L-nets, strategies and proof-nets

Pierre-Louis Curien[1] and Claudia Faggian[2] *

[1] CNRS - Université Paris 7
[2] Universitá di Padova

Abstract. We consider the setting of L-nets, recently introduced by Faggian and Maurel as a game model of concurrent interaction and based on Girard's Ludics. We show how L-nets satisfying an additional condition, which we call logical L-nets, can be sequentialized into traditional tree-like strategies, and vice-versa.

1 Introduction

In the context of Game Semantics several proposals are emerging - with different motivations - towards strategies where sequentiality is relaxed to capture a more parallel form of interaction, or where the order between moves in a play is not totally specified. Such strategies appear as graphs, rather then more traditional tree-like strategies. We are aware of work by Hyland, Schalk, Melliès, McCusker and Wall. Here we will consider the setting of L-nets, recently introduced by Faggian and Maurel [8] as a game model of concurrent interaction, based on Girard's Ludics.

The idea underlying L-nets (as well as other approaches) is to not completely specify the order in which the actions should be performed, while still being 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.

More traditional strategies, and in particular Hyland-Ong innocent strategies [13], are trees. In this paper we are interested in relating some representatives of these two kinds of strategies. We show how strategies represented by graphs, with little ordering information, can be sequentialized into tree-like strategies; conversely, sequential (tree) strategies can be relaxed into more asynchronous ones.

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

A graph strategy (an L-net) is a set of partially ordered views (p.o. views), where a p.o. view is a partially ordered set of moves, which expresses an enabling relation, or a scheduling among moves. The set of such p.o. views forms a directed acyclic graph. Any interaction (play) results into a partially ordered set of moves.

In our setting a tree strategy is, in particular, a graph strategy. Hence we have an homogeneous space, inside which we can move, applying our procedures Seq and $Deseq$, which respectively add or relax dependency (sequentiality).

* Research partially supported by Cooperation project CNR-CNRS Italy-France 2004-2005 (Interaction et complexité, project No 16251).

From graph strategies to tree strategies and vice-versa. The graph strategies we will consider are (a class of) L-nets. The tree-like strategies we will consider are Girard's designs [11] (syntactically, designs are particular sorts of Curien's abstract Böhm trees [4, 5]). As a computational object a design is a Hyland-Ong innocent strategy on a universal arena, as discussed in [7]. An L-net is a graph strategy *on the same arena*.

We will show how to associate a design to certain L-net, in such a way that all constraints expressed by the L-net are preserved. This is not possible for an arbitrary L-net; it is easy to build a counter-example taking inspiration from Gustave function (a well-known example of a non-sequential function, see e.g. [1]). For this reason, we first introduce the notion of *logical L-nets*, which are L-nets satisfying a condition called cycles condition[3]. We then make the following constructions: in section 4, we show how to obtain a set of designs $seq(\mathfrak{D})$ from a logical L-net $\mathfrak{D}$, while in section 5, we show how to obtain a logical L-net $deseq(\mathfrak{D})$ from a design $\mathfrak{D}$, in such a way that for all designs $\mathfrak{D}$ we have $\mathfrak{D} \in seq(deseq(\mathfrak{D}))$.

The proof-net experience. Tree strategies can be seen as abstract (and untyped) sequent calculus derivations. By contrast, L-nets are graphs which 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 derivations, or as proof-nets. Sequent calculus derivations can be mapped onto proof-nets, by forgetting some of the order between the rules, and conversely proof-nets can be sequentialized into proofs. In this paper we use similar techniques in the framework of game semantics. It is a contribution of the paper to transfer the use of proof-net technologies to the semantic setting of Game strategies. This appears to be a natural consequence of a general direction bringing together syntax and semantics.

2 Tree strategies (designs) and sequent calculus derivations

Designs, introduced in [11], 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 derivations (in a focusing calculus, which we will introduce next).

While we do not recall the standard definitions of view and innocent strategy, in the following we review in which sense a tree strategy is a sequent calculus derivation, and viceversa.

2.1 Focalization and synthetic connectives

Multiplicative and additive connectives of Linear Logic separate into two families: positives ($\otimes, \oplus, 1, 0$) and negatives ($\parr, \&, \bot, \top$). 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

[3] This condition is a simplified version Hughes and Van Glabbeek's toggling condition [12].

polarity as a single step (rule). This corresponds to a property known as focalization, discovered by Andreoli (see [2]), and which provides a strategy in proof-search: (i) negative connectives, if any, are decomposed immediately, (ii) we choose a positive focus, and persistently decompose it up to its negative sub-formulas.

Shift. To these standard connectives, it is convenient to add two new (dual) connectives, called Shift [4]: $\downarrow$ (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. When decomposing a positive connective into its negative subformulas (or viceversa), the shift marks the polarity change. The shift is the connective which *captures "time"* (or sequentiality): it marks a step in computation.

Focusing calculus. Focalization is captured by the following sequent calculus, originally introduced by Girard in [10], and closely related to the focusing calculus by Andreoli (see [2]). We refer to those papers for more details.

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 $\otimes L$, $\otimes R$, etc., we label the rules with the active formulas, in the way we describe below.

Positive connectives: Let $P(N_1, \ldots, N_n) = \bigoplus_{I \in \mathcal{N}} (\bigotimes_{i \in I} (\downarrow N_i))$, where I and $\mathcal{N}$ are index sets. Each $\bigotimes_{i \in I} (\downarrow N_i)$ is called an additive component. In the calculus, there is an introduction rule for each additive component. Let us write N_I for $\bigotimes_{i \in I} (\downarrow N_i)$.

$$\frac{\ldots \quad \vdash N_i, \Delta_i \quad \vdash N_{i'}, \Delta_{i'} \ldots}{\vdash P, \Delta} \ (P, N_I)$$

A positive rule is labelled with a pair: (i) the focus and (ii) the $\otimes$ of subformulas which appear in the premises (that is, the additive component we are using).

