ELSEVIER

Available online at www.sciencedirect.com

SCIENCE DIRECT®

Theoretical Computer Science 350 (2006) 213–233

Theoretical Computer Science

www.elsevier.com/locate/tcs

Interactive observability in Ludics: The geometry of tests

Claudia Faggian

Universitá degli Studi di Padova, Italy

Abstract

Ludics [J.-Y. Girard, Locus solum, Math. Structures in Comput. Sci. 11 (2001) 301–506] is a recent proposal of analysis of interaction, developed by abstracting away from proof-theory. It provides an elegant, abstract setting in which interaction between agents (proofs/programs/processes) can be studied at a foundational level, together with a notion of *equivalence from the point of view of the observer*.

An agent should be seen as some kind of black box. An interactive observation on an agent is obtained by testing it against other agents.

In this paper we explore *what can be observed interactively* in this setting. In particular, we characterize the objects that can be observed in a single test: the *primitive observables* of the theory.

Our approach builds on an analysis of the geometrical properties of the agents, and highlights a deep interleaving between two partial orders underlying the combinatorial structures: the spatial one and the temporal one.

Keywords: Ludics; Game semantics; Linear logic; Geometry of interaction

General motivations: A motivation for calculi describing sequential or concurrent computation (λ-calculus, π-calculus, . . .) is to allow formal methods to prove properties of programs/processes. A typical property we are interested in is when two terms are the same *from the point of view of the observer*. A standard approach is to define equivalence relations on terms. Such techniques go back to the works on λ-calculus and PCF and in the context of process calculi have given rise to a whole range of equivalence relations. Still, one can wonder whether the study of equivalence properties could not be simplified by reconsidering the design of the calculus, and tailoring it to have a good theory of interaction. That is, building the syntax on the equivalence relation, rather than the opposite. The point of view that interaction should come before syntax is the base of Ludics.

Ludics: Ludics [13] is a recent proposal of analysis of interaction in a proof-theoretical setting. The whole theory is built on the analysis of the interaction between a design (proof/program) and counter-designs (again proofs/programs), where designs are rather simple, combinatorial structures which abstract away from the syntax of proofs. The emphasis on the symmetry between observed system and observer gives rise to interesting geometrical properties, on which the study of equivalences is found.

Ludics encompasses and builds on ideas from proof theory, proof search and different approaches to semantics. In particular, on the semantical side, Ludics is close to Game Semantics (see [8]). It could be seen as a sort of Game Semantics, equipped with an internal notion of equivalence from the point of view of the observer. On the syntactical

E-mail address: claudia@math.unipd.it.

0304-3975/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2005.10.042

side, designs can be understood as proofs in the logic programming approach (as in the work by Andreoli) or as lambda-terms in the form of abstract Böhm trees [3]. For these reasons, Ludics provides a clean, general setting in which interaction and equivalence can be studied at a foundational level using geometrical intuitions.

Contents of the paper and relevance to Ludics: In the setting of Ludics, the meaning of a proof/program (design) is given by its behaviour, that is by the way it interacts with the other proofs/programs.

Ludics is a setting in which the objects (proofs/programs) should be defined by the results of their interactions. A proof/program should be seen as some kind of "black box," that we can observe only by testing it against other proofs/programs. In particular, if two objects react in the same way to all the tests, we cannot distinguish between them, and they should actually be the same syntactical object. This property is called *separation,* (the analogue in Ludics of Böhm's theorem), and it is a fundamental requirement for Ludics as an interactive theory.

However, separation is not the only property we can be interested in: *all* properties should be (ideally) not only tested but also determined interactively. Therefore, it appears important to the theory to understand what can be observed interactively. This is the main object of our paper.

It is a fact, even if it had not been expected in the beginning of Ludics, that there are properties which cannot be detected in an interactive way. An example is the use of weakening in a proof [13, p. 392], and [7]. We will discuss this issue in Appendix A.1, but the essential reason behind this is that such a property cannot be observed in a single test; recovering the information when splitted among several observations presents serious difficulties.

In this paper we explore *what can be observed interactively* in the untyped setting of Ludics (for the typed case see [6]). More precisely, we characterize the (partial) designs that can be observed at each single test: the *primitive observables*.

To this purpose we establish combinatorial methods, which we believe are of interest in themselves. We develop two approaches which we respectively qualify of (i) "dynamic" and (ii) "static". Approach (i) consists in studying the geometrical properties of the paths induced by normalization, much in the style of geometry of interaction [5]. Approach (ii) analyzes statically the "spatial" properties of the combinatorial structures (designs).

The geometrical properties we study provide a better understanding of the computational structures and the dynamics of their interaction. In particular, we point out a deep interleaving between two key relations: "time" (sequentiality) and "space" (internal dependency). This understanding will be a base and a guide for further developments.

1. Background

Designs are the structures of Ludics which correspond both (syntactically) to proofs/programs, and (semantically) to their interpretation. In the setting of Ludics, everything lives in the same universe: proofs of A and proofs of $A^{\perp}$, player and opponent strategies, both have the same nature, and satisfy the same rules. The issue of distinguishing special "good" objects is postponed, and left to interactive methods.

Let us first introduce the key notions in an informal way.

1.1. A first overview

Daimon and para-proofs: Ludics provides a general setting in which to any proof of A we can oppose (via cut-elimination) a proof of $A^{\perp}$. To obtain this, a new rule, called the *daimon* is introduced. It is indicated by †, and allows us allows us to justify (assume) any conclusion. All proofs are read bottom-up. Seen this way, a daimon stops the flow of computation: it can be seen as the occurrence of an error, or a sort of "quit".

Addresses: A novelty of Ludics are locations. Proofs do not manipulate formulas, but *addresses*. An address is a sequence of natural numbers, which could be thought of as a name, a channel, or as the address in the memory where an *occurrence of formula* is stored. If we give to an occurrence of formula address ξ , its (immediate) subformulas will receive address ξi, ξj, etc.

Actions: An elementary operation is called an *action* (which corresponds to a move in Game Semantics). An action should be thought of as a cluster of operations which can be performed at the same time. This has a precise proof-theoretical meaning in the calculus which underlies Ludics, i.e. second order Multiplicative-Additive Linear Logic (MALL) [10]. Multiplicative and additive connectives of linear logic split in two families: positives ($\otimes$, $\oplus$) and negatives

($\parr$, &). A cluster of connectives 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*.

Each action has an address (a name) and a polarity (positive or negative). Two actions with the same address, but opposite polarity, correspond to occurrences of opposite type (A and $A^{\perp}$) which can be "plugged" together (that is, cut together).

Designs: Designs are *trees of actions*. As we have already said, they are the computational structures of Ludics (proofs/programs/strategies).

Designs capture the geometrical structure of sequent calculus derivations, providing a more abstract syntax. This syntax can be seen as intermediate between sequent calculus and proof-nets. In the next section we provide a proof-theoretical intuition. Still, one can simply think of a design as a combinatorial object, a tree of actions with certain properties that we are going to present.

Interaction and orthogonality: Interaction between designs takes place through normalization. Normalization of designs is the analogue of normalization on proof-nets, or cut-elimination on sequents.

The core of the theory is actually the normalization between (the addresses analogue of) "proofs of A" and "proofs of $A^{\perp}$". The normal form, if it exists, is necessarily a proof of the empty sequent, which actually can only be proved using the daimon rule, as this rule can justify any conclusion. In fact, either normalization fails or it produces the only possible proof of the empty sequent: $\dfrac{}{\vdash}\ \dagger$. In the latter case, the two proofs we have opposed one to the other are called *orthogonal*.

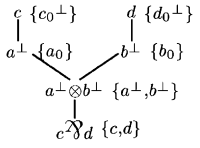
Interactive types: We said that proofs deal with addresses rather than with formulas. The use of addresses makes the designs and the calculus *untyped*, as we abstract from the type annotation. *Types* are recovered in Ludics "semantically" as sets of proof/programs which behave the same way in reaction to the same set of tests: these interactive types are called *behaviours*. Orthogonality is the key notion to define behaviours.

Proof-theoretical intuition: The easiest way to understand the intuition behind designs, is to start from sequent calculus derivations. Consider the following derivation, where each rule is labelled by the active formula and the subformulas which are used in the premises (rather than more usual labels such as $\otimes L$, $\otimes R$, etc.)

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

where c, d, $a^{\perp}$, $b^{\perp}$ denote formulas which respectively decompose into $c_0{}^{\perp}$, $d_0{}^{\perp}$, a_0, b_0 (for example, you could think of c as $c_0{}^{\perp} \oplus c_1{}^{\perp}$, etc.).

Now we forget everything in the sequent derivation, but the labels. The derivation above becomes the following tree of labels (actions):

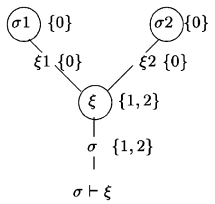


This formalism is more concise than the original sequent proof, but still carries all relevant information. To retrieve the sequent calculus counterpart is immediate. Rules and active formulas are explicitly given. Moreover, we can *retrieve the context dynamically*. For example, when we apply the Tensor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

Reversing the process, each point (each label) in the tree corresponds to a sequent. Such a sequent is conclusion of a rule whose active formula is the one of the label. The other formulas (the context) are those appearing afterwards (ignoring the subformulas). Decomposing a positive formula ($a^{\perp} \otimes b^{\perp}$) creates several premises; in the tree each premise corresponds to a branch. Instead, decomposing a negative formula ($c \parr d$) produces a single premise. This remains true also when working with the additive connectives, even though it is more subtle to explain, because there is

a mismatch between sequent calculus and design syntax. The result is that *the tree of actions only branches on positive nodes.*

To complete the process, we have to abstract from the type annotation (the formulas), and work with 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 $\xi 2$. In the same way, we locate $c \parr d$ at the address σ and so on for its subformulas. Our design becomes the following tree. Because the tree only branches on positive nodes, as a mnemonic aid, we represent the positive actions as nodes (circling them) and the negative actions as edges.



