Travelling on designs

Ludics dynamics

Claudia Faggian

DPMMS – University of Cambridge
United Kingdom
C.Faggian@dpmms.cam.ac.uk

Abstract. Proofs in Ludics are represented by designs. Designs (desseins) can be seen as an intermediate syntax between sequent calculus and proof nets, carrying advantages from both approaches, especially w.r.t. cut-elimination. To study interaction between designs and develop a geometrical intuition, we introduce an abstract machine which presents normalization as a token travelling along a net of designs. This allows a concrete approach, from which to carry on the study of issues such as: (i) which part of a design can be recognized interactively; (ii) how to reconstruct a design from the traces of its interactions in different tests.

Ludics is a new theory recently introduced by Girard in [6]. The program is to overcome the distinction between syntax and semantics: proofs are interpreted via proofs, and all properties are expressed and tested internally. Internally means interactively: the objects themselves test each other. The fundamental artifacts of Ludics are *designs,* which are both (i) an abstraction of formal proofs and (ii) a concretion of their semantical interpretation.

Designs have remarkable properties also as a syntax. They may be seen as an intermediate syntax between sequent calculus and proof-nets. Such a syntax carries advantages from both approaches, in particular w.r.t. cut-elimination. Designs: (i) Offer a concise syntax. (ii) Integrate a good treatment of the additives in a syntax that is still light to manipulate (this is a strong point of Ludics with respect to proof-nets and geometry of interaction). (iii) Are close to implementation, in that they make explicit the "addresses" and use tools typical of implementations, such as a dynamical approach to the context.

To have a concrete approach to designs and develop a geometrical intuition, we introduce an abstract machine, called *Loci Abstract Machine (LAM)*, which allows us to present normalization by a token travelling along a net of designs.

The LAM is the starting point from which we developed several tools for the operational study of designs. The path drawn by the token is a sequence of actions that represents the trace of the interaction between the designs. Conversely, we provide tools for reconstructing the agents from the traces of their interactions. A key operation we use exactly corresponds to a well-know operation of Games Semantics, the computation of the *view* ([7], [8]).

Note 1. By *design* we always intend the tree structure that in [6] is called *dessein.* If we refer to its sequent calculus presentation (i.e. *dessin*) we make it explicit.

1 Ludics in a nutshell

The program of Ludics is to overcome the distinction between *syntax* (the formalism) and *semantics* (its interpretation): proofs are interpreted via proofs. Syntax and semantics meet in the notion of *design*. Designs are both an abstraction of a formal proof, and a concretion of its semantic interpretation. This has been achieved working from two directions.

1. *Making semantics concrete.* This leads to enlarging the universe of proofs, in order to have enough inhabitants to be able to distinguish between them *inside* the system. *Paraproofs* are introduced.

2. *Abstracting from syntax.* The syntax of designs captures the geometrical structure underlying a sequent calculus proof. There are two crucial notions used to obtain this: *focalization* and *locations*. Focalization, which is an essential tool of proof-search ([1]), allows the definition of *synthetic connectives*. Locations are a major novelty of Ludics: proofs do not manipulate formulas, but their *addresses*. These are sequences of natural numbers, which can be thought of as the address in the memory where the formula is stored.

Para-proofs. Ludics provides a setting in which to any proof of A we can oppose (via cut-elimination) a proof of $A^{\perp}$. To this aim, it generalizes the notion of proof (para-proof).

A proof should be thought of in the sense of "proof search" or "proof construction": we start from the conclusion, and guess a last rule, then the rule above. What if we cannot apply any rule? A new rule is introduced, called daimon: $\dfrac{}{\vdash \Gamma}\ \dagger$. It allows us to assume any conclusion, without providing a justification.

Slices. To understand designs, it is useful to have in mind the notion of slice. A &-rule can be seen as the super-imposition of two unary rules: $(a\&b, a)$ and $(a\&b, b)$. Given a 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*. For example, the derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{\cdots}{\vdash a, c}\ a \quad \dfrac{\cdots}{\vdash b, c}\ b}{\vdash a\&b, c}\ (a\&b, \{a\}), (a\&b, \{b\})}{\vdash (a\&b) \oplus d, c}\ ((a\&b) \oplus d, \{a\&b\})}$$

can be decomposed into two slices:

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

The &-rule is a set (the super-imposition) of two unary rules on the same formula. It is important to observe that *normalization is always carried out in a single slice:* selecting one of the premises of a &-rule is exactly what happens during normalization.

Synthetic connectives. The calculus underlying ludics is 2nd order multiplicative-additive Linear Logic ($\mathbf{M}ALL^2$). 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).

In the formula $f = ((p_1 \parr p_2) \oplus q^\bot) \otimes r^\bot$ we have a positive ternary connective $(-\oplus-)\otimes-$. The immediate subformulas of f are $p_1 \parr p_2, q^\bot, r^\bot$ (negative). To introduce this ternary connective there are two possible rules, obtained combining a Tensor-rule with one of the two possible Plus-rules:

$$\frac{\vdash p^\bot, \Gamma \quad \vdash r^\bot, \Delta}{\vdash (p^\bot \oplus q^\bot) \otimes r^\bot, \Gamma, \Delta}\ (f, \{p^\bot, r^\bot\}) \qquad \text{or} \qquad \frac{\vdash q^\bot, \Gamma \quad \vdash r^\bot, \Delta}{\vdash (p^\bot \oplus q^\bot) \otimes r^\bot, \Gamma, \Delta}\ (f, \{q^\bot, r^\bot\})$$

Observe that *each rule* is labelled by a pair: (i) the focus and (ii) the subformulas which appear in the premises.