Negative connectives: Let $N(P_1, \ldots, P_n) = \&_{I \in \mathcal{N}} (\parr_{i \in I} (\uparrow P_i))$. We have a premise for each additive component. Let us write P_I for $\parr_{i \in I} (\uparrow P_i)$.

$$\frac{\ldots \quad \vdash P_I, \Delta \quad \vdash P_J, \Delta \ldots}{\vdash N, \Delta} \ \ldots, (N, P_I), (N, P_J), \ldots$$

A negative rule is labelled by a set of pairs: a pair of the form (focus, $\parr$ of subformulas) for each premise.

[4] The Shift operators have been introduced by Girard as part of the decomposition of the exponentials.

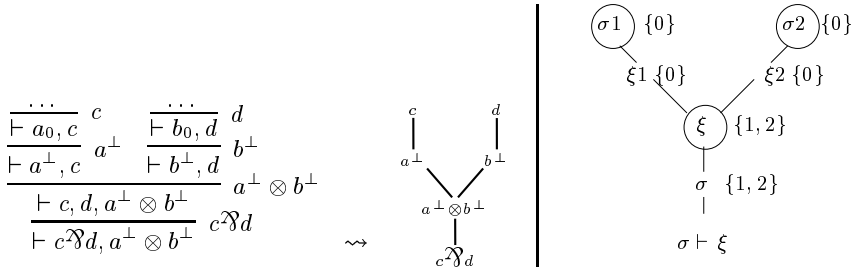


Fig. 1.

We call each of the pairs we used in the labels an *action.* (If a proof does not use $\&$, to each rule corresponds an action. Otherwise, there is an action for each additive component.)

It is important to notice the *duality* between positive and negative rules: to each negative premise corresponds a positive rule. For each action in a negative rule, there is a corresponding positive action, which characterizes a positive rule.

2.2 Designs as (untyped) focusing proofs

Given a focusing proof, we can associate to it a design (forgetting the types). Conversely, given a tree of actions which is a design, we have the "skeleton" of a sequent calculus derivation. This skeleton becomes a concrete (typed) derivation as soon as we are able to decorate it with types. Let us sketch this using an example.

First example. Consider the (purely multiplicative) derivation on the l.h.s. of Figure 1. Each rule is labelled by the active formula. $a^\perp, b^\perp$ denote negative formulas which respectively decompose into a_0, b_0. Notice that we deal with Shift implicitly, writing $a^\perp \otimes b^\perp$ for $\downarrow a^\perp \otimes \downarrow b^\perp$, and so on.

Now we forget everything in the sequent derivation, but the labels. We obtain the tree of labels (actions) depicted in Figure 1.

This formalism is more concise than the original sequent proof, but still carries all relevant information. To retrieve the sequent calculus counterpart is immediate. Rules and active formulas are explicitly given. Moreover we can *retrieve the context dynamically*. For example, when we apply the Tensor rule, we know that the context of $a^\perp \otimes b^\perp$ is c, d, because they are used afterwards (above). After the decomposition of $a^\perp \otimes b^\perp$, we know that c (resp. d) is in the context of $a^\perp$ because it is used after $a^\perp$ (resp. $b^\perp$).

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 name, a channel, or as the address in the memory 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 $a = ((p_1 \parr p_2) \oplus q^\perp) \otimes r^\perp$. If we locate a at the address ξ, we can locate $p_1 \parr p_2, q, r$ respectively in $\xi 1, \xi 2, \xi 3$ (the choice of addresses is arbitrary, as long as each occurrence receives a distinct address).

Let us consider an *action* (P, N_I), where $N_I = \bigotimes_{i\in I}(\downarrow N_i)$ is (ξ, K). Its translation is (ξ, K), where ξ is the address of P, and K is the set of natural numbers corresponding to the relative addresses of the subformulas N_i.

First example, continuation. Coming back to our example (Figure 1), let us abstract from the type annotation (the formulas), and work with addresses. We locate $a^\perp \otimes b^\perp$ 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 $c \parr d$ at the address σ and so on for its subformulas.

To indicate the polarity, in the pictures we circle positive actions (to remind that they are clusters of $\otimes$ and $\oplus$). Our example leads to the tree of actions on the r.h.s. of Figure 1, which is an actual design.

2.3 Understanding the additives (slices)

The treatment of the additive structure is based on the notion of slice.

A $\&$-rule must be thought of as the superposition of two unary rules, $\&_L, \&_R$. We write the two components of the rule which introduces $a\&b$ as $(a\&b, a)$ and $(a\&b, b)$. Given a sequent calculus derivation in Multiplicative Additive Linear Logic (MALL), if for each $\&$-rule we select one of the premises, we obtain a derivation where all $\&$-rules are unary. This is called a *slice* [9]. For example, the derivation on the l.h.s. below, can be decomposed into the slices on the r.h.s..

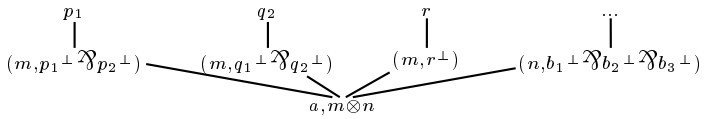
$$\dfrac{\dfrac{\dfrac{}{\vdash a, c}\quad\dfrac{}{\vdash b, c}}{\vdash a\&b, c}}{\vdash (a\&b)\oplus d, c} \quad\rightsquigarrow\quad \dfrac{\dfrac{\dfrac{}{\vdash a, c}}{\vdash a\&b, c}\,(a\&b, a)}{\vdash (a\&b)\oplus d, c} \qquad \text{and} \qquad \dfrac{\dfrac{\dfrac{}{\vdash b, c}}{\vdash a\&b,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

An $\&$-rule is a set (the superposition) of unary rules on the *same formula*. For this reason, we will write $a\&b$ also as $\{(a\&b, a), (a\&b, b)\}$.

