ the set of negative rules that are just above a conclusion (i.e., the set of rules of level 1).*

Lemma Appendix B.12. 1. *The set $Neg_1(\mathfrak{D})$ is finite.*
2. *Every free (negative) rule is in $Neg_1(\mathfrak{D})$.*

Proof. By the finiteness of the interface, there are finitely many conclusions, and since there are are only finitely many rules just above a positive rule, it follows that $Neg_1(\mathfrak{D})$ is finite. Suppose that W is a rule of level > 1, then, going down from

W, we reach a rule W' that dominates W (that $W' \trianglelefteq W$ follows by repeated use of Lemma Appendix B.3). The second part of the statement follows. □

Proof (Negative Splitting Lemma, infinite case). Let $\mathfrak{D}$ be an infinite $\mathrm{L_S}$-net. We concentrate on $Neg_1(\mathfrak{D})$. For each pair $X, X' \in Neg_1(\mathfrak{D})$ such that $X \trianglelefteq X'$, we take an X-zone witnessing this domination. Let $\mathfrak{F}$ be a minimal (finite) $\mathrm{L_S}$-net containing these zones, obtained by possibly adding in a minimal way positive views to the set $\mathfrak{F}'$ of all views $\ulcorner k \urcorner$, where k ranges over the union of the zones (note that $\mathfrak{F}'$ is already a partial L-net by Lemma 6.1). By construction, $Neg_1(\mathfrak{F})$ consists of all sets $X \cap \mathfrak{F}$ such that $X \in Neg_1(\mathfrak{D})$ and $X \cap \mathfrak{F} \neq \emptyset$. Moreover, for any two $X, X' \in Neg_1(\mathfrak{D})$, we have, by construction of $\mathfrak{F}$:

$$X \trianglelefteq_{\mathfrak{D}} X' \qquad \Leftrightarrow \qquad (X \cap \mathfrak{F} \neq \emptyset\ ,\ X' \cap \mathfrak{F} \neq \emptyset\ ,\ \text{and}\ (X \cap \mathfrak{F}) \trianglelefteq_{\mathfrak{F}} (X' \cap \mathfrak{F}))\ .$$

It follows that if $(X \cap \mathfrak{F}) \in Neg_1(\mathfrak{F})$ is free (in $\mathfrak{F}$), then X is free (in $\mathfrak{D}$), using the fact that whenever a rule is dominated, it is dominated by a rule in $Neg_1(\mathfrak{D})$.

Now, suppose that there exists a negative rule X of $\mathfrak{D}$ that is neither free nor dominated by a free rule. We can assume $X \in Neg_1(\mathfrak{D})$ since this property is a fortiori true of any negative rule below. Then, as we noted above, $X \cap \mathfrak{F}$ is not free. Neither can $X \cap \mathfrak{F}$ be dominated by a free rule $X' \cap \mathfrak{F}$, because then we would have that X' is free and $X' \trianglelefteq_{\mathfrak{D}} X$. Therefore $X \cap \mathfrak{F}$ is neither free nor dominated by a free rule in $\mathfrak{F}$, contradicting the Negative Splitting Lemma (finite case). □

We now prove the Splitting Lemma, as a consequence of the Negative Splitting Lemma.

Proof (Splitting Lemma). Let $\mathfrak{D}$ be an $\mathrm{L_S}$-net that has only positive conclusions. We define $size(\mathfrak{D})$ as:

- 0 if at least one of the positive conclusions of $\mathfrak{D}$ is a leaf, and otherwise as
- the cardinal of the set of level 1 negative rules of $\mathfrak{D}$.

Since $\mathfrak{D}$ has finitely many positive conclusions, the size of $\mathfrak{D}$ is finite, even if $\mathfrak{D}$ is not finite.

We apply the Negative Splitting Lemma to $\mathfrak{D}$. We select a splitting negative rule X. Since $\mathfrak{D}$ has no negative conclusion, X is just above a conclusion k of $\mathfrak{D}$. We delete the edges from x to k, for all $x \in X$.

Let us call $\mathfrak{D}_X$ the union of the connected components (in the ordinary unoriented graph-theoretic sense) of the elements of X, and $\mathfrak{D}_k$ the rest of the graph, which contains k. We prove that $\mathfrak{D}_X$ and $\mathfrak{D}_k$ are $\mathrm{L_S}$-nets. Let $\mathfrak{D}'$ stand for either $\mathfrak{D}_X$ or $\mathfrak{D}_k$. We note that if $c \in \mathfrak{D}'$, then $\mathfrak{D}'$ contains every path of $\mathfrak{D}$ starting from c that does not go through one of the deleted edges. It follows that $\ulcorner c \urcorner_{\mathfrak{D}'}$ possibly differs from $\ulcorner c \urcorner_{\mathfrak{D}}$ only by not containing k. We are thus almost in the situation of Lemma 6.2, modulo straightforward adaptations. Let us look for example at the condition Additives: the path leading down from k_1 to w_1 (resp. from k_2 to w_2) does not go through k, and

hence belongs to $\ulcorner k_1 \urcorner_{\mathfrak{D}'}$ (resp. $\ulcorner k_2 \urcorner_{\mathfrak{D}'}$), so condition Additives in $\mathfrak{D}'$ is inherited from condition Additives in $\mathfrak{D}$.

