On the Dynamics of Ludics
A Study of Interaction
Acknowledgements

I am grateful to Jean-Yves Girard whose insights have been determining in the development of my work. His comments have always been inspiring and a source of new understanding and new ideas.

I thank Paul Ruet for his constant attention and the continous fruitful dialogue he offered me. His observations and support have been a precious help all along my PhD.

I greatly benefited from several discussions with Jean-Marc Andreoli; his questions and original point of view have been both helpful and stimulating.

I am indebted to Pierre-Louis Curien for the discussions we had together, his expert reading of the very first version of this thesis and numerous suggestions which led to a revision and (not only formal) reorganization of the material in this thesis.

I am grateful to Martin Hyland. The discussion with Martin go beyond the results in chapter 6. His understanding of the underlying structures has been extremely helpful and determining in allowing me to make some crucial progresses in the development of this work.

I wish to thank Christophe Fouqueré for his comments; the present version of this thesis incorporates his suggestions and remarks.

I am always greatful to Giovanni Sambin who with his enthusiasm introduced me to logic and proof theory.

I thank Marie-Renée Fleury-Donnadieu and Myriam Quatrini for the numerous discussions which have been extremely helpful in the understanding of Ludics and for the pleasure I had to collaborate with them.

I wish to thank Alexandra Bac, Olivier Laurent, Patrick Baillot, Pierre Boude, Sylvain Lippi, Virgile Mogbil, who have been fellow PhD students at IML. Permanent exchanges and discussions, reciprocal support have been an essential help and a stimulating experience.

Thank you also to Laurent Regnier, Thomas Ehrhard, Yves Lafont.
# Contents

1 Preliminaries ................................................. 7
   1.1 Sequent calculus presentation ................................ 8
   1.2 Designs .................................................. 13
      1.2.1 Getting an intuition .................................... 13
      1.2.2 The base .............................................. 16
      1.2.3 A tree of actions ...................................... 16
      1.2.4 A set of chronicles .................................... 18
   1.3 Normalization .............................................. 18
      1.3.1 Cut-nets .............................................. 19
      1.3.2 Orthogonality ......................................... 19
   1.4 Analytical theorems ........................................ 19
   1.5 Behaviours (types) ......................................... 21
      1.5.1 Material designs ....................................... 21
      1.5.2 Some special designs ................................... 21
      1.5.3 Directories, connected and prime behaviours .......... 22
   1.6 Connectives ............................................... 22
      1.6.1 Additives .............................................. 23
      1.6.2 Multiplicatives ....................................... 23
      1.6.3 Shift .................................................. 23
      1.6.4 Decomposition of behaviours .......................... 24

2 From Designs to Disputes: Normalization ................... 25
   2.1 Slices as proof-nets ....................................... 26
      2.1.1 Slices ............................................... 26
      2.1.2 Two orders: chronicles and prefix ..................... 27
   2.2 Loci Abstract Machine ..................................... 30
      2.2.1 \( \text{LAM}_0 \) ........................................... 30
      2.2.2 Balanced slices (Locus Solum) .......................... 35
   2.3 Disputes and chronicles extraction ......................... 36
      2.3.1 Extracting chronicles from a path ..................... 37
      2.3.2 Applications .......................................... 40
2.4 LAM_+: generalized version .................................. 40
2.5 Calculating the pull-back .................................... 41
  2.5.1 Paths and disputes (closure principle) ............... 42
2.6 Discussion .................................................. 43
  2.6.1 Related work ........................................... 43
  2.6.2 Focalization .......................................... 44

3 Visits of Designs and Interactive Observability ............ 45
  3.1 Visits and paths ........................................ 47
    3.1.1 Computing a counter-design that realizes a path 47
  3.2 What can be interactively observed? ..................... 48
    3.2.1 Consequences ..................................... 50
    3.2.2 Some comments .................................... 51
  3.3 Traveling on designs ................................... 51
  3.4 Prefix Tree .............................................. 56
    3.4.1 Polarized trees ................................... 58
    3.4.2 From prefix trees to slices ....................... 59
  3.5 Observability conditions ................................ 60
    3.5.1 Parity ............................................. 61
    3.5.2 Leaves ............................................. 62
    3.5.3 Return ............................................. 63
    3.5.4 Equivalence ...................................... 65
  3.6 Slices and prefix trees .................................. 67
  3.7 Discussion .............................................. 68
    3.7.1 So what? Consequences .......................... 68
    3.7.2 Realizable paths .................................. 69

4 Tests and Behaviours ........................................ 71
  4.1 Observability and behaviours ........................... 72
    4.1.1 Counter-example .................................. 73
  4.2 Tests and behaviours ................................... 75
    4.2.1 One more dog ..................................... 76
    4.2.2 Refining the analysis ........................... 77
    4.2.3 Special behaviours ................................ 78
    4.2.4 Back to strongly material designs ............... 80
  4.3 Properties of strongly material designs w.r.t. order 80
  4.4 Discussion of Counterexample 4.1.1 ..................... 82

5 Decomposition of behaviours ................................ 87
  5.1 Decomposition of behaviours ............................ 87
    5.1.1 Additive-multiplicative decomposition of behaviours 88
    5.1.2 G, G\perp fully concordant ....................... 89
5.1.3 \( \mathbb{G} \) admits MA decomposition \hspace{1cm} 93
5.2 Discussion \hspace{1cm} 93

6 From Disputes to Designs: Ludics and Games \hspace{1cm} 95
6.1 Arenas, players and legal positions \hspace{1cm} 96
6.1.1 Plays \hspace{1cm} 97
6.1.2 Extracting a design and a counter-design from a play \hspace{1cm} 99
6.2 Strategies \hspace{1cm} 101
6.2.1 Generating \( \mathcal{D} \) \hspace{1cm} 101
6.2.2 Strategies \hspace{1cm} 103
6.2.3 Innocent strategies: Views and Plays \hspace{1cm} 105
6.2.4 Designs and innocent strategies \hspace{1cm} 107
6.3 Issues on Linearity \hspace{1cm} 108
6.3.1 Extracting strategies from a play \hspace{1cm} 108
6.3.2 Propagation and Linear Plays \hspace{1cm} 108
6.4 Discussion \hspace{1cm} 111
6.4.1 Comments \hspace{1cm} 111
6.4.2 Further work: Behaviours and Games \hspace{1cm} 111

A An introduction to Uniformity and Completeness \hspace{1cm} 113
A.1 Why uniformity: examples \hspace{1cm} 114
A.1.1 Example: a matter of focalization \hspace{1cm} 115
A.1.2 Example: decomposing \( P \oplus Q \) \hspace{1cm} 116
A.2 From proofs to uniformity \hspace{1cm} 117
A.2.1 Back to the examples \hspace{1cm} 118
A.3 Uniformity and Fax \hspace{1cm} 119
Introduction

Interaction has become an important notion in two deeply connected research areas: proof theory and computer science.

Several developments in computer science converge to models based on a paradigm of computation as interaction. Let us give a few examples. A simple, early example is that of coroutines: a program A has as argument the output of another program B, therefore when it needs that argument, program A passes the control to program B, waiting for B response; when B has finished it passes the control back to A, and so on. A first example of program composition as a dialogue is [BC82]: sequential programs are interpreted by pairs of a function and a computation strategy, which specifies a schedule for the interaction of the function with its value. Interaction is also central in the models of concurrent computation, such as Milner's CCS or the π-calculus [Mil99]. There, programs and agents may be active simultaneously but have to synchronize at appropriate points of their execution. This may involve the transmission of values, one process being the sender and another the receiver.

The paradigm of computation as interaction is particularly significant today, because the typical situation is not anymore that a program is fed with some values and returns a result, and not even that a program interacts with only a single user. Today, we are faced with multiple agents, or users, interacting over a network and exchanging information. Interaction appears to be more visible and even more important than computation.

Logic also has undergone an evolution towards interactive and dynamical models. Two important examples are the Geometry of Interaction and Games Semantics. The Geometry of Interaction [Gir89], developed from Linear Logic [Gir87], interprets normalization (computation) as a flow of information circulating around a net. Games Semantics (into which found their way many insights from GoI) interprets computation as a dialog between two parties, the program (player) and the environment (opponent). Games (cf. [AM99] for a survey) have been an important evolution in logic, but also a successful approach to the semantics of programming language. The strength of these models is to capture the dynamical aspects of computation, so to take into account both qualitative (correctness) and quantitative (efficiency) aspects of the programming languages.

Ludics introduced by Girard in [Gir01b] is another step in this evolution, as it takes interaction as its foundation.

**Ludics**

The program of Ludics is to overcome the distinction between *syntax* (the formalism) and *semantics* (its interpretation). Rather than having two separate worlds, and an *external referee* relating the two, the aim is to have a unique universe, where properties are expressed and tested internally. Internally means interactively: the objects themselves test each other.

Interaction is the basis of the program. Ludics is therefore built on cut-elimination, which not only provides the dynamics (the engine) of the system, but actually represents the most foundational notion. Proceeding from cut-elimination, syntax and semantics meet in the notion of *design*. This is the fundamental artifact of Ludics. Designs are both an abstraction of a formal proof, and a concretion of its semantic interpretation. This has been achieved working from two directions:

- **Making semantics concrete.** The first task leads to *enlarge the universe of proofs*, in order to have enough inhabitants to be able to distinguish between them *inside* the system.

  The key point is that we want proofs and tests to be homogeneous in nature, and that we need to have enough tests. The solution is to complete the universe of proofs, in such a way that to any proof of $A$ we can oppose (via cut-elimination) a proof of $A^\perp$.

- **Abstracting from syntax.** The second task has been to abstract from proofs, in order to obtain an object which has the structure and the dynamic of a proof, but is free from commitments. In a proof we can separate two layers:

  1. "what is actually performed," the mathematical structure and
  2. useful comments on it, typically the name of the formulas.

One can think of the typed lambda calculus: the core notion is the lambda term, which can be obtained by erasing the type decoration. In a similar way, a design corresponds to the geometrical structure underlying a sequent calculus proof.

There are two crucial notions which are necessary to obtain this: *focalization* and *location*. We will discuss the former, which is an essential tool of proof-search, in the next chapter. The latter is a major novelty of Ludics, which takes seriously an other intuition from computer science: proofs (programs) do not manipulate the "idea" of a formula, but the address in the memory where it is stored, its location.

Ludics is distilled from a fine analysis of affine multiplicative-additive Linear Logic ($MALL^2$). The interactive approach allows ludics to generalize the notion of
"type". A type $G$ (called \textit{behaviour}) is a collection of programs that behave the same way in reaction to a set of tests, which are nothing else than other programs and are indicated by $G^\perp$, the orthogonal. What does really matter in a behaviour are the designs that can really be used, those which interact with the orthogonal. Such designs are called \textit{material}.

Beside the logical meaning (the types), the interest of designs lies in their computational content. Designs are concrete and very natural computational objects. They are close to variants of Böhm trees coming from Games Semantics, and in particular to Abstract Böhm Trees introduced by P.L. Curien ([Cur98]) as a generalization of lambda terms and as a concrete syntax for games.

In this thesis we study the dynamical aspects of Ludics, the interaction between designs. Our analysis will focus on the operational aspects and on the combinatorial properties, rather than on the logical ones. As we will emphasize, designs have remarkable properties as syntax. We will also make precise the connection with the notions of Games Semantics.

**Outline of the thesis**

As we have recalled, interaction is the foundation of the Ludics program. The aim for an interactive approach involves the “rules of the game” (no referees) but also the objects themselves, the way we access them. What one knows of a design is what can be observed testing it against a counter-design. For example, if two designs react in the same way to all the tests, we cannot distinguish between them (see the separation theorem that we recall in Section 1.4 and which is the analogous in ludics of the Böhm theorem). Therefore it is important to have a theory telling what can be recognized interactively at each test, and to have tools for reconstructing the objects from the traces of their interaction in different tests.

To study the interaction between designs, we introduce an abstract machine and develop a geometrical interpretation of normalization as a path of computation. This allows us for a concrete approach to designs, from which to carry on the study of interactive observability. Such a study corresponds in a sense to “reverse analysis”. We take care of two main issues: (i) which part of an agent can be tested by another agent, (ii) given a trace of interaction, rebuild the protagonists.

In our work we develop combinatorial methods for the study of designs; some of the techniques may be adapted to different problems.

The plan of the thesis is as follows.
Content of Chapters

Chapter 1 reviews some basic notions of Ludics ([Gir01b]). The bases for our work are then established in Chapter 2:

Chapter 2 From Designs to Disputes: Normalization

We define normalization on designs (dessins) in terms of an abstract machine, called Loci Abstract Machine (LAM). Normalization is presented by a token traveling along a cut-net.

As the token travels on the net, it draws a path: this sequence of actions represents the interaction among the designs of the cut-net. Slightly abusing the terminology in Locus, we call such a sequence a dispute; the part of the cut-net visited by the dispute during the normalization is the pull-back of that dispute.

Once it is clear how normalization works on designs and the notion of a normalization path is introduced, two questions arise.

1. Characterize what can be observed interactively. Can we characterize the (partial) designs that can be covered with a path? This means to characterize the designs used at each single run of normalization, or the designs that are the pull-back of a dispute (Chapter 3 and 4).

2. Study the interactions as primitive. Can we characterize the paths themselves, or rather the sequences of actions representing the interaction of two designs? This will build a bridge with Games Semantics (Chapter 6).

Chapter 3 Visits of Designs and Interactive Observability.

In this chapter we study the interaction among designs as pure (i.e. untyped) objects.

1. We establish some technical tools, by studying the operational aspects, in particular the possible forms of paths induced by normalization. This is much in the style of Geometry of Interactions.

2. We study what can be observed interactively. We remark, for example, that we cannot interactively detect the use of weakening. This lead us to characterize those designs that can be observed (visited) at each single run of normalization. Such designs can be thought of as primitive observable.

Chapter 4 Tests and Behaviours.

We relate the results of the previous chapter to the typed setting of behaviours. Suppose we have a design that is material (thus we know we can explore the whole of it) and with all the good properties to be observable. Unfortunately we are not
guaranteed that we can explore it with a single test. Even if this fact is surprising at first, we present a non trivial counter-example that demonstrates it.

The fact is that even if $\mathcal{D}$ as a pure design admits a test, when $\mathcal{D} \in \mathcal{G}$ that test must belong to $\mathcal{G}^\perp$. This leads us to study a more specific question: if we consider a design $\mathcal{D} \in \mathcal{G}$, under which conditions a counter-design $\mathcal{E} \perp \mathcal{D}$ (that implements a test) belongs to $\mathcal{G}^\perp$?

Chapter 5 Decomposition of behaviours.

The study of the main counter-example in Chapter 4 also leads us to study the decomposition of a behaviour.

We give a characterization, in the finite case, of those behaviours admitting additive-multiplicative decomposition. They correspond to the interpretation of constant-only MALL formulas.

Chapter 6 From Disputes to Designs: Ludics and Games.

We are interested in the description of a design as the collection of its possible interactions. This corresponds to an old idea in Ludics, explored by Girard in an earlier unpublished manuscript.

The approach we take allows us to make precise the correspondence with Game Semantics. This chapter can be read as a two-way dictionary between the basic notions of Ludics and Game Semantics: disputes and designs on one side, plays and strategies on the other side. To have a common language contributes to understand where are the similarities, and where the approaches differ.
Chapter 1

Preliminaries

In this Chapter we review some basic notions of Ludics [Gir01b]. Our presentation takes ideas and examples from [Gir01a] and [Cur]. We refer to those papers for a more complete introduction. The former emphasizes foundational issues, the latter the point of view of computer science.

We start with a review of the three main ingredients of ludics: generalized proofs, location and focalization.

We then describe the main artifact of Ludics, the designs, which are the object corresponding to proofs. A designs capture the geometrical structure of a sequent calculus derivation, and actually can be presented in a sequent calculus style. We do so in Section 1.1. The “true” designs are then introduced in Section 1.2.

Not to be confused by terminology, it is important to notice that by design we always intend the structure that in [Gir01b] is called dessein. When we refer to the sequent calculus counterpart (dessin), we make it explicit.

In Section 1.4 we resume the properties that designs satisfy.

In Section 1.5 we review the notion of type (behaviour) and some of the constructions (connectives).

The universe of proofs

In [Gir01b] the universe of proofs has been enlarged, through a generalization of the notion of proof based on the preservation of cut-elimination. To understand the idea, a proof must 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, up to completion. Most likely the process eventually abort, but we anyway did constructed a truncated proof. Such a truncated proof is a well defined formal object; in particular one can develop a proof-theory and normalize cuts involving such proofs. To do so
a new rules is introduced, the daimon:

$$\Gamma \vdash T \dagger$$

When using such a rule, we assume the sequent, without providing a justification.

Intuitively, a daimon is placed in a proof that we do not want to justify completely, or in a counter-proof (test) that we want to terminate. Girard call dog a proof whose only aim is to test a certain property, or impose a certain constraint (rule), and then aborts with daimon.

With daimon, any formula becomes provable. However, only formulas which are provable in the usual sense can be proved without the use of daimon: usual proof are still there, and appear as daimon-free proofs. The gain is that now the universe of proofs has been closed: to any proof of $A$ we can oppose a proof of $A^\perp$.

Locations

A major novelty of Ludics is that proof do not manipulate formulas, but their location, the address where they are stored (see Section 1.1).

Polarity and synthetic connectives (focalization)

Connectives and constants of Linear Logic separate into two families: positives ($\otimes, \oplus, 1, 0, !$) and negatives ($\forall, \&, \perp, T, ?$). This distinction has its full meaning in proof-search. Negative connectives are reversible, i.e. the rule to decompose them (bottom-up) is deterministic. Instead, on positive connectives there is a real choice (true non-determinism). However, positive connectives enjoy a dual property, called focalization, first observed by Andreoli ([AP91]): given a sequent of only positive formulas, there is at least a formula, the focus, that may be selected as active in the last rule and then entirely decomposed up to its first negative subformulas. 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 subformulas. To apply positive (negative) rules on the same focus in a threat can be seen as a single step.

The most important consequence, from the point of view of logic, is that a cluster of operations of the same polarity can be seen as a single connective, which is called a synthetic connective. A formula is then an alternation of positive and negative layers.

1.1 Sequent calculus presentation

A formula is positive (negative) if its outer-most connective is positive (negative).

In polarized logic, a positive connective can only be applied on positive formulas. Therefore if $P, Q, R$ are positive formulas, we cannot directly form $((P \forall P_2) \oplus Q^\perp) \otimes$
$R^\perp$; we need an operation which changes the polarity. In linear logic, this is done by the exponentials (! and ?). In fact, recent work by Girard shows that each exponential decomposes into two operations: a core operation which takes care of resources management, and an external shell which changes the polarity. Therefore we have $\vdash N = \downarrow (\exists N)$ and dually $\vdash P = \uparrow (\forall P)$.

At present, the good way to deal with the exponentials in Ludics is still object of research; for this reason we are leaving exponentials aside. As in [Gir01b], the connectives we are going to consider are the standard connectives of MALL, plus the new connective called $\mathrm{Shift}$: $\downarrow$. The only role of the $\mathrm{Shift}$ is to change the polarity of a formula: if $N$ is negative, $\downarrow N$ is positive.

\[
\begin{array}{c}
\vdash N, \Gamma \\
\vdash \downarrow N, \Gamma
\end{array}
\]

In order to keep notation as simple as possible, we are going to deal with $\downarrow$ implicitly: we write $((P_1 \exists P_2) \otimes Q^\perp) \otimes R^\perp$ for $(\vdash (\downarrow P_1 \forall P_2 \vdash \downarrow Q^\perp) \vdash \downarrow R^\perp$.

In the formula above, we have two synthetic connectives: a negative binary connective $(- \exists -)$ and a positive ternary connective $(- \otimes -)$.

As we just seen, $A = (P^\perp \otimes Q^\perp) \otimes R^\perp$ can be treated as a single ternary connective $\Phi(P^\perp, Q^\perp, R^\perp)$ which applies to the negative subformulas $P^\perp, Q^\perp, R^\perp$. For this connective there are two possible rules, obtained combining a Tensor-rule with one of the two possible Plus-rules:

\[
\begin{align*}
\vdash P^\perp, \Gamma & \quad \vdash R^\perp, \Delta \\
\vdash (P^\perp \otimes Q^\perp) \otimes R^\perp, \Gamma, \Delta & \quad (A, \{P^\perp, R^\perp\}) \quad \text{or} \quad (A, \{Q^\perp, R^\perp\})
\end{align*}
\]

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

The dual of $A$ is $A^\perp = (P \& Q) \exists R$, which again is treated as a single ternary connective. The rule for this ternary connective combines the Par-rule with the With-rule:

\[
\begin{align*}
\vdash P, R, \Lambda & \quad \vdash Q, R, \Lambda \\
\vdash (P \& Q) \exists R, \Lambda & \quad (A^\perp, \{P, R\}), (A^\perp, \{Q, R\})
\end{align*}
\]

The rule is labelled by a set of pair: a pair (focus, set of subformulas) for each premise. Actually, we rather use the label $(A^\perp, \{P, R\}, \{Q, R\})$ which is short for $\{(A^\perp, \{P, R\}), (A^\perp, \{Q, R\})\}$; a negative label is therefore given by the (unique) focus, and a set of subformulas for each premise.

To fully understand the notation and the symmetry between positive and negative, observe that to each positive rule corresponds a premise of the negative
rule. During cut-elimination, the positive rule will only interact with that negative premise. That is to say, the positive rule will select one of the premises, as we show below.