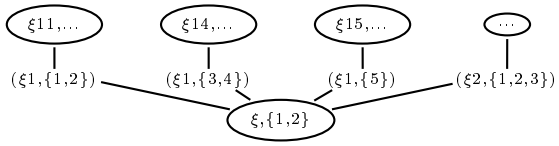
A more structured example. Let $a = (m \otimes n) \oplus c$,
$m = (p_1{}^\perp \parr p_2{}^\perp)\&(q_1{}^\perp \parr q_2{}^\perp)\&r^\perp$, $n = b_1{}^\perp \parr b_2{}^\perp \parr b_3{}^\perp$, with p_i, q_i, b_i positive formulas. Consider the following derivation, where the set of labels R_1 is
$\{(m, p_1{}^\perp \parr p_2{}^\perp), (m, q_1{}^\perp \parr q_2{}^\perp), (m, r^\perp)\}$ and R_2 is $\{(n, b_1{}^\perp \parr b_2{}^\perp \parr b_3{}^\perp)\}$.

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

It is immediate to obtain the corresponding typed design:



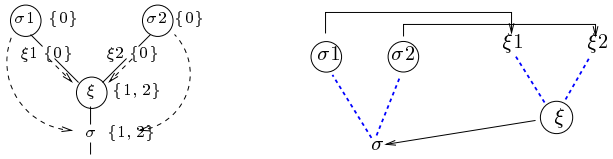
Let us now give addresses to the subformulas of A. The counterpart of the previous tree is the following one, which is actually a design.



Bipoles (reading a design). 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) and all the negative actions which follow it (all being at immediate subaddresses ξi of ξ). *Each address corresponds to a formula occurrence*. The positive action corresponds to a positive connective. The negative actions are partitioned according to the addresses: each address corresponds to a formula occurrence, each action on that address corresponds to an additive component.

Towards proof-nets. Let us consider a multiplicative design (a slice). We are given *two partial orders*, which correspond to two kinds of information on each action $\kappa = (\sigma, I)$: (i) a time relation (*sequential order*); (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 explicit the relation of being a subaddress with 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 [6]), with some additional information on sequentialization. Taking forward this idea of proof-nets leads us to L-nets.

3 Logical L-nets

In this section, we recall the notion of L-net of Faggian and Maurel [8], but we replace the acyclicity condition by the stronger cycles condition.

Actions (arena and moves). An *action* is either the special symbol † (called daimon) or (cf. above) a pair $k = (\xi, I)$ given by an address ξ and a finite set I of indices. When not ambigous, we write just ξ for the action (ξ, I). In the following, the letters k, a, b, c, d vary on actions.

We say that σ is a *subaddress* of ξ if ξ is a prefix of σ (written $\xi \sqsubseteq \sigma$). We say that an action (ξ, I) *generates* the addresses ξi, for all $i \in I$, and write $a \sqsubseteq_1 b$ if the action a generates the address of the action b (a is the parent of b). We will write $a \sqsubseteq b$ for

the transitive closure of this relation. Actions together with the relation $\sqsubseteq_1$ define what could be called a *universal arena*.

A *polarized action* is given by an action k together with a *polarity*, positive (k^+) or negative (k^-). The action $\dagger$ is defined to be positive. When clear from the context, or not relevant, we omit the explicit indication of the polarity.

L-nets (graph strategies) L-nets have an internal structure, described by a directed acyclic graph (d.a.g.) on polarized actions, and an interface, providing the names on which the L-net can communicate with the rest of the world.

An *interface* is a pair of disjoint sets Ξ, Λ of addresses (names), which we write as a sequent $\Xi \vdash \Lambda$. We call Λ the positive (or outer) names, and Ξ the negative (or inner) names. Ξ is either empty or a singleton. We think of the inner names as passive, or receiving, and of the outer names as active or sending.

Directed graphs and notations. We consider any directed acyclic graph G up to its transitive closure, and in fact we only draw the skeleton (the minimal graph whose transitive closure is the same as that of G). We write $a \leftarrow b$ if there is an edge from b to a. In all our pictures, the edges are oriented downwards. We use $\overset{*}{\leftarrow}$ for $\leftarrow\leftarrow \ldots \leftarrow$.

A node n of G is called *minimal* (resp. *maximal*) if there is no node a such that $a \leftarrow n$ (resp. $n \leftarrow a$). Given a node n, we denote by $\ulcorner n \urcorner_G$ (the *view* of n) the sub-graph induced by restriction of G on $\{n\} \cup \{n', n' \overset{*}{\leftarrow} n\}$ (we omit to indicate G whenever possible).

It is standard to represent a strict partial order as a d.a.g., where we have an edge from b to a whenever $a < b$. Conversely, (the transitive closure of) a d.a.g. is a strict partial order.

Definition 1 (pre L-nets). *A* pre L-net *is given by:*

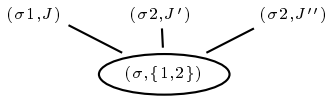
- *An interface $\Xi \vdash \Lambda$.*
- *A set A of nodes which are labelled by polarized actions*[5].
- *A structure on A of* directed acyclic bipartite graph *(if $k \leftarrow k'$, the two actions have opposite polarity) such that:*
  - *i.* Parents. *For any action $a = (\sigma, J)$, either σ belongs to the interface (and then its polarity is as indicated by the base), or it has been generated by a preceding action $c \overset{*}{\leftarrow} a$ of opposite polarity. Moreover, if a is negative, then $c \leftarrow a$.*
  - *ii.* Views. *For each action k, in $\ulcorner k \urcorner$ each address only appears once, i.e. all a's such that $a \overset{*}{\leftarrow} k$ are on distinct addresses.*
  - *iii.* Sibling. *Negative actions with the same predecessor are all distinct.*
  - *iv.* Positivity. *If a is maximal w.r.t. $\overset{*}{\leftarrow}$, then it is positive.*

To complete the definition of logical L-nets, we still need (i) a notion allowing us to deal with multiple copies of the same action induced by the additive structure and (ii) a correctness criterion on graphs. We first give a few definitions.