The pair (ξ, I) is an *action*. ξ is the address of the formula, while the set of natural numbers I correspond to the relative addresses of the immediate subformulas.

A word of warning: Our example is simplified by the fact that all connectives are binary. Each action should actually always be thought of as a tree of connectives with the same polarity, decomposed at once. The accompanying set of indices makes explicit the relative subaddresses of the sub-formulas which are created in the decomposition.

The tree of addresses we obtain has some properties which mirror those of the sequent calculus. To understand them, we have to explain focalization.

Focalization: The property we are using when working with synthetic connective is known as *focalization*, first discovered by Andreoli [1] for proof-search. MALL negative connectives are reversible, i.e. the rule to decompose them (bottom-up) is deterministic and can be immediately applied. Instead, on positive connectives there is a real choice to perform (non-determinism); however, given a sequent of only positive formulas, there is at least a formula, called *focus*, that can be selected as active in the last rule and then entirely decomposed up to its negative sub-formulas. This provides a strategy in proof-search: (i) negative rules are performed immediately, (ii) positive rules, once chosen a focus, are persistently applied up to their negative sub-formulas. The most important consequence, from the point of view of logic, is that a cluster of connectives of the same polarity can be seen as a single connective, a synthetic connective. A formula can then be seen as an alternation of positive and negative layers.

As one of the steps leading to a more abstract syntax, the sequent calculus underlying Ludics is focalized. The proof construction repeats the pattern: "(i) decompose any negative formula (it is easy to show that in any sequent there is at most one negative formula when this strategy is applied since a positive rule produces exactly one subformula in each premise); (ii) choose a positive focus, and decompose it in its negative components." Moreover, to apply positive (negative) rules in a thread can be seen as a single step.

The resulting process of proof construction is mirrored in the tree of actions. In particular:

- polarities alternate;
- a positive focus is always followed by its immediate sub-address;
- after a negative decomposition we select a new positive focus (exactly one).

Normalization and slices: To understand the proof-theory of Ludics, and in particular normalization, it is important to understand the notion of slice. This notion has been introduced as part of the theory of proof-nets of Linear Logic, to deal with the additives [10]. In the same way as there are two $\oplus$-rules ($\oplus_L$ and $\oplus_R$), also a &-rule can be decomposed in two unary rules: $\&_L$ and $\&_R$. A usual binary & is the "superimposition" of two unary $\&_i$. Think of 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.* For example, the derivation

$$\dfrac{\dfrac{\cdots}{\vdash a, c} \quad \dfrac{\cdots}{\vdash b, c}}{\vdash a \,\&\, b, c}\ \&$$

if there is no other application of &-rule, can be decomposed in two slices:

$$\frac{\dfrac{\cdots}{\vdash a, c}}{\vdash a \,\&\, b, c}\ \&_L \quad \text{and} \quad \frac{\dfrac{\cdots}{\vdash b, c}}{\vdash a \,\&\, b, c}\ \&_R$$

It should be clear that the interaction (normalization) between orthogonal proofs (designs) is always carried out in a single slice: *to select* one of the premises of a &-rule is exactly what happens when normalizing against $\oplus$.

In first approximation, one can think of a slice as an MLL (multiplicative linear logic) derivation, but in fact the connectives also include $\oplus$ and unary &. It has been an idea of Linear Logic since long that slices are the perfect syntax for additive proof-nets: a proof should be seen as the superimposition of all its slices. The advantage is that normalization of slices enjoys the same pleasant properties as that of MLL proof-nets. The difficulty is how to "superimpose" the slices. Addresses and sequentialization make this possible in Ludics.

A design can be presented as a set of slices which enjoy some coherence conditions. Normalization on designs works very well by slices. The normal form of cut designs is given by the set of the normal forms of all the possible slices. Conversely, any slice in the normal form comes from the normalization of slices in the original designs. Normalization therefore becomes the following procedure:

- normalize all possible slices;
- put their normal forms together.

The result is a set of slices, which in turn is a design.

Since the properties we are interested in are related to normalization and orthogonality, in this paper we will concentrate on slices.

1.2. Definitions

Actions: An *address* ξ is a sequence of natural numbers. The empty sequence is indicated by $\langle\rangle$. We say that σ is a *subaddress* of ξ if ξ is prefix of σ (written $\xi \sqsubseteq \sigma$); ξi is an immediate subaddress of ξ. If we think of ξ as the address of a formula A, we think of a subaddress of ξ as the address of a subformula of A.

An *action* is either the symbol † or a pair $\kappa = (\xi, I)$ given by an address ξ and a set I of indices which define the possible subaddresses. We say that κ *creates* the addresses ξi, for all $i \in I$. To an action in a design is also associated a polarity, positive (κ^+) or negative (κ^-).

Base: A base provides the addresses which will be used in a design (the conclusion of the proof, the specification of the process). To the base is associated a polarity. Here we only consider designs with a single address as base. This captures the most interesting case, does not imply any loss of generality, and will simplify the presentation.

Designs and slices: A *design* is given by a base and a structure which can be presented in several ways. In particular, a design can be seen as a set of slices, with some coherence properties.

A *slice* is given by a base and a tree of actions satisfying certain conditions. We think of the tree as oriented from the root upwards. If the action κ_1 is *before* κ_2, we write $\kappa_1 < \kappa_2$. A sequence of actions starting from the root is called a *chronicle*. The sequence of actions leading from the root to an action κ is *the chronicle of* κ.

To be a slice, a tree of actions must satisfy the *conditions* below:

Root: The root is an action on the address given by the base.

Polarity: The polarities of the actions alternate. The root has the same polarity as the base.

Sub-address: Given an action κ which is not †, its address either belongs to the base or it is a σi such that in the tree there is an action (σ, I), $i \in I$ and $(\sigma, I) < (\sigma i, K)$, for some K. This mirrors the *sub-formula property*.

Branching: The tree only branches on positive actions.

Negative addresses: The addresses used immediately after a positive action of address ξ are immediate sub-addresses of ξ. Notice that, as a consequence, † can only appear as a leaf.

Leaves: All maximal actions are positive.

Linearity: Each address only appears once.

A design is a set of slices, where the superimposition is obtained as the union of the chronicles. For example, in Fig. 1 the union of the three slices on the left-hand side (notice that a chronicle is in particular a slice) produces the design on the right-hand side, which is a design but not a slice since $\xi 0$ appears twice.

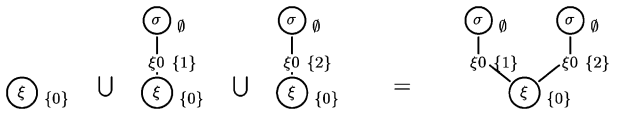


Fig. 1.

We recall the definition of designs for the sake of completeness, but in this paper we will never need them, as the properties we are interested in are only concerned with slices.

Coherence conditions on a design:

- Two chronicles only branch on positive actions (they first differ on negative actions).
- If two positive actions κ_1, κ_2 have the same address, their chronicles first differ on two negative actions $(\xi, I)^-$, $(\xi, J)^-$ having the same address.

If we have in mind the proof-theoretical intuition, the pair of negative actions should be thought of as the unary components of a &. The common address of κ_1, κ_2 is "duplicated" as happens to an additive context. This is exactly the situation illustrated by the design on the right-hand side of Fig. 1 (where $\kappa_1 = \kappa_2 = (\sigma, \emptyset)$). The address $\xi 0$ could be thought of as an occurrence of $a\&b$, respectively decomposed by $\&_L$ $((\xi 0, \{1\}))$ and $\&_R$ $((\xi 0, \{2\}))$.

Simplifications: When working just with slices (instead of designs), a few simplifications are possible. As we focus on the case of a base with a single address, and this is forced to be the initial address used by the slice, we identify the base with the root of the designs. All addresses in the tree will be subaddresses of the root.

As in a slice all addresses are distinct (by the linearity condition), *each action is uniquely determined by its address.* For this reason, we will identify the action $\kappa = (\sigma, I)$ with its address σ.

1.3. Sequential and prefix order

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

1. a time relation (*sequential order*): $\kappa_1 \leqslant \kappa_2$;
2. a space relation (*prefix order*), corresponding to the relation of being sub-address: $\xi \sqsubset \xi'$.

In particular, the sub-address condition can be reformulated as: $\sigma \sqsubset \xi \Rightarrow \sigma \leqslant \xi$.

A contribution of this paper is to evidence how the interplay between these two orders forces structure and remarkable properties, on which normalization ultimately relies. In particular, we will study the properties of the prefix tree alone, and show as some relevant aspects only depend, statically, on the prefix tree alone, and are independent, and invariant, w.r.t. the particular sequentialization.

In a design there are two trees, even though the usual presentation only emphasizes the sequential tree, leaving the prefix tree implicit. Designs, as a syntax, are intermediate between sequent calculus and proof-nets (both these aspects can be made precise). Interestingly, bringing to the foreground one tree or the other emphasizes the relation with one syntax or the other.

In a previous section, we have put forward the connection with a sequent calculus derivation. Let us now consider the design in the left-hand side of Fig. 2, and make the sub-address relation explicit. If we bring the prefix tree on the foreground (Fig. 2, right-hand side) it is the connection with proof-nets which appears (we could type σ^- as $\parr$ and ξ^+ as $\otimes$). Let us illustrate this very informally. In a multiplicative proof-net we have a sub-formula tree with some extra information which are the axiom links. A more general form of extra (sequential) information are boxes, largely used in polarized proof nets [15]. This is exactly what one finds in Fig. 2: a prefix tree, and some extra sequential information which can be seen as boxes. This information in particular allows us to retrieve the axiom links. Notice as it is actually standard to see axioms (and in particular generalized axioms) as a kind of box. We do not go into details here, but making these intuitions precise, one would actually find polarized proof nets. Still, having in mind the proof-net presentation will help to understand normalization.