Normalization

The redex:

\[ \frac{\vdash P^\perp, \Gamma \vdash R^\perp, \Delta}{\vdash (P^\perp \oplus Q^\perp) \otimes R^\perp, \Gamma, \Delta} \]

\[ (A, \{P^\perp, R^\perp\}) \vdash P, R, \Lambda \quad \vdash Q, R, \Lambda \]

\[ \frac{\{A^\perp, \{P, R\}\}, \{A^\perp, \{Q, R\}\}}{\vdash \Gamma, \Delta, \Lambda} \]

reduces to:

\[ \vdash P^\perp, \Gamma \vdash R^\perp, \Delta \vdash P, R, \Lambda \]

\[ \vdash \Gamma, \Delta, \Lambda \]

The redex:

\[ \frac{\vdash Q^\perp, \Gamma \vdash R^\perp, \Delta}{\vdash (P^\perp \oplus Q^\perp) \otimes R^\perp, \Gamma, \Delta} \]

\[ (A, \{Q^\perp, R^\perp\}) \vdash P, R, \Lambda \quad \vdash Q, R, \Lambda \]

\[ \frac{\{A^\perp, \{P, R\}\}, \{A^\perp, \{Q, R\}\}}{\vdash \Gamma, \Delta, \Lambda} \]

reduces to:

\[ \vdash Q^\perp, \Gamma \vdash R^\perp, \Delta \vdash Q, R, \Lambda \]

\[ \vdash \Gamma, \Delta, \Lambda \]

The commutation of rules that one could need to perform to reduce to the above cases are the obvious ones. However, since we have one more rule, the daimon, there is one more case that we need to consider for normalization, namely the case where \( \vdash A, \Gamma, \Delta \) is justified by a *daimon*:

\[ \vdash A, \Gamma, \Delta \]

\[ \vdash A^\perp, \Lambda \]

\[ \vdash \Gamma, \Delta, \Lambda \]

which always reduces to

\[ \vdash \Gamma, \Delta, \Lambda \]

no matter what is the form of the proof cut against it.
Some more details

A choice adopted in [Gir01b] is to use only positive symbols, writing the negative formulas on the left-hand side of the sequent: ⊢ (P & Q) ⊗ R, Λ is the same as (P ⊥ ⊗ Q ⊥) ⊗ R ⊥ ⊢ Λ. Therefore the sequent calculus will consist of sequents of the form Ξ ⊢ Δ. \(^1\) The left-hand side is the negative part of the sequent. Moreover, one can restrict to the case where Ξ has at most one formula.

**Remark 1.1.1** One can easily check that if we start from the proof of a single formula, in all sequents of a focalized proof there is at most one negative formula: this is a property which is preserved by all rules.

A consequence is that when we have a negative rule, we always know which is the focus.

We already explained as negative rule are labelled by the focus and a set of subformulas for each premise. If the set of premises is empty, we therefore have

\[
\frac{A \vdash \top}{A \vdash (A, \emptyset)} \quad \text{as in} \quad \vdash \top, \Gamma
\]

Such a rule performs no action at all. This corresponds to a proof that we could still expand in the process of proof-construction. There is no dual: no positive rule matches a rule which is doing nothing. The only rule that normalize against this is a daemon.

It is important to distinguish the previous from the following ones:

\[
\frac{\vdash \Gamma}{\vdash A, \Gamma \vdash (A, \emptyset)} \quad \text{as in} \quad \vdash \top, \Gamma
\]

This is a positive rule with no premises. The dual rule is:

\[
\frac{\vdash \Gamma}{A \vdash \Gamma \vdash (A \perp, \emptyset)} \quad \text{as in} \quad \vdash \bot, \Gamma
\]

This is a negative rule with a single premise, and no subformulas.

**A sequent calculus of locations**

Next step is to remove logical decorations (formulas). What remains is the location (the address) of the formula. Such an address is called **locus**.

\(^1\)This notational choice has several advantages, and in particular stresses the fact that the two formulas involved in a cut are the same. However, we will often use the one side notation, to make the typing more readable, in particular w.r.t. proof nets: a Tensor or a Plus “on the left” are more immediate to understand as respectively a Par and a With.
Each formula to be decomposed receives an address, or location, the place where
the name of the formula is written. We can assume this space is an infinitely
branching tree. If \( A \) of the previous section is given the address \( \xi \), then \( P, Q, R \)
will respectively be distinguished by indices 1, 2, 3: e.g. \( Q \) is the sub-formula of \( A \)
of relative location 2. \( P, Q, R \) will respectively be located in \( \xi_1, \xi_2, \xi_3 \). The logical rules
written above will be interpreted by the decomposition of \( \xi \) into the sub-addresses
\( \xi_1, \xi_2 \) and/or \( \xi_1, \xi_3 \).

The example we have been working with in the previous section can be rewritten
as

\[
\begin{align*}
\text{Positive rules} \\
\xi_1 \vdash \Gamma \quad \xi_2 \vdash \Delta \\
\hline
\xi, \{1, 3\} \\
\xi_2 \vdash \Gamma \\
\xi_3 \vdash \Delta \\
\xi, \{2, 3\}
\end{align*}
\]

\[
\begin{align*}
\text{Negative rules} \\
\xi_1 \vdash \xi_3, \Lambda & \vdash \xi_2, \xi_3, \Lambda \\
\hline
\xi, \{1, 3\} & \{2, 3\} \\
\xi \vdash \Lambda
\end{align*}
\]

Sequences of addresses are expressions of the form \( \Xi \vdash \Lambda \) where:
\( \Xi, \Lambda \) are finite sets of addresses, pairwise disjoint;
\( \Xi \) contains at most one address.

A sequent of the form \( \vdash \Lambda \) is said positive, a sequent of the form \( \xi \vdash \Lambda \) is said
negative. In particular, the empty sequent \( \vdash \) is positive. A sequent is atomic
when it contains exactly one address.

The rules of the calculus that we have sketched so far fall into three categories:

Rules

Daimon.

\[
\vdash \Gamma \\
\vdash
\]

Positive rules.
Assume \( I \) is a set of indices (called ramification), and for \( i \in I \) the \( \Gamma_i \) are pairwise
disjoint and included in \( \Gamma \). One can apply the following rule (finite, one premise for
each \( i \in I \)):

\[
\begin{align*}
\ldots \\
\xi_i \vdash \Gamma_i \\
\ldots \\
\xi, \Gamma \\
\hline
(\xi, I)
\end{align*}
\]

Negative rules.
Assume \( \mathcal{N} \) is a set of ramifications (called the directory of the rule ), and for all
\( I \in \mathcal{N}, \Lambda_I \subseteq \Lambda \). One can apply the following rule (possibly infinite, one premise for
each \( I \in \mathcal{N} \)):

\[
\begin{align*}
\ldots \\
\vdash \xi I \\
\ldots \\
\xi \vdash \Lambda \\
\hline
(\xi, \mathcal{N})
\end{align*}
\]
The notation $\xi I$ we have used is short for $\{\xi, i \in I\}$.

This rule is a combination of usual rules (grouped into clusters) and weakening (that is why it is demanded $\bigcup \Gamma_i \subseteq \Gamma$ and $\Lambda_I \subseteq \Lambda$ rather than strict equality).

**Definition 1.1.2 (Actions)** The pair $(\xi, I)$ is called an action. As we have seen, $\xi$ is an address (the address of a formula) and $I$ a set of indices, the relative addresses of the immediate relative subformulas we are considering. $\xi$ is called focus of the action, while $I$ is called ramification.

$\dagger$ is also an action, but an improper action. An action of the form $(\xi, I)$ is called proper.

### 1.2 Designs

Designs are the base of our work, so we try to develop a good intuition about them. We start with an informal presentation, then we will describe a design both as a tree of actions and as a set of chronicles.

The analysis of designs will continue in Chapter 2.

#### 1.2.1 Getting an intuition

Designs capture the geometrical structure of sequent calculus derivations. The simplest way to introduce designs is to start with a sequent calculus derivation.

Let us work again with a concrete example. Consider the following sequent calculus derivation

\[
\begin{array}{c}
\vdash a_0, c_0 \perp \\
\vdash a_0, c \quad \{c, \{c_0 \perp\}\} \\
\vdash a_0, d \quad \{d, \{d_0 \perp\}\} \\
\vdash b_0, d \quad \{(b_0 \perp, \{b_0\})\} \\
\vdash b_0, d \quad \{(b_0 \perp, \{b_0\})\} \\
\vdash b_0, d \quad \{(b_0 \perp, \{b_0\})\} \\
\vdash c, d, a \perp \otimes b \perp \\
\vdash c \& d, a \perp \otimes b \perp \\
\end{array}
\]

where $a \perp, b \perp, c, d$ are formulas which respectively decompose as $a_0, b_0, c_0 \perp, d_0 \perp$ (for example $\downarrow a_0, \downarrow b_0 \perp, \downarrow c_0 \perp, \downarrow d_0 \perp$). Let us forget everything in the sequent derivation, but the labels. The derivation above becomes the following tree of actions, which is 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 ("post mortem", as said in [Gir01b]). 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: “Choose a positive focus, decompose it in its negative components, then choose a positive focus, and so on.” This is mirrored in the tree by:

- polarities alternate;
- a positive focus is always followed by its immediate sub-addresses
- after a negative decomposition we select a new positive focus.

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 \( \xi \); 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 $c \land d$ at the address $\sigma$ and so on for its subformulas. Our design becomes:

$$
\begin{array}{c}
\sigma_1 \{0\} \\
\xi_2 \{0\}
\end{array}
\begin{array}{c}
\sigma_2 \{0\} \\
\xi_2 \{0\}
\end{array}
\xi \{1, 2\}
\sigma \{1, 2\}
$$

Where are the additives?

Let us consider an additive derivation

$$
\frac{\vdash A \quad \vdash B}{\vdash A \land B} \{(A\land B, A), (A\land B, B)\}
\Rightarrow \vdash (A\land B) \land C^\perp
$$

If we locate $(A\land B) \land C^\perp$ in $\xi$, we have:

$$
\frac{\vdash \xi_{11} \quad \vdash \xi_{12}}{
\xi_{11} \vdash \xi_{12} \quad \{(\xi_{11}, \{1\}), (\xi_{12}, \{2\})\}}
\Rightarrow \vdash \xi, \{1\}
$$

This will correspond to:

$$
\begin{array}{c}
\xi_{11} \{1\} \\
\xi_{12} \{2\}
\end{array}
\xi \{1\}
$$

The actions $(A\land B, A)$ (that is $(\xi_{11}, 1)$) and $(A\land B, B)$ (that is $(\xi_{12}, 2)$) are to be thought of as unary $\&$. The usual binary rule for $(A\land B)$ is a super-imposition of two unary rules. We will come back to this in Section 2.1.1.

A design is given by a base and a tree of actions with some properties that we are going to present in the next section. A branch in the tree is called a chronicle. If $\kappa_1$ is before $\kappa_2$ we write $\kappa_1 < \kappa_2$. We think of the tree as oriented in the time, starting from the root. Hence the terminology before and after for actions in the same branch (chronicle).
1.2.2 The base

A base is a sequent of addresses, which correspond to the “initial” sequent of the derivation, the conclusion of the proof, the specifications of the process. The base:

1. gives the addresses of the formulas we are going to decompose;
2. establishes the polarity of the addresses;
3. establishes a dependency relation (a partial order\(^2\)) between the addresses, namely \(\xi < \lambda\) if \(\xi \vdash \lambda, \Lambda\).

**Polarity and Parity.** As we have already seen, a sequent has a positive and a negative side. According to its position (r.h.s. or l.h.s.), each address in the base has a *polarity* (positive or negative).

We have also seen that in a synthetic connective the polarity of subformulas alternates at each layer, so if \(\xi\) is positive, \(\xi_i\) is negative, \(\xi_{ij}\) is positive. According to its length, we say that an address is even or odd. This is called the *parity* of an address. Fixed as reference the addresses \(\xi\) given in the base, sub-addresses of \(\xi\) with the same parity as \(\xi\) have the same polarity, sub-addresses of \(\xi\) with opposite parity have opposite polarity.

We call *paritary* a base where all the addresses on the same side have the same parity, opposite to that of the addresses on the other side. A paritary base has the same parity as its positive addresses: \(2 \vdash 33, \vdash 11, 715 \vdash\) are all even bases. Observe that, because of their position, 2 and 715 are negative addresses, 33 and 11 are positive.

A design is positive or negative, even or odd according to its base.

1.2.3 A tree of actions

A design is given by a base and a tree of actions satisfying certain properties. The base induces a polarization of all the addresses (all the actions) we consider.

A design 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).

A design satisfies the following conditions:

*Root.* The root (possibly roots in case of a negative base) focuses on an address of the base; this is chosen minimal w.r.t. the dependency relation (thus if there is a negative address, that will be decomposed first.)

*Polarity.* Polarities alternate.

*Branching.* The tree only branches on positive actions.

\(^2\)To speak of partial order will make more sense when we will consider cut nets, which are sets of designs, and thus sets of bases.
Focalization. The addresses used after a positive action $(\xi, I)$ are immediate subaddresses of $\xi$: $\xi_i, i \in I$.

\[
\xi, K \quad \xi, K' \quad \xi, K''
\]

Observe that $\dagger$ 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 (if we have the focus $\xi_i$ then $\xi < \xi_i$). This simply corresponds to the subformula property.

Leaves. All maximal actions are positives.

Propagation (linearity). Assume $\kappa, \kappa'$ have the same focus $\sigma$, then

1. They are not on the same branch;
2. If the configuration is as follows

\[
\sigma \quad \sigma
\]

\[
\xi, K \quad \xi, K'
\]

then $\xi_i = \xi_j$.

This condition means that an addresses can be duplicated (reused) only in the context of a $\&$.

**Definition 1.2.1 (Slices)** A slice of a tree of actions is a subtree (on the same base) such that the addresses $\xi_i, i \in I$ used immediately after a positive action $(\xi, I)$ are all distinct.

The above definition implies that in the sequent calculus presentation, all negative rules are at most unary.

**Fact 1.2.2** Propagation can be reformulated as follows:

"in all slices each focus only appears once."

1.2.4 A set of chronicles

In [Gir01b] designs are described as set of chronicles. The definition is in two steps:
- definition of chronicle, that is a formal branch in a focalized sequent calculus derivation,
- definition of a coherence condition making a set of chronicle all belong to the same proof.

**Definition 1.2.3 (Chronicle)** A chronicle $\mathcal{c}$ of base $\Xi \vdash \Lambda$ is a non empty sequence of actions $\langle \kappa_0, \kappa_1, \ldots, \kappa_n \rangle$ such that:

- **Alternation.** The polarity of $\kappa_j$ is equal to that of the base for $j$ even, opposite for $j$ odd.
- **Daimon.** For $j < n$, $\kappa_j$ is proper.
- **Positive focuses.** The focus of a positive action $\kappa_p$ either belongs to the basis or is an address $\xi_i$ generated by a previous action: $\kappa_q = (\xi, I), i \in I$ and $\kappa_q < \kappa_p$.
- **Negative focuses.** The focus of a negative action $\kappa_p$ either belongs to the basis or is an address $\xi_i$ generated by the previous action: $\kappa_{p-1} = (\xi, I), i \in I$
- **Destruction of Focuses.** Focuses are pairwise distinct.

**Definition 1.2.4 (Coherence)** The chronicles $\mathcal{c}, \mathcal{c}'$ are coherent when

- **Comparability.** Either one extend the other, or they first differ on negative actions, i.e. if $\mathcal{c} = \mathcal{c} \land \mathcal{c}' \ast k \ast \mathcal{c}$, $\mathcal{c} = \mathcal{c} \land \mathcal{c}' \ast k' \ast \mathcal{c}'$ then $\kappa, \kappa'$ are negative.
- **Propagation.** If $\mathcal{c}, \mathcal{c}'$ first differ on $\kappa, \kappa'$ with distinct focuses, then all ulterior focuses are distinct.

**Remark 1.2.5** Propagation iff in all slices the focuses are pairwise distinct.

**Definition 1.2.6 (Design)** A design $\mathcal{D}$ of base $\Xi \vdash \Lambda$ is a set of chronicles of base $\Xi \vdash \Lambda$ such that:

- **Arborescence.** $\mathcal{D}$ is closed under restriction
- **Coherence.** The chronicles of $\mathcal{D}$ are pairwise coherent
- **Positivity.** If $\mathcal{c} \in \mathcal{D}$ has no extension in $\mathcal{D}$, then its last action is positive
- **Totality.** If the base is positive, then $\mathcal{D}$ is non empty

1.3 Normalization

Designs have been constructed by imitation of cut-free proofs. Let us now introduce cuts. First of all, notice that there is no cut rule. A cut is a coincidence of addresses of opposite polarity in the base of two designs.

We first precise this idea. We then sketch the normalization procedure on sequent of addresses. This mimic normalization in sequent calculus. In chapter 2 we will define normalization directly on the trees of actions.
1.3.1 Cut-nets

**Definition 1.3.1 (Cut-net and main design)** A cut-net is a non-empty finite set $R = \{ \mathcal{D}_1, ..., \mathcal{D}_n \}$ of designs, of respective bases $\Xi_i \vdash \Lambda_i$ such that:

- The loci occurring in the bases are uncomparable or equal.
- Each locus occurs at most in two bases, once positive and once negative. Such a shared locus is said cut.
- The graph whose vertices are the $\Xi_i \vdash \Lambda_i$ and whose edges are the cuts is connected and acyclic.

We orient the edges from positive to negative occurrences. The starting vertex is called the main base of the cut-net. The corresponding design is the main design of the cut-net.

Notice that a design is a special case of cut net ($n = 1$).

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.

**Definition 1.3.2 (Normal form)** The normal form of a cut-net $R$ is indicated by $\ll R \gg$

*Normalization* As in sequent calculus...

1.3.2 Orthogonality

**Definition 1.3.3 (Opposite base)** Given a base $\Xi \vdash \Lambda$, its opposite is the base (or the family of bases) which allows us to close the net. That is, the opposite of $\vdash \xi$ is $\xi \vdash$ (and vice-versa); the opposite of $\xi \vdash \lambda_1, ..., \lambda_n$ is the family $\vdash \xi, \lambda_1 \vdash, ..., \lambda_n \vdash$.

**Remark 1.3.4** When we speak of the closed net on $\Xi \vdash \Lambda$ we mean the cut-net on the closure of such base.

1.4 Analytical theorems

The value of the concept of design lies in a number of remarkable properties, which are the respective analogous of Böhm’s theorem, Church-Rosser property, and stability.
Separation and order

**Theorem 1.4.1 (Separation)** If $\mathcal{D} \neq \mathcal{D}'$ then there exists a counter-design $\mathcal{E}$ which is orthogonal to one of $\mathcal{D}, \mathcal{D}'$ but not to the other.

The proof is based on the following construction, whose purpose is to interactively recognize a chronicle:

**Definition 1.4.2 (Oppc)** If $\mathcal{c}$ is a chronicle, we define the counter-design $\text{Oppc}$ as:

- If $\mathcal{c}$ ends with a negative action, then $\text{Oppc}$ consists exactly of the (opposite of the) actions performed in $\mathcal{c}$.
- If $\mathcal{c}$ ends with a positive action, then $\text{Oppc}$ consists of the (opposite of the) actions performed in $\mathcal{c}$, together with an appropriate daimon.

If $\mathcal{D}, \mathcal{D}'$ are different, they differ at least on a chronicle $\mathcal{c}$ such that for example $\mathcal{c} \in \mathcal{D}$ and $\mathcal{c} \notin \mathcal{D}'$. We have that $[\mathcal{D}, \text{Oppc}] = \dagger$ and $[\mathcal{D}', \text{Oppc}] = \Omega$.

Designs on the same base are ordered w.r.t. convergence:

**Definition 1.4.3** Let $\mathcal{C}, \mathcal{D}$ be designs on the same base. $\mathcal{C} \leq \mathcal{D}$ if $\mathcal{D} \in \mathcal{C} \perp$.

This means that $\mathcal{C}$ is “more defined” than $\mathcal{D}$, or that $\mathcal{D}$ converges more easily than $\mathcal{C}$.

The greatest design is the one whose only action is $\dagger$, because it always converges. The smallest design is the one which performs no action, because it only converges against $\dagger$. Girard summarizes this situation with the inequality:

$$\Omega \preceq (\xi, I) \preceq \dagger$$

The relation $\preceq$ can be explicitly described by means of two operations:

- **Widen**: add more premises to negative rules;
- **Shorten**: replaces positive rules (actually whole subtrees) with daemons.

$\mathcal{D} \subseteq \mathcal{D}'$ indicate the inclusion among designs, as set of chronicle (or as subtrees).

**Associativity, closure principle, stability**

**Theorem 1.4.4 (Associativity)** Let $\{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$ be a net of nets, then

$$\llbracket \mathcal{R}_1 \cup \cdots \cup \mathcal{R}_n \rrbracket = \llbracket \llbracket \mathcal{R}_1 \rrbracket \cup \cdots \cup \llbracket \mathcal{R}_n \rrbracket \rrbracket$$

Associativity together with separation converges in the closure principle: “everything reduces to closed nets.” The normal form of a net $\mathcal{R}$ is determined by the
normal forms of all its completions into a closed net:

if $\mathcal{D}, \mathcal{E}$ are designs of respective base $\vdash \lambda$ and $\vdash \xi$, then the normal form $[\mathcal{D}]\mathcal{E}$ is the unique design $\mathcal{D}'$ of base $\vdash \lambda$ such that for every $\mathcal{F}$ of base $\lambda \vdash$ we have $[\mathcal{D}', \mathcal{F}] = [\mathcal{D}, \mathcal{E}, \mathcal{F}]$.