[5] Hence nodes are *occurrences* of actions, but we freely speak of actions for brevity.

Bipoles and rules. The positive actions induce a partition of the d.a.g. $\mathfrak{D}$ just described. A *bipole* (cf. previous section) is the tree we obtain when restricting $\mathfrak{D}$ either (i) to a positive action and the actions which immediately follow it, or (ii) to the negative actions which are initial (degenerated case).

Let us partition each bipole according to the addresses. A *rule* is a maximal set $\{(\xi, K_j)\}$ of actions which have the same address, and belong to the same bipole. A rule is positive or negative according to the polarity of its actions. When a rule is not a singleton, we call it an *additive rule* (think of each action as an additive component). An *additive pair* is a pair $(\xi, J)^-, (\xi, J')^-$ belonging to an additive rule. Observe that if a rule is not a singleton, it must be negative. If we look at the bipole in the following picture, we have two rules: $R_1 = \{(\sigma 1, J)\}$ and $R_2 = \{(\sigma 2, J'), (\sigma 2, J'')\}$.



Paths. An edge is an *entering edge* of the action a if it has a as target. 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 *path* is a sequence of nodes $k_1, \ldots k_n$ belonging to distinct rules, and such that for each i either $k_i \rightarrow k_{i+1}$ (the path is going down) or $k_i \leftarrow k_{i+1}$ (the path is going up). A *switching path* on a pre L-net is a path which uses at most one switching edge for each negative rule. A *switching cycle* is a cycle (on a sequence of nodes $k_1, \ldots k_n$ belonging to distinct rules) which contains at most one switching edge for any negative rule.

Definition 2 (logical L-net). *A* logical L-net *is a pre L-net such that*

- **Additives.** *Given two positive actions $k_1 = (\xi, K_1), k_2 = (\xi, K_2)$ on the same address, there is an additive pair w_1, w_2 such that $k_1 \overset{*}{\leftarrow} w_1$, and $k_2 \overset{*}{\leftarrow} w_2$.*
- **Cycles.** *Given a non-empty union C of switching cycles, 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 $w_2 \overset{*}{\leftarrow} c_2$.*

L-nets as sets of views / chronicles. We call *chronicle* (view) a set $\mathfrak{c}$ of actions equipped with a partial order, such that: $\mathfrak{c}$ has a unique maximal element (the apex), and satisfies the (analog of the) parent condition.

Any node k in a pre L-net $\mathfrak{D}$ defines a chronicle, which is $\overline{\ulcorner k \urcorner}$, where the overlining operation is defined on directed acyclic graphs G whose nodes are injectively labelled, as follows: replace all nodes of G by their labels, yielding a graph G' isomorphic to G, Then $\overline{G}$ is the transitive closure of G', i.e., G' viewed as a strict partial order (cf. above). We can associate to each L-net $\mathfrak{D}$ a set of chronicles $\phi(\mathfrak{D})$, as follows:

$$\phi(\mathfrak{D}) = \{\overline{\ulcorner n \urcorner} \mid n \text{ is a node of } \mathfrak{D}\}$$

The set $\phi(\mathfrak{D})$ is closed downwards, in the following sense: if $\mathfrak{c} \in \phi(\mathfrak{D})$, if k is the maximal action of $\mathfrak{c}$, and if $k' \in \mathfrak{c}$ is such that k covers k', i.e., $k' < k$ and there exists no $k'' \in \mathfrak{c}$ such that $k' < k'' < k$, then $\ulcorner k' \urcorner$ (taken with respect to $\mathfrak{c}$) belongs to $\phi(\mathfrak{D})$.

Conversely, given a set Δ of chronicles which is closed downwards, we define a directed graph $\psi(\Delta)$ as follows: the nodes are the elements of Δ and the edges are all the

pairs of the form $(\mathfrak{c}', \mathfrak{c})$ such that, if k, k' are the maximal actions of $\mathfrak{c}$, $\mathfrak{c}'$, respectively, then $k' \in \mathfrak{c}'$ and k covers k' (in $\mathfrak{c}$). It is easy to see that for any downwards closed set of chronicles Δ we have $\phi(\psi(\Delta)) = \Delta$. Conversely, given an L-net $\mathfrak{D}$, we have that $\psi(\phi(\mathfrak{D}))$ is isomorphic as a graph to $\mathfrak{D}$.

The functions ϕ and ψ are inverse bijections (up to graph-isomorphisms of L-nets) between the collection of L-nets and the set of downward closed sets of chronicles Δ such that $\psi(\Delta)$ is an L-net.

In this paper, we will largely rely on the presentation of L-nets as sets of chronicles (views). This in particular allows us to treat easily the superposition of two L-nets as the union of the two sets of chronicles (see section 5.2). We shall write write $\mathfrak{c} \in \mathcal{S}$ and $\mathcal{S} \subseteq \mathfrak{D}$ for $\mathfrak{c}, \mathcal{S}, \mathfrak{D}$ respectively a chronicle, a set of chronicles and an L-net.