The dual formula $(p\&q)\parr r$ has a negative connective whose rule combines *the* Par-rule with *the* With-rule:

$$\frac{\vdash p, r, \Lambda \quad \vdash q, r, \Lambda}{\vdash (p\&q)\parr r, \Lambda}\ \{(f^\bot, \{p, q\}), (f^\bot, \{p, r\})\}$$

The rule is labelled by a set of pair: a pair (focus, set of subformulas) for each premise. This makes sense if we understand that each negative premise corresponds to an additive slice. Actually, we rather use the label $(f^\bot, \{\{p, r\}, \{q, r\}\})$ which is short for the one above.

To each positive rule corresponds a premise of the negative rule. During cut-elimination, the positive rule will select a negative premise. That is to say, the positive rule will select one slice. For example, the redex:

$$\dfrac{\dfrac{\vdash p^\bot, \Gamma \quad \vdash r^\bot, \Delta}{\vdash (p^\bot \oplus q^\bot) \otimes r^\bot, \Gamma, \Delta}\ (f, \{p^\bot, r^\bot\}) \quad \dfrac{\vdash p, r, \Lambda \quad \vdash q, r, \Lambda}{\vdash (p\&q)\parr r, \Lambda}\ \{(f^\bot, \{p, r\}), (f^\bot, \{q, r\})\}}{\vdash \Gamma, \Delta, \Lambda}$$

reduces to:

$$\frac{\vdash p^\bot, \Gamma \quad \vdash r^\bot, \Delta \quad \vdash p, r, \Lambda}{\vdash \Gamma, \Delta, \Lambda}$$

Note 2. We write $((p_1 \parr p_2) \oplus q^\bot) \otimes r^\bot$ for $(\downarrow (\uparrow p_1 \parr \uparrow p_2) \oplus \downarrow q^\bot) \otimes \downarrow r^\bot$. A positive rule can only be applied on positive formulas. Therefore we cannot directly form $((p_1 \parr p_2) \oplus q^\bot) \otimes r^\bot$; we need to use an operator which changes the polarity, the *Shift:* $\downarrow$. If N is negative, $\downarrow N$ is positive. However, we are going to deal with $\downarrow$ implicitly.

Locations. Each formula to be decomposed receives an address. Let f of the previous example have address ξ, and p, q, r be respectively located in $\xi 1, \xi 2, \xi 3$. The positive rules in the previous example can be rewritten as

Positive rules

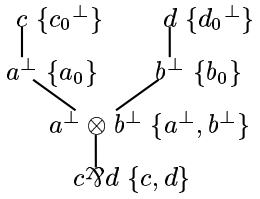
$$\frac{\xi 1 \vdash \Gamma \quad \xi 2 \vdash \Delta}{\vdash \xi, \Gamma, \Delta}\ (\xi, \{1, 3\}) \qquad \frac{\xi 2 \vdash \Gamma \quad \xi 3 \vdash \Delta}{\vdash \xi, \Gamma, \Delta}\ (\xi, \{2, 3\})$$

Sequents of addresses are expressions of the form $\Xi \vdash \Lambda$ where: Ξ, Λ are finite sets of addresses, pairwise disjoint, and Ξ contains at most one address. Notice that negative formulas are written on the left-hand side. There is at most one negative formula.

Designs: getting an intuition. Designs capture the geometrical structure of sequent calculus derivations. To start from the sequent calculus is the simplest way to introduce designs. Consider the following derivation, where $a^\perp, b^\perp, c, d$ denote formulas which respectively decompose as $a_0, b_0, {c_0}^\perp, {d_0}^\perp$.

$$\dfrac{\dfrac{\dfrac{\vdash a_0, {c_0}^\perp}{\vdash a_0, c}\ (c, \{{c_0}^\perp\})}{\vdash a^\perp, c}\ \{(a, \{a_0\})\} \qquad \dfrac{\dfrac{\vdash b_0, {d_0}^\perp}{\vdash b_0, d}\ (d, \{{d_0}^\perp\})}{\vdash b^\perp, d}\ \{(b^\perp, \{b_0\})\}}{\dfrac{\vdash c, d, a^\perp \otimes b^\perp}{\vdash c \parr d, a^\perp \otimes b^\perp}\ \{(c \parr d, \{c, d\})\}}\ (\downarrow a^\perp \otimes b^\perp, \{\ a^\perp, b^\perp\})$$

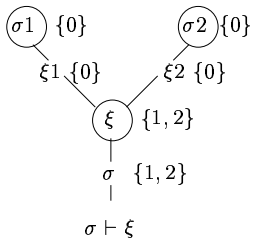
Let us forget everything in the sequent derivation, but the labels. The derivation above becomes the following tree of labels, which is in fact a (typed) design:



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 formulae 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 above. After the decomposition of $a^\perp \otimes b^\perp$, we know that c is in the context of $a^\perp$ because it is used after $a^\perp$, and that d is in the context of $b^\perp$, because it appears after it.