**Theorem 1.4.5 (Stability)** Normalization commutes with compatible intersection.

### 1.5 Behaviours (types)

Designs can be handled in two ways:

- **Untyped.** Designs can be considered as pure (i.e. as themselves).
- **Typed.** Designs can be considered as part of a behaviour.

A type is interactively defined as a “behaviour:"

**Definition 1.5.1 (Behaviour)** A behaviour $\mathbf{G}$ is a set of designs (on a given base) which is equal to its bi-orthogonal.

#### 1.5.1 Material designs

What does really matter in a behaviour is only the part we can actually use, that is the part we can observe with a test.

Observe that if $\mathcal{D} \in \mathbf{G}$, then any $\mathcal{D}' \supset \mathcal{D}$ belongs to $\mathbf{G}$, but for bad reasons. Possibly, nothing in $\mathcal{D}' - \mathcal{D}$ will never be used: if there were in $\mathbf{G}$ a test to access the actions in $\mathcal{D}' - \mathcal{D}$, such a test would diverge against $\mathcal{D}$...

**Definition 1.5.2 (Incarnation, material designs)** Given $\mathcal{D} \in \mathbf{G}$ there is a smallest $\mathcal{D}_0 \subset \mathcal{D}$ which is still a design of $\mathbf{G}$. This design is called the incarnation $|\mathcal{D}_G|$ of $\mathcal{D}$ in $\mathbf{G}$.

A design equal to its incarnation is said material: $\mathcal{D} = |\mathcal{D}|$.

The incarnation $|\mathbf{G}|$ of the behaviour $\mathbf{G}$ is the set of its material designs.

The theorem of stability implies a more operational characterization of a material designs: the incarnation $|\mathcal{D}|$ of a design $\mathcal{D}$ is the part of $\mathcal{D}$ which is visited via normalization with designs in $\mathbf{G}^\perp$.

#### 1.5.2 Some special designs

Let us first give a name to some special designs we are going to use. Many more are catalogued in [Gir01b].

- **Daimon ($\mathfrak{D}$)** is the design:

\[
\begin{array}{c}
\vdash \Lambda \\
\end{array}
\]
\( \Diamond \alpha \) belongs to all positive behaviours, in particular \( |0| = \{ \Diamond \alpha \} \).

The negative Daimon (\( \Diamond \alpha^- \)) is the design:
\[
\begin{align*}
\cdots & \vdash \xi \Gamma, \Lambda \\
\xi & \vdash \Lambda \\
\therefore & \vdash (\xi, P_f(\mathbb{N}))
\end{align*}
\]

\( \Diamond \alpha^- \) belongs to all negative behaviours.

We call \( \mathcal{R}am_{\lambda, I} \) the following design
\[
\begin{align*}
\cdots & \vdash \lambda \xi \Gamma \\
\lambda \xi & \vdash \\
\therefore & \vdash \Lambda \\
\therefore & \vdash (\lambda, P_f(\mathbb{N})) \\
\therefore & \vdash (\lambda, I)
\end{align*}
\]

We have that \( \mathcal{G} = \{ I : \mathcal{R}am_{\lambda, I} \in G \} \) indices the connected components of \( G \).

### 1.5.3 Directories, connected and prime behaviours

\( \mathcal{G} \) is called the directory of \( G \). If \( G \) is a positive behaviour, \( \mathcal{G} \) consists of those \( I \) such that \( (\langle \rangle, I) \) is the first action of a design \( D \in G \). \( \mathcal{G} = \mathcal{G}^\perp \). The incarnation of \( \Diamond \alpha^- \) in \( G^\perp \) is of the form
\[
\begin{align*}
\cdots & \vdash \langle \rangle \Gamma \\
\langle \rangle & \vdash \\
\therefore & \vdash (\langle \rangle, \mathcal{N})
\end{align*}
\]

where \( I \in \mathcal{N} \) exactly when \( (\langle \rangle, I) \) is the first action of some \( D \in G \).

A behaviour is connected when its directory is a singleton \( \{ I \} \).

A behaviour is prime when its directory is a singleton \( \{ i \} \).

### 1.6 Connectives

Ludics is very sensitive to locations. This is one of the reasons which make the notion of type both richer and finer. Usual logical connectives appear when there is no interference between addresses: everything is placed in distinct locations. Here we only remind the constructions which correspond to MALL connectives\(^3\).

**Definition 1.6.1 (Disjoint behaviours)** Two behaviours \( G, G' \) on the same base are disjoint if \( \mathcal{G}(G), \mathcal{G}(G') \) are disjoint sets. If \( G, G' \) are positive, this means in particular that any design in \( G \) has a first action distinct from that of a design in \( G' \).

To force behaviour to be disjoint, one can always use a “delocations” of all the addresses in all designs, for example by letting:
\[
\phi(\sigma \circ i \circ \tau) = \sigma \circ 2i \circ \tau \quad \psi(\sigma \circ i \circ \tau) = \sigma \circ 2i + i \circ \tau
\]

\(^3\)The only chapter of the thesis concerned with logical connectives is Chapter 5.
1.6.1 Additives

Let $A, B$ be disjoint behaviours.

If $A, B$ are positive one defines $A \oplus B = A \cup B$. The new set of designs is a behaviour: $(A \cup B)_{\perp \perp} = A \cup B$.

If $A, B$ are negative one defines $A \& B = A \cap B$. The new set of designs is a behaviour: $(A \cap B)_{\perp \perp} = A \cap B$.

What is remarkable is incarnation, which gives us a simple and complete description of $A \& B$:

$$|A \& B| = |A| \times |B|$$

1.6.2 Multiplicatives

Let $A, B$ be disjoint behaviours.

Let $\mathcal{D}, \mathcal{D}'$ be positive disjoint designs, with first action respectively $(\xi, I)$ and $(\xi, I')$. The operation $\mathcal{D} \otimes \mathcal{D}'$ put the two designs together in a new design defined as follows:

If $I \cap I' = \emptyset$ then the design $\mathcal{D} \otimes \mathcal{D}'$ has first action $(\xi, I \cup I')$ and then continues as $D$ on the addresses $\xi_i, i \in I$ and as $\mathcal{D}'$ on the addresses $\xi_j, j \in I'$.

If either $\mathcal{D}$ or $\mathcal{D}' = \uparrow$, then $\mathcal{D} \otimes \mathcal{D}' = \uparrow$.

If $I \cap I' \neq \emptyset$ then $\mathcal{D} \otimes \mathcal{D}' = \uparrow$ (this last case actually admits several variations, giving rise to non-commutative connectives).

If $A, B$ are positive one defines $A \otimes B = \{\mathcal{D} \otimes \mathcal{D}', \mathcal{D} \in A, \mathcal{D}' \in B\}$. The new set of designs is a behaviour: $\{\mathcal{D} \otimes \mathcal{D}', \mathcal{D} \in A, \mathcal{D}' \in B\}_{\perp \perp} = \{\mathcal{D} \otimes \mathcal{D}', \mathcal{D} \in A, \mathcal{D}' \in B\}$.

If $A, B$ are negative, then $A \triangledown B = (A \otimes B)_{\perp}$.

1.6.3 Shift

Shift simply adds a dummy move to all design in the behaviour, to change the polarity of the designs, without anything structural change in the design itself. If $G$ is negative, $\downarrow G$ is positive, if $G$ is positive, $\uparrow G$ is negative.

If $\tau$ is a chronicle of base $\vdash \Lambda, \xi i$ (resp. $\xi i \vdash \Lambda$) then shift $\tau$ is the chronicle $(\xi, i) \vdash \Lambda$. To shift a design we shift all its chronicles. Then one defines: $\downarrow G = \{\downarrow \mathcal{D}, \mathcal{D} \in G\}_{\perp \perp}$ and $\uparrow G = \{\uparrow \mathcal{D}, \mathcal{D} \in G\}_{\perp \perp}$

One has that the positive behaviour $\downarrow G$ is equal to $\{\downarrow \mathcal{D}, \mathcal{D} \in G\} \cup \{\uparrow \}$.

Any behaviour composed in this way enjoys internal completeness: the set of designs produced by the construction is equal to its biorthogonal. Since the biorthogonal does not introduce new objects, we have a complete description of all the designs
in the behaviour. For examples, for any $\mathcal{D} \in A \otimes B$ we know we can decompose it as $\mathcal{D}_1 \otimes \mathcal{D}_2$, with $\mathcal{D}_1 \in A$ and $\mathcal{D}_2 \in B$. Because of internal completeness, any behaviour formed by using the connectives can be decomposed in its initial components. However, in general a behaviour cannot be decomposed. Let us examine the situation, to which we will come back in Chapter 5.

1.6.4 Decomposition of behaviours

Additive decomposition. Any positive behaviour can be written in unique way as the $\bigoplus$ of connected behaviours:

$$G = \bigoplus_{I \in \mathcal{G}} G_I$$

Shift decomposition. Any prime behaviour is immediately decomposed as $G = \downarrow (G' \downarrow)$.

Multiplicative decomposition. There is no such a thing as a multiplicative decomposition of any behaviour. Let us see what one would like to have.

Definition 1.6.2 (Projection) Let $S \subseteq \mathbb{N}$. If the first action of a design $\mathcal{D}$ is $(\xi, I)$, the projection of $\mathcal{D}$ on $S$ $\mathcal{D} \upharpoonright S$ is the design starting with $(<> , I \cap S)$. $\mathcal{D} \upharpoonright S = \mathcal{D} i$.

Any positive design $\mathcal{D}$ with first action $(\xi, I)$ can uniquely be written as a tensor product $\mathcal{D'} \otimes \mathcal{D''}$, where $\mathcal{D'} = \mathcal{D} \upharpoonright S$ and $\mathcal{D''} = \mathcal{D} \upharpoonright \mathbb{N} \setminus S$.

Assume $G_I$ is a connected behaviour. Any design in $G_I$ can be written as the product of the designs $\mathcal{D} \upharpoonright \{i\}$, $i \in I$; the first rule of $\mathcal{D} \upharpoonright \{i\}$ is thus $(<> , \{i\})$. One can then define $G_I \upharpoonright i = \{ \mathcal{D} \upharpoonright i \}$, $\mathcal{D} \in G_I, i \in I$. However this operation does not provide a multiplicative decomposition of $G_I$. It is evident that $G_I \subseteq \bigotimes_{i \in I} \mathcal{D} \upharpoonright i$, but the converse is not true in general.
Chapter 2

From Designs to Disputes: Normalization

As we have seen, the objects that in Ludics correspond to derivations are the designs. Designs can be presented in a sequent calculus style, but in fact they are trees of actions. In this chapter (and in fact all along this thesis) we stress that the designs as trees (*desseins* in [Gir01b]) represent a very convenient syntax. As we evidence in Section 1, 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:

- Offer a concise syntax.
- 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 GoI)
- Are close to implementation, in that they make explicit the “addresses” and use tools typical of implementations, such us a dynamical approach to the context.

The study of normalization on designs is the central object of this chapter. In section 2 we shall present cut-elimination by means of an abstract machine, called \textit{Loci Abstract Machine (LAM)}. The normal form of a cut-net is calculated by a token traveling along a net. We first work “per slices,” where a slice essentially is a multiplicative derivation. The LAM normalization on designs is analogous to the order quotient defined in [Gir01b], though it has been developed independently.

As a token travels on a cut-net, it draws a path, a sequence of actions that represents the interaction among the designs of the cut-net. With a slight abuse on the terminology in [Gir01b], we call such a sequence dispute; the part of the cut-net visited during the normalization is the \textit{pull-back} of that dispute.

The study of normalization on designs leads in particular two the following outcomes:

1. We are able to calculate the pull-back of a dispute (Section 2.5).
2. The LAM can be generalized to run on any design (Section 2.4). This means that even if we do not select the slice we are working with, all the information we need in order to move to the correct position in the net (i.e., to the correct slice) is carried by the state of the machine.

Both points (1) and (2) extract the necessary information from the normalization path. The key operation we use exactly corresponds to a well-known operation of Games Semantics, the computation of the view.

Notation and conventions

By design we always intend the structure that in [Gir01b] is called desseins. If we refer to their sequent calculus counterparts (i.e., dessins) we make it explicit.

2.1 Slices as proof-nets

2.1.1 Slices

The notion of slice was introduced as part of the theory of proof-nets in Linear Logic [Gir87], to speak of the two components of a \&-rule. Informally speaking, a \&-rule is seen as the super-imposition of two unary rules: \((a \& b, \{a\})\) and \((a \& b, \{b\})\). Given a proof, we obtain a slice of that proof if for any \&-rule we select one of the premises.

Example 2.1.1 The derivation

\[
\frac{\vdash \cdots c}{\vdash a, c} \quad \frac{\vdash \cdots c}{\vdash b, c} \quad \frac{\vdash a \& b, c}{a \& b, \{a\}, \{b\}} \quad \frac{\vdash a \& b, c}{\downarrow a \& b}
\]

can be decomposed as the super-imposition of two slices (in each slice the \&-rule is unary)

\[
\frac{\vdash \cdots c}{\vdash a, c} \quad \frac{\vdash \cdots c}{\vdash b, c} \quad \frac{\vdash a \& b, c}{a \& b, \{a\}} \quad \frac{\vdash a \& b, c}{\downarrow a \& b} \quad \frac{\vdash a \& b, c}{a \& b}
\]

and

\[
\frac{\vdash \cdots c}{\vdash a, c} \quad \frac{\vdash \cdots c}{\vdash b, c} \quad \frac{\vdash a \& b, c}{a \& b, \{a\}} \quad \frac{\vdash a \& b, c}{\downarrow a \& b} \quad \frac{\vdash a \& b, c}{a \& b}
\]

Observe that selecting one of the premises of a \&-rule is exactly what normalization does. Normalization is always carried out in a single slice: when a negative rule (typically a \&-rule) has several premises, the opponent (typically a \oplus-rule) selects one of them.

In the setting of Ludics, a slice \(\mathcal{S}\) of a design \(\mathcal{D}\) is defined as a design \(\mathcal{S} \subseteq \mathcal{D}\), where all negative rules have at most one premise. As a design, a slice is simply a
tree of actions (a pointed tree of addresses, as we will see soon), where each address only appears once. Observe in particular, the immediate sub-addresses of a positive action that are used as focuses are all distinct.

Let us look again at the above example. We locate $c$ in the address $\tau$, $\downarrow (a \& b)$ in the address $\xi$, $(a \& b)$ in $\gamma_0$, $a$ in $\xi_0$, and $b$ in $\xi_02$. The derivation of Example 2.1.1 corresponds to the following design

![Diagram of a tree with labels $\tau$, $\xi_1$, $\xi_2$, and $\xi$.]

whose two slices are

![Diagram of two separate trees with labels $\tau$, $\xi_1$, and $\xi$, and $\tau$, $\xi_2$, and $\xi$.]

**Notation 2.1.2** In a slice, 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 $\sigma$.

### 2.1.2 Two orders: chronicles and prefix

In a design 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").

The order among actions in the chronicles (the solid lines in the picture below) will be indicated by $<$, while the prefix order (the dashed lines) will be indicated by $\sqsupseteq$. We indicate the relation to be an immediate sub-address as $\sqsupseteq_1$. For example, $\xi_i$ is immediate sub-address of $\xi$, noted $\xi_i \sqsupseteq_1 \xi$. We indicate the relation to be an immediate successor in the chronicle-tree as $\succ_1$. 
Let us have again a look at the example in Section 1.2.1. We make explicit the relation of being a sub-address with a dashed arrow connecting $\sigma$ to $\sigma_1$ and $\sigma_2$, and $\xi$ 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 added of some information on the axiom links.

Let us take again the above example, where we have made explicit the information on both orders. 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 information allows us to establish the axiom 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 $\sigma_2$.

Our argument is rather informal, but the above intuitions suggest an interesting question. Since slices seem to be close to multiplicative proof-nets, cannot we deal with normalization as in proof-nets rather than as in sequent calculus? The answer is indeed positive.
Essentially we can mimic proof-nets normalization, as in the following example, where the cut-net

\[
\begin{array}{c}
\sigma_1 \quad \sigma_2 \quad \xi_1 \quad \xi_2 \\
\xi \quad \zeta \\
\sigma \\
\end{array}
\]

once written as

\[
\begin{array}{c}
\sigma_1 \quad \sigma_2 \quad \xi_1 \quad \xi_2 \\
\xi \quad \zeta \\
\sigma \\
\end{array}
\]

and then to

\[
\begin{array}{c}
\sigma_1 \quad \sigma_2 \quad \tau_1 \quad \tau_2 \\
\tau \\
\end{array}
\]
The situation is in general slightly more complex than in the above example, because we are working in a setting which is not typed. Thus for example $\xi$ 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 [Gir01b] and the abstract machine we define in the next section.

2.2 Loci Abstract Machine

Normalization of a cut-net $\mathcal{R}$ 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 $\text{LAM}_0$, working on slices. In Section ?? we will define a generalized version, indicated by $\text{LAM}_+$, working directly on designs.

The definitions of cuts, cut-net ($\mathcal{R}$), normal form ($[\mathcal{R}]$), main design were given in Chapter 1.

2.2.1 $\text{LAM}_0$

We first work only with slices. Since in a slice there is no “additive duplication,” normalization of slices is simpler than normalization of general designs. However, it is not a simplification, in the sense that working “by slices” is enough to recover the normal form of any cut-net.

Given a cut-net $\mathcal{R}$, the machine takes as input a token. While the token moves around following the instructions of the machine, it drafts a path on the cut-net. The token records such a path; each path will represent a chronicle of the normal form $[\mathcal{R}]$.

To distinguish between the actions which are cut and those which are not we call an address closed if it is a sub-address of a cut, open otherwise. We extend this definition to actions.

Figure 2.1 and 2.2 graphically present the machine; the token is represented by a square. Observe that the token is always going upwards. The key point is that when the same address $\sigma$ appears in distinct designs, we can move from one design to the other, passing from $\sigma^+$ to $\sigma^-$. 

**Initialization** (Fig.2.1): The token enters the net on the first action of the main design.

**Transitions** (Fig.2.2): When the token is on an open action, it follows the chronicles order, moving upwards from $\kappa$ to the actions which immediately follow $\kappa$ in
the slice. When the token enter a closed positive action, the token exits at the corresponding negative action (then changing of design).

The transitions shown in the pictures are formalized below. At the end of this section we will give an example of execution (2.2.1).

**Definition of LAM**

A token is given by a pair \((\kappa, [p])\). The first element \(\kappa\) is either an action, which represents the current position of the token, or a special symbol \(Halt\), indicating that computation has finished. The second element 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. \(\epsilon\) indicates the empty sequence of actions. A multi-token \(S\) is a set of tokens.

Transitions are directed by the two functions \(First\) and \(Succ\). \(First(\mathcal{R})\) is the collection of the first actions in the main design. Given an action \(\kappa\) in a slice \(\mathcal{G}\), we indicate by \(Succ(\kappa)\) the collection of all actions \(\kappa_i\) such that \(\kappa <_1 \kappa_i\) in \(\mathcal{G}\).

**Initialization.**

If \(First(\mathcal{R}) \neq \emptyset\) then \(S = \{\kappa[c] : \kappa \in First(\mathcal{R})\}\)
else \(S = \{Halt[c]\}\)

**Transitions.**

Let \(t \in S\), \(S = \{t\} \uplus S'\) (where \(\uplus\) is the disjoint union), and \(t = \eta[p]\).

If \(\eta\) is an open action (recall that open means not cut) then we proceed with

\[
S = \begin{cases} 
\{\kappa[p, \eta] : \kappa \in Succ(\eta)\} \uplus S' & \text{if } Succ(\eta) \neq \emptyset \\
\{Halt[p, \eta]\} \uplus S' & \text{if } Succ(\eta) = \emptyset
\end{cases}
\]

(2.1)
Transitions: $\sigma$ open

Transitions: $\sigma$ closed

Figure 2.2: LAM
If \( \eta \) is a closed action (the focus is sub-address of a cut), then \( \eta \) is necessarily positive, and we proceed moving to the corresponding negative action:

\[
S = \begin{cases} 
\{ \kappa[p] : \kappa \in \text{Succ}(\eta^-) \} \uplus S' & \text{if } \eta^- \in \mathcal{R} \\
\{ \text{Halt}[p, \Omega] \} \uplus S' & \text{if } \eta^- \notin \mathcal{R}
\end{cases}
\]  

(2.2)

Termination.
The computation halts when all the tokens \( t_i \) in \( S = \{ t_1, \ldots, t_n \} \) are of the form \( \text{Halt}[p_i] \). If \( \text{Halt}[p, \Omega] \in S \) we say that the computation fails, otherwise that it succeeds.

If the normalization succeeds, each maximal path describes a maximal chronicle of the normal form. Given a cut net \( \mathcal{R} \), we indicate by \( LAM_0(\mathcal{R}) \) the result of the execution of the abstract machine on the cut-net \( \mathcal{R} \):

\[
LAM_0(\mathcal{R}) = \{ p : \text{Halt}[p] \}
\]

The normal form \( [\mathcal{R}] \) is the closure under prefix of \( LAM_0(\mathcal{R}) \).

Let us comment on the above definition.

Initialization. The "else" case arises when the main design is a negative empty design (which in [Gir01b] is called \( \mathcal{G}kunk \)).

