Ludics nets, a game model of concurrent interaction

Claudia Faggian
Universitá di Padova, Italy

Francois Maurel
PPS, Paris 7, France

Abstract

We propose L-nets as a game model of concurrent interaction*. L-nets, which correspond to strategies (in Games Semantics) or designs (in Ludics), are graphs rather than trees; the interactions (plays) result into partial orders.*

1 Introduction

Game Semantics has been a successful approach to model sequential computation. Its strength is to capture the dynamical aspects of computation. Typically, the possible interactions (*plays*) between a program and its environment are represented as linearly ordered sequences of moves, forming a tree. More recent work has been directed to extend this approach, either to describe a more concurrent form of interaction ([1, 9]) or to liberalize the notion of game in order to play on graph structures ([10, 12]).

Ludics [8] has been proposed by Girard as an abstract setting for proof-theory, founded on interaction. Ludics is a game model which presents remarkable structural properties. In particular, it comes equipped with a built-in notion of observational equivalence, in the sense that two agents reacting the same way to any test are actually equal. The central role of addresses (names) and the interactive methods open a bridge with concurrency theory. At front of this, the original theory imposes constraints which induces a strict sequentiality. It is a very natural step to seek a more asynchronous or "concurrent" notion of interaction.

The work we present[1] stems from both of these directions. We introduce abstract structures corresponding to proofs/programs: L-nets. L-nets correspond to what is called design in Ludics, and strategy in Game Semantics. Standard strategies and designs are trees, and interactions are linear sequences of moves. L-nets are graphs, stating a controlled amount of dependency between actions, and the interactions are partial orders, allowing for parallelism. If tree strategies can be seen as *abstract sequent calculus derivations*, L-nets are *abstract proof-nets*, in multiplicative additive linear logic (MALL). While the multiplicative structure deals with parallelism, the additive structure accounts for a proof-theoretical counterpart of non determinism.

Sections 3 and 4 are concerned simply with Game Semantics: we introduce parallel strategies, the L-nets, describe composition, and show that it is associative.In Section 5 we study observational equivalence on L-nets. In Section 6 we move to types, and develop the high-level architecture of Ludics on L-nets.

Several intuitions underly our work, and they will represent directions for further development: proof-nets (as in [6, 7]), Winskel's event structures [13]), multi-focalization (focusing proof-nets) as raised by Andreoli in the distributed construction of proof-nets ([2]).

The intuition provided by proof-nets has actually been a strong guide, and we are going to use it all along the paper, even though the formal development of the proof-nets syntax we have in mind will be postponed to future work (see [3] for a first account in this direction). Our approach departs from the standard ones to proof-nets in the way it deals with sequentiality and with the additives. To handle these aspects, we rather exploit and take forward ideas proposed by Girard in [8].

Additives and slices. To understand our approach to the additives, it is important to understand the notion of slice. Let us (informally) introduce it.

A $\&$-rule must be thought of as the "superposition" of two unary $\&$ rules, $\&_L, \&_R$. Given a sequent calculus derivation, if for any $\&$-rule we select one of the premises, we obtain a derivation where all $\&$-rules are unary. This is called a *slice* ([6]). We write the two components of a rule which introduces $a\&b$ as $(a\&b, a)$ and $(a\&b, b)$, and write $a\&b$ also as $\{(a\&b, a), (a\&b, b)\}$.

It has been an idea of Linear Logic since long that slices are the perfect syntax for additive proof-nets: a MALL proof should be seen as the superposition of all its slices. How to "superimpose" the slices is the difficult point. Normalization of designs in [8] works exactly in this way: by slices. The same approach is followed by [11] for po-

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

larized proof-nets. In both cases, sequentiality (given by boxes in polarized proof-nets) offers enough "synchronization points" to be able to recompose slices after normalization.

A major achievement of this paper is to show that even relaxing sequentiality enough to take away boxes, it is still possible to work by slices. Here we do this in an abstract setting, but we expect to be able to have the same result for typed proof-nets, that is MALL proof nets.

Parallelism and non-determinism. It is well known that multiplicative (MLL) proof-nets allow for parallelism. Why do we insist on additives? The underlying insight, which is also not new, is that additives are the proof-theoretical counterpart of non-determinism. A & rule allows us to superimpose different proofs, that is (read bottom-up) different possible continuations for the process, different possible ways to evolve in the future. Each slice represents a possible branching in the evolution of the process.

Note. In this paper we had to cut the proofs, which are provided in the full version.

2 An overview: strategies and interaction

The role of a strategy in Game Semantics is to tell the player how to respond to a counter-player move. The dialogue between the two players (let us call them P and O) will produce an interaction (a play).

Figure 1 presents a very simplified example of two tree strategies (we forget any detail about pointers). A specific move is played by (belongs to) only one of the players, so there are P-moves and O-moves. The active (positive) move of P are those that P plays, while its passive (negative) moves are those played by O, and to which P has to respond. In the picture, for each player strategy we distinguish the actives (positive) moves, i.e. those which belong to that player, with circles.

Let us look at the strategies (1). According to the P-player strategy, it will start with b_0, then respond with a_0 to Opponent move b_1, and with † (termination) to Opponent move b_2. Let us make it interact with the O-player strategy. The interaction goes as follows: O answer to b_0 is b_1, P answer to b_1 is a_0, O answer to a_0 is a_1, and so on.

Our algorithm to calculate the interaction is simple. (i.) Start from P-player initial move, (ii.) Check counter-player answer to that move, that is, go to the corresponding opposite action, and take the following move. (iii.) Repeat step (ii.) until terminating on †.

Figure 2 illustrates the idea of strategy we are going to develop.

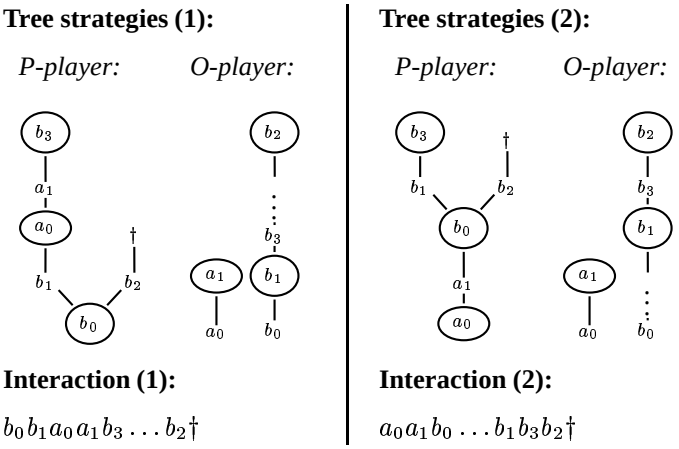


Figure 1. Tree strategies

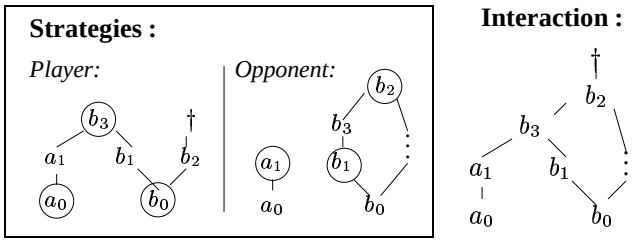


Figure 2. Graph strategies

The strategies are graphs and the interaction (which we are going to describe in a later section) is now a partial order. The way to calculate it is similar to the previous one, but (1.) there are several threads running in parallel, (2.) on certain moves we need to synchronize.