1.4. Normalization and orthogonality

As in the λ-calculus, normalization between designs is deterministic but not necessarily terminating. The most important use of normalization in Ludics is to oppose a proof of A to a proof of $A^\perp$ (a design $\mathfrak{D}$ of base ξ^+ to a design $\mathfrak{E}$

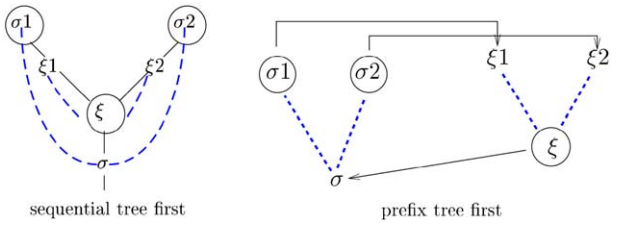


Fig. 2.

of base ξ^-). In the closed case of normalization (that is A against $A^\perp$), the result can only be an empty conclusion (the empty base). There are only two possible outcomes: either normalization fails (diverges) or it terminates, producing as result a daimon †, as daimon is the only rule that can "prove" everything, and in particular an empty conclusion.

If normalizing $\mathfrak{D}$ against $\mathfrak{E}$ results in a †, we say that $\mathfrak{D}$ and $\mathfrak{E}$ are *orthogonal*: $\mathfrak{D} \perp \mathfrak{E}$. Orthogonality is a key notion in Ludics. A *type* (called *behaviour*) is a set $\mathbf{G}$ of designs equal to its biorthogonal ($\mathbf{G} = \mathbf{G}^{\perp\perp}$).

Normalization produces a merging of the cut slices, and the normal form is then obtained by hiding the "private communication", that is those actions which belong (with opposite polarity) to both slices. In the closed case of normalization, the whole interaction is "private communication". Nothing is visible from "outside", but in fact what we are interested in is the process of interaction itself, and in particular in those actions which actually take part in it.

As normalization on proof-nets coincides with connecting nodes with opposite label, so normalizing on slices essentially means to identify opposite labels. This is the idea which underlies both the presentation of normalization as a "quotient" between the (sequential) orders associated to each slice in [13], and the operational procedure given in [7], which we will use here. We think of a token traveling on the slices to be cut, and describing the visit performed by normalization. In the closed case, the procedure simply becomes:

- *Start* on the root of the positive slice.
- From a *positive proper action* κ^+ move to the corresponding negative action κ^- (changing slice). If κ^- does not belong to the cut slices, the process fails.
- From a *negative action* κ^- move upwards to the unique action which follows κ in the same slice.
- On the special action †, the process successfully terminates.

Not all actions are used by normalization. The part of the slices which actually interacts is the part which matters, the part which actually has a chance to determine properties via orthogonality. While it is clear that actions which are never reached do not play any role, equally important is the order in which the actions are reached.

Normalization between two slices $\mathfrak{S}$ and $\mathfrak{T}$ establishes a linear order among the actions which are used. The sequence of visited actions, which represents the trace of the *interaction* between $\mathfrak{S}$ and $\mathfrak{T}$, is indicated by $[\mathfrak{S} \rightleftharpoons \mathfrak{T}]$ and called *dispute*. This exactly corresponds to a play in Game Semantics. This sequence induces a path on each of the two cut slices. When we speak of *visiting a slice* (or a design), we always mean visiting it by normalizing. We also call the sequence of visited actions a *normalization path*. Notice how each slice determines a traversal strategy for the opponent tree.

1.5. Computing a counter-design

Given a slice and a path on it, is there a counter-design which realizes that path? We illustrate here a construction that follows immediately from the procedure of normalization described above. We prefer to spell out because we will use it all the time.

By *path* p on a slice $\mathfrak{S}$ we intend a sequence of actions starting from the root of $\mathfrak{S}$ and such that the restriction of $\mathfrak{S}$ to the actions in any $p' \sqsubseteq p$ is a subtree. Assume we have a slice $\mathfrak{S}$ and a path $p = \kappa_0, \ldots, \kappa_n$ on it. Our aim is to build a counter-design $\mathfrak{T}$ such that $[\mathfrak{S} \rightleftharpoons \mathfrak{T}] = p$.

Procedure: To *build* $\mathfrak{T}$, we progressively place the actions of p to form a tree, that we want to be a counter-design. 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 (†) 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 slice. The only property that is not guaranteed by the construction is that of *sub-address* (on the positive focuses).

Conventions: The leaves condition on slices allows us to deal with daimon implicitly: we can assume that any maximal action which is negative is followed by a †.

Unless needed, we will not make explicit the polarity of the actions in a design. However, in all pictures, when representing a slice, we indicate the polarities by *circling the positive nodes.*

2. Questions

2.1. Slices and prefix trees

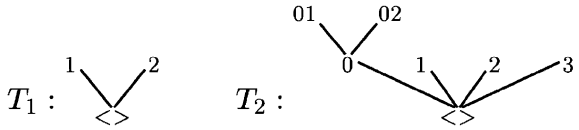
Given a slice $\mathfrak{S}$, we can associate to it the tree of the addresses which appear in $\mathfrak{S}$, with the relation $\sqsubseteq$. We call such a tree the prefix tree of the slice. More generally, let us consider an address σ and the tree corresponding to the prefix order on its sub-addresses. We call *prefix tree* any finite sub-address of such a tree. The prefix tree associated to a slice $\mathfrak{S}$ is the addresses analogue of the "sub-formula tree." We indicate it by $\mathbf{T}(\mathfrak{S})$.

When considering the prefix tree of a slice, we forget some information about sequentiality. For example, looking at Fig. 3, to the slice consisting of the single chronicle $\mathfrak{C}_1$ corresponds the prefix tree in the middle.

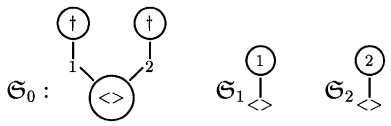
In the prefix tree we forget that the action of address 1 is performed before the action of address 2. Notice that the same prefix tree is associated also to the chronicle $\mathfrak{C}_2$ where the address 2 is scheduled before 1.

Could we deal with slices "modulo sequentiality"? Actually, a natural question which arises is whether given a prefix tree, we can associate to it a slice using the same addresses [1]. If this is possible, in general there will be several ways to do so, corresponding to several ways to sequentialize the actions (add a sequential order to).

Given a prefix tree T, *a slice* associated to it is any slice $\mathfrak{S}$ such that $\mathbf{T}(\mathfrak{S}) = T$. It is not always possible to associate a slice to a prefix tree. To give an *example*, let us consider the prefix trees T_1 and T_2 in the picture below.



To T_1 we can associate a positive design (which is $\mathfrak{S}_0$ in the following picture), but there is no way to associate to T_1 a negative design.



We can build $\mathfrak{S}_1$ or $\mathfrak{S}_2$, but there is no space to add a second positive action. In fact, when we sequentialize, each negative action has space to allow only one positive action. For similar reasons, to T_2 we cannot associate neither a positive nor a negative slice (the reader should try).

2.2. What can be observed interactively?

What part of a design can be visited by a closed normalization? We know that normalization is always carried out in a single slice. Given a slice, can we build a counter-design which is able to completely recognise (visit) it? Even if we only consider finite slices, the answer is no, as shown by the following example, which we have introduced in [7].

[1] We deal with daimon implicitly: any maximal branch terminating on a negative action is completed by a daimon.

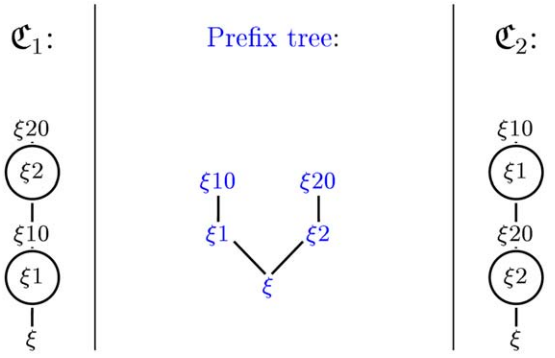


Fig. 3.

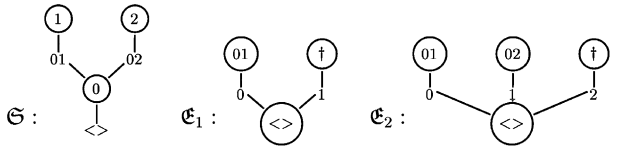
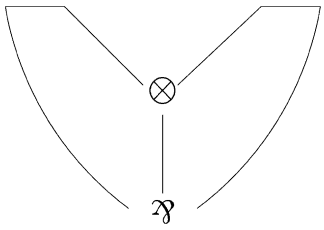


Fig. 4.

Example 1 (*Main example*). Let us try to build a counter-design to explore the slice $\mathfrak{S}$ in the picture. Normalization must start with $\langle\rangle$, use 0, and then, depending on the counter-design, it will choose one of the branches, going either to 01 or 02. The two choices are symmetrical, so let us take 01. At 1 we are forced to stop, because there is no way to move to the other branch. In fact the counter-design we have implicitly built is $\mathfrak{E}_1$ (picture above), corresponding to the path $p = \langle\rangle, 0, 01, 1$, while the path we would like to have is $p' = \langle\rangle, 0, 01, 1, 02, 2$. The tree of actions that would realize this path is actually $\mathfrak{E}_2$ (in the picture above), where we dispose the action in a configuration that via normalization would produce p'. However, this is not a design, because it does not satisfy the sub-address condition ($0 \not\prec 02$).