Transitions. (3.1) When the actions are open, there is no distinction between positive and negative actions, because in both cases the token is "going up." We distinguish instead if the action is a leaf or not. If not, the token continues upwards, and possibly multiplies (multi-token). It the action is a leaf (then it is necessarily positive), that branch of computation terminates (termination with success).

(3.2) On this step the token changes of design, moving from the positive occurrence of an action to the negative one. If the action does not appear negative in the net, this branch of computation terminates with failure, indicated by \( \Omega \).

To normalize we need to know the action, not simply the address. This has two reasons: (i) we work with slices (thus we can have \( (A \& B, A) \) against \( (A \oplus B, B) \) (ii) the addresses are not typed (thus the same addresses could correspond to different formulas).

Observe that:

Multi-tokens vs. tokens. In the definition above, we have chosen to work with a multi-token. We could also use a single token, ending at each time with a single chronicle of the normal form. Anyway, on any branching positive node, we would need to choose a subtree in which to continue. For this reason we rather prefer to "multiply" the token.
The role of paths. When working with slices, the path recorded by the token has no role in the computation: it is only used to store the result. In fact each path will represent a chronicle of the normal form.

Let us give an example of execution.

Example 2.2.1 (Execution) Consider the following cut-net, where the bases are respectively $\alpha \vdash \beta, \gamma$ and $\beta \vdash \sigma, \tau$.

![Diagram 1](image1)

We decorate it with the path followed by the tokens: $i$ indicate the $i$-ary step.

![Diagram 2](image2)

On $\sigma$ 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 records the open actions. As the computation proceeds, the normal form grows as follows:

![Diagram 3](image3)

$\alpha \rightarrow \alpha \rightarrow \alpha \rightarrow \alpha$
Designs vs. sequent calculus normalization

We could have presented the same design with the syntax of sequent calculus.

\[
\begin{align*}
\frac{\alpha_0 \vdash \beta_{10}, \alpha_0}{\beta_1 \vdash \alpha_0} & \quad \frac{\beta_{20}, \gamma \vdash \beta_2}{\gamma \vdash \beta_{21}} \quad \frac{\beta_1 \vdash \beta_{2}, \gamma}{\beta \vdash \{1, 2\}} \\
\frac{\sigma_1 \vdash \sigma_{10}, \beta_2}{\sigma_1 \vdash \beta_2} & \quad \frac{\sigma_2 \vdash \sigma_{20}, \tau}{\tau \vdash \sigma_2} \quad \frac{\beta \vdash \sigma_2, \tau}{\beta \vdash \sigma_1, \gamma} \\
\frac{\alpha \vdash \beta, \gamma}{\alpha \vdash \beta, \gamma}
\end{align*}
\]

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. Namely:

\[
\begin{align*}
\frac{\sigma_1 \vdash \gamma}{\sigma_1 \vdash \gamma} & \quad \frac{\sigma_2 \vdash \tau}{\sigma_2 \vdash \tau} \\
\frac{\alpha_0 \vdash \sigma, \gamma, \tau}{\alpha \vdash \sigma, \gamma, \tau}
\end{align*}
\]

Properties

Let us state some results. It is easy to verify that

**Proposition 2.2.2** If normalization succeeds, the closure under prefix of \(LAM_0(\mathcal{R})\) is a design (actually a slice) on the base of \(\mathcal{R}\).

Since normalization was only defined for the sequent calculus, we need to verify that our procedure lead to the same result.

**Proposition 2.2.3 (Normalization of slices)** Let us indicate by \(LAM_0^*(\mathcal{R})\) the closure under prefix of \(LAM_0(\mathcal{R}) = \{p : Halt[p]\}\).

Given a slice \(S\) written in sequent calculus style, let us indicate by \(Dess(S)\) the corresponding design. We have that:

\[
LAM_0^*(\text{Dess}(S_1),...,	ext{Dess}(S_n)) = \text{Dess}([S_1,...,S_n])
\]

where \([S_1,...,S_n]\) is the normal form calculated in the "sequent calculus" fashion.

**Proof.** Easy proof. One verifies that at any reduction in sequent calculus, the label of the main base is exactly the action to which the token moves. □

2.2.2 Balanced slices (Locus Solum)

Our procedure is a step-by-step analogous to the "quotient order" defined by Girard in [Gir01b].

In an ideal world, positive and negative actions should match each other. Such an ideal world is called in [Gir01b] balanced.
Definition 2.2.4 If $R = \{ \mathcal{O}_1, \ldots, \mathcal{O}_n \}$ and $\mathcal{S}_i \subseteq \mathcal{O}_i$ are slices, then we call $\{ \mathcal{S}_1, \ldots, \mathcal{S}_n \}$ a slice of $R$.

$R$ said balanced if all closed actions appear both as positive and negative.

It is an easy remark that the core task of the LAM is actually to calculate a balanced slice of a cut-net (if we were not interested in the normal form, but only in the result of normalization, we could forget the path).

It is immediate that

Proposition 2.2.5 A closed cut-net converges iff it contains a balanced slice.

Moreover, Girard has proved the following important result:

Proposition 2.2.6 (Girard). If the cut-net $R$ is a balanced slice, normalization uses all its actions.

We are going to use this result in the following.

Observe that to be balanced is not necessary for a cut-net to normalize. The key point is that the part of the net visited by the normalization is balanced. Each time we enter a positive action, we should be able to exit in the corresponding negative action. If we enter a positive action for which there is no corresponding exit action, the algorithm diverges. A negative action to which does not correspond an entrance is much less dramatic: it simply will never be accessed.

2.3 Disputes and chronicles extraction

We have seen that normalization can be presented by a token traveling around the cut-net. Each token drafts a path. Each path is a chronicle of the normal form, as soon as we hide the closed actions. That is why to calculate the normal form we only record the open actions.

Anyway, 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 it converges, the normal form reserves no surprise: it is $\frac{F}{\top}$ What is interesting is the interaction itself, that is in the sequence of actions that have actually been visited during the normalization.

Definition 2.3.1 (Disputes) We call normalization path the sequence of actions visited during the normalization of a cut-net.

We call dispute the sequence of actions visited during the normalization of a closed net. If the net is $\{ \mathcal{O}, \mathcal{E} \}$, we indicate the dispute by $[\mathcal{O} = \mathcal{E}]$.

Notice that we reserve the name dispute to the closed case.
Definition 2.3.2 We indicate by $\text{Paths}(R)$ the collection of all normalization paths on $R$. Observe that if $R$ is closed, $\text{Paths}(R)$ is a singleton.

Fact 2.3.3 (LAM) It is immediate to modify the abstract machine $LAM_0$ into a machine $LAM$ that keeps track of all the visited actions. When the machine terminates we then have that $LAM(R) = \text{Paths}(R)$. To obtain the normal form we need to hide the close actions.

The part of a cut-net actually used in the normalization, that is the part that is covered by the path, is itself a cut-net. In Locus Solum, the minimal $R_0 \subseteq R$ s.t. $[R_0] = [R]$ is called the pull-back of $[R]$ along $R$. With a slight abuse of terminology we define

Definition 2.3.4 (Pull-back) Let $R$ be a cut-net, and let its normalization produce the dispute $p$. We call pull-back of $p$ the $R_0 \subseteq R$ whose actions are visited during the normalization.

For each design $D \in R$, we also say that $p$ covers $D_0 \subseteq D$ if $p$ visits all the actions in $D_0$.

In the next section we will show that the pull-back of a dispute does not depend on the cut-net that generated it.

2.3.1 Extracting chronicles from a path

In a design, the same action may appear several times, because of the use of n-ary negative rules (additives!). What allows us to identify a specific occurrence of an action $\kappa$ is the chronicle that leads us from the base to that action.

As we shall see, the normalization path allows us to retrieve the chronicle that identifies any of its actions.

The key here is an operation that arises naturally in this context: we invert the process of constructing the path. However, anyone familiar with Games Semantics will recognize in it a well known operation of HO games, the view. The notion of view is relative to a player, or to a parity in our setting. Let us recall some notions:

**Parity** Remember that the space of addresses, and thus of actions, is split into two parties: Even and Odd, accordingly to the length of the address. At this point it becomes convenient to use paritary bases, that is bases where all addresses on the right-hand side (positive) have the same parity, opposite to that of the address on the left-hand side. A base has the same parity (even or odd) as the addresses on its positive side, and opposite parity w.r.t. the address on the negative side. The empty base $\perp$ is fixed positive. A design is even or odd accordingly to its base. An action is even or odd accordingly to its focus.
Observe that also any cut-net \( \{D_i\} \) splits into two components: the collection of even designs \((\mathcal{D}_E)\), and the collection of odd designs \((\mathcal{D}_O)\). Hence we can write \( \mathcal{R} \) as \( \{(\mathcal{D}_E), (\mathcal{D}_O)\} \). We extend the notation for disputes to this case, writing \([(\mathcal{D}_E) \equiv (\mathcal{D}_O)]\).

**Polarity** Remind that the polarity (positive, negative) of the addresses (and hence of the actions) in a design is relative to the parity (even, odd) of the base: positive means same parity, negative means opposite parity. An even address is positive even-wise, and negative odd-wise.

**Notation 2.3.5** When we need to specify a player (Even or Odd) but do not wish to commit ourselves, we use the variable \( X \), for \( X \) either Even or Odd, and \( \overline{X} \) for the dual.

To explicit if an action \( \kappa \) is \( E \), \( O \), positive or negative we use the notation: \( \kappa^E, \kappa^O, \kappa^+, \kappa^- \).

When an action (or a base, or a design) has parity Even (Odd) we also say that it belongs to Even (Odd).

Let us define the function \( \text{view} \). Observe that the actions of a normalization path have a parity (even/odd) but are not polarized in themselves. They have a polarity (positive/negative) w.r.t. a fixed player. If \( \kappa \) belongs to \( X \), it is \( X \)-positive and \( \overline{X} \)-negative.

**Definition 2.3.6 (Views)** Let \( p \in \text{Paths}(\{\mathcal{D}, \mathcal{E}\}) \) and \( q \subseteq p \). W.r.t. either player, all addresses have a polarity. Let us fix a player \( X \), either Even or Odd. Its view \( \gamma q^X \) on \( q \) is inductively defined by:

- \( \gamma e^X = e \);
- \( \gamma s^X \kappa = \gamma s^X \kappa \) if \( \kappa \) is positive w.r.t. \( X \);
- \( \gamma s(\xi, I)^X = (\xi, I) \) if \( \xi \) is \( X \)-negative and belongs to a base (i.e. \( \xi \) is not sub-address of any address in \( q \));
- \( \gamma s(\xi, I) t(\xi, K)^X = \gamma s^X(\xi, I)(\xi, K) \) if \( (\xi, K) \) is \( X \)-negative.

We indicate the Odd view by \( \gamma q^O \) and the Even view by \( \gamma q^E \).

It is convenient to adopt the following convention:

by \( \gamma q^\kappa^+ \) we mean the view of the player for which \( \kappa \) is positive. If \( \kappa \) belongs to \( X \), then \( \gamma q^\kappa^+ = \gamma q^\kappa^X \) and \( \gamma q^\kappa^- = \gamma q^\kappa^\overline{X} \).

The notion of view easily generalize to any \( p = [(\mathcal{D}_E) \equiv (\mathcal{D}_O)] \).
Lemma 2.3.7 Let \( \mathcal{E}_i \) be a cut-net of slices, and \( p \) a normalization path on it. If \( \kappa \in p \) and \( \kappa' < \kappa \) in one of the slices, then \( \kappa' < \kappa \) in \( p \). Thus in particular:

(i) if \( \kappa \in \mathcal{E}_i \) appears in \( p \), any action that precedes \( \kappa \) in a chronicle \( \mathcal{C}_k \) is visited by \( p \) before \( \kappa \);

(ii) for any \( q \subseteq p \) the part of each slice covered by \( q \) is a tree which contains the root.

Proof. By induction on the length of \( q \kappa q \subseteq p \). For \( \kappa_n \) either open or positive the result is immediate by induction, because the action that precedes \( \kappa_n \) in the slice also precedes \( \kappa_n \) in the path. Let us consider the negative closed action \( \kappa^{-}_n \), assuming \( \kappa^{-} \in \mathcal{E}_i \). Either the focus belongs to the base of the design, and in such a case there is nothing to prove, or \( \kappa_n = (\xi_i, J) \) and \( (\xi, I)^+ < (\xi_i, J) \) in \( \mathcal{E}_i \). In an other slice \( \mathcal{E}_j \), \( \xi_i \) is a positive address which has been produced by an action \( (\xi, I') \) s.t. \( (\xi, I') < (\xi_i, J)^+ \) in \( \mathcal{E}_j \). Thus \( (\xi, I') < (\xi_i, J) \) in \( p \), and \( I = I' \).

\[ \square \]

Remark 2.3.8 Designs only communicates along shared addresses. Acyclicity of the cut-net implies that if normalization moves from the design \( \mathcal{D} \) to the design \( \mathcal{E} \), then the normalization path can only enter again \( \mathcal{D} \) from \( \mathcal{E} \).

Notation 2.3.9 We indicate the sequence \( q \kappa q \subseteq p \) by \( p \leq_{k} \).

Let \( \mathcal{R}_0 \) be a cut-net of slices and \( p \in Paths(\mathcal{R}_0) \). An action \( \kappa \) occurs twice if it is closed, once if it is open. Each occurrence appears in a specific position, identified by a chronicle. Then

Proposition 2.3.10 (Chronicles extraction) Let \( \mathcal{R}_0 \) be a cut-net of slices, \( p \in Paths(\mathcal{R}_0) \) and \( q \kappa q \subseteq p \) (we indicate \( q \kappa q \subseteq p \)). Assume \( \kappa \) has parity \( X \). If \( \kappa \) appears positive in \( \mathcal{E}_0 \), the chronicle that identifies \( \kappa^+ \) is \( \tau q \kappa \gamma^+ \) (that is \( \tau q \kappa \gamma^X \)). If \( \kappa \) appears negative in \( \mathcal{R}_0 \), then the chronicle that identifies it is \( \tau q \kappa \gamma^- \) (that is \( \tau q \kappa \gamma^X \)).

Notice that an open action \( \kappa \) will appear in \( \mathcal{R}_0 \) either positive or negative. Proof. The proof is by induction on the length of \( q \kappa q \subseteq p \). If \( \kappa \) is an open action then the action that precedes \( \kappa \) in the appropriate chronicle. Let \( q = q' \eta \), and assume \( \kappa \) is an even action (its focus is an address of even length). If \( \kappa \) is positive, the chronicle associated with \( \kappa \) is \( \tau q' \eta \gamma^E \kappa \), where \( \tau q' \eta \gamma^E \) is by induction the chronicle associated to \( \eta^- \); if \( \kappa \) is negative, the chronicle associated with \( \kappa \) is \( \tau q' \eta \gamma^O \kappa \), where \( \tau q' \eta \gamma^O \) is by induction the chronicle associated to \( \eta^+ \) and \( \tau q' \eta \gamma^O \) is \( \tau q' \eta \gamma^O \kappa \), because the focus of \( \kappa \) is generated by \( \eta \).

If \( \kappa \) is closed, the chronicle associated to the positive action \( \kappa^+ \) is \( \tau q \gamma^E \kappa \), as above. The chronicle associated to the negative action \( \kappa^- = (\xi, J)^- \) is of the form \( c(\xi, I)^+ (\xi, J)^- \), where \( (\xi, I) < (\xi, J)^- \) in \( p \), by Lemma 2.3.7. Hence \( \tau q \kappa \gamma^O = \tau q \kappa \gamma^O (\xi, I) \) and \( \tau q \kappa \gamma^O \).
2.3.2 Applications

Proposition 2.3.10 has immediate consequences which we develop in the next sections:

1. We can *generalize* LAM normalization to all cut-nets, without need to select a slice: the path allows us to calculate the next position of the token.

2. Given a dispute \( p = [\mathcal{D} = \mathcal{E}] \) we can *calculate the pull back*, that is the minimal cut-net \( \mathcal{G}, \mathcal{T} \) such that \([\mathcal{G}, \mathcal{T}] = p\).

3. We can establish a closure principle for normalization paths.

2.4 LAM\(_+\): generalized version

The normalization procedure given in Section 2.2 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 \( \sigma^+ \) to \( \sigma^- \). Moreover, it would not be feasible if we were not working by slices (though we can always work “by slices”).

Observe that if we do not work with slices, in a design the same action may appear several times, because of additive duplications. However, the sequence of visited actions carries all information needed to retrieve the position of any its action (Proposition 2.3.10). In particular, when we enter a positive action \( \kappa^+ \) we are able to retrieve the chronicle that identifies the negative action \( \kappa^- \) to which we have to move. Assume \( p \) is the sequence of actions we have visited so far, and we enter \( \sigma^+ \). We then move to the action \( \kappa^- \) identified by the chronicle \( \mathcal{d} = \gamma_{p_{\kappa^+}^{-}} \).

A *token* is now given by a pair \( (\epsilon, [p]) \). The first element \( \epsilon \) is a chronicle that represents the current position of the token. The second element, is a sequence of actions, that faithfully represents the path followed by the token: each time the token enters an action (open or closed) that action is copied in the list.

Given a design \( \mathcal{D} \), and a chronicle \( \epsilon \eta \in \mathcal{D} \), \( \text{Succ}(\epsilon \eta) \) indicate the actions which extend \( \epsilon \eta \) in \( \mathcal{D} \).

**Definition of LAM\(_+\)**

Let \( S \) be a set of tokens (multi-token).

**Initialization.**

If \( \text{First}(\mathcal{A}) \neq \emptyset \) then \( S = \{ \kappa[\epsilon] : \kappa \in \text{First}(\mathcal{A}) \} \)
else \( S = \{ \text{Halt}[\epsilon] \} \)

**Transitions.**

Let \( t \in S \) and \( t = \epsilon \eta[p] \).

If the action \( \eta \) is open then
\{t\} \rightarrow \begin{cases} \{\kappa[p, \eta] : \kappa \in \text{Succ}(\eta)\} & \text{if } \text{Succ}(\eta) \neq \emptyset \\ \{\text{Halt}[p, \eta]\} & \text{if } \text{Succ}(\eta) = \emptyset \end{cases} \quad (2.3)

If the action \(\eta\) is closed, then \(\eta\) is necessarily positive, and we proceed moving to the negative action \(\eta^-\) identified by the chronicle \(\delta = \uparrow p\eta^-\).

\{t\} \rightarrow \begin{cases} \{\emptyset[k] : k \in \text{Succ}(\emptyset)\} & \text{if } \emptyset = \uparrow p\eta^- \in \mathcal{R} \\ \{\text{Halt}[p, \Omega]\} & \text{if } \uparrow p\eta^- \notin \mathcal{R} \end{cases} \quad (2.4)

**Multi-token rule.**

Let \(S = S_1 \sqcup S_2\) then:

\[
\frac{S_1 \rightarrow S'_1}{S \sqcup S_2 \rightarrow S'_1 \sqcup S_2}
\]

**Termination.**

The computation halts when all the tokens \(t_i\) in \(S = \{t_1, \ldots, t_n\}\) are of the form \(\text{Halt}[p_i]\). If \(\text{Halt}[p, \Omega] \in S\) we say that the computation fails, otherwise that it succeeds.

**Result.**

Let \(\mathcal{R}\) b a cut-net on which we execute the machine above. The normal form is given by the closure under prefix of \(\{\text{hide}(p) : \text{Halt}[p]\}\), where \(\text{hide}(p)\) is \(p\) where we have deleted (hidden) all closed actions.

**Proposition 2.4.1 (Normalization of designs)** \(\text{LAM}^+_\mathcal{R}(\mathcal{R}) = [\mathcal{R}]\).

More precisely, given a “sequent calculus” design \(D\), let us indicate by \(\text{Dess}(D)\) the associated design (dessein). We have that:

\[
\text{LAM}^+_\mathcal{R}(\text{Dess}(D_1), \ldots, \text{Dess}(D_n)) = \text{Dess}([D_1, \ldots, D_n])
\]

where \([D_1, \ldots, D_n]\) is the normal form calculated in the sequent calculus fashion.

### 2.5 Calculating the pull-back

The normalization of a closed cut-net produces a dispute. If we are given a dispute, we can calculate the minimal cut-net that produces it, the pull-back of \(p\). We indicate this operation by \(\text{Pull}(p)\).

**Definition 2.5.1 (Pull(p))** Let \(p = [\mathcal{D} = \mathcal{E}]\). \(\text{Pull}^E(p)\) is defined as follows:

1. calculate \(\uparrow p\mathcal{E} \sqcup \{\uparrow q\mathcal{E} : q \sqsupset p, q \neq \mathcal{E}\}\)
2. complete with \(\dagger\) all maximal views which terminate with a negative action. Observe that there is at most one such view.

\(\text{Pull}^O(p)\) is defined symmetrically. \(\text{Pull}(p) = \{\text{Pull}^E, \text{Pull}^O\}\).
It is immediate, and it is important to notice, that

**Lemma 2.5.2**  Pull\((p)\) only depends on \(p\). Thus for any cut-net \(R\), the normalization produces the dispute \(p\) iff \(\text{Pull}(p) \subseteq R\).

As a consequence

**Proposition 2.5.3**  Given a cut-net \(R\) whose normalization produces the dispute \(p\), Pull\((p)\) is the pull-back of \(p\).