Observe as both tree strategies respect all precedence constraints described by the graph strategy. Both interactions (1) and (2) are linear extensions of the partial order which describe the interaction between the graph strategies (however, (1) and (2) schedule the actions in two different ways).

3 Statics: strategies (L-nets)

Moves: addresses and actions. An *address* ξ is a sequence of indices, which are just natural numbers. An *action* is either the special symbol † (called daimon) or a pair $k = (\xi, I)$ given by an address ξ and a finite set I of indices. In the following, the letters k, a, b, c, d vary on actions.

We say that σ is a *sub-address* of ξ if ξ is a prefix of σ (written $\xi \sqsubset \sigma$); ξi is an immediate sub-address of ξ. We say that an action (ξ, I) *generates* the addresses ξi, for all $i \in I$, and write $a \sqsubset_1 b$ if the action a generates the address of b (b is justified by a). We will write $a \sqsubset b$ for the transitive closure of this relation.

A *polarized action* is given by an action k together with

a *polarity*, positive (k^+) or negative (k^-). The action † is defined to be positive. When clear from the context, or not relevant, we omit the explicit indication of the polarity. In all our pictures, the positive actions will be circled. When not ambiguous, we write just ξ for the action (ξ, I).

Pre L-nets. 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. In all our pictures, the edges are oriented downward. We consider any directed acyclic graph (d.a.g.) up to its transitive closure. An edge from a to b is transitive if there is an oriented path from a to b not using that edge. We draw explicitly only edges which are not transitive, denoted $a \leftarrow b$ (a immediately precedes b). We use $\overset{*}{\leftarrow}$ for $\leftarrow\leftarrow ... \leftarrow$.

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

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

Definition 3.1 (Pre L-net) *A* pre L-net *is given by:*

- *An interface* $\Xi \vdash \Lambda$.
- *A set A of nodes which are labeled by polarized actions. For any action $a = (\sigma, J)$, σ is sub-address of a name in the interface. If σ belongs to the positive (resp. negative) names of the interface, then a is positive (resp. negative). If the action a is generated by an action k, then a and k have opposite polarity.*
- *A structure on A of* directed acyclic bipartite graph *(if $k \leftarrow k'$, the two actions have opposite polarity) such that:*
  - *i.* Parents (justification). *For any action $a = (\sigma, J)$, either σ belongs to the interface, or it has been generated by a preceding action c. If c is positive, and $c^+ \leftarrow a^-$, then $c \sqsubset_1 a$ (a is justified by c).*
  - *ii.* Unicity (linearity). *Given an action k, in $\ulcorner k \urcorner$ each address only appears once.*
  - *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.*

Chronicles (views). We call *chronicle* (view) a structure (of directed acyclic bipartite graph) with a unique maximal action (the apex), and satisfying conditions (i) and (ii) above. Any action k in a pre L-net $\mathfrak{D}$ defines a chronicle, which is $\ulcorner k \urcorner$. Conversely, a pre L-net can be described as a set of views (as standard for innocent strategies).

Fact 3.2 (Views as partial orders) *In a chronicle (or view), any action occurs at most once. The directed acyclic graph therefore defines a partial order on its actions: $k < k'$ iff $k \overset{*}{\leftarrow} k'$.* We identify a chronicle with its actions equipped with the partial order $<$. *The same applies to slices, which we define next.*

Slices. A subgraph $\mathfrak{S}$ of $\mathfrak{D}$ which is closed under view ($\ulcorner k \urcorner_{\mathfrak{S}} = \ulcorner k \urcorner_{\mathfrak{D}}$) and satisfies (iv) is a *slice of* $\mathfrak{D}$ if each address only appears once.

Comments. Here we think of a view (a chronicle) as a "desequentialization" of a standard innocent strategy view. The intuition is that some order information is irrelevant[2]. Observe that the address codes the justifier. The extra structure (the set of indices) will manage the additive structure.

Condition (i) says that *the relation $\overset{*}{\leftarrow}$ respects the prefix order*: $\sigma \sqsubset \tau \Rightarrow \sigma \overset{*}{\leftarrow} \tau$. Notice that if we read positive as Player, and negative as Opponent, the condition on negative actions corresponds to the usual condition on views for innocent strategies. This condition subsumes both justification and innocence. Observe also that, as † has no subaddresses, such an action is always maximal. Condition (iii) implies that if two sibling negative actions have the same address, they must have different sets of indices.

By construction, a pre L-net is well founded. Observe that if the interface is $\xi \vdash \Lambda$, an action is minimal and negative iff it has address ξ (hence all minimal negative actions have the same address).

Bipoles and sequential links. To understand an L-net, it is useful to decompose it into conceptual units. The positive actions induce a partition of the d.a.g., as follows. A *bipole* is what we obtain when restricting a pre L-net either (i) to a positive action and the set of actions which immediately follow it, or (ii) to the negative actions which are minimal (degenerated case). Definition 3.1 implies that the bipole in case (i) is a depth-one tree, where the root is positive and generates the negative actions (the two relations $\sqsubset_1$ and $\leftarrow$ coincide).

[2] Similar generalizations on views were propounded by Martin Hyland.

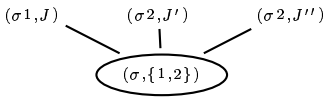
We call *sequential link* an *oriented* edge $a^- \leftarrow b^+$, going from a positive action to a negative one. *A pre L-net can be seen as built from bipoles, connected together using sequential links.*

L-nets. In an action there is more than just an address and a polarity. The extra information (the set of indices) allows us to manage the additive structure of L-nets.

To complete the definition of L-nets, we still need (i) a correctness criterium on graphs, to guarantee a good behaviour when normalizing (as it is standard in the theory of proof-nets) and (ii) a notion allowing us to deal with multiple copies of a same action, a situation which is induced by the additive structure. Before giving the definition of L-net, let us try to understand the additive structure.

Let us partition a 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). Observe that if a rule is not a singleton, it must be negative. An *additive pair* is a pair $(\xi, J)^-, (\xi, J')^-$ of negative actions on the same address, and belonging to the same bipole.