Slices A *slice* is an L-net in which there is no additive pair (or, equivalently, no repetition of addresses). A slice $\mathfrak{S}$ of an L-net $\mathfrak{D}$ is a maximal subgraph of $\mathfrak{D}$ which is closed under view ($\ulcorner k \urcorner_{\mathfrak{S}} = \ulcorner k \urcorner_{\mathfrak{D}}$) and it is a slice.

L-nets and logical L-nets Our definition of logical L-net differs from the defininition of L-nets in [8] in the cycles condition, which replaces the acyclicity condition of L-nets, which asserts that there are no switching cycles in a slice. It is immediate that our *cycles condition* implies the acyclicity condition. Hence, a logical L-net is, in particular, an L-net. Notice that while acyclicity is a property of a slice, the new condition speaks of cycles which traverse slices.

Designs. The designs of [11], can be regarded as a special case of L-nets: they are those L-nets such that each positive node is the source of at most one negative node, and each negative node has a single entering edge. Equivalently, the L-nets corresponding to designs are those which are trees that branch only on positive nodes.

4 Sequentializing a graph strategy

A node in an L-net should be thought of as a cluster of operations which can be performed at the same time. An edge states a dependency, an enabling relation, or a precedence among actions. Let us consider a very simple example: a chronicle $\mathfrak{c}$, i.e. a partially ordered view (p.o. view). 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 which extends c will define a complete scheduling of the tasks, in such a way that each action is performed only after all of its 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 aim of this section is to provide a procedure, which takes an L-net and adds sequentiality in such a way that the constraints specified by the L-net are respected. In particular, all actions in a p.o. view of $\mathfrak{D}$ will be contained in a (totally ordered) view of the tree $Seq(\mathfrak{D})$. The process of sequentialization is non-deterministic, as one can expect, i.e. there are different ways to produce a design from a logical L-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 chronicle is completely determined, 2. separate slices which are shared through additive superposition.

The key point in sequentialization is to select a rule which does not depend on others. This is the role of the Splitting lemma.

Lemma 1 (Splitting lemma). *Given an L-net $\mathfrak{D}$ which satisfies the cycles condition, if $\mathfrak{D}$ has a negative rule, then it has a splitting negative rule. A negative rule $W = \{\dots, w_i, \dots\}$ is splitting if either it is conclusion of the L-net (each w_i is a root), or if deleting all the edges $w_i \to w$ there is no more connection (i.e., no path) between any of the w_i and w.*

The proof is an adaptation to our setting of the proof of the similar lemma in [12]. Moreover, the proof implies that

Proposition 1. *The splitting negative rule W can always be chosen of minimal height: either it is conclusion of the L-net, or it is above a positive action, which is conclusion.*

Remark 1. A consequence of the previous proposition is that, when applying the splitting lemma, we are always able to work "bottom up".

4.1 Sequentialization

An L-net does not need to be connected. This is a natural and desirable feature if we want both parallelism and partial proofs, that is proofs which can be completed into a proper proof. Actually, non-connectedness is an ingredient of Andreoli's concurrent proof construction. On the logical side, non-connectedness corresponds to the mix rule.

There is no special problem for sequentializing non-connected L-nets, except that we need to admit the mix-rule. But as the (controversial) mix rule is refused by designs, we distinguish logical L-nets which are connected.

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 edge for each negative node, and deleting all the other ones. A slice is *S-connected* if all its possible switching graphs are connected. Finally, we call an L-net S-connected if all its maximal slices are.

Proposition 2. *A logical L-net $\mathfrak{D}$ which is S-connected can be sequentialized into a design, or (equivalently) into its sequent calculus presentation.*

Remark 2. If we admit mix, it is easy to adapt the procedure below to sequentialize any logical L-net.

Proof. The proof is by induction on the number N of negative nodes of the L-net $\mathfrak{D}$.

Case 1: $N = 0$. $\mathfrak{D}$ consists of a single positive action k, which does not need further sequentialization.

Case 2: $N > 0$ and there are negative initial nodes. By definition of L-net, all negative nodes which are initial belong to the same rule $W = \{\dots, w_i, \dots\}$.

Let $\mathfrak{D}_i$ be the union of all slices $\mathfrak{S} \subseteq \mathfrak{D}$ such that $w_i \in \mathfrak{S}$. That is, $\mathfrak{D}_i$ is the maximal L-net obtained as set of all chronicles $\mathfrak{c}$ such that $w_j \notin \mathfrak{c}$, for any $w_j \neq w_i$. It is immediate that, operationally, $\mathfrak{D}_i$ is the graph obtained from $\mathfrak{D}$ following these two steps: (i) delete all nodes c such that $w_j \leftarrow^* c$, for $j \neq i$; (ii) delete any negative node which has become a leaf.

$\mathfrak{D}_i$ is S-connected. Let $\mathfrak{D}'_i$ be the tree obtained from $\mathfrak{D}_i$ by removing w_i and by sequentializing the resulting L-net. $\mathfrak{C}_i = \overset{\mathfrak{D}'_i}{\underset{w_i}{|}}$ is a design. The forest given by the union of all $\mathfrak{C}_i$ is a design:

$$\dfrac{\dfrac{\dots}{\vdash \xi_I, \Delta} \quad \dfrac{\dots}{\vdash \xi_J, \Delta}}{\xi \vdash \Delta}\ W$$

Case 3: $N > 0$ and there are no negative initial nodes. We select a splitting negative rule $X = \{x_1 = (\xi i, J_1), \dots, x_n = (\xi i, J_n)\}$. This rule is part of a bipole, with root $k = (\xi, I)$ and possibly other negative rules Y_j. We delete the edges from $x \in X$ to k, disconnecting $\mathfrak{D}$.