Consequences: As a *consequence* there are properties which we *cannot interactively detect*. One paradigmatic example is the use of *weakening*, whose relevance is related to full completeness. As this fact is important to the theory of Ludics, we discuss it in the Appendix A.1.

Comments: The slice $\mathfrak{S}$ in Fig. 4 corresponds to a purely multiplicative structure. In fact we could easily type it as follow: $F(\langle\rangle) = F(0)\parr F(1)\parr F(2)$, $F(0) = F(01) \otimes F(02)$, where by $F(*)$ we indicate the formula associated to the address $*$. The result has the following form:



What is striking with this example is that there is no communication between the two branches of the slice $\mathfrak{S}$. They behave as distinct components. This goes against the intuition on slices which comes from Linear Logic (when thinking of a slice as a multiplicative proof-net). All correctness criteria, such as Girard's long trip or Danos–Regnier (for all switching, the graph is *connected* and acyclic) correspond to the idea that all formulas are connected via normalization. For a finite slice of a design, this idea to be "connected" by the normalization does not longer hold.

It could even make sense to introduce a notion of *strong* slice, as a slice that can actually be visited in a single normalization. In this case, there is a communication among all the actions, in the sense that all the actions can be connected by a normalization path.

2.3. Plan

In this section we introduced two questions:

- What is the relation between slices and prefix trees?
- What can be interactively observed?

These two questions are closely related, and actually appear as two aspects of the same problem. Indeed, to be able to visit all the actions of a slice $\mathfrak{S}$ via normalization means that we have a counter-design which is also a slice, and whose actions are the same as those of $\mathfrak{S}$ with opposite polarity. This corresponds to being able to sequentialize those actions into a slice.

It is fairly easy to find a sufficient condition on prefix trees to guarantee that we can associate to it a slice. To establish that it is also necessary is harder. It will follow from a study of the properties of normalization paths. Moreover, we will be able to characterize those prefix trees to which we can associate both a slice and a counter-slice.

The key point is that if we want that both the design and the counter-design are able to develop the dialogue (interact) on all the addresses, we need a balance between positive and negative addresses. In the following sections we will make this precise, establishing three equivalent conditions. Each of them captures a slightly different intuition on what makes interaction possible.

3. Static approach

3.1. Polarized trees of addresses

Polarized trees: Given a tree (and in particular a prefix tree), we can associate a polarity to the nodes. A *polarized tree* T is a tree whose nodes are alternatively labelled with opposite polarities. A polarized tree is positive or negative according to the polarity of the root. We indicate by $N(T)$ the number of negative nodes in T, and by $P(T)$ the number of positive nodes. We indicate by $N_k(T)$ the number of negative nodes of arity k and by $P_k(T)$ the number of positive nodes of arity k. A *leaf* has arity $n = 0$ and an *internal node* arity $n > 0$. Hence, in particular, $N_0(T)$ is the number of negative leaves. We will omit to explicitly mention T when not ambiguous.

We call *branching node* a node of arity $n \geqslant 2$.

The following lemma relates a property on the total number of positive and negative nodes with a property on the number of nodes of the same polarity.

Lemma 2. *Let T be a polarized tree.*
If the root is negative, the two following conditions are equivalent:

$$\text{(i) } N(T) \geqslant P(T) \quad \text{(ii) } N_0 \geqslant N_2 + 2N_3 + \cdots + (k-1)N_k + \cdots$$

If the root is positive, the two following conditions are equivalent:

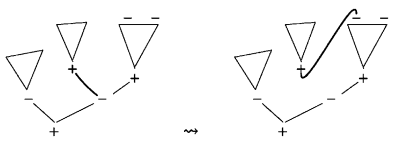
$$\text{(i}') \ P(T) \geqslant N(T) \quad \text{(ii}') \ P_0 \geqslant P_2 + 2P_3 + \cdots + (k-1)P_k + \cdots$$

Proof. Easy counting. Let us assume the root is negative. How many positive nodes are there in T? Each positive node is above a negative one, and each n-ary negative node introduces n positive nodes. Hence $P = \sum_k kN_k$. On the other side, $N = \sum_k N_k$. Thus (i) is equivalent to:

$$N_0 + N_1 + N_2 + N_3 + \cdots + N_n \geqslant 0N_0 + N_1 + 2N_2 + 3N_3 + \cdots + nN_n. \qquad \square$$

3.2. From polarized prefix trees to slices

A polarized prefix tree is "almost" a slice, except for the fact that it branches also on negative addresses. We have already seen that not any prefix tree can be associated to a slice. However, under certain conditions (condition (*), Proposition 5), we can produce a "sequentialization"of a prefix tree into a slice. We proceed as follows: for each n-ary negative node with $n \geqslant 2$, we prune the exceeding $n-1$ positive subtrees, and *graft* them on top of a negative leaf (see Fig. 3.2 below). Condition $(*)$ in Proposition 5 ultimately guarantees that there are "enough leaves" to do so. The crucial point is that the operations we perform preserve the sub-address condition, which is an invariant of any transformation we perform. Notice that the condition is satisfied by any prefix tree.



Definition 3 (*Sub-address condition*). A tree of addresses satisfies the *sub-address condition* if $\xi \sqsubset \xi'$ implies that $\xi \leqslant \xi'$ (therefore ξ' belongs to the subtree induced by ξ).

Let us define the following operations on trees:

- Given a tree T and a subtree T_0, $(T \setminus T_0)$ is the tree obtained from T by removing T_0.
- Given two trees T_1, T_2 and a leaf λ of T_1, $graft(T_1, \lambda, T_2)$ is the tree obtained from T_1 adding T_2 as subtree of λ.

Observe that: if T is polarized, then $(T \setminus T_0)$ is polarized; if T_1, T_2 are polarized and λ and $root(T_2)$ have opposite polarity, then $graft(T_1, \lambda, T_2)$ is polarized.

Lemma 4. *Let T be a polarized tree of addresses.*
(i) *If T satisfies the sub-address condition so does $(T \setminus T_0)$.*
(ii) *Assume λ is a leaf of T with same polarity as T, and T_i is an immediate subtree of T not containing λ. Then if T satisfies the sub-address condition so does $graft((T \setminus T_i), \lambda, T_i)$.*

Proposition 5. *Let T be a polarized prefix tree. If T satisfies the condition*

$(*)$ *$N(T') \geqslant P(T')$ for any negative subtree T'*

then there is a slice $\mathfrak{S}$ with the same polarity as T, such that $\mathbf{T}(\mathfrak{S}) = T$.

That is, T is the prefix tree of $\mathfrak{S}$.

Proof. We define an application S mapping polarized trees of addresses which satisfy $(*)$ into polarized trees. We show that $S(T)$ satisfies the following conditions:

1. $S(T)$ has the same polarity as T and the same set of addresses;
2. if T satisfies the sub-address condition then so does $S(T)$;
3. $S(T)$ satisfies $(*)$;
4. $S(T)$ branches only on positive nodes.

As a result, given a prefix tree T which satisfies $(*)$, $S(T)$ is a slice whose prefix tree is T.

To define S we proceed by induction on the size of the tree T:

If T is a single node, let us set $S(T) = T$.

Otherwise let us call $T_1, \ldots, T_n$ its immediate subtrees. It is obvious that if T satisfies $(*)$ and the sub-address condition, so does each of the subtrees. Let T' be the tree with the same root as T and immediate subtrees $S(T_1), \ldots, S(T_n)$.

We distinguish two cases:

1. If T is positive, let $S(T) = T'$. By induction, $S(T)$ satisfies all properties.
2. If T is negative, then T' satisfies conditions (1), (2), and (3). We can transform such a tree into a tree $\Phi(T')$ which satisfies also condition (4). Let us proceed by induction on the arity k of T.

If $k = 1$ then T' satisfies also (4), and $\Phi(T') = T'$.

Otherwise, let us transform T' into a tree T'' which has arity $k - 1$ and satisfies (1), (2), (3). Since the root of T' has arity $\geqslant 2$, condition (3) and Lemma 2 guarantee that there is at least a negative leaf. Let λ be such a leaf, and T'_i an immediate subtree of T' not containing λ. Define $U = (T' \setminus T'_i)$ and $T'' = \mathit{graft}(U, \lambda, T'_i)$. □

3.3. Observability conditions: parity and leaves

We are ready to give two (equivalent) conditions which guarantee that a slice can be visited with a single test (in a single run of normalization). We can already show that they are sufficient to guarantee observability. The fact that they are also necessary will follow from Proposition 10 in the next Section.

Parity: Let $\mathfrak{S}$ be a slice and $\mathbf{T}(\mathfrak{S})$ its prefix tree. $\mathfrak{S}$ satisfies the *parity condition* if in all the *positive* subtrees T' of $\mathbf{T}(\mathfrak{S})$, $P(T') \geqslant N(T')$.
It is immediate that $P - N \leqslant 1$.

We can reformulate the previous condition to consider only the positive nodes. It is enough to look at positive leaves and branching nodes.

Leaves: Let $\mathfrak{S}$ be a slice and $\mathbf{T}(\mathfrak{S})$ the corresponding prefix tree. $\mathfrak{S}$ satisfies the *leaves condition* if all the positive subtrees T' of $\mathbf{T}(\mathfrak{S})$ satisfy the following expression: $P_0(T') \geqslant \sum (i-1)P_i(T')$.