To each address in a bipole corresponds a formula occurrence. If we look at the bipole in the picture below, we have two rules: $R_1 = \{(\sigma 1, J)\}$ and $R_2 = \{(\sigma 2, J'), (\sigma 2, J'')\}$. The actions $(\sigma 2, J')$ and $(\sigma 2, J'')$ form an additive pair. We can think of σ as the address of a formula $A \otimes (B \& C)$, and think of $\sigma 1$ as A and $\sigma 2$ as $B \& C$.



It is immediate that we have a slice iff there are no additive pairs.

Let us consider a pre L-net. 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 *correction path* on a pre L-net is a path which uses at most one switching edge for each negative rule.

Definition 3.3 (L-net) *An* L-net *is a pre L-net $\mathfrak{D}$ such that*

- Acyclicity. *In a slice, no correction path is cyclic.*
- Additives. *Given two positive actions $k_1 = (\xi, K_1), k_2 = (\xi, K_2)$ on the same address, there is an additive pair $(\tau, I)^-, (\tau, J)^-$ such that $k_1 \overset{*}{\leftarrow} (\tau, I)^-$, and $k_2 \overset{*}{\leftarrow} (\tau, J)^-$.*

The additive condition is about the actions which are duplicated by effect of the additive structure. Figure 3 aims at suggesting the correspondence between what we call additives in our setting and the &-rule. Think of $(\sigma 0, I), (\sigma 0, J)$ as the two unary components of a &-rule: $\&_L$ and $\&_R$, decomposing $A \& B$ respectively into A and B. Now think of k_1, k_2 as two decompositions of occurrences of the same formula C belonging to the context of both A and B. In the graph, we have a sequential link from each occurrence of C to the additive component on which it depends.

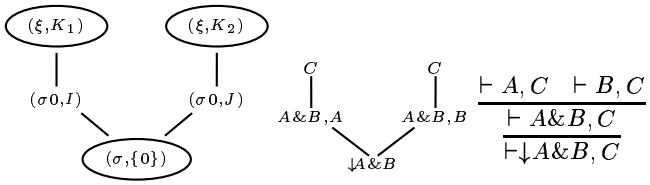


Figure 3.

L-nets as sets of chronicles. An L-net can be seen as a set of chronicles: the set of all the chronicles defined by its actions. This allows us to write $\mathfrak{c} \in \mathcal{S}$ and $\mathcal{S} \subseteq \mathfrak{D}$ for $\mathfrak{c}, \mathcal{S}, \mathfrak{D}$ respectively a chronicle, a slice and an L-net. In the following, we will largely rely on the presentation of L-nets as sets of chronicles. This will allow us to treat easily the superposition of slices as the union of the two sets of chronicles.

3.1 The intuitions behind

The d.a.g. A node 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. In particular, an edge imposes sequentiality. The idea underlying L-nets (as well as other approaches) is to not completely determine the order in which the actions should be performed, while still being able to express precedence constraints. For example, certain tasks may have to be performed before other tasks.

Consider a configuration such the one appearing on the l.h.s. of Figure 4. The actions α and β can be performed in parallel, or scheduled in any order, as long as they are performed before γ is performed. The action γ acts as a *point of synchronization* (this will be made precise when looking at the dynamics, that is normalization).

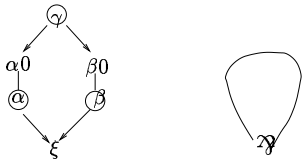


Figure 4.

Game-semantical intuition.
L-nets vs. designs and innocent strategies. Designs, as in [8], can be described as a special case of L-nets, those which

are trees, branching only on positive nodes: a negative action is followed by a single positive action.

An innocent strategy can be presented either as a set of plays, or as a set of views, as this is enough to describe all interactions. In [5] we have shown that designs are sorts of innocent strategies (on the universal arena), with the view presentation. The key correspondence is in fact chronicle = view.

Views/chronicles. In the same sense as L-nets generalize designs, they are a generalization of innocent strategies (we still need to investigate the extent of this). Here we think of a view/chronicle as a "desequentialization" of a view/chronicle in the usual sense of innocent strategies or Ludics.

Given a chronicle $\mathfrak{c}$, it is always possible to add sequentiality to make it a linear order. A total order which extends $\mathfrak{c}$ 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. The graph is Figure 4 is actually a chronicle (view), which could be sequentialized into a chronicle (view) in the original sense either as $\xi, \alpha, \alpha 0, \beta, \beta 0, \gamma$ or as $\xi, \beta, \beta 0, \alpha, \alpha 0, \gamma$.

Proof-theoretical intuition.

Addresses and actions. The notions of address and action have been introduced in [8]. In Ludics proofs do not manipulate formulas but their *addresses*. An address is a name, given as a sequence of natural numbers. If we give to an occurrence of formula address ξ, its immediate subformulas will receive address $\xi i, \xi j$ etc.

An *action* should be thought of as a cluster of operations which can be performed at the same time. This intuition has a precise proof-theoretical meaning in the calculus which underlies Ludics, i.e. second order multiplicative-additive Linear Logic. Multiplicative and additive connectives of LL separate into two families: positives $(\otimes, \oplus, 1, 0)$ and negatives $(\parr, \&, \bot, \top)$. A cluster of operations of the same polarity can be decomposed in a single step, and can be written as a single connective, which is called a *synthetic connective*. A formula is positive (negative) if its outer-most connective is positive (negative).

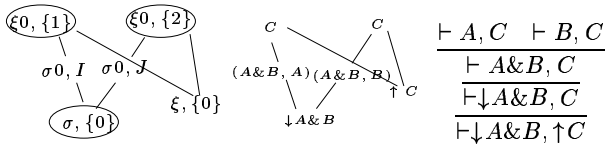
Think of an action as a tree of connectives with the same polarity, decomposed at once The accompanying set of indices make explicit the relative subaddress of the subformulas which are created in the decomposition.

Bipoles. Notice as in Figure 4 we have four bipoles. Our bipoles correspond (in an untyped setting) to Andreoli's notion of bipole. Think of a bipole as a step in the decomposition of a focusing proof.

Termination (†). The symbol † can be seen as a marking for termination. A nice interpretation, due to Curien, is to read it as an "*error*" case, (a "recoverable" error) which, if reached, makes the program quit.

Proof-nets. The role of sequential links is played in proof-nets by boxes and axioms (a box is a sort of axiom). Notice that sequential links are more flexible than boxes: they do not need to be nested. A close relative of sequential links in the theory of proof-nets are the so-called jumps. We think of a synchronization point as an axiom (possibly a generalized one, as those associated to boxes). If we gives types (formulas) to addresses, the example of Figure 4 could be written as the proof-net appearing on the r.h.s. of the picture.

The most interesting consequence of the use of sequential links rather than boxes is the additive structure. Let us consider again the example in Figure 3, which now we slightly expand adding one more decomposition. Again, we use a typed image to clarify the situation. Think of $\uparrow$ just as a generic negative connective. It helps to think of C as $C_1 \oplus C_2$. C is in the context of both premisses of $A\&B$, but is decomposed in two different ways. The parent formula $\uparrow C$ is shared from both slices. On $\uparrow C$ (on $(\xi, \{0\})$) we have what (with proof-nets terminology) is called an additive contraction.