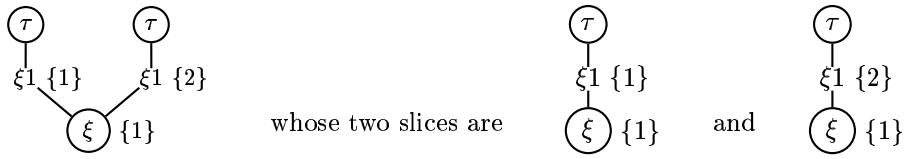
Since the sequent calculus is focalized, the proof construction follows the pattern: " (i) Decompose any negative formula; (ii)choose a positive focus, decompose it in its negative components, decompose the negatives; repeat (ii)." This is mirrored in the tree. In particular, polarities alternate, and a positive focus is always followed by its immediate sub-addresses.
Observe that the tree only branches on positive nodes. As a mnemonic aid, we represent the *positive nodes as vertices and the negative nodes as edges.*

To complete the process, let us now abstract from the type annotation (the formulas), writing only the addresses. In the example above, we locate $a^\perp \otimes b^\perp$ at the address ξ; for its subformulas a and b we choose the sub-addresses $\xi 1$ and $\xi 2$. Finally we locate a_0 in $\xi 10$ and b_0 in $\xi 20$. In the same way, we locate $c \parr d$ at the address σ and so on for its subformulas. Our design becomes:



The pair (ξ, I) is called an *action*. As we have seen, ξ is an address (the address of a formula) and I a set of natural numbers, the relative addresses of the immediate subformulas we are using. ξ is called *focus* of the action. The daimon † is also an action.

Where are the additives. The key to understand the &-rule in terms of design is to remember that the &-rule is a set (the super-imposition) of two actions on the (*same address*). Let us revisit our example of slices. Let us locate c in the address τ, $(a\&b) \oplus d$ in the address ξ, $(a\&b)$ in $\xi 1$, a in $\xi 11$, and b in $\xi 12$. The derivation of our previous example corresponds to the following design



The actions $(\xi 1\{1\})$ and $(\xi 1\{2\})$ should be be thought of as unary &; the usual binary rule is recovered as the set of actions on the address $\xi 1$.

Design: syntax. A *design* is given by *a base* and *a tree of actions* with some properties which we recall below.

A branch in the tree is called a *chronicle*. We think of the tree as oriented from the root upwards. If the action κ_1 is *before* κ_2, we write $\kappa_1 < \kappa_2$. We write $\kappa_1 <_1 \kappa_2$ if κ_2 immediately follows κ_1.

The base. A base is a sequent of addresses, which corresponds to the "initial" sequent of the derivation, the conclusion of the proof, the specification of the process. The base: (i) gives the addresses of the formulas we are going to decompose; (ii) induces a polarization of all the addresses (all the actions) in the design. According to its position, each address in the base has a *polarity*: positive (right hand side) or negative (l.h.s.). As in a synthetic connective the polarity of subformulas alternates at each layer, if ξ belongs to the base and is positive, ξi is negative, ξij is positive, and so on.

The tree of actions. A design $\mathfrak{D}$ of base $\Xi \vdash \Delta$ is: (i) a non empty tree of actions if the base is positive (there is only one first action), (ii) a (possibly empty)

forest of actions on the same initial focus if the base is negative (we can have a set of first actions on the same address).

Such a tree of actions satisfies the following conditions:
Root. The root (possibly roots in case of a negative base) focuses on an address of the base. If there is a negative address, that will be decomposed first.
Polarity. Polarities alternate.
Branching. The tree only branches on positive actions.
Focalization. The addresses used as focuses after a positive action (ξ, I) are immediate sub-addresses of ξ. Observe that † can only appear as a leaf, because it has no sub-addresses.
Sub-addresses. An address is either chosen in the base or has been created before (always $\xi < \xi i$). This simply corresponds to the *subformula property.*
Leaves. All maximal actions are positives.
Propagation (linearity). In all slices of $\mathfrak{D}$ each focus only appears once, where given a tree of action, a *slice* is a subtree such that the addresses $\xi i, i \in I$ after a positive action (ξ, I) are all distinct. This condition means that an addresses can be duplicated (reused) only in the context of a &.

Normalization. In Ludics there is no cut rule; a *cut* is a coincidence of addresses of opposite polarity in the base of two designs. A *cut-net* is a finite set $\mathfrak{R} = \{\mathfrak{D}_1, ...\mathfrak{D}_n\}$ of designs of respective bases $\Xi_i \vdash \Lambda_i$ The graph whose vertices are the $\Xi_i \vdash \Lambda_i$ and whose edges are the cuts is connected and acyclic. If we orient the edges from positive to negative, the design corresponding to the starting vertex is the *main design* of the cut-net. The uncut loci form a base, the *base of the cut-net.* A cut-net whose base is the empty sequent is said to be *closed.*

We call an address *closed* if it is a sub-address of a cut, *open* otherwise. The definition extends to actions.

The *normal form* of a cut net $\mathfrak{R}$ is indicated by $[\![\mathfrak{R}]\!]$. The normalization procedure on sequents of addresses mimics normalization in sequent calculus. In the next section, we will define normalization directly on the trees of actions.

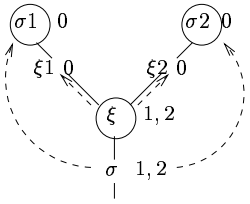
2 Slices as proof-nets

As a design, a slice is simply a tree of actions, where each address only appears once. Each action is uniquely determined by its focus. For this reason, when working with slices we often identify an action $\kappa = (\sigma, I)$ with its focus σ.