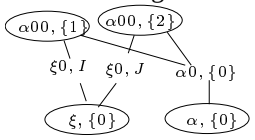
Let us call G_X the part of the graph containing X, and G_k the other part. Let us check that the cycles condition is preserved for both G_X and G_k (preservation of all other properties is immediate). In the case of G_k it is obvious, in the case of G_X it comes from the fact that k determines a "bottle-neck" in the graph, as any path going down from G_X to G_k must traverse k. Let us assume that there are switching cycles in G_X, hence a fortiori in $\mathfrak{D}$. The cycles condition for $\mathfrak{D}$ implies that there is an additive pair w_1, w_2 such that each w_i is below a node c_i in one of the cycles. If w_1, w_2 were in G_k, any path going down from c_i to w_i should traverse k. This would mean that there is a path down from k to w_i for each w_i, and hence that both w_i belong to $\ulcorner k \urcorner$, which is against the definition of L-net.

We conclude by applying induction. G_k will sequentialize into a design containing the node k. G_X will sequentialize into a set of trees of roots respectively $x_1, \dots, x_n$. We obtain a design by having each of these trees pointing to k.

$$\dfrac{\dfrac{\overline{\xi i \vdash \Delta_i}\ X \quad \dots \quad \overline{\xi j, \Delta}\ Y_j}{\vdash \xi, \Delta}\ (\xi, I) \qquad \dots}{\dots}$$

4.2 Examples of sequentialization

Let us consider the following L-net $\mathfrak{R}$, where we have two negative rules, both splitting:



$X = \{(\xi 0, I), (\xi 0, J)\}$ and $A = \{(\alpha 0, \{0\})\}$.

If we choose X, we obtain the two trees on the left-hand side of Figure 2, and then the design $\mathfrak{X}$. Instead, choosing A we obtain the design $\mathfrak{A}$ (on the r.h.s.).

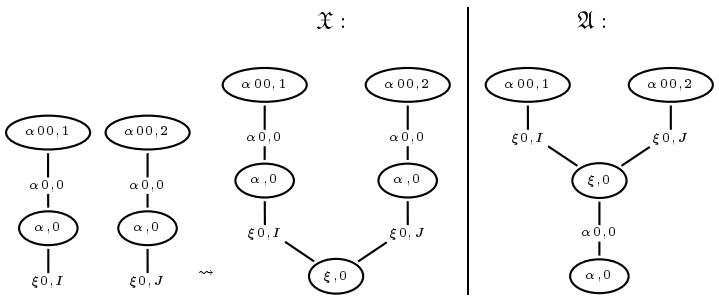


Fig. 2.

5 Desequentializing a tree strategy

Beyond the fact that an action can be seen as a cluster of operations that can be performed together thanks to focalization, in a design (actually, in any tree strategy) remains a lot of artificial sequentiality, just as in sequent calculus proofs for Linear Logic. In the case of proofs, the solution has been to develop proof-nets, a theory which has revealed itself extremely fruitful.

We want to apply similar techniques to designs. Our aim in this section is to remove some artificial sequentialization, while preserving essential sequentialization, namely that allowing to recover axioms and to deal with additives.

All dependency (sequentialization) which is taken away by desequentialization can be (non-deterministically) restored through sequentialization (Theorem 1).

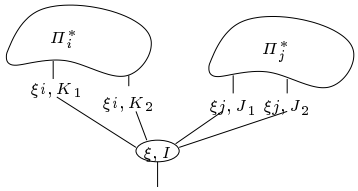
5.1 Desequentialization

It is rather immediate to move from designs to an explicit sequent calculus style representation. We already sketched this with an example, and refer to [11] for the details (notice that, because of weakening, there are several sequent calculus representations of a design). To each node k in a design we can associate a sequent of addresses, corresponding to the sequent on which the action is performed. We choose an algorithm which performs weakening as high as possible in the derivation, pushing it to the leaves.

Leaves. For each leaf k in a design, we can recover the sequent of addresses corresponding to the sequent on which that action is performed.

Given a leaf k in the design, its translation k^* is the same node k, to which we explicitly associate a set of addresses, which we call $link(k)$, in the following way:
if k is either the action of address ξ on the sequent $\dfrac{}{\vdash \xi, \Gamma}\ k = (\xi, I)$ or the special action $\dagger$ on $\dfrac{}{\vdash \Gamma}\ k = \dagger$, we have $link(k) = \Gamma$.

Positive conclusion. Let us condider a design whose root is a positive action (ξ, I), and call Π_i the forest of subtrees whose conclusions have address ξi. The design translates



into the L-net
in the following way. Associate the L-net Π_i^* to each Π_i. Take the union of all Π_i^*. Add $(\xi, I)^+$ to the nodes, and extend the set of edges with a relation $(\xi, I) \leftarrow k$ for each action k of address ξi.

Negative conclusion. Let us consider a design having as conclusion the negative rule $X = \{x_i = (\xi, I)^-, x_2 = (\xi, J)^-, ...\}$. Let us call Π_I the subtree above (ξ, I). A design of negative conclusion translates into an L-net in the following way.

1. For each subtree (premiss) Π_I do the following.
   - Associate the L-net Π_I^* to Π_I.
   - Add $(\xi, I)^-$ to the nodes of Π_I^*.
   - Extend the set of edges with a relation $(\xi, I)^- \leftarrow k$ for each action k such that:
     - k has address ξi ($i \in I$), or
     - k is a leaf such that $\xi i \in link(k)$.

   Let us call $\mathfrak{D}_I$ the resulting graph (which is an L-net).

2. Consider $\mathfrak{D}_I, \mathfrak{D}_J, \ldots$. Obtain $\mathfrak{D}'_I, \mathfrak{D}'_J, \ldots$ by extending the set of edges of each $\mathfrak{D}_I$ with a relation $(\xi, I)^- \leftarrow k$ for each positive node k such that $\ulcorner k \urcorner \in \mathfrak{D}_I, \ulcorner k \urcorner \notin \mathfrak{D}_J$, for some $J \neq I$.