We take more space to illustrate these ideas in [3].

4 Dynamics: composition (normalization)

In this section we define *normalization* of L-nets, that is to say composition. We proceed in two steps: we first define normalization on slices, and then normalization on general L-nets. Normalization on slices is as straightforward as normalization on MLL (multiplicative linear logic) proof-nets (as slices are sort of purely multiplicative proof-nets). This is the core of L-net normalization.

Normalization of L-nets is indeed reduced to slice normalization. As illustrated in Figure 6, we: (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 their chronicles. We will comment Figure 6 in Section 4.2.

We can compose two L-nets $\mathfrak{D}_1, \mathfrak{D}_2$ which have compatible interfaces, that is when a positive (outer) name σ in the interface of $\mathfrak{D}_1$ coincides with the negative (inner) name in the interface of $\mathfrak{D}_2$, as for example in $\Xi \vdash \Lambda, \sigma$ and $\sigma \vdash \Delta$. The shared name σ is called a *cut*.

A *cut-net* is a finite set $\mathfrak{R} = \{\mathfrak{D}_1, ...\mathfrak{D}_n\}$ of L-nets such that: *(i)* each address occurs at most in two interfaces, once as a positive name and once as a negative name; *(ii)* the

graph whose vertices are the interfaces and whose edges are the cuts is connected and acyclic.

We call an address *internal* if it is a sub-address of a cut, *visible* otherwise. The definition extends to actions. The interface of the cut-net is the interface induced by the visible addresses of the interfaces ($\Xi \vdash \Lambda, \Delta$ in our example). A cut-net whose interface is the empty sequent (there are no visible actions) is said a *closed* cut-net. Two "complementary" interfaces which put together produce an empty interface are called *opposite* ($\vdash \xi$ and $\xi \vdash$ are opposite interfaces).

A *slice of a cut-net* $\mathfrak{R}$ is a cut-net $\mathfrak{S} \subseteq \mathfrak{R}$, such that $\mathfrak{S}$ is a cut-net of slices. Given a cut-net $\mathfrak{R} = \{\mathfrak{D}_1, ...\mathfrak{D}_n\}$ we call slice of $\mathfrak{R}$ a set $\{\mathfrak{S}_1, ...\mathfrak{S}_n\}$ where $\mathfrak{S}_i$ is a slice of $\mathfrak{D}_i$.

Let $[\![\mathfrak{R}]\!]$ denote the **normal form** of $\mathfrak{R}$. We will have that $[\![\mathfrak{R}]\!] = \cup[\![\mathfrak{S}]\!]$, for all slices $\mathfrak{S} \subseteq \mathfrak{R}$.

4.1 Normalization of slices

In this section we exploit Fact 3.2. Normalization on slices follows the standard paradigm of parallel composition plus hiding of internal communication. There are several possible ways to present it:

– by rewriting rules in the style of MLL proof-nets;

– by means of an abstract machine;

– as merging of orders (keeping in mind Fact 3.2).

Here we are going to present normalization by means of an abstract machine (which is an adaptation of our LAM machine [4]). The interaction among the L-nets forming a cut-net is described by a wave (we think of this as a multi-token) traveling on the cut-net. The wave moves upwards on visible actions, while on internal actions it moves from the positive action k^+ to the corresponding negative action k^-. The order in which the nodes are reached during the interaction establishes a new partial order, the one underlying the normal form.

Our procedure is reminiscent of event structures, or Petri nets. Let us denote by $Prec(k)$ the set $\{k', k' \leftarrow k\}$ of nodes which immediately precede k (the preconditions of k). The key point is that *we can reach an action only k if we have reached all action in $Prec(k)$*: one action is enabled (accessed) by the enabling of all actions in $Prec(k)$. Because of the structure of L-nets, moving from positive to negative is "asynchronous", while reaching a positive action k needs a synchronization among all the negatives nodes in $Prec(k)$.

Let $\mathfrak{R}$ be a cut-net of slices. $\widetilde{\mathfrak{R}}$ is the partial order associated to the graph (V,E) on the (non polarized) actions of $\mathfrak{R}$ obtained as follows:

- If k is a visible action of $\mathfrak{R}$ (positive or negative) and $Prec(k) \subseteq V$ then $k \in V$ and $k' \leftarrow k \in E$, for all $k' \in Prec(k)$. Observe that if k is initial, then $k \in V$.
- If k^+ is an internal action of $\mathfrak{R}$, $Prec(k) \subseteq V$ and $k^- \in \mathfrak{R}$, then $k \in V$, and $k' \leftarrow k \in E$, for all $k' \in Prec(k^+)$.

The graph (V,E) describes a d.a.g.. If $\mathfrak{R}$ is closed, we call the partial order $\widetilde{\mathfrak{R}}$ a *play*. If $\mathfrak{R} = \{\mathfrak{S}, \mathfrak{T}\}$, we indicate $\widetilde{\mathfrak{R}}$ with $[\mathfrak{S} \rightleftharpoons \mathfrak{T}]$.

The *normal form* $[\mathfrak{R}]$ *of* $\mathfrak{R}$ is obtained by hiding (i) the internal actions and (ii) any negative action which is maximal (the actions in (i) correspond to internal communication, while the actions in (ii) are garbage, left from failed communication).

Proposition 4.1 $[\mathfrak{R}]$ *is an L-net.*

The only delicate condition to check is the acyclicity of the correction paths. To do this we reformulate normalization as graph rewriting, in the style of proof-nets normalization.The argument is then rather standard: if there is a cycle after a rewriting step, we find a cycle in the graph before performing that step. Figure 5 sketches the rewriting technique.

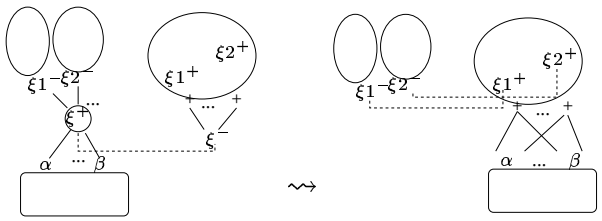


Figure 5. Proof-nets style rewriting

Normalization as merging. A slice $\mathfrak{S}$ of a cut-net is *balanced* when any internal action k appears in $\mathfrak{S}$ also with opposite polarity. We can reformulate the process of normalization we described as

Proposition 4.2 *Let $\mathfrak{R}$ be balanced. We merge the partial orders corresponding to each slice by identifying each pair of internal actions of opposite polarity. The induced relation on the actions of $\mathfrak{R}$ is a partial order, which we indicate with $<<_{\mathfrak{R}}$.$[\mathfrak{R}]$ consists of the visible actions of $\mathfrak{R}$, with the order induced by $<<_{\mathfrak{R}}$.*

Properties of normalization.