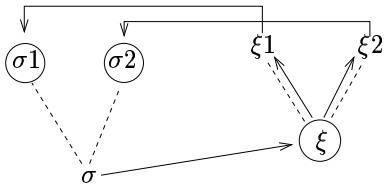
In a slice we are given *two orders,* corresponding to two kinds of information on the actions:

- the succession in time, recorded by the chronicles (the chronicles tree);
- the succession in space, corresponding to the relation of being sub-address (the prefix tree, which is analogous to a "sub-formula tree").

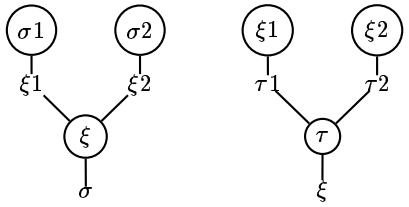
Let us again have a look at our previous example of design. We make explicit the relation of being a sub-address with a dashed arrow connecting σ to $\sigma 1$ and $\sigma 2$, and ξ to its sub-addresses, as follows:



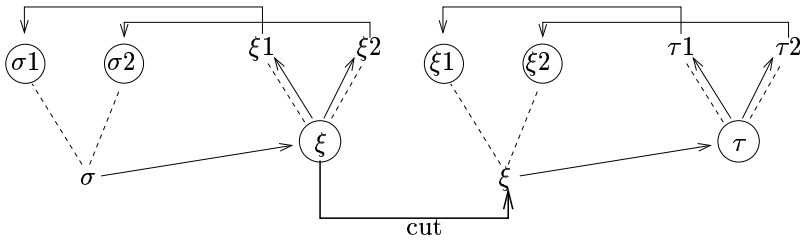
Consider a multiplicative proof-net, where the axioms are possibly "generalized axioms," that is hypothesis of the form $\vdash \Gamma$. Such a proof-net is a sub-formula tree with some extre information on the axiom links. If we emphasize the formula-tree rather than the chronicles-tree, we recognize something similar to a proof-net, added of some information on sequentialization. In particular this extre information allows us to establish the axioms links (generalized axioms, of the form $\xi \vdash \Gamma$) between the last-focused addresses, which are the leaves in the prefix tree. As we see below, in our example $\xi 1$ is connected to $\sigma 1$ and $\xi 2$ to $\sigma 2$.



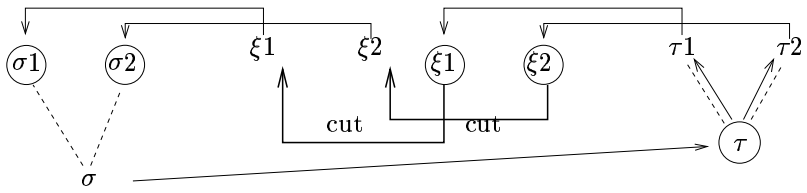
This suggests dealing with normalization as in proof-nets rather than as in sequent calculus. Essentially we mimic proof-nets normalization, as in the following example, where the cut-net



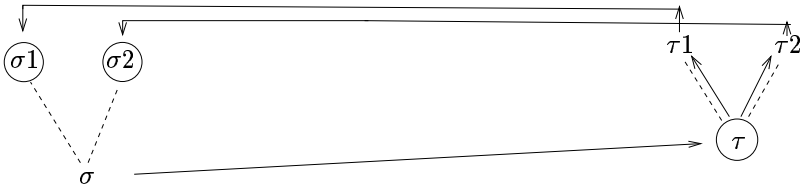
once written as



reduces as follows



and then to



In Ludics the situation is in general slightly more complex than in the above example, because the setting is not typed. Thus for example ξ could correspond on one side to the action $(\xi, \{1,2,3\})$ and on the other side to the action $(\xi, \{1,2\})$, or just not appear at all. Observe however that what we actually do on proof-nets is to connect (or to identify) two nodes with the same label. This can be done on designs. This idea underlies both the normalization as "quotient of orders" described in [6] and the abstract machine we define in the next section.

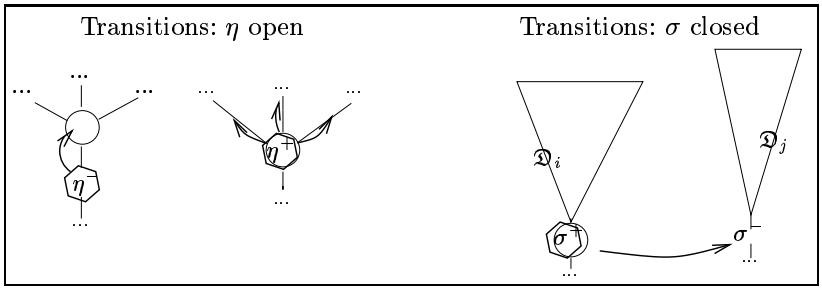
3 Loci Abstract Machine

Normalization of a cut-net $\mathfrak{R}$ can be presented by a token traveling along the net. This is implemented by a machine which we call Loci Abstract Machine (LAM). We first present a minimal version, which we indicate by $\mathbf{LAM_0}$, working on slices. Since in a slice there is no "additive duplication," normalization of slices is simpler than normalization of general designs. However, one could always work "by slices:" normalize slice by slice, and then put them together. In Section 5 we will generalize it.

The figure below presents the machine graphically. The key point is that when the same address σ appears in distinct designs, we can move from one design to the other, passing from σ^+ to σ^-. Observe that the token is always going *upwards*. While the token moves around, it draws a path on the cut-net. Each path will represent a chronicle of the normal form $[\![\mathfrak{R}]\!]$, as soon as we hide the closed actions (internal communication).