**Remark 2.5.4**  It is easy to extend the definition above to any closed cut net \(R\). 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. Observing that the pull-back of \(p\) is a net of slices, and that each slice has an unique root, we consider the maximal subsets of Pull\(^E\)(\(p\)) (resp. Pull\(^O\)(\(p\))) of chronicles with the same first action. Let us denote them by \(S_1, \ldots, S_n (\xi_1, \ldots, \xi_m)\). Then \(\text{Pull}(p) = \{(S_i), (\xi_j)\}\).

**Notation 2.5.5**  \((\text{Pull}^+, \text{Pull}^-)\) Sometimes we have a natural point of view, for example when we work with a fixed \(D\) and study the disputes \(p = [\mathcal{E} = \mathcal{E}], \) for all \(\mathcal{E} \in D\). In such a case, we will use the notation Pull\(^+\)(\(p\)) for the component of Pull\((p)\) with the same parity as \(D\), and Pull\(^-\)(\(p\)) for the opposite.

**Remark 2.5.6**  \((\text{Opp}_+)\) It is worth observing that when one calculates \(\text{Opp}_+\), one actually calculates Pull\(^-\)(\(c\)).

### 2.5.1 Paths and disputes (closure principle)

We reserve the name dispute to the (unique) normalization path generated by a closed net. However, any normalization path of a cut-net \(R\) can be seen as a dispute. More precisely, we can complete \(R\) into a closed net \(R_p \supseteq R\) such that \(p\) is the dispute generated by the normalization of \(R_p\). This establishes a closure principle for normalization paths, in the sense that to study properties of normalization paths it is enough to study the properties of disputes (i.e. paths on closed nets).

**Proposition 2.5.7**  If \(R\) is a cut-net and \(p \in \text{Paths}(R)\) then there a is closed net \(R'\) such that \(R \subseteq R'\) and \(p\) is the dispute generated by the normalization of \(R'\).

**Proof.**

Both Pull\(^+\)(\(p\)) and Pull\(^-\)(\(p\)) are slices. Observe that the actions \(\kappa\) such that \(\kappa^- \in R, \kappa^+ \not\in R\) will appear also as positive in Pull\((p)\). The essential point is to verify that the newly created chronicles \(\partial \kappa^+\) (those corresponding to actions \(\kappa\) which are open in \(R\)) satisfy the sub-address condition on positive focus. This is easy (for the same argument as in the Separation Theorem): on open actions \(p\) is going
upwards. Therefore if \((\xi, K)^+ \in \text{Pull}(p)\) is an action which does not belong to \(\mathcal{R}\), then \((\xi, K)^-\) is open and negative, and in the normalization path it is immediately proceeded by \((\xi, I)\), because normalization is going upwards. Therefore we have 
\[
\delta((\xi, K)^+) = \text{Pull}'((\xi, I))^- (\xi, I)^+ = \delta'(x^i, I)^- (\xi, K)^+.
\]

We postpone the other details of the proof to Chapter 6. They are not difficult, but will be better dealt with using--once for all--the tools we develop in that chapter. The idea is as follows:

\(p\) is a dispute, in fact the dispute generated by \(\text{Pull}(p)\).

The pull-back of \([\mathcal{R}]\) is included in \(\text{Pull}(p)\), and \(\text{Pull}(p) \cup \mathcal{R}\) is also a cut-net.

We can then consider \(\mathcal{R}_p = \text{Pull}(p) \cup \mathcal{R}\). This is again a closed net, producing the dispute \(p\).

\(\square\)

## 2.6 Discussion

There are two natural directions that arise. We will develop them in the remaining chapters.

1. **Characterize what can be observed interactively.** Can we characterize the (partial) designs that can be covered with a path? This means to characterize the designs used at each single run of normalization, or the designs that are the pull-back of a dispute. (Chapter 3 and 4)

2. **Study the interactions as primitive.** Can we characterize the paths themselves, or rather the sequences of actions representing the interaction of two designs? This will build a bridge with Games Semantics. (Chapter 6)

### 2.6.1 Related work

Our normalization on designs (rather than on the sequent calculus) is analogous to the order quotient defined in [Gir01b], though it was developed independently. Our approach is more local, hence easier to use for actual computations. On the other hand, Girard’s theory provides a synthetic view, in which better fit the development of general results.

The presentation of composition of designs by the Loci Abstract Machine is reminiscent of the analysis of composition of strategies (in HO and AJM games settings) in terms of abstract machines carried out in [DHR96].

The notion of desseins is very close to that of abstract Bhôm 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 with the abstract machines studied by Curien and Herbelin in [CH98]. Our generalized LAM is actually an instance of View abstract machine, introduced by Coquand in [Coq95].
2.6.2 Focalization

In Chapter 1 we recalled that the key points on which desseins are based are: Actions/Addresses, Contexts retrieved dynamically, Focalization. Normalization exploits all of them. We would like in particular to stress that if focalization is a well-established tool for effective proof-search, it also appears as a tool for effective cut-elimination (see also [Coq95], [CH98]).
Chapter 3

Visits of Designs and Interactive Observability

In this Chapter we study fundamental properties of the interaction between designs as pure (untyped) objects. The main outcome is a characterization of the designs that 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.

The best example of this is the Separation Theorem: we separate two designs only if we can interactively separate them. In fact, if the two designs differ on a chronicle $c$, we can build the counter-design $Opp_c$, whose purpose is exactly to visit all the actions of $c$. Given a slice $\mathcal{E}$, can we follow the same idea and build a counter-design $Opp_{\mathcal{E}}$ such that the normalization visits all the actions?

To study this question corresponds to study what can be interactively recognized in Ludics. We observe, in particular, that is not possible to interactively detect the use of weakening (Section 3.2).

In this chapter:

1. We study geometrical properties of the normalization paths, in the style of Geometry of Interaction. These properties represent “technical tools,” which we are going to use in other parts of this Thesis.

2. Given a slice $\mathcal{E}$, we can associate to it the prefix tree of the loci that appear in $\mathcal{E}$. Conversely, given a prefix tree, can we associate a slice to it? This question is closely connected to the study of which design we can visit, and in fact underlies it: given a slice $\mathcal{E}$, to build a counter-design that explores it is equivalent build a counter-design which uses the same actions as $\mathcal{E}$ (with opposite polarity).

3. We study the question of which designs, or part of them, we can observe
interactively. The designs that can be explored in a test (in a single run of normalization) represent the primitive units of observability. Though one could think that it is enough to work by slices, we show that the answer is no (the definition of slice given in [Gir01b] is “external,” as we shall make precise in Section 3.2). Thus we provide a characterization of those designs that can be explored through a single counter-design.

There are several intuitions that correspond to these designs:
- the design we can visit in a single run of normalization, and that in this sense are “primitive observable”;
- the part of a design that can be visited;
- the pull-back of disputes;
- designs whose actions can be organized in a counter-design. This means to characterize, modulo sequentiality, the “slices” that can be in the intersection of two designs.

Plan
In Section 1 we make precise the notion of visit of a design, and describe how to build a counter-design which realizes a given visit. Section 2 introduces and discusses the question of observability. We then develop two approaches to the study of interaction between designs. We could respectively qualify them as “dynamic” (Section 3) and “static” (Section 4). In Section 3 we study the geometrical properties of a visit, in the style of GoI. In Section 4 we study the space of addresses (the prefix tree) associated to a design. In Section 5 we relate the results of the previous sections, to give a characterization of observability for designs.

Notations and conventions
< indicates the order between actions in a sequence, either a chronicle or a dispute. Given a normalization path p, we write \( \alpha <_p \beta \) if \( p = p_1 \alpha p_2 \beta p_3 \). Given a slice \( \mathcal{S} \) we write \( \alpha <_{\mathcal{S}} \beta \) if for a chronicle \( \epsilon \) in \( \mathcal{S} \) we have that \( \epsilon = \epsilon_1 \alpha \epsilon_2 \beta \epsilon_3 \). The terms “before” and “after” refer to the < relation.

\( \sqsubseteq \) is the prefix order between addresses. The terms “child” and “parent” refer to the “immediate sub-address” relation. We indicate by \( \sigma \ast \) a generic suffix of \( \sigma \).

When working with slices, each focus uniquely determines an action, thus we often identify an action \((\xi, I)\) with its focus \(\xi\).

To avoid unnecessary complications, most of the time we consider design on an atomic base (i.e. \( \vdash \xi \) or \( \xi \vdash \)). The result are easily generalized.

In this chapter we are going to manipulate designs (more precisely slices) focusing on their structure of trees. It is convenient do define once for all the notion of subtree:
Definition 3.0.1 (Sub-designs, sub-slices) If the locus ξ occurs in the slice $\mathcal{S}$ (in the design $\mathcal{D}$), then the subtree induced by $\mathcal{S}$ (by $\mathcal{D}$) above ξ is a slice (design) of root ξ, that we indicate by $\mathcal{S}_\xi$ ($\mathcal{D}_\xi$) and call a sub-slice (a sub-design).

3.1 Visits and paths

The visit of a tree converts it into a linearly ordered structure (a list). Let us remind the two main traversal strategies for trees (non necessarily binary):

*Preorder traversal:* First visit the root, then traverse each of the subtrees in preorder.

*Postorder traversal:* First traverse the left and right subtrees in postorder and then visit the root.

A slice is a tree. Normalization induces a traversal strategy for a slice. When we speak of visiting a slice (or a design), we always intend visiting it by normalizing. We call the sequence of visited actions a normalization path.

This visit must obey geometrical rules we analyze in the next sections. Because of these constraints, it is not always possible to visit a slice.

Definition 3.1.1 A slice $\mathcal{S}$ admits a visit if it is possible to normalize it against a counter-design in such a way that the normalization path visits all the actions of $\mathcal{S}$.

3.1.1 Computing a counter-design that realizes a path

We have already seen that given a closed net $\{\mathcal{D}, \mathcal{E}\}$, the normalization produces a path (the dispute) on each of the designs. The part of the cut-net which is actually visited is called the pull-back. Given a dispute, we are able to recover the pull-back.

Another way to use the LAM “the other way round”: given a slice and a path on it, calculate a counter-design that realizes the path. Let $\{\mathcal{S}_i\}$ be a cut-net of slices, and $p$ a normalization path on it. It is immediate that for any $q \sqsubseteq p$ the part of each slice covered by $q$ is a tree which contains the root. Let us define a notion of path (candidate to be a normalization path):
Definition 3.1.2 A path \( p \) on a slice \( \mathcal{S} \) is a sequence of actions such that for any \( p'/\subseteq p \) the region of \( \mathcal{S} \) covered by \( p' \) contains the root and is a tree.

Suppose we freely draft a connected path on a slice, and want to build a counter-design that realizes it. If we know that the path is a dispute, we could calculate the pull-back. The characterization of dispute we give in Chapter 6 will allows us to proceed in this way. Otherwise, we can build the counter-design "by hand". To build the counter-design by hand is useful in practice, and helps intuition.

**Procedure.** Assume we have a slice \( \mathcal{S} \) and a path \( p = \kappa_0, \ldots, \kappa_n \) on it. Our aim is to build a counter-design \( \mathcal{I} \) such that \( [\mathcal{S} = \mathcal{I}] \) is \( p \). We focus the discussion on the case of \( \mathcal{S} \) having atomic base (\( \vdash \xi \) or \( \xi \vdash \)); the general case (\( \mathcal{S} \) has base \( \Xi \vdash \Lambda \)) is similar.

The base of \( \mathcal{I} \) is determined, being opposite to the base of \( \mathcal{S} \). To build \( \mathcal{I} \), we progressively place the actions of \( p \) to form a tree.

The polarity of the actions in \( \mathcal{I} \) is opposite to that in \( \mathcal{S} \), as is the polarity of the base. If \( \kappa_i \) is negative in \( \mathcal{I} \), there is no ambiguity on where to place it: either it is the root, or it is of the form \( \xi \top \), and we place it just after \( \xi \) (which is positive). If \( \kappa_{i+1} \) is positive in \( \mathcal{I} \), we need to place it just after \( \kappa_i \) (which is negative in \( \mathcal{I} \)). In fact once the normalization is on a positive action \( \kappa_i^+ \) in \( \mathcal{S} \), it moves to the negative action \( \kappa_i^- \) in \( \mathcal{I} \), and then to \( \text{Succ}(\kappa_i^-) \), which is \( \kappa_{i+1} \).

It is immediate that at any stage in \( \mathcal{I} \) there is at most one maximal branch terminating with a negative action. If \( \kappa_n \), the last action of \( p \), is negative in \( \mathcal{I} \), we complete \( \mathcal{I} \) with a daimon (\( \top \)) after \( \kappa_n \). By construction, the normalization applied to \( \{\mathcal{S}, \mathcal{I}\} \) produces \( p \). We need to check that the tree we build is actually a design. The only property that is not guaranteed by the construction is that of subaddress on positive focus. (see the definition of designs in Chapter 1).

### 3.2 What can be interactively observed?

Which part of a design can be visited during normalization? Normalization is always carried out in a single slice: when a negative rule has several premises (think of \&), the opponent (think of \oplus) selects one of them. Being a slice is thus a first, obvious condition for a design to admit a visit. 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:
Example 3.2.1

\[ \xi_1 \quad \xi_2 \quad 1, 2 \]

\[ \mathcal{G} : (\xi, \sigma, \tau) \]

Such a design corresponds to a purely multiplicative structure. In fact we can easily type it. For example we can let \( F(\xi) = F(\xi_1) \otimes F(\xi_2) \), \( F(\langle \rangle) = F(\xi) \otimes F(\sigma) \otimes F(\tau) \), where by \( F(*) \) we indicate the formula associated to the address *_. The result has then the following form:

Let us build a counter-design to explore the slice of example 3.2.1. The path will start with \( \langle \rangle \), move to \( \xi \), 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 \( \sigma \) we are forced to stop, because there is no way to move to the other branch.

The counter-design we have used is the following one:

The corresponding path is \( \langle \rangle, \xi, \xi_1, \sigma \), while the path we would like to have is: \( \langle \rangle, \xi, \xi_1, \sigma, \xi_2, \tau \). This would demand having \( \xi_2 \) after \( \sigma \) in the counter-design. This is against the constraint of positive focus, which demands to have \( \xi < \xi_1 \):
3.2.1 Consequences

The fact that the two branches (the two subtrees) in $\mathfrak{S}$ behave as independent components affects what can be interactively recognized. An immediate consequence is that we cannot interactively detect the use of weakening in a slice (cf. [Gir01b], pag. 85). Consider again Example 3.2.1, assuming that the root is the action $(\xi, \{\xi, \sigma, \tau, \lambda\})$. The root creates a forth address, $\lambda$, which is never used. However, we cannot interactively detect that $\lambda$ is weakened. Either we explore the left branch, or the right one. In the first case we see that $\sigma$ is used. The other addresses, $\tau$ and $\lambda$, are possibly used after $\xi_2$. In the second case we see that $\tau$ is used, $\sigma$ and $\lambda$ being possibly used after $\xi_1$.

It is interesting to take a look at the sequent calculus counterpart what we see at each test. Consider, for example, the first one. What we observe is the following slice:

\[
\begin{array}{c}
\sigma J \\
\xi_1 I \\
\xi 1,2 \\
\xi, \sigma, \tau, \lambda
\end{array}
\]

It translates into the following derivation:

\[
\frac{\sigma j \vdash (\sigma, J)}{} \\
\frac{\xi_1 \vdash \sigma (\xi_1, I)}{} \\
\frac{\xi_2 \vdash (\xi_2, \emptyset)}{} \\
\frac{\xi, \sigma, \tau, \lambda \vdash (\xi, \{1, 2\})}{\emptyset \vdash (\{\xi, \sigma, \tau, \lambda\})}
\]

The point to be noticed is the premise $(\xi_2, \emptyset)$ which actually corresponds to a subdesign still to be developed. Therefore, for what we know, we can reasonable assume that $\tau$ and $\lambda$ will be used above it.
3.2.2 Some comments

What is striking in Example 3.2.1 is that there is no communication between the two subtrees (the two branches) of the slice \( \mathcal{S} \). They behave as distinct components. This goes against the intuition on slices which comes from Linear Logic (remember to think 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, as defined in [Gir01b], this idea to be “connected” by the normalization does not longer hold. The definition of slice is “external.” For this reason, we are going to call strong a slice that can actually be visited in a single normalization. There is a communication among all the actions, in the sense that all the actions can be connected by a normalization path.

Definition 3.2.2 (strong slices) A slice \( \mathcal{S} \) is strong if it is possible to normalize it against a counter-design in such a way that the normalization path visits all the actions of \( \mathcal{S} \).

Observe that

Fact 3.2.3 If a slice is strong, there is a counter-design which uses the same actions, with opposite polarity.

We will develop this idea in Section 3.4.

3.3 Traveling on designs

In this section we study the geometrical properties of a visit generated by normalization. Our point of view here is very concrete and operational. The main result is Proposition 3.3.4.

Notation

Given two addresses \( \alpha, \beta \), we indicate by \( \alpha \wedge \beta \) the longest common prefix.

Given two chronicles \( \xi_1, \xi_2 \) in a slice \( \mathcal{S} \), we indicate by \( \xi_1 \wedge_{\mathcal{S}} \xi_2 \) the longest common prefix (as chronicle).

Any action in a slice identifies a chronicle. Given a slice \( \mathcal{S} \) and two actions \( \kappa_1, \kappa_2 \) which belong to distinct chronicles, we indicate by \( \kappa_1 \wedge_{\mathcal{S}} \kappa_2 \) the node on which the two chronicles are branching.

Remark 3.3.1 Given a slice \( \mathcal{S} \), two actions \( \kappa_1, \kappa_2 \) in two different chronicles determine a positive node \( (\xi, I) \) where the two chronicles are branching. Thus \( \kappa_1 \in \mathcal{S}_{\xi_i}, \kappa_2 \in \mathcal{S}_{\xi_j}, i \neq j \).
We isolate two easy properties of slices that will be of frequent use.

**Lemma 3.3.2** Given two actions \((\sigma, I), (\tau, J)\) belonging to the same chronicle, either the addresses \(\sigma, \tau\) are comparable \((\sigma \sqsubseteq \tau\) or \(\tau \sqsubseteq \sigma\)) or \(\sigma \land \tau\) is a negative address.

**Proof.** Suppose \(\sigma, \tau\) incomparable. Let \(\sigma = \eta_i \sigma'\) and \(\tau = \eta_j \tau'\), where \(i \neq j\). If \(\eta\) is positive, then \(\eta_i\) and \(\eta_j\) belong to two distinct chronicles. The fact that \(\eta_i < \sigma\), while \(\eta_j < \tau\) is thus against the hypothesis that \(\sigma, \tau\) belong to the same chronicle. \(\square\)

**Lemma 3.3.3** Given two actions \((\sigma, I), (\tau, J)\) in two distinct chronicles of a slice \(\mathcal{S}\), if \(\sigma \land \tau\) is positive, then \(\sigma \land_{\mathcal{S}} \tau = \sigma \land \tau\).

**Proof.** Let \(\sigma \land_{\mathcal{S}} \tau = \xi\), and let \(\sigma = \eta_i \sigma'\) and \(\tau = \eta_j \tau'\), where \(i \neq j\). Therefore \(\eta = \sigma \land \tau\) is either \(\xi\), or it has been used as focus before \(\xi\). 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. \(\square\)

Let us consider the normalization of a cut net. Let \(p = [\mathcal{S} = \mathcal{T}]\), where \(\mathcal{S}, \mathcal{T}\) are slices. The normalization traces a path on each slice. The path moves from one branch to another, entering and exiting the subtrees induced by each positive action. To fix ideas let us give a concrete example, where the superscribed indices indicate the order of visit:
The path induced by the two designs above is: \langle\langle, 1, 10, 101, 1010, 2, 20, 102\rangle. Notice how this path visits the first design \(\mathcal{G}\). The path exits \(\mathcal{G}_{10}\) in 1010, moves outside \(\mathcal{G}_{10}\) and then enters it again on 102.

Let us consider a positive action \((\xi, I)\) in a slice \(\mathcal{G}\). It induces some subtrees \(\mathcal{G}_{\xi_i}, (i \in I)\). In general the normalization path does not visit the slice one branch after the other. To move from \(\mathcal{G}_{\xi_i}\) to \(\mathcal{G}_{\xi_j}\) the path will exit \(\mathcal{G}_{\xi_i}\) on a positive action \(\alpha\), then possibly move around outside \(\mathcal{G}_{\xi_i}\), and then enter \(\mathcal{G}_{\xi_j}\) on a negative action \(\beta\).

Proposition 3.3.4 says that the path must leave \(\mathcal{G}_{\xi_i}\) and enter \(\mathcal{G}_{\xi_j}\) on a sub-address of \(\xi\). That is \(\alpha \wedge \beta = \alpha \wedge \beta\).

If \(p = p_1 \alpha p_2 \beta p_3\) we write \(\alpha <_p \beta\); we indicate \(p_2\) as \(|\alpha, \beta|\).

**Proposition 3.3.4** Let \(p = [\mathcal{G} = \mathcal{E}]\), where \(\mathcal{G}, \mathcal{E}\) are slices, and let \(\alpha, \beta\) be any two actions such that \(\alpha <_p \beta\). If in one of the two slices (let us indicate it by \(\mathcal{I}\)):

(i) \(\alpha, \beta\) belong to distinct chronicles and

(ii) \(|\alpha, \beta[ \cap \mathcal{I}_\alpha \wedge \mathcal{I}_\beta = \emptyset\)

then \(\alpha \wedge \beta = \alpha \wedge \beta\).