We have that $size(\mathfrak{D}_k) < size(\mathfrak{D})$, because every conclusion k' of $\mathfrak{D}_k$ is a conclusion of $\mathfrak{D}$, and every negative rule of $\mathfrak{D}_k$ is a negative rule of $\mathfrak{D}$. Moreover, every free splitting negative rule of $\mathfrak{D}_k$ is a splitting negative rule of $\mathfrak{D}$: indeed, if $\mathfrak{D}_k$ splits into $\mathfrak{D}_{k'}$ and $\mathfrak{D}_{X'}$, then $\mathfrak{D}$ splits into $(\mathfrak{D}_{k'} \cup \mathfrak{D}_X)$ and $\mathfrak{D}_{X'}$. We are now ready to prove that $\mathfrak{D}$ has a positive splitting conclusion, by induction on $size(\mathfrak{D})$:

- Base case. Obvious. Since one positive conlusion is a leaf, it is splitting vacuously.
- Induction case. Let k' be a splitting positive conclusion of $\mathfrak{D}_k$. If $k' \neq k$, then it is also a splitting positive conclusion of $\mathfrak{D}$, since every negative rule just above k' is splitting in $\mathfrak{D}_k$, hence in $\mathfrak{D}$. If $k' = k$, let Y be a negative rule just above k. If $Y = X$, it is splitting by construction; if $Y \neq X$, Y belongs to $\mathfrak{D}_k$ by construction, and hence Y is splitting in $\mathfrak{D}_k$, and hence also in $\mathfrak{D}$, so that k is a positive splitting conclusion. This completes the proof. □

[1] S. Abramsky, P.-A. Melliès, Concurrent games and full completeness, in: Proceedings of the Fourteenth International Symposium on Logic in Computer Science, LICS, pp. 431–442.

[2] M. Hyland, A category of partial orders and merging, 2001. Oral communication at Rencontres Franco-Américaines de Mathématiques (AMS-SMF).

[3] P.-A. Melliès, Asynchronous games 2 : The true concurrency of innocence, in: Proc. CONCUR'04, volume 3170 of *LNCS*, Springer Verlag, 2004.

[4] C. Faggian, F. Maurel, Ludics nets, a game model of concurrent interaction, in: Proc. of LICS'05, IEEE Computer Society Press, 2005.

[5] M. Hyland, A. Schalk, Games on graphs and sequentially realizable functionals, in: LICS 02, IEEE, 2002, pp. 257–264.

[6] G. McCusker, M. Wall, Categorical and game semantics for SCIR, in: Proc. Galop'05 (Games for Logic and Programming Languages), workshop affiliated with Etaps'05, pp. 157–178.

[7] A. Schalk, J. Palacios-Perez, Concrete data structures as games, in: CTCS 04, volume 122 of *Electr. Notes Theor. Comput. Sci.*

[8] P.-A. Melliès, Asynchronous games 2: The true concurrency of innocence, TCS 358(2-3) (2006) 200–228.

[9] P.-A. Melliès, S. Mimram, Asynchronous games: Innocence without alternation, in: Proc. of CONCUR 2007 - LNCS, volume 4703 of *LNCS*, Springer, 2007, pp. 395–411.

[10] S. Rideau, G. Winskel, Concurrent strategies, in: Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011,, pp. 409–418.

[11] M. Hyland, L. Ong, On full abstraction for PCF: I, II, and III, Inf. Comput. 163 (2000) 285–408.

[12] P.-L. Curien, C. Faggian, L-nets, strategies and proof-nets, in: CSL 05 (Computer Science Logic), LNCS, Springer, 2005.

[13] J.-Y. Girard, Locus solum, Mathematical Structures in Computer Science 11 (2001) 301–506.

[14] C. Faggian, M. Hyland, Designs, disputes and strategies, in: CSL'02, volume 2471 of *LNCS*, Springer Verlag, 2002.

[15] M. Basaldella, C. Faggian, Ludics with repetitions (exponentials, interactive types and completeness), Logical Methods in Computer Science 7 (2011) 1–88.