Initialization: The token enters the net on the root of the main design ($Main$).

Transitions: When the token is on an *open action,* κ it follows the chronicles order, moving *upwards* to the actions which immediately follow κ in the slice. When the token enters a *(positive) closed action*, it exits at the corresponding negative action (then changing of design).

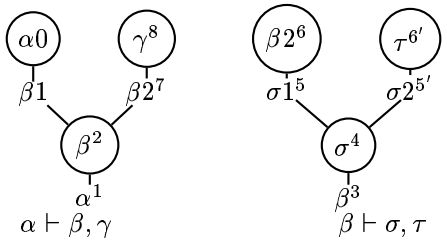


Below we give a formal definition of the machine. At the end of this section we will give an example of execution.

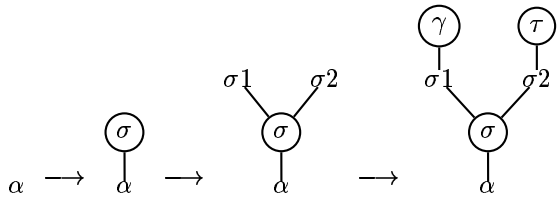
A *token* is given by a pair (s, κ). The action κ represents the current position of the token, while s is a list of actions, which records the path followed by the token. Each time the token enters an open action, that action is attached to the list. The transitions only depend on the position; the sequence of actions is only recorded to produce the normal form. We denote the empty sequence by ϵ.

Let T be the set of all positions reached by the tokens.
Initialization. If $Main \neq \emptyset$ then $T := \{(\epsilon, \kappa)\}$, where κ is the root of the main design $(Main)$.
Transitions.
(i) Let η be an *open* action (recall that open means not cut).
If $(s, \eta) \in T$ then $T := T \cup (s\eta, \kappa)$ for all $\kappa >_1 \eta$.
(ii) Let σ be a *closed* action (the focus is sub-address of a cut).
If $(s, \sigma) \in T$ and $\sigma^- \in \mathfrak{R}$ then $T := T \cup (s, \kappa)$ for $\kappa >_1 \sigma^-$.
Result. $[\![\mathfrak{R}]\!] = \{c : c \sqsubseteq s^+ \text{ and } (s^+, \kappa) \in T\}$, where s^+ is a sequence whose last action is positive.

Comments. When we enter a closed action σ, it is necessarily positive. We proceed to the corresponding negative action (then changing of design). If σ^- exists we move to the (unique) action which follows σ^-. If not, there is no way to extend s, and we are finished with that token. Notice that in this case s terminates on a negative action. Each maximal positive path describes a maximal chronicle of the normal form.
Example of execution. Consider the following cut-net, where the bases are respectively $\alpha \vdash \beta, \gamma$ and $\beta \vdash \sigma, \tau$. We decorate it with the path followed by the tokens: i indicate the i-ary step.



On σ the computation splits in two flows. There are two normalization paths, which are: $\alpha, \beta, \sigma, \sigma 1, \beta 2, \gamma$ and $\alpha, \beta, \sigma, \sigma 2, \tau$. As the token travels along, we only record the open actions and the normal form grows as follows:



From here it is immediate to recover the sequent calculus presentation:

$$\dfrac{\dfrac{\dfrac{}{\vdash \sigma 10, \gamma}\quad \dfrac{}{\vdash \sigma 20, \tau}}{\sigma 1 \vdash \gamma \qquad \sigma 2 \vdash \tau}}{\dfrac{\vdash \alpha 0, \sigma, \gamma, \tau}{\alpha \vdash \sigma, \gamma, \tau}}$$

Designs vs. sequent calculus normalization We could have presented the same cut-net with the syntax of sequent calculus.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \beta 10, \alpha 0}\ \alpha 0 \quad \dfrac{}{\vdash \beta 20, \gamma}\ \gamma}{\beta 1 \vdash \alpha 0}\ \beta 1 \qquad \dfrac{}{\beta 2 \vdash \gamma}\ \beta 2}{\vdash \alpha 0, \beta, \gamma}\ \beta, \{1,2\}}{\alpha \vdash \beta, \gamma}\ \alpha \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \sigma 10, \beta 2}\ \beta 2 \quad \dfrac{}{\vdash \sigma 20, \tau}\ \tau}{\sigma 1 \vdash \beta 2}\ \sigma 1 \qquad \dfrac{}{\sigma 2 \vdash \tau}\ \sigma 2}{\vdash \beta 1, \beta 2, \sigma, \tau}\ \sigma}{\beta \vdash \sigma, \tau}\ \beta$$

The reader is free to normalize on the sequent calculus, to check that the resulting normal form is actually the one associated to the result on designs.

4 Disputes and chronicles extraction

In the previous section we presented normalization by a token traveling around the cut-net. The token draws a path, which is a chronicle of the normal form, as soon as we ignore the closed actions. To calculate the normal form we only need to record the open actions. However, the *normal form is not necessarily the most interesting thing in normalization.* In Ludics, the most important case of cut-net is by far the closed one. If normalization converges, the normal form reserves no surprise: it is $\dfrac{}{\vdash}\ \dagger$. What is interesting is the interaction itself, that is the sequence of actions that have actually been visited (used) during the normalization.

We call *normalization path* the sequence of actions visited during the normalization of a cut-net. We indicate by $Paths(\mathfrak{R})$ the collection of all normalization paths on $\mathfrak{R}$. We call *dispute* the sequence of actions visited during the normalization of a *closed* net. If the net is $\{\mathfrak{D}, \mathfrak{E}\}$, we indicate the dispute by $[\mathfrak{D} \rightleftharpoons \mathfrak{E}]$.