Lemma 4.3 *Let $\mathfrak{S}_0 \subseteq \mathfrak{S}$ be slices. $[\mathfrak{S}_0] \subseteq [\mathfrak{S}]$.*

Lemma 4.4 *Let $\mathfrak{S}$ be a slice of a cut-net and $[\mathfrak{S}] = \mathfrak{A}$. Then (i) $\mathfrak{A}$ is a slice, and (ii) there exists a balanced slice $\mathfrak{S}' \subseteq \mathfrak{S}$, s.t. $[\mathfrak{S}'] = \mathfrak{A}$.*

The following proposition is easily obtained using confluence of the proof-net style graph rewriting.

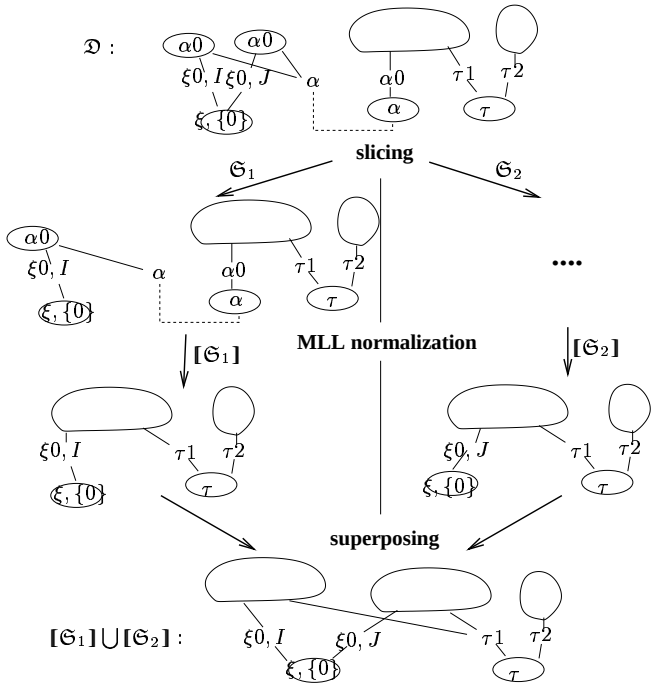


Figure 6. L-nets normalization

Proposition 4.5 (Associativity) *Normalization of slices is associative. Let $\mathcal{A}, \mathcal{B}, \mathcal{C}$ be slices.* $[[\mathcal{A}, \mathcal{B}], \mathcal{C}] = [\mathcal{A}, \mathcal{B}, \mathcal{C}] = [\mathcal{A}, [\mathcal{B}, \mathcal{C}]]$

4.2 Normalization of L-nets

Definition 4.6 (Normalization) *Let $\mathfrak{R}$ be a cut-net. We define $[\![\mathfrak{R}]\!] = \bigcup[\mathfrak{S}]$, for all slices $\mathfrak{S} \subseteq \mathfrak{R}$.*

As $[\![\mathfrak{S}]\!] = [\mathfrak{S}]$, from now on we use only the notation $[\![\ ]\!]$. The key result is the following one, which allow us to reduce all properties of $[\![\ ]\!]$ to properties on slices.

Proposition 4.7 *Let $\mathfrak{A} \subseteq [\![\mathfrak{R}]\!]$ s.t. $\mathfrak{A}$ is a slice. There exists a slice $\mathfrak{S} \subseteq \mathfrak{R}$ s.t. $[\![\mathfrak{S}]\!] = \mathfrak{A}$.*

By exploiting Proposition 4.7 we obtain

Proposition 4.8 *If $\mathfrak{R}$ is a cut net, $[\![\mathfrak{R}]\!]$ is an L-net.*

Proposition 4.9 *Normalization is associative: let $\mathfrak{A}, \mathfrak{B}, \mathfrak{C}$ be L-nets. $[\![[\![\mathfrak{A}, \mathfrak{B}]\!], \mathfrak{C}]\!] = [\![\mathfrak{A}, \mathfrak{B}, \mathfrak{C}]\!] = [\![\mathfrak{A}, [\![\mathfrak{B}, \mathfrak{C}]\!]]\!]$*

Proof of 4.7 (sketch). Let $\mathfrak{A} = [\![\mathfrak{S}_1]\!] \cup [\![\mathfrak{S}_2]\!]$, for $\mathfrak{S}_1$ and $\mathfrak{S}_2$ balanced. If $\mathfrak{S}_1 \cup \mathfrak{S}_2$ is not a slice of $\mathfrak{R}$, it contains at least one additive pair, and any such a pair must be on cut addresses (otherwise it would be in $\mathfrak{A}$). We build a contradiction by using the fact that if we take a pair $(\xi, I_1)^-, (\xi, I_2)^-$, the opposite actions $(\xi, I_i)^+$ belong to $\mathfrak{S}_1 \cup \mathfrak{S}_2$. Hence by the additive condition on L-nets (remember that $\mathfrak{S}_1 \cup \mathfrak{S}_2 \subseteq \mathfrak{R}$), there exists an additive pair w_1, w_2, such that $(\xi, I_1)^+ > w_1, (\xi, I_2)^+ > w_2$.

Example. In Figure 6 we give an example of L-net normalization. The L-net on the l.h.s. of the cut has the same form of that in Section 3.1. The actions $(\xi 0, I), (\xi 0, J)$ form an additive pair. Think of them as the two components of a $\&$-rule, and of α as an additive contraction. We separate the two slices, and normalize them. Normalization is just MLL proof-net rewriting. Finally we superimpose the slices. We find that only the minimum necessary is duplicated: the parts which are above $(\xi 0, I)$ and $(\xi 0, J)$); the right-most part of the L-net is shared. Observe that now the additive contraction is on $\tau 1$. The behaviour of additive normalization seems to us extremely interesting, as it maximizes (in a sense to be better understood) the sharing.

5 Observational equivalence

The question of *when two terms are the same from the point of view of the observer* (or of the environment) is a fundamental question in the study of calculi for either sequential or concurrent computation.

The approach to such a question (and the stress on it) is one feature distinguishing Ludics from Game Semantics. Ludics comes equipped with a built-in notion of observational equivalence: if we cannot distinguish two agents by the way they interact with the other agents, they must actually be (syntactically) equal. Hence there is no need to impose an equivalence relation. This property, called *separation*, is a fundamental requirement for Ludics as an interactive theory, but actually has a long-standing tradition, going back to Böhm theorem for the λ-calculus. We can see separation as a game-semantical analogue of the Böhm theorem. Separation shows in some sense that there is no redundancy in the syntax (or in the model), and gives a feedback of the quality of the syntax w.r.t. the operational semantics.

In this section we will study a separation property for L-nets.