3. Superpose $\mathfrak{D}'_I, \mathfrak{D}'_J, \ldots$. Superposition is obtained by taking the union of the chronicles (see [8] and the examples below).

Superposition is the only step which can introduce cycles. However, if a new cycle C is introduced we find a node $c > x_i$ and a node $c' > x_j$, for $x_i, x_j \in X$.

We have the following result, relating desequentialization and sequentialization.

Theorem 1. *Given (a sequent calculus representation of) a design $\mathfrak{D}$, let us desequentialize it into the L-net $\mathfrak{R}$. There exists a strategy of sequentialization (section 4.1) which allows us to sequentialize $\mathfrak{R}$ into $\mathfrak{D}$.*

The proof comes from the fact that for each step in the desequentialization there is a step of sequentialization which reverses it.

5.2 Examples of superposition

The superposition of two L-nets is their union as sets of chronicles. Let us see an example. Consider the two L-nets $\mathfrak{D}_1, \mathfrak{D}_2$ in Figure 3. 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 chronicles of $\mathfrak{D}_1$ is the set of chronicles $\ulcorner \kappa \urcorner$ defined by each of its

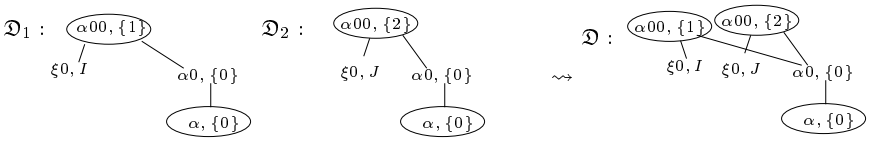


Fig. 3.

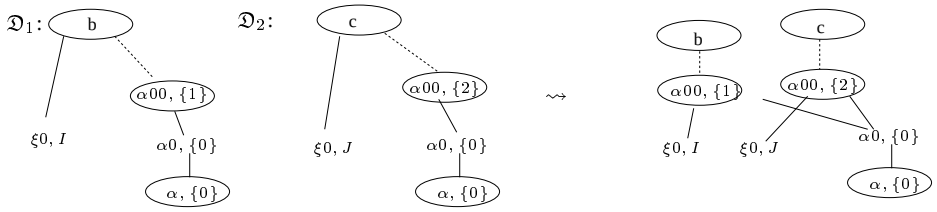
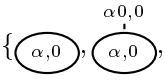
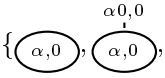
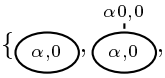


Fig. 5.

actions κ, that is:

$\{$  , $(\xi 0, I), \ulcorner(\alpha 00, \{1\})\urcorner = \mathfrak{D}_1\}$. The set of chronicles of $\mathfrak{D}_2$ is:

$\{$  , $(\xi 0, J), \ulcorner(\alpha 00, \{2\})\urcorner = \mathfrak{D}_2\}$. The resulting union is:

$\{$  , $(\xi 0, I), (\xi 0, J), \mathfrak{D}_1, \mathfrak{D}_2\}$, which corresponds to $\mathfrak{D}$.

5.3 Examples of desequentialization

Example 1. Desequentializing either of the designs $\mathfrak{A}$ or $\mathfrak{X}$ in our previous example of sequentialization yields the original L-net $\mathfrak{R}$ (cf. section 4.2).

Example 2. Let us consider the design in Figure 4, where we just omit an obvious negative action at the place of . . . .

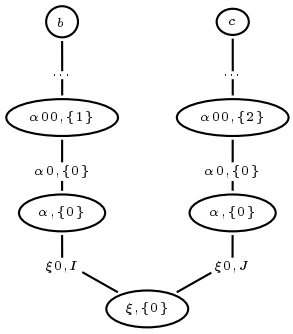


Fig. 4.

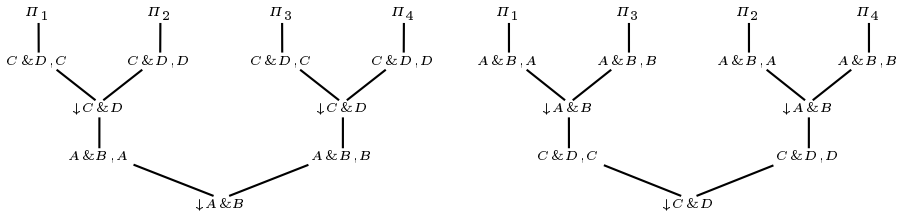
Following the procedure for desequentializing given above, a few easy steps produce the two L-nets $\mathfrak{D}_1, \mathfrak{D}_2$, represented in Figure 5. Observe that we have a chronicle for each node; $\mathfrak{D}_1 \bigcap \mathfrak{D}_2$ is equal to $\{\ulcorner(\alpha, \{0\})\urcorner, \ulcorner(\alpha 0, \{0\})\urcorner\}$. We obtain $\mathfrak{D}'_1$ by adding the relation $(\xi 0, I) \leftarrow (\alpha 00, \{1\})$, and $\mathfrak{D}'_2$ in a similar way. Remember that we consider each chronicle in the graph modulo its underlying partial order, that is why it is not necessary to explicitly write the edge $(\xi 0, b)$. The union $\mathfrak{D}'_1 \bigcup \mathfrak{D}'_2$ produces the L-net on the right-hand side of Figure 5.