Remark 1. It is immediate to modify the abstract machine given in the previous section into a machine that keeps track of all the visited actions.

Views. In a design, action with the same focus may appear several times, because of the use of n-ary negative rules (additives!). Each occurrence of an action κ is identified by the minimal chronicle $\mathfrak{c}\kappa$ in which it appears. We can see this as the position of that κ. As we shall see, for each action κ used in the normalization, the normalization path allows us to retrieve its position.

The key is to invert the process of constructing the path. This is in fact a well-known operation of HO-Nickau games [7], [8] the *view* operation. The notion of view is relative to a player, or to a parity in our setting. Let us recall some technical notions we need.

The space of addresses, and thus of actions, is split between two players: Even and Odd, according to the length of the address. A *base* has the same parity (even or odd) as the addresses on its positive side (all addresses on the right-hand side –positive– have the same parity, opposite to that of the address on the left-hand side). The empty base $\vdash$ is defined positive. A *design* is even or odd according to its base. An *action* is even or odd according to its focus. When an action (or a base, or a design) *has parity Even (Odd)* we also say that it *belongs to Even (Odd).*

The polarity (positive or negative) of each action in a design is *relative* to the parity (even, odd) of the design. In a design of base X, each X action is positive. We use the variable X, for X either Even or Odd, and $\overline{X}$ for the dual. To explicit if an action κ is Even or Odd, positive or negative we use the notation: $\kappa^E, \kappa^O, \kappa^+, \kappa^-$.

Any *cut-net* $\{D_i\}$ splits into two components: the collection of even designs $(\mathfrak{D}_i^E)$, and the collection of odd designs $(\mathfrak{D}_j^O)$. Hence we can write $\mathfrak{R}$ as $\{(\mathfrak{D}_i^E), (\mathfrak{D}_j^O)\}$. We extend the notation for disputes to this case, writing $[(\mathfrak{D}_i^E) \rightleftharpoons (\mathfrak{D}_j^O)]$.

Let us define the function *view* on $p \in Paths(\mathfrak{R})$. Observe that each action κ in p has a parity (Even/Odd). If κ belongs to X, it is X-positive and $\overline{X}$-negative. Given an action $(\xi, I) \in p$, we say that it is *initial* if ξ is not a sub-address of any other address in p (ξ belongs to the base of one of the designs in the cut-net).

Definition 1 (Views). *Let $p \in Paths(\mathfrak{R})$ and X be either Even or Odd. Its view $\ulcorner p \urcorner^X$ of p is defined as follows (positive and negative is relative to X).*

- $\ulcorner \epsilon \urcorner = \epsilon$;
- $\ulcorner s\kappa^+ \urcorner = \ulcorner s^- \urcorner \kappa$;
- $\ulcorner s\kappa^- \urcorner = \kappa$ *if* κ *is initial;*
- $\ulcorner s\kappa' t\kappa^- \urcorner = \ulcorner s \urcorner \kappa' \kappa$ *if* $\kappa = (\xi i, K)$ *and* $\kappa' = (\xi, I)$.

We denote Odd view *by* $\ulcorner q \urcorner^O$ *and the* Even view *by* $\ulcorner q \urcorner^E$. *It is convenient to adopt the following convention: by* $\ulcorner q\kappa^+ \urcorner$ *we mean the view of the player for which* κ *is positive. If* κ *belongs to* X, *then* $\ulcorner q\kappa^+ \urcorner = \ulcorner q\kappa \urcorner^X$ *and* $\ulcorner q\kappa^- \urcorner = \ulcorner q\kappa \urcorner^{\overline{X}}$

Notice that the notion of view applies to any $p = [(\mathfrak{D}_i^E) \rightleftharpoons (\mathfrak{D}_j^O)]$.

Chronicles extraction. Let $\mathfrak{R}$ be a cut-net whose designs are all slices and $p \in Paths(\mathfrak{R})$. We have that:

Proposition 1. *Let $\mathfrak{R}$ be a cut-net of slices, $p \in Paths(\mathfrak{R})$ and $q\kappa \sqsubseteq p$. If κ appears positive in $\mathfrak{R}$, then the chronicle $\mathfrak{c}\kappa^+ \in \mathfrak{R}$ is given by $\ulcorner q\kappa^+ \urcorner$. If κ appears negative in $\mathfrak{R}$, then the chronicle $\mathfrak{c}\kappa^- \in \mathfrak{R}$ is $\ulcorner q\kappa^- \urcorner$.*

Notice that an open action κ will appear in $\mathfrak{R}$ either positive or negative, never both.

Proof. The proof is by induction on the length of $q\kappa$. Let κ be an *open* action. The action η visited just before κ by normalization is the action that precedes κ in the chronicle. Let $q = q'\eta$ and $\mathfrak{c}\kappa = \mathfrak{c}'\eta\kappa$. By induction, $\ulcorner q'\eta \urcorner = \mathfrak{c}'\eta$. If κ is positive, $\ulcorner q'\eta\kappa^+ \urcorner = \ulcorner q'\eta \urcorner\kappa$. If κ is negative, $\ulcorner q'\eta\kappa^- \urcorner = \ulcorner q \urcorner \eta^+\kappa$, because the focus of κ is sub-address of that of η.