Notice that only one of the two slices \(\mathcal{G}, \mathcal{E}\) can satisfy the conditions. In fact the conditions imply that in the the slice \(\mathcal{I}\) \(\alpha\) is positive and \(\beta\) is negative, because the path exits in \(\alpha\) and enters in \(\beta\).

**Proof.**

It is enough to show that \(\alpha \wedge \beta\) is always positive in the slice \(\mathcal{I}\) in which we calculate \(\alpha \wedge \beta\). Then \(\alpha \wedge \beta = \alpha \wedge \beta\) by Lemma 3.3.

The proof is by induction on \(d(\alpha, \beta)\), the number of actions between \(\alpha\) and \(\beta\) in \(p\). Such a number is necessarily even.

Let \(d(\alpha, \beta) = 0\). In the slice \(\mathcal{I}\) (which is either \(\mathcal{G}\) or \(\mathcal{E}\)), when \(p\) moves from \(\alpha\) to \(\beta\) it changes of branch. This means that in the counter-design we must have a chronicle \(\alpha^- \beta^+\). By Lemma 3.3.2, \(\alpha \wedge \beta\) is negative in the counter-design, and then positive in \(\mathcal{I}\), where we calculate \(\alpha \wedge \beta\).

Let \(d(\alpha, \beta) = n \geq 2\). In order to simplify the notation, let assume that the conditions hold for the slice \(\mathcal{G}\) (\(\mathcal{I} = \mathcal{G}\)), and let \(\xi = \alpha \wedge \mathcal{G} \beta\).

We are in the situation illustrated by (a) in the picture below.
Notice that for any action $\tau \in |\alpha, \beta|$:

- $\tau \not\in S_\xi$, by hypothesis;

- $\tau \not< \xi$. In fact all the addresses below $\xi$ in $S$ are visited before $\xi$, which in turn is visited before $\alpha$ (because $\tau \prec_{\xi} \sigma$ implies $\tau \prec \sigma$).

- $\tau$ belongs (at least) to one of the subtrees induced by the nodes $\sigma_i < \xi$, that is the nodes between the root of $S$ and $\xi$.

Let us consider the positive actions $\sigma_i <_0 \xi$, that is the nodes below $\xi$ in $S$. Let $\sigma$ be the maximal node such that $\sigma <_0 \xi$ and $S_\sigma \cap |\alpha, \beta| \neq \emptyset$. Next figure illustrates this:
Notice that:

$\mathcal{G}_\xi \subseteq \mathcal{G}_{\sigma_i}$, for some $\sigma_i$ sub-address of $\sigma$, and $\mathcal{G}_{\sigma_i} \cap ]\alpha, \beta[ = \emptyset$, because of the choice of $\sigma$.

Let call $\gamma$ the first point of $]\alpha, \beta[$ that belongs to $\mathcal{G}_\sigma$, and let $\delta$ be the last point of $\mathcal{G}_\sigma$ appearing in $]\alpha, \beta[$.

We are then in the situation illustrated by (b.).

Since $\alpha \in \mathcal{G}_{\sigma_i}$, $\gamma \in \mathcal{G}_{\sigma_j}$, and $\mathcal{G}_\sigma \cap [\alpha, \gamma] = \emptyset$, we can apply the inductive hypothesis, obtaining that $\alpha = \sigma i \ast$, $\gamma = \sigma j \ast$. In a similar way, we have that $\delta = \sigma k \ast$ and $\beta = \sigma i \ast$.

We know then that $\sigma i \subseteq \alpha \land \beta$. If we now assume that $\alpha \land \beta$ is negative in $\mathcal{G}$, then in $\mathcal{E}$ $\alpha \land \beta$ is positive and, by Lemma 3.3.2 $\alpha, \beta$ belong to distinct subtrees of $\alpha \land \beta$. This means in particular that inside $]\alpha, \beta[$ the normalization path move from one subtree to the other. By the inductive hypothesis, the exit and the enter points should be sub-addresses of $\alpha \land \beta$ and then of $\sigma i$. But, $\mathcal{G}_{\sigma_i} \cap ]\alpha, \beta[ = \emptyset$, giving a contradiction.

\[\square\]

Proposition 3.3.4:

(i) establishes a relation between the chronicles order and the prefix order.

(ii) establishes a necessary condition a path must satisfy when moving from a branch to another; this reminds of analogous conditions on the tensor in Geometry of Interaction and Games Semantics.
3.4 Prefix Tree

Given a slice $\mathcal{S}$, we can associate to it the prefix tree of the addresses which appear in $\mathcal{S}$, indicated by $T(\mathcal{S})$.

**Definition 3.4.1 (Prefix tree)** Let us consider an address $\sigma$ and the tree corresponding to the prefix order on its sub-addresses. A prefix tree is a finite subtree of such a tree.

**Definition 3.4.2 (Prefix tree associated to a slice)** The prefix tree $T(\mathcal{S})$ associated to a slice $\mathcal{S}$ is the tree corresponding to the prefix order on the addresses which appear in the slice. We will indicate by $T_{\xi}$ the subtree of the prefix tree $T$ induced above $\xi$ (cf. Definition 3.0.1)

A natural question that arises is whether given a prefix tree, we can associate to it a slice using the same addresses. If this is possible, in general there will be several ways to do so, and thus several slices associated to a tree. Each one corresponds to a way of introducing sequentiality on the use of the addresses.

**Definition 3.4.3 (Slices associated to a prefix tree)** Given a prefix tree $T$, a slice associated to it is any slice $\mathcal{S}$ such that $T(\mathcal{S}) = T$.

To associate a slice to a prefix tree is not always possible. As a first example, let us consider the following prefix tree:

![Diagram of prefix tree](image)

There is no way to associate to it a negative design (that is a design of base $<>^\bot$), but we can associate a positive design (of base $^+<>$). In the latter case we have
In the former case we would have

\[ \begin{array}{c}
1 \\
\downarrow \\
\end{array} \quad \text{or} \quad \begin{array}{c}
2 \\
\downarrow \\
\end{array} \]

and no way (no space) to add a second positive action.

As a second example, let us consider the following tree

\[ \begin{array}{c}
01 \\
\downarrow \\
0 \\
\downarrow \\
1 \\
\downarrow \\
2 \\
\downarrow \\
3 \\
\end{array} \]

To this tree of addresses, it is not possible to associate neither a positive nor a negative design (you have to try).

In section 3.6 we characterize the prefix trees to which we can associate both a design and a counter-design. Observe that

**Proposition 3.4.4** If \( S \) is a strong slice, the counter-slice \( T \) which visits all the actions of \( S \) is itself strong and \( T(S) = T(T) \). Conversely, given a prefix tree \( T \), if we can associate to it both a slice \( S \) and a counter-slice \( T \) then \( S, T \) are strong slices.

**Proof.** The first proposition is immediate. The second one is consequence of Proposition 2.2.6 \( \blacksquare \)

It is easy to give a sufficient condition on prefix trees to guarantee we can associate a slice.

Given a prefix tree \( T \), the choice of a polarity for the base induces the polarization of the whole tree. A polarized prefix tree essentially is a slice with no sequentiality information. Prefix and chronicle trees coincide on the positive nodes, that in both case are followed by the immediate sub-addresses. However, a prefix tree also branches on negative addresses.

The easiest way to associate a slice to a prefix tree is to directly manipulate it: we break the prefix tree, and rebuild it as a slice. In practice, for any negative n-ary node, \( n \geq 2 \) we prune \( n - 1 \) positive subtrees and we graft each of them on top of a negative leaf.
3.4.1 Polarized trees

Definition 3.4.5 We call leaf a node of arity $n = 0$ and internal node a node of arity $n > 0$. We call branching node a node of arity $n \geq 2$.

Definition 3.4.6 A polarized tree is a tree whose nodes are alternatively labelled with opposite polarity. 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 $k$. Hence, for example, $N_0(T)$ is the number of negative leaves.

We will omit to explicit $T$ when not ambiguous.

The following proposition relates an easy to express property of a polarized tree as “$N \geq P$,” with a property on the number of negative leaves ($N_0$).

Lemma 3.4.7 Let $T$ be a polarized tree.
If the root is negative, the two following conditions are equivalent:
(i) $N(T) \geq P(T)$
(ii) $N_0 \geq N_2 + 2N_3 + \ldots + (k - 1)N_k + \ldots$

If the root is positive, the two following conditions are equivalent:
(i') $P(T) \geq N(T)$
(ii') $P_0 \geq P_2 + 2P_3 + \ldots + (k - 1)P_k + \ldots$

Proof. Easy counting. Let us assume the root is negative. How many positive nodes are there in $T$? 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:

$$N_0 + N_1 + N_2 + N_3 + \ldots + N_n \geq 0N_0 + N_1 + 2N_2 + 3N_3 + \ldots + nN_n$$

which reduces to (ii). \qed

Definition 3.4.8 Let us define the following operations on trees:
- Given a tree $T$ and a subtree $T_0$, $(T \setminus T_1)$ is the tree obtained from $T$ by removing $T_0$.
- Given two trees $T_1, T_2$ and a leaf $L$ of $T_1$, $\text{graft}(T_1, L, T_2)$ is the tree obtained from $T_1$ adding $T_2$ as subtree of $L$. 

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

**Definition 3.4.9 (Sub-address condition)** A tree of addresses satisfies the sub-address condition if $\xi \subseteq \xi'$ implies that $\xi'$ belongs to the subtree induced by $\xi$ ($\xi'$ is above $\xi$).

**Lemma 3.4.10** Let $T$ be a polarized tree of addresses.

(i) If $T$ satisfies the sub-addresses condition so does $(T \setminus T_0)$.

(ii) Assume $L$ is a leaf of $T$ with same polarity as $T$, and $T_i$ is an immediate subtree of $T$ not containing $L$. Then if $T$ satisfies the sub-address condition so does $\text{graft}((T \setminus T_i), L, T_i)$

### 3.4.2 From prefix trees to slices

A polarized prefix tree is “almost” a slice, except for the fact that it branches also on negative loci. Our aim is, for each $n$-ary negative branching, to prune the exceeding $n - 1$ positive subtree, and to graft them on top of a negative leaf. The condition that we give below ($\text{Neg}$) 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 the same than the positive focus condition on chronicles). Observe that such a condition is satisfied by any prefix tree.

**Definition 3.4.11** Given a prefix tree $T$, the choice of a base, and thus of a polarity for the root, induces the polarization of the tree. As usual, addresses of the same parity (even or odd length) have the same polarity (positive or negative), addresses of opposite parity have opposite polarity.

**Proposition 3.4.12** Let $T$ be a polarized prefix tree. If $T$ satisfies the following condition

$$\text{(Neg)} \quad N(T') \geq P(T') \text{ or, equivalently, } N_0(T') \geq \sum (i - 1)N_i(T')$$

then there is a slice $\mathcal{S}$ with the same polarity as $T$, such that $T$ is the prefix tree of $\mathcal{S}$: $T(\mathcal{S}) = T$.

**Proof.** We define an application $S$ mapping polarized trees of addresses which satisfy ($\text{Neg}$) 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 ($\text{Neg}$);
4. $S(T)$ branches only on positive nodes.
As a result, given a prefix tree $T$ which satisfies $(N eg)$, $S(T)$ 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 $(N eg)$ 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 $\geq 2$, condition (3) guarantees that it has at least a negative leaf. Let $L$ be such a leaf, and $T'_1$ an immediate subtree of $T'$ not containing $L$. Define $U = (T' \setminus T'_1)$ and $T'' = graft(U, L, T'_1)$.

We end by setting $S(T) = \Phi(T')$.

\hfill \Box

Notice that, in general, the way in which we can “sequentialize” a prefix tree into a slice is not unique.

### 3.5 Observability conditions

The designs that can be explored in a single run are the primitive “units” of observability. We give a characterization of them, using the results we established in the previous two sections.

Let us start by discussing the counter-example 3.2.1. The point with this example is that we need enough addresses if we want that both the design and the counter-design are able to develop the dialogue interacting on all the addresses. In our example there is not “enough space” to move from one branch to the other, since as we have seen (Proposition 3.3.4) we need to change of branch on a subadress of $\xi$.

However, it is enough to slightly expand the design to be able to explore it in a single run. Let us do it as follows:
Example 3.5.1

Running this design against the following one allows us to completely cover it

To have “enough space” is really a matter of balance between positive and negative addresses. In the following section we will make this precise in three distinct ways, each corresponding to a condition that guarantees the observability of the slice. Each condition catches a slightly different intuition. We will first prove that the three conditions are equivalent. Then we will show that the “observability conditions” actually characterize the slices that can be covered with a normalization path.

It is important to notice that the first two conditions are on the prefix tree, while the third condition is on the chronicle tree itself.

3.5.1 Parity

The first observability condition is only concerned with the prefix tree associated to a design. It guarantees that the loci that compose the design can also be arranged
in a counter-design.

**Definition 3.5.2 (Parity)** Let $\mathcal{S}$ be a slice and $T(\mathcal{S})$ its prefix tree. $\mathcal{S}$ satisfies the parity condition if in all the positive subtrees $T'$ of $T(\mathcal{S})$ $P(T') \geq P(T)$.

Notice that we can check this condition with a single postorder traversal of $S$ (cf. Proposition 3.6.2).

**Fact 3.5.3** It is immediate that the difference is at most 1: $P - N \leq 1$

Let us check the conditions on our leading example. The prefix tree associated to the design of example 3.2.1 is the following:

$$\begin{align*}
\xi_1 &\quad \xi_2 \\
\xi &\quad \sigma &\quad \tau
\end{align*}$$

When we move to the expanded design of example 3.5.1 the prefix tree becomes:

$$\begin{align*}
\xi_{10} \\
\xi_1 &\quad \xi_2 &\quad \sigma_0 \\
\xi &\quad \sigma &\quad \tau
\end{align*}$$

### 3.5.2 Leaves

We already know we can reformulate the previous condition (Lemma 3.4.7). In this way we only need to consider the positive nodes. We also neglect the unary nodes, looking only at (positive) leaves and (positive) nodes that are $n$-ary, for $n \geq 2$.

**Definition 3.5.4 (Leaves)** Let $\mathcal{S}$ be a slice and $T(\mathcal{S})$ the corresponding prefix tree. $\mathcal{S}$ satisfies the leaves condition if all the positive subtrees of $T(\mathcal{S})$ satisfy the following expression:

$$P_0 \geq \sum (i - 1) P_i$$

**Remark 3.5.5** Notice that the arity of a positive node is the same in a slice $\mathcal{S}$ and in the corresponding prefix tree $T(\mathcal{S})$. In particular, a positive leaf of $\mathcal{S}$ is a leaf in $T(\mathcal{S})$. 
3.5.3 Return
The third condition establishes an effective description of a way to cover a slice with a normalization path. More precisely, it ensures that the slice can be traversed in preorder.

Remember that a preorder traversal of an n-ary tree $T$ first visits the root and then visits in preorder each of the subtrees.

Let us first give an example on slices.

Example 3.5.6 As an example, let us consider the following design $\mathcal{D}$:

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

![Diagram](image-url)
Normalizing against $\mathcal{E}_1$ gives the path $\langle \emptyset, \xi, \xi_1, \xi_{10}, \xi_2, \xi_{20}, 200, \tau \rangle$, that is a visit in preorder, while the visit resulting from the normalization against $\mathcal{E}_2$ is not.

Later in this section we will show that when it is possible to visit a design, then it is always possible to visit it in preorder.

We start with a few remarks on the preorder traversal of a tree. 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 $\phi$ is necessarily a leaf, and either it terminates the visit of $T$ itself, or after it we start the visit of another subtree $T_j$. If this is the case, we call $\phi$ a return point for the root of $T$.

Given a tree $T$, each leaf but the last visited one is necessarily a return point for exactly one of the internal nodes.

**Fact 3.5.7** Any visit in preorder of an n-ary tree $T$ induces a (partial) function $G$ from the leaves into the internal nodes, defined by:

$$\xi = G(\sigma) \text{ if } \sigma \text{ is a return point for } \xi$$

It satisfies the following property:

(*) Each internal node has exactly one subtree that does not contain a return point for that node.

Observe that the function $G$ is defined on all the leaves but one (which is the last visited); (ii) $G^{-1}(\xi)$, where $\xi$ varies on the internal nodes, induces a partition of the leaves.

Conversely, given a tree consider a function $G$ from the leaves into the internal nodes which satisfies (*). Necessarily, all return points for a node $\xi$ belong to distinct subtrees of $\xi$. Therefore, any choice of an order between the elements in $G^{-1}(\xi)$ induces a visit in preorder of the tree.

**Definition 3.5.8** (Return ) A finite slice $\mathcal{S}$ satisfies the return condition if we can define a partial function $G$ from the leaves into the internal nodes, which satisfies the property (*) above and such that:

$$G(\sigma) \text{ is a prefix of the return point } \sigma.$$ 

Observe that the return point of $\xi$ in the sub-tree $\mathcal{S}_{\xi_i}$ is forced to be a sub-address of $\xi_i$.

The return condition really means that, for an opportune ordering of the subtrees, we can visit the slice in preorder, persistently completing the visit of a subtree before starting a new one.
Proposition 3.5.9 Assume the slice $\mathcal{S}$ satisfies the return condition. Then any ordering of the return points induces a preorder traversal of $\mathcal{S}$ that can be realized by normalization of $\mathcal{S}$ with a counter-design.

Proof. We proceed essentially as in the separation theorem. We can always go “upwards” in a chronicle, the only delicate point is when we change of branch.

The proof is easy, by induction on the length of $q \subseteq p$. Given a slice $\mathcal{S}$, we show that there is a counter-design $\mathcal{T}_q$ which realizes $q$. The delicate point is when $q$ moves from a positive action $\kappa_1$ to a negative action $\kappa_2$ in $\mathcal{S}$, because this means that in $\mathcal{T}$ we have a positive action $\kappa_2$ which appear immediately after the negative action $\kappa_1$. We need to check that $\mathcal{T}_q$ 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 $\kappa_1$ is of the form $\xi$ then $\kappa_2$ is of the form $(\xi, J)$.

2. When we change of branch we are in the following situation:

\begin{center}
\begin{tikzpicture}
  \node[draw, circle] (xi) {$\xi$};
  \node[draw, circle, above of=xi] (xi*) {\(\xi\)*};
  \node[draw, circle, below of=xi] (xij) {$\xi_j$};
  \draw[->] (xi) -- (xi*);
  \draw[->] (xi) -- (xij);
\end{tikzpicture}
\end{center}

Therefore at the orthogonal we need to have $\xi_j$ just after $\xi*$*. This guarantees that the subaddress condition is satisfied on positive focuses, because the parent address of $\xi_j$ is $\xi$, which by induction is below $\xi*$. \(\square\)

This is not the only possible argument. In particular, the tools in Chapter 6 will allow us a more “high-level” proof (it is immediate to check that the path induced is a dispute.)

3.5.4 Equivalence

Proposition 3.5.10 The three observability conditions are equivalent.
Proof. We have already shown in section 3.4 that the conditions of Parity and Leaves are equivalent.

Return ⇒ Leaves. Since G(σ) is a prefix of σ, σ is above G(σ) in T(Ø). The conditions on the function G ensure that in any subtree of T(Ø), at each n-ary node ξ correspond n-1 leaves, which are the inverse image of ξ.

Leaves ⇒ Return. On any sub-design, we define an assignment that for each n-ary node ξ selects n - 1 leaves which are sub-addresses of ξ and lie in distinct subtrees Øξi. Each subtree treated in this way has exactly one leaf that has not been selected.

The leaves condition tells us that there are “enough” leaves to define a partial function G, but the delicate point is to check that the the leaves we are going to use as “return points” are well distributed.

The proof is by induction on the size of the slice Ø, where the size is given by the number of branching nodes (nodes of arity ≥ 2) in Ø.

Size(Ø) = 0. Immediate, since we are considering a single chronicle, we select no leaf and there is one non selected leaf.

Size(Ø) = 1. In Ø there is only one n-ary node ξ. The Leaves condition on T(Ø)_ξ ensures that in Ø there are at least n - 1 leaves that are sub-addresses of ξ. This guarantees that we can select n - 1 such leaves. Since there is only one branching node, each of the n subtree is reduced to a single branch, having only one leaf. Thus the n - 1 selected leaves are forced to lay in distinct subtrees.

Size(Ø) ≥ k. Let ξ be the lowest n-ary node. By induction, we can select all but one of the leaves in each of the sub-slices Øξi. The Leaves conditions guarantees that in T(Ø)_ξ there are at least enough leaves to cover the positive nodes above ξ plus n - 1 leaves for ξ itself. Thus among the leaves that have not yet been selected, there must be n - 1 sub-addresses of ξ. Since the unselected leaves are one for each subtree Øξi, this guarantees that the leaves we associate to ξ lay in distinct sub-slices.

□

Next result establishes that the conditions we have given actually characterize the slices which admit a visit by normalization. Moreover, a non intuitive outcome is that if it possible to visit a design, it is also possible to visit it in preorder.

Theorem 3.5.11 Let Ø be a slice. The following properties are equivalent:

(i) Ø satisfies the observability conditions;

(ii) Ø admits a visit;

(iii) Ø admits a visit in preorder.
Proof.  
(iii) $\Rightarrow$ (i). The fact that $\mathcal{G}$ admits a visit in preorder is immediate by the return condition.