5.4 A typed example: additives

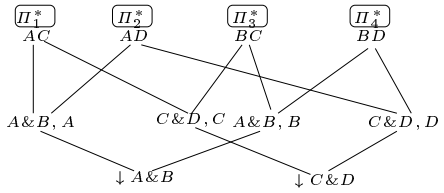
The following (typical) example with additives illustrates what it means to have more parallelism. Assume we have derivations $\Pi_1, \Pi_2, \Pi_3, \Pi_4$ of (respectively) $\vdash A, C$, $\vdash A, D$, $\vdash B, C$, $\vdash B, D$. In the sequent calculus (and in proof-nets with boxes) there are two distinct ways to derive $\vdash A\&B, C\&D$, and the two derivations differ only by commutations of the rules.

$$\dfrac{\dfrac{\dfrac{\Pi_1}{\vdash A, C}\quad\dfrac{\Pi_2}{\vdash A, D}}{\vdash A, C\&D}\,C\&D \qquad \dfrac{\dfrac{\Pi_3}{\vdash B, C}\quad\dfrac{\Pi_4}{\vdash B, D}}{\vdash B, C\&D}\,C\&D}{\vdash A\&B, C\&D}\,A\&B \qquad\qquad \dfrac{\dfrac{\dfrac{\Pi_1}{\vdash A, C}\quad\dfrac{\Pi_2}{\vdash A, D}}{\vdash A\&B, C}\,A\&B \qquad \dfrac{\dfrac{\Pi_3}{\vdash B, C}\quad\dfrac{\Pi_4}{\vdash B, D}}{\vdash A\&B, D}\,A\&B}{\vdash A\&B, C\&D}\,C\&D$$

The same phenomenon can be reproduced in the setting of designs, or in the setting of polarized linear logic. Very similar to the above derivations are the two following (typed) designs, where we introduced some $\downarrow$ to have distinct binary connectives. We write formulas instead of addresses, to make the example easier to grasp.



The desequentialization of either of the trees above is the following L-net $\mathfrak{R}$:



Conversely, when sequentializing $\mathfrak{R}$, we get back either one or the other, depending on whether we choose to start from $A\&B$ or from $C\&D$. Notice that both $A\&B$ and $C\&D$ are splitting.

6 Discussion and further work

We can isolate two classes of L-nets, those of maximal sequentiality (the tree strategies), which are idempotent with respect to Seq and those of minimal sequentiality. Notice that while Seq applies to arbitrary L-nets, here we have defined $Deseq$ only on trees. This is still enough to characterize also the class of L-nets of minimal sequentiality, as those for which we have $Deseq(Seq(\mathfrak{D})) = \mathfrak{D}$, for any choice in $Seq(\mathfrak{D})$.

We expect to be able to define the desequentialization of arbitrary L-nets, by using the splitting Lemma. Moreover, we believe that sequentialization and desequentialization can be extended to infinite L-nets, by working bottom-up lazily, or stream-like.

In the setting we presented, if we have just enough sequentiality to recover axioms and dependencies from the additives, we obtain (an abstract counter-part of) MALL

proof-nets. At the other extreme, all sequentiality can be made explicit, and we have designs "à la locus solum" [11] (or abstract polarized MALL $\downarrow\uparrow$ proof nets as in [14]). L-nets allow us to vary between these extremes, and hence provide us with a framework in which we can graduate sequentiality.

Here we are strongly inspired by a proposal by Girard, to move from proof-nets to their sequentialization (sequent calculus derivation) in a continuum, by using jumps. It must be noticed that edges inducing sequentiality in L-nets actually correspond to Girard's jumps.

We need to understand better this gradient of sequentiality. (i) In this paper we saturate L-nets to maximal sequentiality. We intend to study ways to perform sequentialization gradually, adding sequential edges progressively. (ii) We would like to have a more precise understanding of what it means to have maximal or minimal sequentiality, and to investigate the extent of our desequentialization.

In future work, we wish to investigate a typed setting. The immediate typed counterpart of logical L-nets should be focusing proof-nets [3]. While previous work on focusing proof-nets was limited to multiplicative linear logic, our framework extends to additive connectives.

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

References

1. R. Amadio and P.-L. Curien. *Domains and Lambda-calculi*. Cambridge University Press, 1998.
2. J.-M. Andreoli. Focussing and proof construction. *Annals of Pure and Applied Logic*, 2001.
3. J.-M. Andreoli. Focussing proof-net construction as a middleware paradigm. In *Proceedings of Conference on Automated Deduction (CADE)*, 2002.
4. P.-L. Curien. Abstract bohm trees. *MSCS*, 8(6), 1998.
5. P.-L. Curien. Introduction to linear logic and ludics, part ii. to appear in Advances of Mathematics, China, available at www.pps.jussieu.fr/curien, 2004.
6. C. Faggian. Travelling on designs: ludics dynamics. In *CSL'02*, volume 2471 of *LNCS*. Springer Verlag, 2002.
7. C. Faggian and M. Hyland. Designs, disputes and strategies. In *CSL'02*, volume 2471 of *LNCS*. Springer Verlag, 2002.
8. C. Faggian and F. Maurel. Ludics nets, a game model of concurrent interaction. In *Proc. of LICS (Logic in Computer Science)*. IEEE Computer Society Press, 2005.
9. J.-Y. Girard. Linear logic. *Theoretical Computer Science*, (50):1–102, 1987.
10. J.-Y. Girard. On the meaning of logical rules i: syntax vs. semantics. In Berger and Schwichtenberg, editors, *Computational logic*, NATO series F 165, pages 215–272. Springer, 1999.
11. J.-Y. Girard. Locus solum. *MSCS*, 11:301–506, 2001.
12. D. Hughes and R. van Glabbeek. Proof nets for unit-free multiplicative-additive linear logic. *ACM Transactions on Computational Logic*, 2005.
13. M. Hyland and L. Ong. On full abstraction for PCF. *Information and Computation*, 2000.
14. O. Laurent. *Etude de la polarisation en logique.* PhD thesis, 2002.