Remark 6. Notice that the arity of a positive node is the same in a slice $\mathfrak{S}$ and in the corresponding prefix tree $\mathbf{T}(\mathfrak{S})$. In particular, a positive leaf of $\mathfrak{S}$ is a leaf in $\mathbf{T}(\mathfrak{S})$.

It is immediate that the two conditions above (Parity and Leaves) are equivalent. They guarantee that the addresses composing the slice can be rearranged in a counter-design. We call them *observability conditions*.

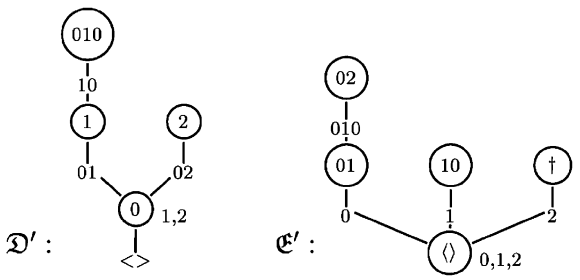
Lemma 7 (*Girard*). *Given two slices $\mathfrak{S}$ and $\mathfrak{T}$ whose sets of actions are dual, then the normalization uses all the actions* (*Section* 3.2 *in* [13]).

Proposition 8. *Let $\mathfrak{S}$ be a slice. If $\mathfrak{S}$ satisfies the observability conditions then $\mathfrak{S}$ admits a complete visit by normalization. That is, there is a counter-design $\mathfrak{E}$, such that $[\mathfrak{S} \rightleftharpoons \mathfrak{E}]$ uses all the actions of $\mathfrak{S}$.*

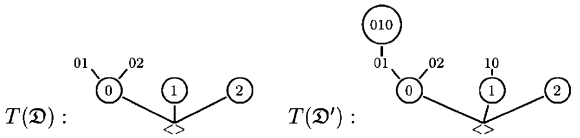
Proof. By Proposition 5, the parity condition ensures that we can exhibit a slice $\mathfrak{T}$ with the same actions as $\mathfrak{S}$ but opposite polarity. By Lemma 7, this implies that the normalization between $\mathfrak{S}$ and $\mathfrak{T}$ terminates, and uses all of the actions. □

3.4. Example 1 (continuation)

Let us examine again our counter-example 1. In our example there was not "enough space" to move from one branch to the other. However, it is enough to slightly expand the design $\mathfrak{D}$ into the new design $\mathfrak{D}'$ to be able to explore it in a single run. In fact, we can visit $\mathfrak{D}'$ normalizing it against $\mathfrak{E}'$ below.



It is easy to check our conditions against the polarized prefix trees $\mathbf{T}(\mathfrak{D})$ and $\mathbf{T}(\mathfrak{D}')$, respectively, associated to the design of Example 1 and to $\mathfrak{D}'$:



4. Dynamic approach

4.1. Normalization paths

Let us study the geometrical properties of a path generated by normalization. Proposition 10, which is the key result of the paper, establishes a relation between the sequential order and the prefix order.

Notations: Given two addresses α, β, we indicate by $\alpha \sqcap \beta$ their longest common prefix. In the same way, since a chronicle is a sequence of addresses, given two chronicles we can consider their longest common prefix. Given two actions κ_1, κ_2 in a slice $\mathfrak{S}$, we indicate by $\kappa_1 \bigwedge_{\mathfrak{S}} \kappa_2$ the last action of the longest common prefix of their chronicles (we recall that the chronicle of an action κ is the path from the root to κ).

It is convenient to fix a notation for the *subtrees*: if ξ occurs as a node in the tree T (resp. in the slice $\mathfrak{S}$), then we indicate by T_ξ (resp. $\mathfrak{S}_\xi$) the subtree induced by T (resp. by $\mathfrak{S}$) above ξ.

It is immediate by the branching condition that given a slice $\mathfrak{S}$ and two actions κ_1, κ_2 which are sequentially incomparable, we have that $\xi = \kappa_1 \bigwedge_{\mathfrak{S}} \kappa_2$ is a positive node. Therefore, there exist i, j such that $\kappa_1 \in \mathfrak{S}_{\xi i}$, $\kappa_2 \in \mathfrak{S}_{\xi j}, i \neq j$.

Lemma 9. (i) *Given a slice $\mathfrak{S}$ and two actions having addresses σ, τ such that $\sigma < \tau$ (they belong to the same chronicle), either $\sigma \sqsubseteq \tau$ or $\sigma \sqcap \tau$ is a negative address.*

(ii) *Assume that σ, τ are sequentially incomparable in $\mathfrak{S}$ (that is neither $\sigma < \tau$ nor $\tau < \sigma$) If $\sigma \sqcap \tau$ is positive, then $\sigma \bigwedge_{\mathfrak{S}} \tau = \sigma \sqcap \tau$.*

Proof. (i) Suppose σ, τ incomparable. Let $\sigma = \eta i \sigma'$ and $\tau = \eta j \tau'$, where $i \neq j$. If η is positive, then ηi and ηj belong to two distinct chronicles. The fact that $\eta i < \sigma$, while $\eta j < \tau$ is thus against the hypothesis that σ, τ belong to the same chronicle.

(ii) Let $\sigma \bigwedge_{\mathfrak{S}} \tau = \xi$, and let $\sigma = \eta i \sigma'$ and $\tau = \eta j \tau'$, where $i \neq j$. Therefore, $\eta = \sigma \sqcap \tau$ is either ξ, or it has been used as focus before ξ. In such a case, either $\eta i < \xi$ or $\eta j < \xi$. We can suppose $\eta i < \xi$. But $\eta j < \tau$, thus we cannot have $\xi < \tau$ as we are assuming. □

Consider the path induced by the closed normalization of two slices $\mathfrak{S}, \mathfrak{T}$. The sequence of actions $p = [\mathfrak{S} \rightleftharpoons \mathfrak{T}]$ induces a path on each slice.

Let us consider a positive action ξ in the slice $\mathfrak{S}$. Each of the negative subaddress ξi which immediately follow ξ induces a subtree $\mathfrak{S}_{\xi_i}$. In general, the normalization path does not traverse the slice completing each subtree before entering another subtree (as it would in a preorder traversal). When moving from $\mathfrak{S}_{\xi_i}$ to $\mathfrak{S}_{\xi_j}$ the path will exit $\mathfrak{S}_{\xi_i}$ after a positive action α, then possibly move around outside $\mathfrak{S}_\xi$, and then enter $\mathfrak{S}_{\xi_j}$ on a negative action β.

Proposition 10 will show that such a path must leave the subtree $\mathfrak{S}_{\xi i}$ and enter the subtree $\mathfrak{S}_{\xi j}$ on a sub-address of ξ.

Before going into it, let us give a concrete example in order to fix ideas. In Fig. 5 the supscripts on the nodes of $\mathfrak{S}$ indicate the order of visit induced by normalization against the slice $\mathfrak{T}$ consisting only of linearly ordered actions:

The path p induced by $\mathfrak{S}$ and $\mathfrak{T}$ is exactly: $\langle \langle \rangle, 1, 10, 101, 1010, 2, 20, 102 \rangle$. On $\mathfrak{S}$, the path p exits the subtree $\mathfrak{S}_{10}$ (the subtree induced by the node 10) in 1010, continues outside $\mathfrak{S}_{10}$ and then enters it again on the node 102.

Given a path $p = p_1\ \alpha\ p_2\ \beta\ p_3$ we write $\alpha \ll_p \beta$; we indicate p_2 as $]\alpha, \beta[$. We indicate by $d(\alpha, \beta)$ the number of actions between α and β in p. Such a number is necessarily even if α is positive and β is negative as in the sequel.

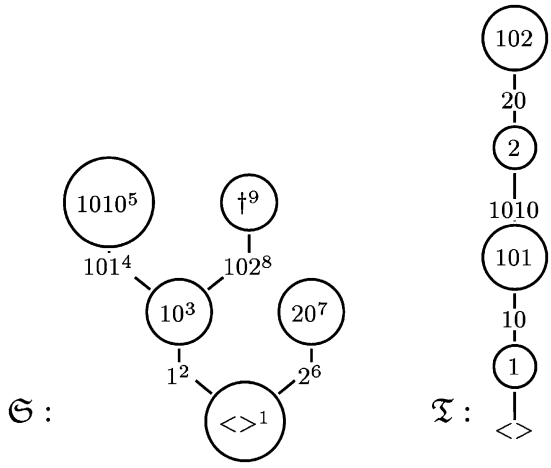


Fig. 5.

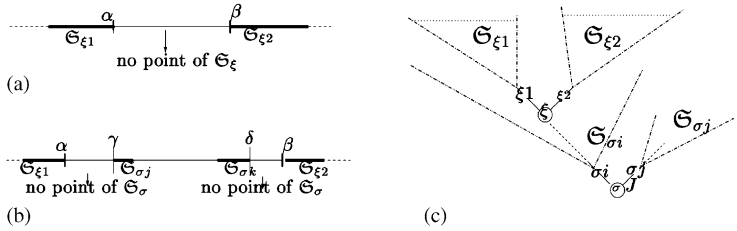


Fig. 6.

Proposition 10. *Let $p = [\mathfrak{S} \rightleftharpoons \mathfrak{E}]$, where $\mathfrak{S}, \mathfrak{E}$ are slices, and let α, β be any two actions such that $\alpha \ll_p \beta$. Assume that in one of the two slices (let us indicate it by $\mathfrak{T}$)* (i) *α, β belong to distinct chronicles of $\mathfrak{T}$, and* (ii) *no action κ such that $\alpha \ll_p \kappa \ll_p \beta$ belongs to the subtree induced by $\alpha \bigwedge_{\mathfrak{T}} \beta$. Then $\alpha \sqcap \beta = \alpha \bigwedge_{\mathfrak{T}} \beta$.*

Condition (ii) means that the path p *exits* that subtree $\mathfrak{T}_{\alpha \bigwedge_{\mathfrak{T}} \beta}$ in α and *enters* in β. Notice that only one of the two slices $\mathfrak{S}, \mathfrak{E}$ can satisfy the conditions, because α must be positive and β negative.