[16] P.-L. Curien, Abstract Böhm trees, Mathematical Structures in Computer Science 8 (1998).

[17] P.-L. Curien, Introduction to linear logic and ludics, part II, Advances of Mathematics, China 35 (2006).

[18] S. Abramsky, R. Jagadeesan, Games and full completeness for multiplicative linear logic, J. Symb. Log. 59 (1994) 543–574.

[19] R. Amadio, P.-L. Curien, Domains and Lambda-calculi, Cambridge University Press, 1998.

[20] D. Hughes, R. van Glabbeek, Proof nets for unit-free multiplicative-additive linear logic, ACM Transactions on Computational Logic 6 (2005) 784–842.

[21] S. Mimram, Sémantique des jeux asynchrones et réécriture 2-dimensionnelle, Ph.D. thesis, Paris 7 University, 2008.

[22] C. Faggian, M. Piccolo, Partial orders, event structures, and linear strategies, in: Proceedings of Typed Lambda Calculi and Applications, TLCA, volume 5608 of *LNCS*, Springer, 2007.

[23] M. Piccolo, Linearity and Beyond in Denotational Semantics, Ph.D. thesis, Univ. Torino - Univ. Paris 7, 2009.

[24] P. Di Giamberardino, Jump from parallel to sequential proofs: Additives, Draft, 2011.

[25] J.-Y. Girard, Proof-nets : the parallel syntax for proof-theory., Logic and Algebra (1996).

[26] J.-M. Andreoli, Focussing and proof construction, Annals of Pure and Applied Logic (2001).

[27] J.-Y. Girard, On the meaning of logical rules II: multiplicative/additive case, in: Foundation of Secure Computation, NATO series F 175, IOS Press, 2000, pp. 183–212.

[28] J.-Y. Girard, On the meaning of logical rules I: syntax vs. semantics, in: Computational logic, NATO series F 165, Springer, 1999, pp. 215–272.

[29] J.-Y. Girard, The Blind Spot. Lectures on Logic., European Mathematical Society, 2011.

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

[31] C. Faggian, Travelling on designs: ludics dynamics, in: CSL'02 (Computer Science Logic), volume 2471 of *LNCS*, Springer Verlag, 2002.

[32] P.-L. Curien, Notes on game semantics, 2007. Available from `www.pps.jussieu.fr/~curien`.

[33] C. Faggian, M. Piccolo, Ludics is a model for the finitary linear pi-calculus, in: S. R. della Rocca (Ed.), Proceedings of TLCA 2007, volume 4583 of *Lecture Notes in Computer Sciences*, Springer, 2007, pp. 148–162.

[34] M. Berger, K. Honda, N. Yoshida, Sequentiality and the π-calculus, in: Proceedings of TLCA 2001, volume 2044 of *Lecture Notes in Computer Sciences*, Springer, 2001, pp. 29–45.

[35] J.-M. Andreoli, Focussing proof-net construction as a middleware paradigm, in: Proceedings of CADE'02 (Conference on Automated Deduction).

[36] B. Jacobs, J. Rutten, A tutorial on (co)algebras and (co)induction, Bulletin of EATCS 62 (1997) 222–259.

[37] O. Laurent, Intuitionistic dual-intuitionistic nets, Journal of Logic and Computation 21 (2011) 561–587.

[38] P. Di Giamberardino, C. Faggian, A jump from parallel to sequential proofs. Multiplicatives., in: CSL'06 (Computer Science Logic), volume 4207 of *LNCS*.

[39] P. D. Giamberardino, C. Faggian, Proof nets sequentialisation in multiplicative linear logic, Ann. Pure Appl. Logic 155 (2008) 173–182.

[40] P. Di Giamberardino, Jump from Parallel to Sequential Proofs : On Polarities and Sequentiality in Linear Logic, Ph.D. thesis, Univ. Roma3- Univ. Aix-Marseille II, 2008.

[41] C. Faggian, M. Piccolo, A graph abstract machine describing event structure composition, in: GT - VC, Graph Transformation for Verification and Concurrency. ENTCS.

[42] M. Nielsen, G. Plotkin, G. Winskel, Event structures and domains 1, Theoretical Computer Science 13 (1981) 85–108.

[43] D. Varacca, N. Yoshida, Typed event structures and the pi-calculus, in: MFPS.

[44] C. Faggian, Linear logic games: sequential and parallel, 2006. Preprint.

[45] F. Maurel, Un cadre quantitatif pour la Ludique, Ph.D. thesis, Université de Paris 7, 2004.

[46] M. Basaldella, K. Terui, Infinitary completeness in ludics, in: LICS 2010, Symposium on Logic in Computer Science, IEEE Computer Society Press, 2010.