Orthogonality. We define a notion of orthogonality. The idea is to evaluate an L-net inside a closed cut-net. This idea is similar to that of contextual observational semantics, where we evaluate a piece of program M in C[M], where C ranges over full program contexts.

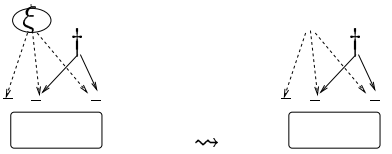
The normal form of a closed cut-net is either the empty graph, or a set of nodes all labeled by †. We call $\mathfrak{Dai}$ the L-net which consists of the single action †. Given two L-nets $\mathfrak{D}, \mathfrak{E}$ on opposite interface, they are *orthogonal*, written $\mathfrak{D} \perp \mathfrak{E}$, if $\mathfrak{Dai} \subseteq [\![\mathfrak{D}, \mathfrak{E}]\!]$.

For any slice $\mathfrak{S}$, we can define its "canonical opponent" $Opp(\mathfrak{S})$, that is an L-net which is orthogonal to $\mathfrak{S}$ and uses all of its actions. Let $\mathfrak{S}$ be a slice. We obtain $Opp(\mathfrak{S})$ in two steps. (i) (V,E) is the graph where V is given by the actions of $\mathfrak{S}$ which are different from †, taken with opposite polarity, and E is the $\leftarrow$ structure induced by the parent

order. (ii) For any negative action k which is maximal w.r.t. $\leftarrow$, we add an action $\dagger$ and the edge $k \leftarrow \dagger$. Observe that $Opp(\mathfrak{S})$ it is just a prefix tree (i.e. the address counter-part of a formula tree), completed with actions $\dagger$

Proposition 5.1 *(i) For any slice S, $Opp(\mathfrak{S})$ is a slice (on the opposite interface). Given a chronicle $\mathfrak{c}$, $Opp(\mathfrak{c})$ has at most one occurrence of $\dagger$. (ii) $\mathfrak{S} \perp Opp(\mathfrak{S})$.*

Garbage collection. Let us consider a configuration as the one on the left-hand side of the picture below. When



normalizing, before reaching ξ, we have already reached $\dagger$. If we are ultimately interested only in orthogonality, we cannot do better than reaching a $\dagger$; the presence of ξ does not add any contribution to this. Let us define a rewriting rule that we will call "garbage collection".

The **GC rewriting rule** is: if $\ulcorner a \urcorner \supseteq Prec(\dagger)$ for an occurrence of $\dagger$, then a is erased as well as any k, s.t. $a \overset{*}{\leftarrow} k$.

The rewriting system defined by the GC rule is terminating and confluent. We denote by $GC(\mathfrak{D})$ the L-net obtained by repeated application of this rule.

It is immediate that garbage collection respects the following equations: $GC(\mathfrak{D}) \subseteq \mathfrak{D}$; $GC(GC(\mathfrak{D})) = GC(\mathfrak{D})$; $\mathfrak{D}_1 \subseteq \mathfrak{D}_2 \not\Rightarrow GC(\mathfrak{D}_1) \subseteq GC(\mathfrak{D}_2)$. The third point says that *GC is not monotonous* w.r.t. $\subseteq$, which is easy to understand as the larger is the net, the more we can have of $\dagger$ inducing the GC.

Example: $\mathfrak{D}_1 = \overset{\alpha}{\bigcirc}$, $\mathfrak{D}_2 = \overset{\alpha}{\bigcirc}\ \dagger$, $\mathfrak{D}_1 \subseteq \mathfrak{D}_2, GC(\mathfrak{D}_1) = \mathfrak{D}_1$ while $GC(\mathfrak{D}_2) = \dagger$.

Lemma 5.2 *$\mathfrak{E} \perp \mathfrak{D}$ iff $\mathfrak{E} \perp GC(\mathfrak{D})$.*

Definition 5.3 (Pure L-nets) *An L-net $\mathfrak{D}$ is pure if $GC(\mathfrak{D}) = \mathfrak{D}$.*

Separation.

Proposition 5.4 (Separation) *Let $\mathfrak{D}, \mathfrak{D}'$ be pure L-nets. There exists an L-net $\mathfrak{E}$ which is orthogonal to one and not to the other.*

Proof of 5.4 (sketch). Define the *size* of a chronicle as the number of its actions which are not $\dagger$. Let $\mathfrak{c}$ be a minimal chronicle which belongs to one of the two L-nets $\mathfrak{D}, \mathfrak{D}'$ and no to the other. Assume $\mathfrak{c} \in \mathfrak{D}$. We know that $[\![\mathfrak{D}, Opp(\mathfrak{c})]\!] \supseteq \mathfrak{Dai}$, and we want to show that $\dagger \notin [\mathfrak{T} \rightleftharpoons Opp(\mathfrak{c})]$ for any slice $\mathfrak{T} \subseteq \mathfrak{D}'$. We prove that composing $\mathfrak{T}$ with $Opp(\mathfrak{c})$, (i) we cannot trigger "by mistake" an occurrence of $\dagger$ which was present in $\mathfrak{T}$, and (ii) if $\dagger \in Opp(\mathfrak{c})$, we cannot reach it using less actions than those in $\mathfrak{c}$.

Observational equivalence. Let $\mathfrak{A}, \mathfrak{B}$ be L-nets. $\mathfrak{A} \sim \mathfrak{B}$ iff $\mathfrak{A}^\perp = \mathfrak{B}^\perp$. We have that $\mathfrak{A} \sim GC(\mathfrak{A})$, $\mathfrak{A} \sim \mathfrak{B} \Rightarrow GC(\mathfrak{A}) = GC(\mathfrak{B})$.

6 Ludics on Graphs

Ludics has been introduced in [8] as a modular construction.

- At the low level there is the definition of the *untyped computational structures* (called designs) and their dynamics (*normalization*). Normalization allows for the definition of *orthogonality*.
- The computational objects satisfy certain remarkable properties, called *analytical theorems*, in particular *associativity* of normalization, and *separation*. Analytical theorems are the *interface* with the rest of the construction.
- At the high-level there is the interactive definition of *types* (set of proof/programs equal to their biorthogonal). The analytical theorems make types satisfy *internal completeness*: any design in a composed type can be decomposed into a design in each of the component of the type (for example any $\mathfrak{D} \in \mathbf{A} \otimes \mathbf{B}$ can be decomposed into $\mathfrak{D}_1 \in \mathbf{A}$ and $\mathfrak{D}_2 \in \mathbf{B}$).

The high-level architecture of Ludics is independent from the low-level structures, as long as they satisfy the analytical theorems.

What we do in this Section is to change the low level "module" of Ludics (computational structures and dynamics) with a more asynchronous one. The structures we use satisfy the analytical theorems, which allows us to preserve the high-level architecture of Ludics (types) with its remarkable results, in particular internal completeness.