Proof. It is enough to show that $\alpha \sqcap \beta$ is always positive in the slice $\mathfrak{T}$ in which we calculate $\alpha \bigwedge_{\mathfrak{T}} \beta$. Then $\alpha \sqcap \beta = \alpha \bigwedge_{\mathfrak{T}} \beta$ by Lemma 9. The proof is by induction on $d(\alpha, \beta)$.

Let $d(\alpha, \beta) = 0$. In the slice $\mathfrak{T}$ (which is either $\mathfrak{S}$ or $\mathfrak{E}$), when p moves from α to β it changes branch. This means that in the counter-design we must have a chronicle $c\alpha^- \beta^+$. By Lemma 9, $\alpha \sqcap \beta$ is negative in the counter-design, and then positive in $\mathfrak{T}$, where we calculate $\alpha \bigwedge_{\mathfrak{T}} \beta$.

Let $d(\alpha, \beta) = n \geqslant 2$. Assume that $\mathfrak{T}$ is $\mathfrak{S}$, and let $\xi = \alpha \bigwedge_{\mathfrak{S}} \beta$. Condition (ii) writes as : $]\alpha, \beta[\cap \mathfrak{S}_\xi = \emptyset$. This situation is illustrated in Fig. 6(a).

Notice that for any action $\kappa \in]\alpha, \beta[$: (i) $\kappa \notin \mathfrak{S}_\xi$, by hypothesis, and (ii) $\kappa \not< \xi$. In fact all the actions below ξ in $\mathfrak{S}$ are visited before ξ, which in turn is visited before α (because $\tau < \sigma$ implies $\tau \ll_p \sigma$); (iii) τ belongs (at least) to one of the subtrees induced by the nodes $\sigma_i < \xi$ in $\mathfrak{S}$ (the actions between the root of $\mathfrak{S}$ and ξ).

Let us consider the positive actions $\sigma_i < \xi$ in $\mathfrak{S}$ (those actions below ξ in $\mathfrak{S}$). Let σ be the maximal node such that $\sigma < \xi$ in $\mathfrak{S}$ and $\mathfrak{S}_\sigma \cap]\alpha, \beta[\neq \emptyset$. Fig. 6(c) illustrates this. Notice that: (i) $\mathfrak{S}_\xi \subseteq \mathfrak{S}_{\sigma i}$, for some σi sub-address of σ, and (ii) $\mathfrak{S}_{\sigma i} \cap]\alpha, \beta[= \emptyset$, by maximality of σ. Let us call γ the first point of $]\alpha, \beta[$ that belongs to $\mathfrak{S}_\sigma$, and let δ be the last point of $\mathfrak{S}_\sigma$ appearing in $]\alpha, \beta[$. This situation is illustrated in Fig. 6(b).

Since $\alpha \in \mathfrak{S}_{\sigma i}$, $\gamma \in \mathfrak{S}_{\sigma j}$, and $\mathfrak{S}_\sigma \cap]\alpha, \gamma[= \emptyset$, we can apply the inductive hypothesis, obtaining that $\alpha = \sigma i *$, $\gamma = \sigma j *$. In a similar way, we have that $\delta = \sigma k *$ and $\beta = \sigma i *$. We know then that $\sigma i \sqsubseteq \alpha \sqcap \beta$. If we now assume that $\alpha \sqcap \beta$ is

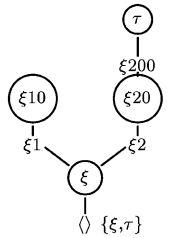
negative in $\mathfrak{S}$, then in $\mathfrak{E}$ $\alpha \sqcap \beta$ is positive and, by the proof of Lemma 9(i), α, β belong to distinct subtrees of $\alpha \sqcap \beta$. This means in particular that inside $[\alpha, \beta]$ the normalization path moves from one subtree to the other. By the inductive hypothesis, the exit and the enter points should be sub-addresses of $\alpha \sqcap \beta$ and then of σi. But, $\mathfrak{S}_{\sigma i} \cap]\alpha, \beta[= \emptyset$, giving a contradiction. So $\alpha \sqcap \beta$ is positive in $\mathfrak{S}$ and we are done by Lemma 9(ii). □

4.2. Return condition

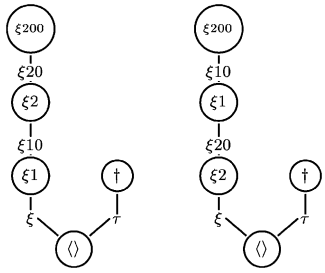
Proposition 10 allows us to write a third condition. This is more technical, but we will be able to show that it is necessary to observability. The condition really says that we can visit the slice in preorder, persistently completing the visit of a subtree before starting visiting a new one. Conversely, if a slice admits a visit, it admits a visit in preorder.

Let us give an example of such a visit on a slice.

Example 11 (*Preorder traversal of a slice*). Let us consider the following design $\mathfrak{D}$:



There are two possible ways to visit $\mathfrak{D}$, each corresponding to the normalization against one of the following counter-designs $\mathfrak{E}_1$, $\mathfrak{E}_2$.



Normalizing against $\mathfrak{E}_1$ gives the path $\langle \langle\rangle, \xi, \xi 1, \xi 10, \xi 2, \xi 20, \xi 200, \tau \rangle$, that is a visit in preorder, while the visit resulting from the normalization against $\mathfrak{E}_2$ is not.

Preorder traversal: We start with a few remarks on the preorder traversal of a tree. A *preorder traversal* of an n-ary tree T first visits the root and then visits in preorder each of the subtrees. Consider an n-ary tree T with $n \neq 0$; we indicate by $T_1, \ldots T_n$ its *immediate* subtrees. For each T_i, the last visited node σ is necessarily a leaf. The visit of T either terminates on that leaf σ, or it continues starting the visit of another subtree T_j. If this is the case, we call σ a *point of return* for the visit of T, and for the node ξ in which T is rooted. The same inductively applies to the subtrees.

Each leaf but the last one visited is necessarily a point of return for exactly one internal node.

Therefore, we have the following:

Fact 12. *Any visit in preorder of an n-ary tree T induces a* (*partial*) *function G from the leaves to the internal nodes, defined by*

$\xi = G(\sigma)$ *if* σ *is a point of return for the tree rooted in* ξ.

The visit satisfies the following property:

(**) " each internal node has exactly one immediate subtree that does not contain a point of return for that node."

Observe that: (i) *the function G is defined on all the leaves but one, and* (ii) $G^{-1}(\xi)$, *where* ξ *varies on the branching nodes, induces a partition of the leaves.*

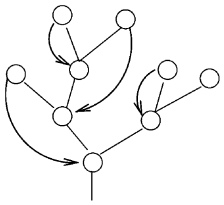
Conversely, given a tree we can consider a function G from the leaves into the internal nodes which satisfies the property $(**)$ above. Necessarily, all points of return of a node ξ belong to distinct subtrees of ξ. For each node ξ of the tree, we can choose an order between its points of return; this will then induce a visit in preorder of the tree.

Return: A finite slice $\mathfrak{S}$ satisfies the *return condition* if we can define a partial function G from the positive leaves to the internal nodes, which satisfies the following two properties.

(i) $G(\sigma)$ is a prefix of the leaf σ;
(ii) each ξ internal node has exactly one immediate subtree which does not contain a leaf σ s.t. $G(\sigma) = \xi$.

We call a leaf σ s.t. $\xi = G(\sigma)$ a *point of return* for ξ. Observe that the point of return of ξ in the sub-tree $\mathfrak{S}_{\xi i}$ is forced to be a sub-address of ξi.

Remark 13. In practice, to define G, for each branching node we chose a point of return in each of its sub-trees but one, as illustrates the following picture.



Using Proposition 10 we are able to show that the return condition is a necessary condition for observability.

Proposition 14. *If a slice admits a visit by normalization, then it satisfies the return condition.*

Proof. Let $\mathfrak{S}$ be a slice, and ξ a positive node of arity $n \geqslant 2$. Each of the n subtrees has as root a subaddress ξi; we indicate the subtree by $\mathfrak{S}_{\xi i}$. Since $\mathfrak{S}$ admits a visit, the normalization path must complete the visit of all the subtrees $\mathfrak{S}_{\xi i}$. We can order them accordingly to the order in which their visit is completed. Let $\mathfrak{S}_{\xi j}$ be any of the first $n-1$ subtrees, and α be its last visited action, which must be a leaf. Since there is at least one subtree of ξ whose visit is still to be completed, we are sure that after α the normalization path will enter $\mathfrak{S}_{\xi}$ again. By Proposition 10, α is a sub-address of ξ. Hence we can choose it as the point of return for ξ in the subtree $\mathfrak{S}_{\xi j}$. □

The return condition is also a sufficient condition. It really means that we can visit the slice in preorder, *for a suitable ordering of the subtrees*, persistently completing the visit of a subtree before starting a new one.

Proposition 15. *If a slice $\mathfrak{S}$ satisfies the return condition, any ordering of the points of return induces a preorder traversal of $\mathfrak{S}$ that can be realized by normalization of $\mathfrak{S}$ with a counter-design.*

Proof. The argument is similar to that used to prove the separation theorem in [13]. We can always produce a counter-design which allows normalization to go "upwards" in a chronicle. The delicate point is changing of branch.