(i) $\Rightarrow$ (ii). By the previous discussion (which uses the return condition), or by the following argument (which uses parity condition). Observe that, by the results in Section 3.4, the parity condition ensures we can exhibit a slice $\mathcal{T}$ of base opposite to that of $\mathcal{G}$ and such that $|\mathcal{G}| = |\mathcal{T}| = |T|$. The fact that all the actions in $\mathcal{G}$ appear with opposite polarity in $\mathcal{T}$ implies that the normalization between $\mathcal{G}$ and $\mathcal{T}$ converges, and uses all of the actions (cf. Proposition 2.2.6).

(ii) $\Rightarrow$ (i). To prove this we exploit Proposition 3.3.4. Any positive node $(\xi, I)$ of $\mathcal{G}$ induces the subtrees $\mathcal{G}_{\xi_i}, i \in I$. Proposition 3.3.4 states that if the normalization path moves from one subtree $\mathcal{G}_{\xi_i}$ to another subtree $\mathcal{G}_{\xi_j}$, then the path must leave $\mathcal{G}_{\xi_i}$ on a sub-address of $\xi$.

Since $\mathcal{G}$ admits a visit, the normalization path must complete the visit of all the subtrees. Let us indicate as $i_1, \ldots, i_n$ the order in which the visit of the subtrees is completed. If $\mathcal{G}_{\xi_i}$ is any of the first $n - 1$ subtrees, when we complete its visit, the visit of the remaining subtrees of $\mathcal{G}_\xi$ is still to be completed. Hence the normalization path will enter $\mathcal{G}_\xi$ again after the last visited action in $\mathcal{G}_{\xi_i}$.

Then we can choose a return point for $\xi$ in each subtree $\mathcal{G}_{\xi_{i_k}}, k \in \{i_1, \ldots, i_{n-1}\}$, choosing the last visited action of that subtree.

\[ \square \]

3.6 Slices and prefix trees

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

**Proposition 3.6.1** To a prefix tree $T$ we can associate both a positive and a negative slice off, as soon as we fix a polarization:

(i) in all the negative subtrees $T'$, $N(T') \geq P(T')$

and

(ii) in all the positive subtrees $T''$, $P(T') \geq N(T')$

**Proposition 3.6.2** It is immediate that for all subtrees the difference is always at most 1: $|N - P| \leq 1$

Both directions are an immediate consequence of the results already proved in this chapter.

Observe that we should really speak of even and odd trees. This because positive and negative only make sense when we choose one of the players, when we are given a design, when we fix a base...

It is important to observe that:
Fact 3.6.3 We can check that the above condition holds for all the subtrees in a time which is linear in the size of the tree, using a single postorder traversal of $T$.

The following function $Balance$ calculates the difference between the number of positive and negative nodes in a tree $T$ of subtrees $T_1, ..., T_n$:

$Balance(\emptyset) = 0$;
$Balance(T) = \sum Balance(T_i) + 1$ if $T$ is positive;
$Balance(T) = \sum Balance(T_i) - 1$ if $T$ is negative.

When we calculate $Balance(T)$ we calculate $Balance(T')$, for all the subtrees of $T$. We simply need to check during this computation that $Balance(T) \in \{0, 1\}$ on positive nodes, $Balance(T) \in \{0, -1\}$ on negative nodes.

Remark 3.6.4 The fact that globally “the number of negative actions is at least equal to the number of positive action” is not sufficient to make a prefix tree into a slice. Our leading example 3.2.1 is a good example of this.

Remark 3.6.5 The condition “in all the negative subtrees, the number of negative actions is at least equal to the number of positive actions” is not necessary for a prefix tree to be associated to a slice, as shown by the following (legal) slice, and its prefix tree:

Anyway, this is not a strong slice.

3.7 Discussion

3.7.1 So what? Consequences

The main application is to delineate the properties that can be expressed interactively. We know that we can only check properties of strong slices. Parts of slices
that are not connected behave as disjoint additive components: they are completely independent w.r.t. normalization.

1. As we have seen in Section 3.2.1, a typical phenomenon is that we cannot detect interactively the use or non use of weakening. Indeed if we do not detect weakening in any strong slice, this does not mean that there is no weakening in the design.

2. Another good example can be given anticipating on Chapter 6. We can represent a design as the collection of its interactions, that is as a collection of disputes. It turns out that the difficult point is to characterize linearity. The natural solution would be to require all disputes to be linear, that is to use each address at most once. This is the kind of linearity condition we can test interactively. Unfortunately this does not imply linearity in the whole slice, only in strong ones. We will discuss this in Section 6.3.

On the other hand, *w.r.t. computation*, an object that satisfies linearity locally (on strong slices) will behave as an object that satisfies that property globally (on any slice). This is true exactly for the same reasons that make it impossible to detect a property that is not local to strong slices.

We will give an example of this in Section 6.3, where we try to study and control the property of propagation. Propagation is the property ensuring linearity in designs. Though we cannot abolish it without reworking all constructions and results of Ludics, we can easily make it local.

### 3.7.2 Realizable paths

Proposition 3.3.4 established a necessary condition that a path must satisfy. However, if we draw a path on a design which satisfies this condition we are not guaranteed that there is a counter-design allowing us to realize that path.
As an example, let us take the following design:

![Design Diagram]

We consider two paths $p_1, p_2$ on that design:

$$p_1 : \tau, \tau 2, \tau 20, \tau 1, \xi, \xi 2, \tau 12, \tau 120, \xi 20, \xi 1, \tau 10, \tau 201, \tau 2010$$

$$p_2 : \tau, \tau 1, \xi, \xi 2, \tau 12, \tau 2, \tau 20, \tau 120, \xi 20, \xi 1, \tau 10, \tau 201, \tau 2010$$

Both paths satisfy the necessary conditions on branching, and both paths move from $\tau 10$ to $\tau 201$, but only $p_1$ admits a counter-design that realizes it.

Another working hypothesis could be to draw a path in such a way that each prefix satisfies the parity condition. Again, the above example gives a counterexample: the prefix tree associated to any $q \subseteq p_2$ satisfies the parity condition. Such a condition guarantees that there is a path ($p_1$) but gives no information on the path itself. Not any path will do.

Though the conditions we have are not sufficient, it is tempting to study the paths on a design that can be realized by normalization against a counter-design. We will do this in Chapter 6.
Chapter 4

Tests and Behaviours

In this Chapter we extend our analysis of designs to the typed setting, i.e. the
designs as part of a behaviour.

First (Section 4.1) we investigate the possibility to transfer to behaviours the re-
results on observability that we established in the previous chapter. One could think
that it is enough to work with material design. Unfortunately a material design with
all good properties for being observable is not necessarily so inside a behaviour. In
Section 4.1.1 we give a (non-trivial) counter-example that illustrates this.

When we work within a behaviour, we test a design in $G$ only using designs in
$G^\perp$. The properties we studied in the previous chapter guarantee that given a slice
$\mathcal{S} \subseteq \mathcal{D}$ we can produce a counterdesign $\mathcal{E}$ which visits $\mathcal{S}$. If now $\mathcal{D}$ belongs to a
behaviour $G$, even if an appropriate $\mathcal{E}$ exists, we still need that $\mathcal{E} \in G^\perp$. This lead
us to focus on a more specific question (Section 4.2): which tests are available when
working with behaviours? If $\mathcal{E}$ is a counterdesign of $\mathcal{D}$ and $\mathcal{D} \in G$, under which
conditions $\mathcal{E}$ belongs to $G^\perp$? To answer, we first need to formulate the question in a
good way. What we are really interested in is not the specific counter-design $\mathcal{E}$, but
any counter-design that interacts with $\mathcal{D}$ in the same way as $\mathcal{E}$. It is then natural
to consider the equivalence relation $\mathcal{E} \sim_D \mathcal{E}'$ that holds when $[\mathcal{D} = \mathcal{E}] = [\mathcal{D} = \mathcal{E'}]$. Our
question is now precise: Given a design $\mathcal{D} \in G$ and $\mathcal{E} \perp \mathcal{D}$, under which condi-
tions there exists an $\mathcal{F} \in G^\perp$ s.t. $\mathcal{F} \sim_D \mathcal{E}$?

In Section 4.2.3 we study the behaviours in which we can realize all possible
disputes. In Section 4.3 we also discuss a situation where we want to limit the
possible visits, allowing only some selected disputes to be realized.
4.1 Observability and behaviours

The operational way to characterize a material design $D \in G$ is as a design whose actions are all visited during the normalization with a counter-design in $G^\perp$. However, this does not mean that all actions are used in the same normalization with a single counter-design. In general a part of $D$ is visited by a counter-design, another part by another counter-design, but there is not a counter-design able to visit all the actions of $D$.

This phenomenon, that we have already met in the previous chapter, is particularly important when we want to interactively establish properties of designs in a behaviour. As we have already observed (Section 3.2.1) 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 this partial pieces of information together, the information is not necessarily complete.

The geometrical constraints we have studied in Chapter 3 still apply. However, the situation is more complex when working with behaviours, because the designs of a behaviour interact with each other to determin the orthogonal. To be a strong slice is still a necessary condition for a slice to admit a visit, but it is not sufficient. The following one is a simple example of a new situation which can produce.

**Example 4.1.1** Let us consider the behaviour $G$ generated by the two following designs $D_1, D_2$

$$
\begin{align*}
D_1 : &\quad \xi_1 \to \xi_2 \\
D_2 : &\quad \xi
\end{align*}
$$

$D_1$ includes both both the following strong slices:

$$
\begin{align*}
\xi_1 \to \xi_2 &\quad \xi_1 \to \xi_2 \\
\xi &\quad \xi
\end{align*}
$$

and
Observe that within the behaviour \( \{\mathcal{D}_1, \mathcal{D}_2\}^{\perp \perp} \) 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 \( \mathcal{D}_2 \). 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: \( \langle \sigma, \sigma_1, \tau, \tau_0, \sigma_10, \sigma_2, \sigma_20 \rangle \).

This example leads us to the following definition.

**Definition 4.1.2** Given a finite slice \( \mathcal{S} \subseteq \mathcal{D} \subseteq \mathcal{G} \), we call it strongly material if there exists a counter-design \( \mathcal{E} \in \mathbf{G}^{\perp} \) such that \( [\mathcal{D} = \mathcal{E}] \) contains all the actions of \( \mathcal{S} \).

Since only strong slices may be visited, a slice which is strongly material is necessarily contained in a strong slice. Assume that the design \( \mathcal{D} \) is finite and material, and \( \mathcal{S} \subseteq \mathcal{D} \) is a "maximal" strong slice. Is it strongly material? The answer is negative. In the next section we build a (non trivial) counter-example.

### 4.1.1 Counter-example

Let us consider the following design \( \mathcal{D} \).

\[
\begin{array}{c}
\sigma_{10} \quad \eta_{20} \quad \sigma_{20} \quad \tau_{20} \\
\eta_1 \quad \eta_2 \quad \tau_1 \quad \tau_2 \\
\sigma_1 \quad \tau \quad \sigma_1, \tau, \eta \\
\end{array}
\]

\( \mathcal{D} \) is a strong slice. Thus \( \mathcal{D} \) is strongly material in its principal behaviour \( \mathcal{D}^{\perp \perp} \).

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

\[ \mathcal{E}_1 : \]

\[ \mathcal{E}_2 : \]

Now, let us consider the behaviour \( \mathbf{G} \) generated by \( \mathcal{D} \) together with the following two designs \( \mathcal{D}_1, \mathcal{D}_2 \)
As design of $G = \{D_1, D_2, \mathcal{D}_1\}$, $D$ is still material, but is not strongly material. There is no way to visit all its action with a design $\epsilon \in G$.

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

### 4.2 Tests and behaviours

In the previous section, we were looking for any counter-design in $G$ allowing us to completely visit a slice. We have seen that even if it is possible to build a counter-design $\epsilon$ to visit $G$, $\epsilon$ could no be in $G$.

This lead us to focus on a more specific question: under which conditions do all the counter-design of $\mathcal{D}$ have, in a sense we shall explain, a “representative” in $G$?

We have two reasons to do this:

1. It is evident that within a behaviour (that is when we test a design in $G$ only using designs in $G$) divergence does not discriminate between designs. However there are situations where we want a distinction, and to do so we need a finer analysis. Since here we are interested in the interaction, we will discriminate designs according to the trace of their interaction with a counter-design.

2. For the same reasons as above, tests we use on pure designs may not be available when working with behaviours. A trivial example is the prototype of test, $\overline{OPP_c}$ (cf. Section 1.4). It separates two designs when they differ on a chronicle $c$. Obviously, as soon as two material designs of $G$ differ on the chronicle $c$, $\overline{OPP_c} \not\in G$.

It is easy to give other examples:
i We already observed in 4.1.1 that within that behaviour we cannot realize the

dispute $\langle \sigma, \sigma_2, \sigma_20 \rangle$.

ii Observe also that it is no longer true that if a slice admits a visit, it admits

a visit in preorder. Let us take again the design of Example 3.5.6, that was

possible to visit in preorder. Let us consider the behaviour $G$ generated by:

\[ \begin{array}{ccc}
\xi & \xi_1 & \xi_2 \\
\xi_1 & 1,2 & \xi, \tau \\
\xi_2 & 1,2 & \xi, \tau \\
\end{array} \]

The first design is the same as in Example 3.5.6. However, in $\{D_1, D_2\}^{\down}$ it

still admits a visit, but no longer a visit in preorder. The only possible com-

plete visit is the sequence $\langle < >, \xi, \xi_2, \xi_20, \xi_1, \xi_10, \xi_200, \tau \rangle$. Any normalization

choosing instead the path $\langle < >, \xi, \xi_1 \rangle$ would fall against $D_2$, while persisting on

the branch of $\xi_2$ would make it impossible to move to $\xi_1$ after having reached

$\tau$. In other words, if we consider the counter-designs described in Example

3.5.6, we have that $\xi_2$ is in $G^{\down}$, but $\xi_1$ is not.

4.2.1 One more dog

In this section we introduce a construction that we are going to use in the

next section. We mention it separately, because it is one more example of

these peculiar objects of Ludics which Girard often calls “dogs”.

Definition 4.2.1 Given a slice $T$, we indicate by $\hat{T}$ the maximal design that

contains $T$, that is $F \preceq \hat{T}$ for all $F$ such that $T \subseteq F$ (cf. the definition of order

in Section 1.4).

$\hat{T}$ is obtained from $T$ by completing the negative rules with all possible rami-

fications and closing the added branches with a daimon. Thus $\text{Succ}_{\hat{T}}(\xi, I) =$
\[(\xi_i, K) : i \in I, K \in \mathcal{P}_f(\mathbb{N})\]. Moreover, if \(\kappa^- \in \hat{\mathcal{F}}\) and \(\kappa^- \not\in \mathcal{F}\), then 
\[\text{Succ}_\mathcal{F}(\kappa^-) = \dagger\]

Observe that:

(i) the positive proper actions of \(\hat{\mathcal{F}}\) are exactly the same as those of \(\mathcal{F}\);
(ii) all actions (and in particular all positive addresses) only appear once; for this reason \(\hat{\mathcal{F}}\) is very close to a slice.

### 4.2.2 Refining the analysis

In chapter 3, we have studied the slices \(\mathcal{S}\) whose actions can be organized into a counter-slice \(\mathcal{E}_\mathcal{S}\). In this case, \(\{\mathcal{S}, \mathcal{E}_\mathcal{S}\}\) is exactly the pull-back associated to the dispute \([\mathcal{S} \Rightarrow \mathcal{E}_\mathcal{S}]\) (cf. Section 2.5). \(\mathcal{E}_\mathcal{S}\) is the minimal design that allows us to cover \(\mathcal{S}\) with a single dispute. However any design \(\mathcal{E}\) containing \(\mathcal{E}_\mathcal{S}\) would produce the same effect.

Since we are actually interested in the interaction among designs, it is natural to consider the equivalence class of counter-designs that produce the same interaction with \(\mathcal{D}\).

**Definition 4.2.2** \(\mathcal{E} \sim_\mathcal{D} \mathcal{E}' \iff [\mathcal{D} = \mathcal{E}] = [\mathcal{D} = \mathcal{E}']\)

This can be expressed in terms of a familiar notion, the incarnation. It was already observed by Girard that producing the same dispute means to have the same incarnation. Since \(\mathcal{E}, \mathcal{E}'\) both belong to \(\mathcal{D}^{\perp}\) then

**Fact 4.2.3** \(\mathcal{E} \sim_\mathcal{D} \mathcal{E}' \iff [\mathcal{E}^{\perp}] = [\mathcal{E}'^{\perp}] = [\mathcal{D}^{\perp}]\)

Any dispute \(p\) determines a class of designs producing that same interaction. In this class there are two distinguished elements: the minimum and the maximum (w.r.t. the order among designs). We can calculate both of them from \(p\).

**Proposition 4.2.4** For any class of designs \(\mathcal{F}\) such that \([\mathcal{D} = \mathcal{F}] = p\), there are a minimal and a maximal element \(\text{Min}_p, \text{Max}_p\).

\(\text{Min}_p \subseteq \mathcal{F}\) characterizes the designs whose interaction with \(\mathcal{D}\) gives \(p\).

**Proof.** It is immediate (cf. 2.5.2) that the pull-back of \(p\) gives us the minimal element. Moreover, for any \(\mathcal{F}'\) s.t. \([\mathcal{D} = \mathcal{F}'] = p\), \(\text{Min}_p \subseteq \mathcal{F}'\). Therefore we have also that \(\mathcal{F}' \leq \text{Min}_p\), by construction. Then \(\text{Max}_p\) is \(\text{Min}_p\).

The interest of the maximal element is that it belongs to any behaviour that contains a representative of the class.

**Corollary 4.2.5** If \(\mathcal{D} \in \mathcal{G}\), \(\mathcal{F} \in \mathcal{G}^{\perp}\), \([\mathcal{D} = \mathcal{F}] = p\) then \(\text{Max}_p \in \mathcal{G}^{\perp}\)
This will allow us to simplify the work, because instead of looking in $G^\perp$ for a
generic $\mathfrak{F}$, on which we have no information except that it contains the slice $\text{Min}_p$,
we can look for an element that we exactly know, $\text{Max}_p$.

Let us consider all possible disputes on a certain design:

$$\text{Disp } D = \{ p : p = |D = G|, G \perp D \}$$

We can then reformulate Corollary 4.2.5 as follows:

**Corollary 4.2.6** Let $D \in G$, $p \in \text{Disp } D$.
There is an $\mathfrak{F} \in G^\perp$ s.t. $|D = \mathfrak{F}| = p$ iff $\text{Max}_p \in G^\perp$.

In Chapter 6 we will be able to characterize the sequence of actions that can
be produced by normalization of two designs, independently from the particular
designs we use. Anticipating on it, let assume that we can speak of disputes without
referring to a particular cut-net producing it. Then we can restate the previous
result in general:

**Proposition 4.2.7** For any dispute $p$, there are two minimal designs $\mathfrak{S}_p, \mathfrak{T}_p$ and
two maximal designs $\mathfrak{H}_p, \mathfrak{F}_p$ such that their dispute is $p$. We then have:

$$\mathfrak{S}_p \subseteq D \leq \mathfrak{H}_p \quad \text{and} \quad \mathfrak{T}_p \subseteq \mathfrak{F}_p \leq D$$

for any $D, \mathfrak{F}$ s.t. $|D = \mathfrak{F}| = p$.

It is immediate that $\mathfrak{S}_p$ and $\mathfrak{T}_p$ are strong slices.

**Remark 4.2.8** Observe that $\text{Max}_p$ has a very simple structure:
(i) it is very close to $\text{Min}_p$;
(ii) it is completely determined as soon as we know $p$ (because $\text{Min}_p$ is determined
by $p$);
(iii) all its positive actions are negative actions of $D$;
(iv) all actions (and all positive addresses) only appear once: w.r.t. normalization
it behaves as a slice.

### 4.2.3 Special behaviours

There is at least one special case where it is immediate that

1. If $G \in \mathfrak{S}^\perp$, then $G \in G^\perp$.
2. If a strong slice $\mathfrak{S} \subseteq D \in G$ is material then it is strongly material.

This is the case when $G$ is the principal behaviour of $D$. In this section we establish
a condition on behaviours sufficient to entail the same properties.

A strong slice $\mathfrak{S}$ is approved by the design $D$ if $D$ contains a slice $\mathfrak{T} \subseteq D$ such
that any positive action appearing both in $\mathfrak{S}$ and $\mathfrak{T}$ is followed by the same negative
actions. Formally:
Definition 4.2.9 A strong slice $\mathcal{S}$ is approved by the design $\mathcal{D}$ if for some slice $\mathcal{I} \subseteq \mathcal{D}$
if $(\xi, I) \in \mathcal{S}$ and $(\xi, I) \in \mathcal{I}$, then $(\xi, K) \in \mathcal{S}$ implies that $(\xi, K) \in \mathcal{I}$.
A design $\mathcal{E}$ is approved by $\mathcal{D}$ if any strong slice $\mathcal{S} \subseteq \mathcal{E}$ is approved by $\mathcal{D}$.

Proposition 4.2.10 Assume $\mathcal{D} \in \mathcal{G}$ is approved by all material designs in $\mathcal{G}$.
Then:
(i) if $\mathcal{E} \perp \mathcal{D}$, and $[\mathcal{E} \perp \mathcal{D}] = q$, then there is an $\mathfrak{g} \in \mathcal{G}^\perp$ s.t. $\mathcal{E} \sim_q \mathfrak{g}$.