Let κ be *closed*. The positive case is as before. $\mathfrak{c}\kappa^-$ is of the form $\mathfrak{c}(\xi, I)^+(\xi i, J)^-$, where $(\xi, I) < (\xi i, J)^-$ in p. Hence $q'(\xi, I) \sqsubseteq q$, $\ulcorner q\kappa^- \urcorner = \ulcorner q'(\xi, I) \urcorner (\xi i, J)$ and $\ulcorner q'(\xi, I) \urcorner = \mathfrak{c}(\xi, I)$.

Proposition 1 has immediate *consequences* which we develop in the next sections.

5 LAM$_+$: generalized version

The normalization procedure given in Section 3 is well defined since in the case of slices there is only one occurrence of any focus. At the same time, it is idealized in the sense that we assume that the machine is able to find the next action by itself, in particular when moving from σ^+ to σ^-. Moreover, it would not be feasible if we were not working by slices: in a general design, the same action may appear several times (additive duplications). However, the sequence of visited actions carries all information needed to retrieve the position of any of its action (Proposition 1). In particular, when we enter a positive action κ^+ we are able to retrieve the chronicle that identifies the negative action κ^- to which we have to move. Assume p is the sequence of actions we have visited so far, and we enter the positive action κ^+. We then move to the action κ^- identified by the chronicle $\mathfrak{d} = \ulcorner p\kappa^- \urcorner$.

We can therefore define the following general machine to normalize arbitrary designs. Let $Paths(\mathfrak{R})$ be the set of all paths described on $\mathfrak{R}$. We have that:

- $\epsilon \in Paths(\mathfrak{R})$
- Let η be open and of polarity $x \in \{+, -\}$.
  If $p\eta \in Paths(\mathfrak{R})$ and $\ulcorner p\eta^x \urcorner\kappa \in \mathfrak{R}$ then $p\eta\kappa \in Paths(\mathfrak{R})$.
- Let σ be a closed action.
  If $p\sigma \in Paths(\mathfrak{R})$ and $\ulcorner p\sigma^- \urcorner\kappa \in \mathfrak{R}$ then $p\sigma\kappa \in \mathfrak{R}$.

Let $Norm(\mathfrak{R}) = \{hide(p) : p \in Paths(\mathfrak{R})\}$, where $hide(p)$ is p where we have deleted (hidden) all closed actions. We have that $[\![\mathfrak{R}]\!] = \{s, s \sqsubseteq q^+, q^+ \in Norm(\mathfrak{R})\}$, where q^+ is a sequence whose last action is positive.

6 Calculating the pull-back

The normalization of a closed cut-net produces a unique maximal path, the dispute. If we are given a dispute, we can calculate the minimal cut-net that produces it. We indicate this operation by $Pull(p)$.

Let $p = [\mathfrak{D} \rightleftharpoons \mathfrak{E}]$. $Pull^E(p)$ is defined as $\{\ulcorner q \urcorner^E : q \sqsubseteq r^+ \sqsubseteq p, q \neq \epsilon\}$. $Pull^O(p)$ is defined symmetrically. $Pull(p) = \{Pull^E, Pull^O\}$.

It is immediate, and it is important to notice, that $Pull(p)$ only depends on p. Thus for any cut-net $\mathfrak{R}$, the normalization produces the dispute p iff $Pull(p) \subseteq \mathfrak{R}$. As a consequence

Proposition 2. *Given a cut-net $\mathfrak{R}$ whose normalization produces the dispute p, $Pull(p)$ gives the the minimal $\mathfrak{R}_0 \subseteq \mathfrak{R}$ which produces p.*

In [6] $\mathfrak{R}_0$ is called the *pull-back* of p along $\mathfrak{R}$.

It is easy to extend the definition above to any closed cut net $\mathfrak{R}$. In such a case $Pull^E(p)$ and $Pull^O(p)$ are a set of chronicle that we can split into a collection of designs.

7 Computing a counter-design

Let us present another way to use the same machine "the other way round:" given a slice and a path on it, we calculate a counterdesign realizing the path.

A *path* p on a slice $\mathfrak{S}$ is a sequence of actions such that for any $p' \sqsubseteq p$ the region of $\mathfrak{S}$ covered by p' contains the root and is a tree. Now suppose we freely draw such a path on a slice. Is there a counter-design which realizes that path? Can we produce it? If we know that the counter-design exists, we can calculate the pull-back. Otherwise, we can build the counter-design "by hand" as follows.

Procedure. Assume we have a slice $\mathfrak{S}$ and a path $p = \kappa_0, ...\kappa_n$ on it. Our aim is to build a counter-design $\mathfrak{T}$ such that $[\mathfrak{S} \rightleftharpoons \mathfrak{T}] = p$. (We focus our discussion on the case where $\mathfrak{S}$ has base $\vdash \xi$ or $\xi \vdash$; the case of base $\Xi \vdash \Lambda$ is similar, but we have a family $\mathfrak{T}_i$ of counter-designs).

The *base* of $\mathfrak{T}$ is determined. To *build* $\mathfrak{T}$, we progressively place the actions of p to form a tree. The polarity of the actions in $\mathfrak{T}$ is opposite to that in $\mathfrak{S}$, as is the polarity of the base. If κ_i is negative in $\mathfrak{T}$, there is no ambiguity on where to place it: either it is the root, or it is of the form ξi, and we place it just after ξ (which is positive). If κ_{i+1} is positive in $\mathfrak{T}$, we need to place it just after κ_i (which is negative in $\mathfrak{T}$). In fact once the normalization is on a positive action κ_i^+ in $\mathfrak{S}$, it moves to the negative action κ_i^- in $\mathfrak{T}$, and then κ_{i+1}.