Given the slice $\mathfrak{S}$, let p be a traversal obtained by ordering the points of return. We want to produce a counter-design $\mathfrak{T}$ such that $[\mathfrak{S} \rightleftharpoons \mathfrak{T}] = p$. The construction is by induction on the length of $q \sqsubseteq p$. We show that there is a counter-design $\mathfrak{T}_q$ which realizes q. The delicate point is when q moves from a positive action κ_1 to a negative action κ_2 in $\mathfrak{S}$, because this means that in $\mathfrak{T}$ we have a positive action κ_2 which appears immediately after the negative action κ_1. We need to check that $\mathfrak{T}_q$ satisfies the subaddress condition on positive focus. There are two cases to consider:

1. When going upwards in a chronicle, the condition is satisfied because if the focus of κ_1 is of the form ξ then κ_2 is of the form ξi.
2. When we change the branch we are in the situation described by Fig. 7, where $\kappa_1 = \xi i*$, $G(\kappa_1) = \xi$ and $\kappa_2 = \xi j$. (Notice that κ_1 is a positive leaf, $G(\kappa_1)$ is positive (being a branching node) and κ_2 is negative.)

To conclude, at the orthogonal we need to have ξj just after $\xi i*$. This guarantees that the subaddress condition is satisfied on positive focuses, because the parent address of ξj is ξ, which by induction is below $\xi i*$. □

We now can go back to the conditions we already established, to show that they are all equivalent.

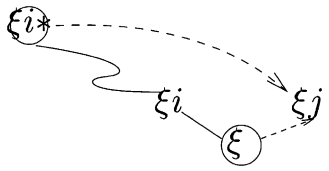


Fig. 7.

Lemma 16. Return *condition* ⇒ Leaves *condition*.

Proof. Since $G(\sigma)$ is a prefix of σ, σ is above $G(\sigma)$ in $\mathbf{T}(\mathfrak{S})$. The conditions on the function G ensure that in any subtree of $\mathbf{T}(\mathfrak{S})$, at each n-ary node ξ correspond $n-1$ leaves, which are the inverse image of ξ. □

5. Results

Putting it all together, we have shown that

Proposition 17. *The three conditions of* parity, leaves and return *are equivalent and characterize observability*.

In fact, $\mathfrak{S}$ admits a visit by normalization ⇒ *Return* ⇒ *Leaves* ⇒*Parity* ⇒ $\mathfrak{S}$ admits a visit by normalization.

Moreover, a not so intuitive outcome is that if it is possible to visit a slice, then it is always possible to visit it in preorder.

We also have enough information to characterize the prefix trees to which we can associate two slices $\mathfrak{S}, \mathfrak{T}$, such that $\mathfrak{S} \perp \mathfrak{T}$.

Proposition 18. *To a prefix tree T we can associate both a positive slice* $\mathfrak{S}$ *and a negative slice* $\mathfrak{E}$ *s.t.* $|\mathfrak{S}| = |\mathfrak{E}| = T$ *iff, as soon as we fix a polarization*:

(i) *in all the negative subtrees* T', $N(T') \geqslant P(T')$, *and*
(ii) *in all the positive subtrees* T'', $P(T') \geqslant N(T')$.

We can check that the conditions hold *for all the subtrees* with a single *postorder* traversal of T.

It is immediate that for all subtrees T' we have $|N(T') - P(T')| \leqslant 1$.

6. Conclusions and perspectives

Ideally, in Ludics we should be able to determine, test and express interactively all properties we require of designs. For this reason it is important to know what can be observed at each single test. The designs that can be visited in a single run of normalization represent the *primitive units of observability.* To establish our results we have developed combinatorial methods which we think may be of independent interest as tools to study the theory of Ludics from an operational point of view.

We believe that an important contribution of our work is to put forward the role of the two orders on a slice (sequential and spatial order), and to evidence how the interplay between these two orders forces structure and remarkable properties. Normalization ultimately relies on such properties. We expect these properties to be an important guide if we want to generalize the structures.

One reason for wanting to do so is for considering concurrency. The use of names, the interactive methods, the notion of equivalence from the point of view of the observer which is internal to the system: all these aspects seem to call for going in this direction. Even though Ludics, as introduced in [13], accounts for sequential computation, the sequentiality assumptions are not essential to the theory, and it seems natural to extend it to a concurrent setting. There

has been progress on this issue, based on the definition of a more parallel syntax, as reported in [9], and the geometrical properties we have highlighted here have been a base and a guide.

While focusing on the sequential order evidences the connection with sequent calculus, putting in the foreground the prefix order makes the proof-net style presentation stand out [6,7]. Developing further this direction towards a parallel syntax such as the original one of proof-nets appears as an interesting direction, which we are currently exploring. In the same way as one can see a (multiplicative) proof-net as a formula tree plus axiom links connecting some of the leaves, a Ludics proof-net is a prefix tree with just enough sequentiality to recover the axioms. From this proof-nets perspective, our observability conditions then appear as *an abstract way to speak about sequentialization*.

In the background of our study there is *a notion of slice which abstracts over sequentiality details.* This also can be seen as a step towards a more asynchronous notion of interaction, following an orthogonal (static rather than dynamic) direction. It seems possible, by exploiting our characterization, to define a new notion of *Slice*, as a prefix tree whose actions could be sequentialized in both a design and a counter-design. It would then be possible to follow an idea that was present in unpublished work by Girard, and say that two designs are orthogonal if they have such a Slice in common: $\mathfrak{D} \perp \mathfrak{E} \iff \exists \mathfrak{S} \in \mathfrak{D} \cap \mathfrak{E}$. Recent developments by Pierre Boudes relating denotational semantics to games through Ludics go in a similar direction, and take further the static (or "asynchronous") approach to slices [2].

The work we have presented here does not take types into account, i.e. does not deal with the high-level architecture of Ludics. Note however that designs can be handled in two ways: (i) *untyped*, i.e. as themselves, pure, or (ii) *typed*, that is as part of a set of designs with all the properties to be a type (behaviour). In this paper we provided a characterization of what can be observed interactively, for designs as untyped objects. When we move to the typed setting, the situation is more delicate and demands further study (see [6]). In Appendix A.2 we give an overview of some of the questions related to this issue.

Let us conclude by putting the question we have addressed in this paper in a more general perspective. A way to reformulate it is: "given a piece of code P, and an environment in which it is executed, how many lines of P are actually run or visited during its execution?" It would be intriguing to study possible relations with static analysis or dead code elimination.

Acknowledgments

I wish to thank Jean-Yves Girard for his insightful advice and Pierre-Louis Curien for his suggestions and stimulating comments. I am also grateful to the referees, whose comments have been of great help to improve the paper.

Appendix A.

In the appendix we discuss some more technical issues related to Ludics. While we tried to make this paper self-contained, the following discussion involves the full high-level architecture of Ludics, and assumes some familiarity with the theory in [13].

A.1. Weakening and interactive observability (exact vs. parsimonious)

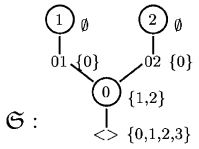
Let us discuss the consequences of our Example 1. The fact that the two branches (the two slices) in $\mathfrak{S}$ behave as independent components affects what can be interactively recognized. An example is that we cannot interactively detect the use of weakening in a slice (see the discussion in [13], p. 392).

In Section 1 we have provided an intuition of the relation between sequent calculus and designs. An address (a formula) is "used" when we use a rule that decomposes it (appearing as an action on ξ). The decomposition of ξ by the action (ξ, I) produces the sub-addresses (subformulas) ξi. We have explained that we can find out that a formula σ is in the context of a certain formula (or of a certain address) because σ is used afterwards. An address which is produced but never used can be seen as weakened. To detect the use of weakening becomes relevant when seeking for full completeness w.r.t. the calculus *MALL*2 (second order MALL). A proof of a formula A will correspond to a design in the behaviour associated to $\mathbf{A}$. Among all the designs in $\mathbf{A}$, we want to distinguish some special ones, corresponding to real proofs. They are those which satisfy certain conditions, called *winning conditions.* The first two are quite intuitive

(given the goal): no use of daimon, and no use of weakening. The third one, *uniformity*, is more technical. Ideally, all winning conditions should be detected interactively. However, (and surprisingly) this is not possible in the case of weakening.

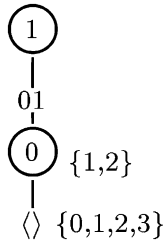
A design where all addresses are used (that is, there is no weakening) is called *exact*. The interactive formulation of this property, known as "parsimony" is expressed by: "The design $\mathfrak{D}$ in the behaviour $\mathbf{G}$ is *parsimonious* when for all $\mathfrak{E} \in \mathbf{G}^{\perp}$ the slice which has been consumed during normalization is exact." These two notions do not coincide, as is easily seen.

Consider again Example 1, assuming that the root of the slice $\mathfrak{S}$ is the action $(\langle\rangle, \{0, 1, 2, 3\})$.



The root creates four addresses, but the address 3 is never used. However, we cannot interactively detect that 3 is never used, and hence weakened. Either we explore the left branch, or the right one. In the first case we see that 1 is used. The other addresses, 2 and 3, are possibly used after 02. In the second case we see that 2 is used, 1 and 3 being possibly used after 01.

It is interesting to take a look at the sequent calculus counterpart of what we see at each test. Consider one of the two possible tests of our example. What we observe is the following slice:



It translates into the following derivation:

$$\dfrac{\dfrac{\dfrac{}{\vdash 01.0,\ 1}\ (1, \emptyset)}{01 \vdash 1}\ (01, \{0\}) \qquad \dfrac{}{02 \vdash}\ (02, \emptyset)}{\dfrac{\vdash 0,\ 1,\ 2,\ 3}{<> \vdash}\ (<>, \{0, 1, 2, 3\})}\ (0, \{1, 2\})$$

Notice the premise $(02, \emptyset)$ which actually corresponds to a sub-design still to be developed. For what we know, we can reasonably assume that 2 and 3 will be used above it.