Remark 6.1 *We actually build the high-level structures of Ludics on a weaker form of the analytical theorems (we show that stability is not needed as long as we have a weaker property). For this reason, we actually have to reprove some of the results in [8]. However, our weaker proofs apply to the original version of Ludics.*

6.1 The low-level architecture

We can build GC into normalization; this way we are able to deal only with pure L-nets ($GC(\mathfrak{D}) = \mathfrak{D}$).

Definition 6.2 (GC-normalization) $[\![\mathfrak{R}]\!]^\bullet = GC[\![\mathfrak{R}]\!]$.

If $\mathfrak{D}, \mathfrak{E}$ are L-nets, we have that $GC[\![GC(\mathfrak{D}), \mathfrak{E}]\!] = GC[\![\mathfrak{D}, \mathfrak{E}]\!]$. From this we obtain:

Proposition 6.3 (Associativity) *Let $\mathfrak{A}, \mathfrak{B}, \mathfrak{C}$ be pure nets.* $[\![[\![\mathfrak{A}, \mathfrak{B}]\!]^\bullet, \mathfrak{C}]\!]^\bullet = [\![\mathfrak{A}, \mathfrak{B}, \mathfrak{C}]\!]^\bullet = [\![\mathfrak{A}, [\![\mathfrak{B}, \mathfrak{C}]\!]^\bullet]\!]^\bullet$

Because of GC, GC-Normalization is not monotonous w.r.t. inclusion. Moreover, it is not stable (at least when † appears in the L-nets.), where by *stability* we mean the following property: let $\mathfrak{D}_1, \mathfrak{D}_2 \subseteq \mathfrak{D}$ then $[\![\mathfrak{F}, \mathfrak{D}_1 \cap \mathfrak{D}_2]\!] = [\![\mathfrak{D}_1, \mathfrak{F}]\!] \cap [\![\mathfrak{D}_2, \mathfrak{F}]\!]$. However, $[\![\ ]\!]^\bullet$ is monotonous w.r.t. $\subseteq$ in the closed case, the one on which the whole high-level architecture of Ludics is build. Even though [8] uses stability to define incarnation and establishing internal completeness, the weaker property stated in Lemma 6.4 is actually just enough.

Lemma 6.4 *If $\mathfrak{R}_0 \subseteq \mathfrak{R}$ are closed cut-nets, then $[\![\mathfrak{R}_0]\!]^\bullet \subseteq [\![\mathfrak{R}]\!]^\bullet$. As a consequence, if $\mathfrak{D}_0 \subseteq \mathfrak{D}$ and $\mathfrak{E} \perp \mathfrak{D}_0$, then $\mathfrak{E} \perp \mathfrak{D}$ ($\perp$ w.r.t. $[\![\ ]\!]^\bullet$).*

Let us summarize the situation with a table:

	$[\![\ ]\!]$	$[\![\ ]\!]^\bullet$
Stability w.r.t. $\subseteq$	NO	NO
Monotonicity w.r.t. $\subseteq$	$\surd$	NO
Monotonicity w.r.t. $\subseteq$ (closed)	$\surd$	$\surd$
Separation	NO	$\surd$
Associativity	$\surd$	$\surd$

Associativity, separation and the property in Lemma 6.4 which enjoys $[\![\ ]\!]^\bullet$ is all we need for rebuild the high-level architecture of Ludics.

6.2 The high-level architecture

From now on, we consider only pure nets and GC-normalization. Orthogonality is always meant w.r.t. $[\![\ ]\!]^\bullet$. Given a set S of L-nets, its orthogonal $S^\perp$ is the set of all L-nets $\mathfrak{E}$, such that $\mathfrak{E} \perp \mathfrak{D}$, for all $\mathfrak{D} \in S$.

A **behaviour (or interactive type)** is a set $\mathbf{G}$ of L-nets of a given interface, which is equal to its biorthogonal. A behaviour is positive or negative, according to its interface.

A typical interactive construction of Ludics is incarnation. If an L-net $\mathfrak{D} \in \mathbf{G}$ and $\mathfrak{D} \subseteq \mathfrak{E}$, then $\mathfrak{E} \in \mathbf{G}$ but none of the new actions in $\mathfrak{E}$ is really used. W.r.t.any interaction, $\mathfrak{D}$ and $\mathfrak{E}$ are equivalent in $\mathbf{G}$, and $\mathbf{G}$ is naturally equipped with an equivalence relation. We can distinguish one L-net $|\mathfrak{D}|_{\mathbf{G}}$ in each class, so that $\mathfrak{D} \cong \mathfrak{E}$ iff $|\mathfrak{D}|_{\mathbf{G}} = |\mathfrak{E}|_{\mathbf{G}}$.

Incarnation. Given an L-net $\mathfrak{D}$ in a behaviour $\mathbf{G}$, its *incarnation* $|\mathfrak{D}|_{\mathbf{G}}$ is the part of $\mathfrak{D}$ that can be interactively reached via normalization with L-nets in $\mathbf{G}^\perp$. Incarnation can be defined as the union of all the minimal $\mathfrak{D}_i$ s.t. $\mathfrak{D}_i \subseteq \mathfrak{D}$ and $\mathfrak{D}_i \in \mathbf{G}$. By Lemma 6.4, $|\mathfrak{D}|_{\mathbf{G}}$ belongs to $\mathbf{G}$.

Remark 6.5 *In [8] stability implies that there exists a smallest design, which is not true in our more general setting.*

An L-net $\mathfrak{D}$ is called incarnated or *material* when $\mathfrak{D} = |\mathfrak{D}|$. The incarnation $|\mathbf{G}|$ of $\mathbf{G}$ is defined as the set of its material L-nets. The property we need for the internal completeness is the following:

Proposition 6.6 $|\mathbf{G}|^{\perp\perp} = \mathbf{G}$.

Proof. One inclusion is obvious as $|\mathbf{G}| \subseteq \mathbf{G}$. We show that $|\mathbf{G}|^\perp \subseteq \mathbf{G}^\perp$. Take $\mathfrak{F} \in |\mathbf{G}|^\perp$. For any $\mathfrak{D} \in \mathbf{G}$, $\mathfrak{F} \perp |\mathfrak{D}|_{\mathbf{G}}$ and since $|\mathfrak{D}|_{\mathbf{G}} \subseteq \mathfrak{D}$, $\mathfrak{F} \perp \mathfrak{D}$ by Lemma 6.4.