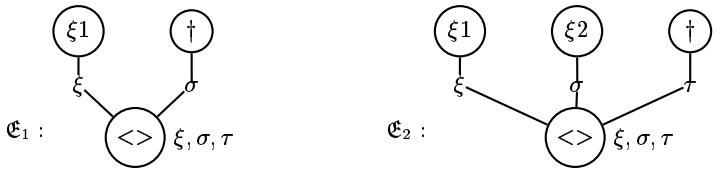
At any stage in $\mathfrak{T}$ there is at most one maximal branch terminating with a negative action. If κ_n, the last action of p, is *negative* in $\mathfrak{T}$, we complete $\mathfrak{T}$ with a daimon ($\dagger$) after κ_n. By construction, the normalization applied to $\{\mathfrak{S}, \mathfrak{T}\}$ produces p. We need to check that the tree we build is actually a design. The only property that is not guaranteed by construction is that of *sub-address* on positive focus.

8 An application: What can be observed interactively?

The program of Ludics is that of an interactive approach to logic. Ideally, we should be able to express and to test interactively the properties we ask to designs. Therefore what we know of a design is what we can see testing it against a counter-design. What part of a design can be visited during normalization? Normalization is always carried out in a single slice. Given a slice, can we build a counter-design which is able to completely explore it? Even if we only consider finite slices, the answer is no, as shown by the following example:

As we have sketched on the right hand side, such a design corresponds to a purely multiplicative structure. In fact we can easily type it, for example letting $F(\xi) = F(\xi 1) \otimes F(\xi 2)$, $F(<>) = F(\xi) \parr F(\sigma) \parr F(\tau)$, where by $F(*)$ we indicate the formula associated to the address $*$.

Let us build a counter-design to explore this slice. The path will start with $<>$, move to ξ, and then choose one of the branches, going either to $\xi 1$ or $\xi 2$. The two choices are symmetrical, so let us take $\xi 1$. At σ we are forced to stop, because there is no way to move to the other branch. The counter-design we have built is the following one ($\mathfrak{E}_1$).



The corresponding path is $<>, \xi, \xi 1, \sigma$, while the path we would like to have is $<>, \xi, \xi 1, \sigma, \xi 2, \tau$. $\mathfrak{E}_2$ (above) is the tree of actions that would realize this path. However, it is not a design, because it does not satisfy the sub-address condition ($\xi \not< \xi 2$).

An immediate consequence is that we *cannot interactively detect* the use of *weakening*, even in a slice. Consider again the example above, now assuming that the root is the action $(\xi, \{\xi, \sigma, \tau, \lambda\})$. The root creates an address, λ, which is never used. However, we cannot interactively detect that λ is weakened. Either we explore the left branch, or the right one. In the first case we see that σ is used. The other addresses, τ and λ, are possibly used after $\xi 2$. In the second case we see that τ is used, σ and λ being possibly used after $\xi 1$.

9 Related and further work

Interaction is central in Ludics, so it is important to have a theory telling what can be interactively recognized, and it is rather natural to take interaction traces as primitive and study designs from them.

In this paper we developed a concrete approach to designs, which gives us effective tools to address issues such as the following ones (see [4]).

(i) Study geometrical properties of the normalization paths, in the style of Geometry of Interaction.

(ii) Rebuild a slice out of a prefix tree of addresses.

(iii) Characterize the (parts of) designs that can be observed interactively: the designs that can be explored in a test (in a single run of normalization) represent the primitive units of observability.

(iv) Present designs as the collections of their disputes, which allows then establish a bridge with Games Semantics [5].

Related work. Our normalization on designs (rather than on the sequent calculus) is analogous to the order quotient defined in [6], though it was developed independently. Our approach is more local, hence easier to use for actual computations. Actually, what the machine does is to calculate the balanced slice. On the other hand, Girard's theory provides a synthetic view, which better suits the development of general results.

The notion of design is very close to that of abstract Böhm tree introduced by Curien as a generalization of lambda terms and as a concrete syntax for games. The way we proceed closely relates our work to the abstract machines studied by Curien and Herbelin in [3]. Our generalized LAM is actually an instance of the View abstract machine, introduced by Coquand in [2].

References

1. J.-M. Andreoli and R. Pareschi. Linear objects: logical processes with built-in inheritance. *New Generation Computing*, 9(3-4):445–473, 1991.
2. T. Coquand. A semantics of evidence for classical arithmetic. *Journal of Symbolic Logic*, (60), 1995.
3. P.-L. Curien and H. Herbelin. Computing with abstract bohm trees. In *Third Fuji International Conference on Functional and Logic Programming*, Kyoto, 1998. Word Scientific.
4. C. Faggian. *On the Dynamics of Ludics. A Study of Interaction.* PhD thesis, Université Aix-Marseille II, 2002.
5. C. Faggian and M. Hyland. Designs, disputes and strategies. In *CSL 2002 (this volume)*, LNCS. Springer, 2002.
6. J.-Y. Girard. Locus solum. *Mathematical Structures in Computer Science*, 2001.
7. M. Hyland and L. Ong. On full abstraction for PCF. *Information and Computation*, 2000.
8. H. Nickau. Hereditarily sequential functionals. In *Proceedings of the Symposium on Logical Foundations of Computer Science: Logic at St. Petersburg*, LNCS. Springer, 1994.