A.2. Moving to the typed setting

In this section we discuss the extension of our analysis of observability to the typed setting, i.e. designs as part of a behaviour. We recall that a behaviour $\mathbf{G}$ is a set of designs equal to its biorthogonal ($\mathbf{G} = \mathbf{G}^{\perp\perp}$). This issue is studied in [6], providing a partial, not fully satisfactory solution. In this section we want to give a feeling for the difficulties and problems related.

When we work within a behaviour, we test a design in $\mathbf{G}$ only using designs in the orthogonal $\mathbf{G}^{\perp}$. The question of interactive observability becomes much harder when working with behaviours, because the designs of a behaviour interact with each other to determine the orthogonal.

The properties we studied in this paper guarantee that given a slice $\mathfrak{S} \subseteq \mathfrak{D}$ we can produce a counter-design $\mathfrak{E}$ which visits $\mathfrak{S}$. If now $\mathfrak{D}$ belongs to a behaviour $\mathbf{G}$, even if an appropriate $\mathfrak{E}$ exists, we still need to ensure that $\mathfrak{E} \in \mathbf{G}^{\perp}$.

A key notion when working with types in Ludics is that of *material design*, which characterizes the really interesting designs. In fact, the operation of orthogonality produces lots of designs: all those which successfully normalize against

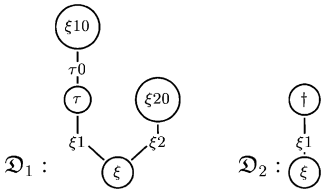
the given designs. Most of these designs are created, but there is no way to interact with them. For this reason, they do not play an essential role in generating the behaviour. The really interesting designs are the material ones. The operational way to characterize a *material* design $\mathfrak{D} \in \mathbf{G}$ is as a design of $\mathbf{G}$ whose actions are all interactively recognized via cuts with counter-designs in $\mathbf{G}^{\perp}$. Let us be careful! This does not mean that all actions are used in the same normalization with a single counter-design. In general a part of $\mathfrak{D}$ is accessed by a counter-design, another part by another counter-design. There is no need of a counter-design able to visit all the actions of $\mathfrak{D}$.

This fact is particularly important when we want to interactively establish properties of designs in a behaviour. As we have already seen, to observe a design as a whole, or to observe it by sort of "windows" is not quite the same. Even if our partial visits eventually cover the whole design, when we put these partial pieces of information together, the information is not necessarily complete.

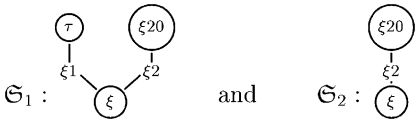
Let us call *strongly material* a slice $\mathfrak{S} \in \mathbf{G}$ which can be visited in a single normalization with a counter-design in $\mathbf{G}^{\perp}$.

The geometrical constraints we have studied still apply. To satisfy the observability conditions is still necessary for a slice to admit a visit. However, it is not sufficient, as shows the following simple example.

A simple example: Let us consider the behaviour $\mathbf{G} = \{\mathfrak{D}_1, \mathfrak{D}_2\}^{\perp\perp}$ generated by the two following designs $\mathfrak{D}_1$, $\mathfrak{D}_2$



$\mathfrak{D}_1$ includes both the following strongly material slices:



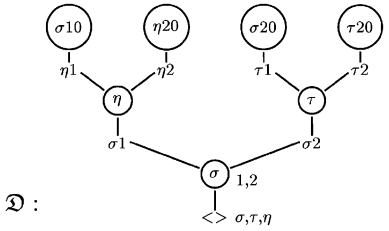
Observe that no counter-design in $\mathbf{G}^{\perp}$ will allow us to realize neither the visit $\langle \xi, \xi 2, \xi 20 \rangle$ nor the visit $\langle \xi, \xi 2, \xi 20, \xi 1, \tau \rangle$. Any counter-design containing the chronicle $\langle \xi —, \xi 2^{+} \rangle$ would fail against $\mathfrak{D}_2$. Therefore, any visit is forced to first enter the branch which starts with $\xi 1$, before accessing $\xi 2$. We can still visit all the actions of the two slices above, but only as part of the following visit: $\langle \xi, \xi 1, \tau, \tau 0, \xi 10, \xi 2, \xi 20 \rangle$.

A tempting and natural idea is to work with material designs. Unfortunately even a maximal slice of a material design, with all good properties for being observable is not necessarily so inside a behaviour.

Assume that the design $\mathfrak{D}$ is finite and material, and $\mathfrak{S} \subseteq \mathfrak{D}$ is a "maximal" such slice. Is it strongly material?

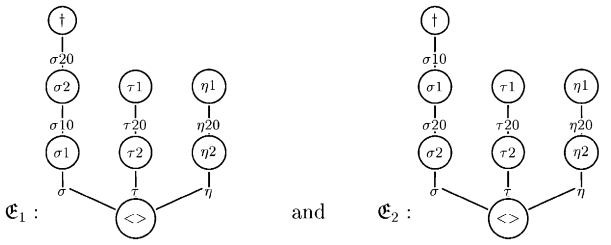
The answer is negative, and it is worthwhile seeing a counter-example, as it is not trivial to build one.

A subtle counter-example: Let us consider the following design $\mathfrak{D}$, which happens to be a slice itself:

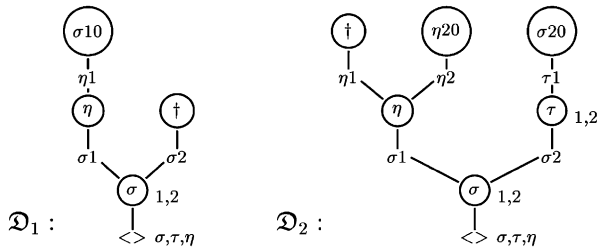


$\mathfrak{D}$ is a slice with all good properties of observability. Therefore, $\mathfrak{D}$ is strongly material in its principal behaviour $\mathbf{D} = \mathfrak{D}^{\perp\perp}$, as all designs which are orthogonal to $\mathfrak{D}$ belong to $\mathbf{D}^{\perp}$.

In fact, there are two ways to completely visit $\mathfrak{D}$, corresponding to the two following counter-designs $\mathfrak{E}_1$, $\mathfrak{E}_2$ in $\mathfrak{D}^{\perp}$:



Now, let us consider the behaviour $\mathbf{G}$ generated by $\mathfrak{D}$ together with the following two designs $\mathfrak{D}_1$, $\mathfrak{D}_2$:



As a design of $\mathbf{G} = \{\mathfrak{D}, \mathfrak{D}_1, \mathfrak{D}_2\}^{\perp\perp}$, $\mathfrak{D}$ is still material, but is not strongly material. There is no way to visit *all* of its action with a *single* design $\mathfrak{E} \in \mathbf{G}^{\perp}$.

Observe that the asymmetry of $\mathfrak{D}_1$, $\mathfrak{D}_2$ is essential for not losing materiality of the original design $\mathfrak{D}$. In general, one has to be very careful, because if a design is material in a behaviour $\mathbf{H}$, and we add other designs to produce the behaviour $\mathbf{H}'$, we add constraints on the way to visit $\mathfrak{D}$ (we have less tests at the orthogonal), and we easily loose the fact that $\mathfrak{D}$ is material.

References

[1] J.-M. Andreoli, R. Pareschi, Linear objects: logical processes with built-in inheritance, New Generation Comput. 9 (3–4) (1991) 445–473.
[2] P. Boudes, Unifying static and dynamic denotational semantics, Draft.
[3] P.-L. Curien, Abstract Böhm trees, Math. Structures in Comput. Sci. 8 (6) (1998).
[5] V. Danos, L. Regnier, Proof-nets and the Hilbert space, in: L. Regnier, J.-Y. Girard, Y. Lafont (Eds.), Advances in Linear Logic, London Mathematical Society Lecture Notes Series, Vol. 222, Cambridge University Press, Cambridge, 1995.
[6] C. Faggian, On the Dynamics of Ludics, Vol. 222, A Study of Interaction, Ph.D. Thesis, Université Aix-Marseille II, 2002.
[7] C. Faggian, Travelling on designs: ludics dynamics, in: CSL'02, Lecture Notes in Computer Science, Vol. 2471, Springer, Berlin, 2002.
[8] C. Faggian, M. Hyland, Designs, disputes and strategies, in: CSL'02, Lecture Notes in Computer Science, Vol. 2471, Springer, Berlin, 2002.
[9] C. Faggian, F. Maurel, Ludics on graphs, towards concurrency, draft (short presentation at LICS 2004).
[10] J.-Y. Girard, Linear logic, Theoret. Comput. Sci. 50 (1987) 1–102.
[13] J.-Y. Girard, Locus solum, Math. Structures in Comput. Sci. 11 (2001) 301–506.
[15] O. Laurent, Etude de la polarisation en logique, Thèse de doctorat, Université Aix-Marseille II, March 2002.

Further Reading

[4] P.-L. Curien, Introduction to linear logic and ludics, Adv. Math., to appear.
[11] J.-Y. Girard, On the meaning of logical rules i: syntax vs. semantics, in: Berger, Schwichtenberg (Eds.), Computational Logic, NATO Series F, Vol. 165, Springer, Berlin, 1999, pp. 215–272.
[12] J.-Y. Girard, On the meaning of logical rules ii: multiplicative/additive case, in: Foundation of Secure Computation, NATO Series F, Vol. 175, IOS Press, Amsterdam, 2000, pp. 183–212.
[14] J.-Y. Girard, From foundations to ludics, Bull. Symbolic Logic, 2003.