6.3 Internal completeness

Let us sketch the constructions on behaviours (types) which allow us to obtain new behaviours (compounded types). The main property of these constructions is *internal completeness*: the set S of L-nets produced by the construction is equal to its biorthogonal ($S = S^{\perp\perp}$). Since the biorthogonal does not introduce new objects, we have a complete description of all L-nets in the behaviour. Typically, if $\mathbf{A}, \mathbf{B}$ are disjoint behaviours, we define

- $\mathbf{A} \oplus \mathbf{B} \equiv \mathbf{A} \cup \mathbf{B}$, which turns out equal to $(\mathbf{A} \cup \mathbf{B})^{\perp\perp}$.
- $\mathbf{A} \otimes \mathbf{B} \equiv \{\mathfrak{A} \otimes \mathfrak{B}, \mathfrak{A} \in \mathbf{A}, \mathfrak{B} \in \mathbf{B}\}$, which turns out equal to $\{\mathfrak{A} \otimes \mathfrak{B}, \mathfrak{A} \in \mathbf{A}, \mathfrak{B} \in \mathbf{B}\}^{\perp\perp}$.

Because of internal completeness, if $\mathfrak{D} \in \mathbf{A} \otimes \mathbf{B}$ we know we can decompose it as $\mathfrak{D}_1 \otimes \mathfrak{D}_2$, with $\mathfrak{D}_1 \in \mathbf{A}$ and $\mathfrak{D}_2 \in \mathbf{B}$. Any behaviour formed by using the connectives can be decomposed in its initial components. Again, [8] uses stability to prove this result, but it is rather straightforward to show that the same result can actually be proved only using monotonicity w.r.t. inclusion, and we only need it in the closed case (the property stated in Lemma 6.4).

7 Discussion and further work.

Towards concurrency. Restricted to slices, our setting provides as much parallelism as MLL proof-nets. What do we gain then? On one side, additive superposition. On the other side, we recover MLL proof-nets, but with all novelties introduced by Ludics: names (addresses) rather than formulas, and a calculus built on the equivalence relation (rather than the opposite). It will now be important to relate L-nets both to process calculus terms and to true concurrency models.

Innocent strategies. This paper is also a proposal for strategies in game semantical terms, namely for strategies in which sequentiality is relaxed. Our strategies are presented as sets of views rather than sets of plays; it is well

known that both presentations are possible. We expect to be able to further develop this line, to define games. We are interested in better understanding if and in which sense L-nets are (de-sequentialized) innocent strategies. It will be important to relate our work to the work by Mellies [12].

Acyclicity. In a preliminary version (short presentation at LICS 04) we had a stronger acyclicity condition, forbidding cycles *across* slices. Such a condition insures sequentiability but kills interesting designs. The acyclicity condition here is minimalist.

Our choice is to accept all computational objects which behave well w.r.t. composition, even though they are "intrinsically" parallel. They are not "logical," but why should we restrict our interest only to sequentializable objects?

Relating trees and graphs. How tree and graph strategies should relate? An L-net is a set of partially ordered views (p.o. views); each partial order specifies some constraints. We think of its sequentialization as a tree-strategy where each view is a linear extension of a p.o. view.

As said above, here we do not force sequentiability. Taking inspiration from the Gustave function, it is easy to produce additive L-nets which are "intrinsically parallel" (all chronicle can be individually sequentialized, but they are not "coherent" all together). In recent work with Pierre-Louis Curien [3], we study a finer acyclicity condition. Such finer condition guarantees that L-nets are sequentializable, while allowing an interesting proof-net behaviour. We are then able to apply proof-net techniques to strategies, and establish a correspondence between our graph and tree strategies similar to that relating proof-nets and sequent calculus.

Syntax. We are interested in developing the proof-nets syntax underlying L-nets, and relating it with the existing ones. [3] takes us one step further in this direction.

We postpone the study of results on completeness until the development of such a syntax. The closer the syntax is to the semantics, the more natural is completeness...

L-nets provide a setting in which several degrees of sequentiality can live; sequentiality is regulated by the sequential links. With minimal sequentiality, we have just prefix trees (which we have seen to play a key role in testing observational equivalence). To recover (an abstract counterpart of) MALL proof-nets, we need just enough sequentiality to recover axioms and additive superposition. At the other extreme, all sequentiality can be made explicit, and we have designs "à la locus solum." All this needs further investigation.

Our objective would be to have the same possibility of graduating sequentiality inside the concrete (typed) proof-nets world. On the fully sequentialized side, designs correspond both to sequent calculus derivations and to Laurent's polarized proof-nets for MALL$\downarrow\uparrow$. On the liberal side we expect to have a syntax for (focusing?) MALL proof-nets.

Separation. In a parallel setting, separation is a challenging issue. The solution for separation we present here is not yet totally satisfactory. Our approach reduces any possibility for distinguishing among graphs to orthogonality, and we have a rather radical notion of equivalence. A finer way to separate, would be to separate on the normal form. We believe it is possible to refine separation by refining the notion of termination. Maybe providing errors with labels.

Acknowledgments. The structure of partial orders underlying our work has been strongly inspired by a reformulation of Games as an abstract category of partial orders and merging put forward by Martin Hyland in a talk at the meeting *Rencontres Franco-Américaines de Mathématiques (AMS-SMF)* in July 2001. Martin Hyland suggested that the LAM machine ([4]) could provide a concrete way to realize the merging.

Many thanks to Pierre-Louis Curien for his precious remarks.

References

[1] S. Abramsky and P.-A. Mellies. Concurrent games and full completeness. In *Proceedings LICS 99*. IEEE Computer Society Press, 1999.

[2] J.-M. Andreoli. Focussing proof-net construction as a middleware paradigm. In *Proceedings of Conference on Automated Deduction (CADE)*, 2002.

[3] P.-L. Curien and C. Faggian. L-nets, strategies and proof-nets. Submitted, available at www.math.unipd.it/claudia.

[4] C. Faggian. Travelling on designs: ludics dynamics. In *CSL 02*, volume 2471 of *LNCS*. Springer Verlag, 2002.

[5] C. Faggian and M. Hyland. Designs, disputes and strategies. In *CSL 02*, volume 2471 of *LNCS*. Springer Verlag, 2002.

[6] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[7] J.-Y. Girard. Proof-nets: the parallel syntax for proof-theory. In Ursini and Agliano, editors, *Logic and Algebra*. Marcel Dekker, New York, 1996.

[8] J.-Y. Girard. Locus solum. *Mathematical Structures in Computer Science*, 11:301–506, 2001.

[9] R. Harmer and G. McCusker. A fully abstract game semantics for finite nondeterminism. In *Proceedings LICS 99*. IEEE Computer Society Press, 1999.

[10] M. Hyland and A. Schalk. Games on graphs and sequentially realizable functionals. In *Proceedings LICS 02*, pages 257–264. IEEE Computer Society Press, 2002.

[11] O. Laurent and L. Tortora-de Falco. Slicing polarized additive normalization. In T. Ehrhard, J.-Y. Girard, P. Ruet, and P. Scott, editors, *Linear Logic in Computer Science*, volume 316 of *LMSLNS*. Cambridge University Press, Nov. 2004.

[12] P.-A. Mellies. Asynchronous games 2 : The true concurrency of innocence. In *CONCUR 04*, volume 3170 of *LNCS*. Springer Verlag, 2004.

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