Fact 4.2.11 Proposition 4.2.10 can be reformulated as follows:
(ii) if $p \in \text{Disp} \mathcal{D}$, then there is an $\mathfrak{g} \in \mathcal{G}^\perp$ s.t. $[\mathcal{D} \perp \mathfrak{g}] = p$.

Each of the two formulations above conveys a slightly different intuition:
(i) w.r.t. interaction, $\mathcal{G}^\perp$ contains a representative for any equivalence class;
(ii) for any possible dispute of $\mathcal{D}$, $\mathcal{G}^\perp$ contains a design that realizes it.

Given $\mathcal{D}$, the idea is to consider $\text{Min}_p$ for any $p \in \text{Disp} \mathcal{D}$. We know that $\text{Min}_p \in \mathcal{G}^\perp$, and we want to prove that $\bar{\text{Min}}_p \in \mathcal{G}^\perp$. This is quite a permissive (big) design. However, observe that the fact that $\bar{\text{Min}}_p \in \mathcal{G}^\perp$ is not true in general.

For example, consider the behaviour of example 4.1.1. The design $\mathcal{E} = \xi$ belongs to $\mathcal{D}_1^\perp$. As discussed, $\mathcal{E}$ does not belong to $\mathcal{G}^\perp$; for exactly the same reasons, also $\mathcal{E}$ does not belong to $\mathcal{G}^\perp$.

**Proof.** Let us fix a dispute $p = [\mathcal{D} \perp \mathcal{E}]$. We show that for any material design $\mathfrak{g} \in \mathcal{G}$, $[\mathfrak{g}, \text{Max}_p] = \uparrow$. To this purpose, it is enough to show that each design $\mathfrak{g}$ contains a slice $\mathcal{I}$ s.t. $[\mathcal{I}, \text{Max}_p] = \uparrow$. This is true for any slice $\mathcal{I} \subseteq \mathfrak{g}$, such that $\mathcal{I}$ approves $\mathfrak{g}$. Notice that:

1. The positive proper actions of $\mathcal{E}$ are exactly the negative actions in a strong slice of $\mathcal{D}$ ($\mathcal{D}_\mathcal{E}$), by construction. Any positive focus of $\text{Max}_p$ only appears once: $\text{Max}_p$ is not a slice, but w.r.t. normalization behaves as if it was.
2. The normalization between two slices fails when to a positive action does not correspond the same negative action.
Since $Max_p$ has a finite number of positive actions, normalization will be finite. To prove that it succeed, we want to prove that if $\kappa^+ \in \mathfrak{T}$ then $\kappa^+ \in Max_p$, and vice-versa.

If the focus is the root $<>$, the result is immediate. Let the focus be of the form $\xi_i$.

Assume $\kappa = (\xi_i, K)^+ \in \mathfrak{T}$. Necessarily $(\xi, I)^- <_\mathfrak{T} (\xi_i, K), (\xi, I) < (\xi_i, K)$, then $(\xi, I)^+ \in Max_p$. By construction, $Max_p$ accepts any action with focus $\xi_j$, for $j \in I$. Next action will be either $\hat{\dagger}$ (success) or a proper positive action.

Assume $\kappa = (\xi_i, K)^+ \in Max_p$. We know that $(\xi, I) < (\xi_i, K)$, and then $(\xi, I)^+ \in \mathfrak{T}$, and we know that $(\xi_i, K)^- \in \mathfrak{D}$. By hypothesis, $(\xi_i, K)^- \in \mathfrak{T}$. Next action will be either $\hat{\dagger}$ (success) or a proper positive action. 

Observe that Proposition 4.2.10 establishes a condition which is sufficient but not necessary.

### 4.2.4 Back to strongly material designs

It is an immediate consequence of proposition 4.2.10 that:

**Corollary 4.2.12** If each material design of $G$ is approved by the other material designs, then any strong slice which is material is strongly material.

### 4.3 Properties of strongly material designs w.r.t. order

In this section we want to exploit the constraints imposed on the visit of a design by the other designs in a behaviour. Given a design $\mathfrak{D}$ in a behaviour, we force the fact to have *only one possible way to visit it*. The method is to add enough designs to prevent certain visits (certain tests), but still have enough counter-designs to visit all the actions of $\mathfrak{D}$.

Given a design $\mathfrak{D}$, any counter-design determines a dispute which represent a traversal of part of $\mathfrak{D}$. This induces a linear order among the actions of $\mathfrak{D}$ covered by the dispute. In general, for any strong slice $\mathfrak{S}$ there are several of such orders, each corresponding to a different visit of the slice. This is still true within a behaviour.

Inside a behaviour we can also order designs, and in particular material designs, with respect to the order $\preceq$. As an example, consider the principal behaviour gen-
erated by the following design $\mathcal{D}$:

$$
\begin{array}{c}
70 \\
\downarrow 7 \\
2 \\
\mathcal{D}:
\end{array}
$$

The material designs of $\mathcal{D}^\perp$ are:

$$
\begin{array}{c}
\triangleright 20 \\
\downarrow 7 \\
3 \\
\mathcal{D}_1 : \langle> \\
\end{array}
\begin{array}{c}
\triangleright 70 \\
\downarrow 7 \\
2 \\
\mathcal{D}_2 : \langle> \\
\end{array}
\begin{array}{c}
\triangleright 70 \\
\downarrow 7 \\
2 \\
\mathcal{D}_3 : \langle> \\
\end{array}
\end{array}
$$

Observe that w.r.t. the order $\preceq$ among designs, $\mathcal{D}_1$ and $\mathcal{D}_2$ are incomparable. Starting from the behaviour $\mathcal{D}^\perp$, we can add designs to obtain a behaviour where all designs greater than $\mathcal{D}$ are linearly ordered. This is closely related to the possibility of linearly order in an unique way the actions of $\mathcal{D}$.

If we consider the above example, we can select a visit of $\mathcal{D}$, for example $\langle> , 7, 70, 2, 20 \rangle$. To make it the only possible visit we add to the behaviour the following designs

$$
\begin{array}{c}
\triangleright \\
\downarrow 3 \\
\mathcal{D}_4 : \langle> \\
\end{array}
\begin{array}{c}
\triangleright 70 \\
\downarrow 7 \\
\mathcal{D}_5 : \langle> \\
\end{array}
\end{array}
$$

Now the only material designs are $\mathcal{D}, \mathcal{D}_1$ and $\mathcal{D}_4$. 

Proposition 4.3.1 Let $\mathcal{G} \subseteq \mathcal{D} \subseteq \mathcal{G}$ be strongly material. We can extend $\mathcal{G}$ to a “maximal” behaviour $\mathcal{G}' \supseteq \mathcal{G}$ such that $\mathcal{G}$ is still material in $\mathcal{G}'$ (in fact, strongly material), and there is a linear order among the designs $\mathcal{D}' \supseteq \mathcal{D}$ which are material in $\mathcal{G}'$.

There is one such construction for any linear order $p$ among the actions of $\mathcal{G}$, where $p$ is a dispute $[\mathcal{D} = \mathcal{E}]$, $\mathcal{E} \in \mathcal{G} \perp$.

Procedure. Given a positive action $\kappa \in \mathcal{G}$, let $p_\kappa = \langle \ldots \kappa \rangle \subseteq p$. Let $\mathcal{D}_\kappa$ be $\mathcal{D} \upharpoonright p_\kappa$, in which we have substituted $\upharpoonright$ for $\kappa$. Let $\mathcal{G}' = (\mathcal{G} \cup \{\mathcal{D}_\kappa, \kappa \in p, \kappa \text{ positive in } \mathcal{G}\}) \perp \perp$

$[\mathcal{C}, \mathcal{D}_\kappa] = \upharpoonright$ for any $\kappa$, thus $\mathcal{C} \in \mathcal{G}' \perp$, and $\mathcal{D}$ is strongly material in $\mathcal{G}'$

Moreover $\mathcal{D}_\kappa$ is covered by $[\mathcal{D}_\kappa = \mathcal{C}]$, thus all $\mathcal{D}_\kappa$ are strongly material. $\square$

If $\kappa < p \kappa'$, then either $\mathcal{D}_\kappa \supsetneq \mathcal{D}_{\kappa'}$ or $\mathcal{D}_\kappa, \mathcal{D}_{\kappa'}$ are incomparable. Since $\mathcal{D}_{\kappa_n} \supsetneq \mathcal{D}$, then

Proposition 4.3.2 The relation between the designs $\{\mathcal{D}_\kappa : \mathcal{D}_\kappa \supsetneq \mathcal{D}\}$ is a linear order.

If $\mathcal{C} \supsetneq \mathcal{D}$ is material in $\mathcal{G}'$, then $\mathcal{C} \in \{\mathcal{D}_\kappa\}$.

4.4 Discussion of Counterexample 4.1.1

A simple property that the counter-example 4.1.1 does not satisfy is that the same address appears in different slices “typed” in different ways. If we consider $\tau 2$, one time we have the rule $(\tau 2, \emptyset)$ and the other time we have the rule $(\tau 2, \{0\})$. This is the property that is forbidden by the conditions in section 4.2.3.

A second property that we can observe is that the components “depend” on each other. Notice that even if $\mathcal{G} \perp$ is a positive, connected behaviour we cannot decompose it as a tensor of behaviours: the behaviours in the example we have studied does not admit a multiplicative decomposition.

As we already observed, $\mathcal{D} \perp$ contains both $\mathcal{C}_1$ and $\mathcal{C}_2$, while $\mathcal{G} \perp$ does not contain either of them. The relevant counter-design in $\mathcal{G} \perp$ are the following ones, which we indicate by $\mathfrak{A}$ and $\mathfrak{B}$.
It is immediate to see that $H = G^\perp$ cannot be multiplicatively decomposed. Referring to our example, for a design $\mathfrak{F} \in H$ and $\xi \in \{\sigma, \tau, \eta\}$, let us indicate by $\mathfrak{F}_\xi$
the designs \( \mathfrak{F} \mid \xi \). For example, \( \mathfrak{A} \mid \eta \) is

\[
\begin{align*}
&\eta^4 \\
&\downarrow \\
&\eta \\
&\mathfrak{A}_\eta: \langle \rangle
\end{align*}
\]

We indicate by \( H_\xi \) the set \( \{ \mathfrak{F} \mid \xi, \mathfrak{F} \in H \} \).

We have that \( H \subseteq H_\sigma \otimes H_\tau \otimes H_\eta \) and \( H \neq H_\sigma \otimes H_\tau \otimes H_\eta \), because \( \mathcal{E}_1, \mathcal{E}_2 \not\in H \), while \( \mathcal{E}_1, \mathcal{E}_2 \in H_\tau \otimes H_\tau \otimes H_\eta \) and:

\[
\begin{align*}
\mathfrak{A} &= \mathfrak{A}_\sigma \circ \mathfrak{A}_\tau \circ \mathfrak{A}_\eta, \\
\mathfrak{B} &= \mathfrak{B}_\tau \circ \mathfrak{B}_\tau \circ \mathfrak{B}_\eta, \\
\mathcal{E}_1 &= \mathfrak{B}_\tau \circ \mathfrak{A}_\tau \circ \mathfrak{B}_\eta, \\
\mathcal{E}_2 &= \mathfrak{A}_\tau \circ \mathfrak{B}_\tau \circ \mathfrak{A}_\eta.
\end{align*}
\]

This leads us to a characterization of the behaviours that can be written as \( G = \bigoplus (\otimes \downarrow G_i) \), at least in the finite case. We will discuss this in chapter 5.

Observe however that to admit a multiplicative decomposition is not at all necessary to have a test for any strong slice. Let us consider the behaviour \( H \) generated by the two following designs:

\[
\begin{align*}
\xi &\mid K \\
\xi &\mid I
\end{align*}
\]
The relevant material counter-designs of $H^\perp$ are:

![Diagram]

This behaviour cannot be decomposed, because any other combinations of the chronicles in the two designs of $H^\perp$ does not belong to it. Anyway any strong slice in $H$ is strongly material. The same fact is not true for $H \perp$. 


Chapter 5

Decomposition of behaviours

5.1 Decomposition of behaviours

In Chapter 1 we reviewed some constructions on behaviours which correspond to logic connectives. Any behaviour composed in this way enjoys internal completeness: the set of designs produced by the construction is equal to its biorthogonal. Since the biorthogonal does not introduce new objects, we have a complete description of all the designs in the behaviour. For examples, $A \otimes B$ is defined as $\{A \otimes B, A \in A, B \in B\}^\perp$. Since this is equal to $\{A \otimes B, A \in A, B \in B\}$, then for any $\mathfrak{D} \in A \otimes B$ we know we can decompose it as $\mathfrak{D}_1 \otimes \mathfrak{D}_2$, with $\mathfrak{D}_1 \in A$ and $\mathfrak{D}_2 \in B$.

Because of internal completeness, any behaviour formed by using the connectives can be decomposed in its initial components. However, in general a behaviour cannot be decomposed. Let us examine the situation.

1. Any positive behaviour admits an additive decomposition in its connected components:

   $$G = \bigoplus_{I \in G} G_I$$

   where $G_I = \{\mathfrak{D} \in G$ which have as first rule $(\langle, I \rangle) \cup \mathfrak{D}_i\}$.

2. Moreover, any prime behaviour, that is a behaviour where all proper designs different from $\mathfrak{D}_i$ have the same first rule $(\langle, \{i\})$, is immediately decomposed as $G = \left(\mathfrak{G}'\right)^\perp$.

3. What is missing in between these two results is a multiplicative decomposition of the connected components $G_I$ into prime behaviours. Unfortunately, there is no such a result in general, as we recall below.

   Let us recall what is meant by multiplicative decomposition. Assume $G_I$ is a positive behaviour whose designs have the same first rule $(\langle, I \rangle)$. Any design in $G_I$ can be written as the product of the designs $\mathfrak{D} \upharpoonright \{i\}, i \in I; \text{the first rule of } \mathfrak{D} \upharpoonright \{i\}$ is thus $(\langle, \{i\})$. In the case of $\mathfrak{D}_i$, we set $\mathfrak{D}_i \upharpoonright \{i\} = \mathfrak{D}_i$. One can then define
In this chapter we present a characterization of the behaviours that can hereditarily and \textit{finitely} be decomposed as

\[
\bigoplus_{I \subseteq \mathfrak{G}} \left( \bigotimes_{i \in I} 
\downarrow P^\perp \right)
\]

We limit our study to behaviours whose material designs are finite. In particular, this gives a characterization of those behaviours which correspond to the \textit{interpretation of constant-only MALL formulas}.

\textbf{Note.} To read this chapter some familiarity with [Gir01b] would be useful, as we make use of more advanced notions than in the rest of the thesis.

In this chapter we will identify designs with their sequent calculus presentation.

\section{Additive-multiplicative decomposition of behaviours}

\begin{definition}[MA decomposition] A positive behaviour \(G\) \textit{admits multiplicative-additive decomposition (MA decomposition)} if:
\begin{itemize}
  \item \(G = 0\)
  \item \(G = 1\)
  \item \(G = \bigoplus_{I \subseteq \mathfrak{G}} \left( \bigotimes_{i \in I} P_i \right)\), where \(P_i = \downarrow Q^\perp\) and \(Q\) admits MA decomposition.
\end{itemize}
\end{definition}

\begin{remark} If \(\mathfrak{G} = \emptyset\) then \(G = 0\); \(G_\emptyset = 1\).
\end{remark}

\begin{definition} Let us associate a size to a decomposition:
\(\text{Size}(0) = 1, \text{Size}(1) = 1, \text{Size}(G) = \sum \text{Size}(P_i)\).
We say that a behaviour \(G\) which admits MA decomposition \textit{admits finite MA decomposition} if \(\text{Size}(G)\) is finite.

We also associate a depth to a decomposition:
\(\text{Depth}(0) = 1, \text{Depth}(1) = 1, \text{Depth}(G) = \max \{\text{Depth}P_i\} + 1\).
We say that a behaviour \(G\) \textit{admits MA decomposition of finite depth} if \(\text{Depth}(G)\) is finite.
\end{definition}

\begin{fact} The class of positive behaviours with finite MA decomposition is the minimal class of behaviours that contains \(1, 0\) and is closed by \(\oplus, \otimes,\) and \(\downarrow (-)^\perp\).
\end{fact}
Definition 5.1.5 (Fully concordant behaviour) A behaviour is fully concordant if for any two material designs \( \mathcal{D}, \mathcal{D}' \) and any negative address \( \xi \):
\[
(\xi, \mathcal{M}) \text{ occurs in } \mathcal{D} \text{ and } (\xi, \mathcal{N}) \text{ occurs in } \mathcal{D}' \text{ implies } \mathcal{M} = \mathcal{N}.
\]
Notice that \( \mathcal{M} \) can also be the empty set (no action is performed).

Definition 5.1.6 We call height of a behaviour the maximal height of its material designs, where the height of a design is the maximal length of its branches as sequent calculus derivation.

Observe that if \( \mathcal{D} = \{\} \) or \( \mathcal{D} = ((\xi, \emptyset)^+) \) then \( \text{Height}(\mathcal{D}) = 1 \). If \( \mathcal{D} = ((\xi, i)^+) \) the \( \text{Height}(\mathcal{D}) = 2 \), because the corresponding sequent derivation is:
\[
\frac{\xi i \vdash (\xi, \emptyset)}{\vdash (\xi, \{i\})}
\]

Theorem 5.1.7 Let \( G, G^\perp \) be behaviours of finite height. \( G, G^\perp \) admit an MA decomposition of finite depth iff \( G, G^\perp \) are fully concordant. Moreover, if the material designs of \( G, G^\perp \) are finite designs, then \( G, G^\perp \) admit finite MA decomposition.

Remark 5.1.8 Observe that the fact that the material designs in \( G, G^\perp \) are finite implies that also \( \mathcal{Q}(G) \) is finite. Otherwise \( |\mathcal{Q}(i)| \) in \( G^\perp \) would be infinitely branching.

5.1.2 \( G, G^\perp \) fully concordant

To prove one direction of Theorem 5.1.7 we will establish the following steps:

1. Assume \( G, G^\perp \) are fully concordant. If \( I \in \mathcal{Q}G \) and \( i \in I \), then \( \bigodot_{i \in I} D_i \in G \) for any \( D_i \in G | \{i\} \)
2. \( G | \{i\} = G_I | \{i\} \) for any \( I \in \mathcal{Q}G \) such that \( i \in I \).
3. \( G | \{i\} = (G | \{i\})^{\perp^{\perp}} \)
4. Assume \( G | \{i\} = \downarrow (P^{\perp}); P, P^{\perp} \) are fully concordant.

The proof is by induction on \( \text{Min}\{\text{Height}(G), \text{Height}(G^{\perp})\} \). Observe that the decomposition of the Shift makes decrease the height of all designs in \( G \) and \( G^{\perp} \). Observe also that if \( \text{Height}(G) = 1 \) then \( G \) is either 0 or 1.

Notation 5.1.9 We generally omit the bracketing for the singleton, so we write \( G | i \) for \( G | \{i\} \).

All along this section \( G, G^\perp \) are behaviours whose material designs are finite.

Remark 5.1.10 We will make use of the following immediate properties:

(i) If \( \mathcal{C}, \mathcal{D} \in G, \mathcal{C} \subseteq \mathcal{D} \) and \( \mathcal{D} \in G_I \) then \( \mathcal{C} \in G_I \) and \( \mathcal{C} | i \subseteq \mathcal{D} | i \).

(ii) If \( \mathcal{C}_i \subseteq \mathcal{D}_i \in G | i \) then \( \bigodot \mathcal{C}_i \subseteq \bigodot \mathcal{D}_i \).
Proof.
(i) Immediate from the fact that $c$ is forced to have the same first action as $d$.
(ii) We just need to notice that $D_i \subseteq D$ implies $D = D_i$. This means that if
$c_j = D_i$, for some $j \in I$, and thus $\bigcap_{i \in I} c_i = D_i$, we also have that $\bigcap D_i = D_i$.
\qed

**Proposition 5.11** If $G, G^\perp$ are fully concordant, then for any $I \in \Phi G$, $\bigcap_{i \in I} (G \mid i) = G_I$

**Proof.** It is always true that $G_I \subseteq \bigcap_{i \in I} G \mid i$.
We need to show that for any $D_i \subseteq G \mid i$, $\bigcap_{i \in I} D_i \subseteq G_I$. We first obtain
the result for $D_i = D_i \mid i$ when $D_i$ is a material design in $G$. This, proved in the
following lemma, entails the general result. We just observe that:
(i) If $D \subseteq G_I$ then $|D|_G \subseteq G_I$, because the first action $(\xi, I)$ belongs to $|D|_G$.
(ii) Let $D^{(i)}, i \in I$ be a family of designs in $G$ s.t. $D^{(i)} \mid i \in G \mid i$. $|D^{(i)}|_G \mid i \subseteq
D^{(i)} \mid i$ and thus $\bigcap (|D^{(i)}|_G \mid i) \subseteq \bigcap (D^{(i)} \mid i)$
By the following lemma, $\bigcap (|D^{(i)}|_G \mid i) \subseteq G$, hence also $\bigcap (D^{(i)} \mid i) \subseteq G$.
\qed

**Lemma 5.12** Let $I \in \Phi G$, and for $i \in I$ let $D^{(i)}$ be a family of material designs in $G$ such that $D \mid i \subseteq G \mid i$. If $G, G^\perp$ are fully concordant, then:

$$\bigcap_{i \in I} (D^{(i)} \mid i) \subseteq G$$

**Remark 5.13** Notice that $D^{(i)} \subseteq G_I$, where $i \in J$, but in general $I \neq J$. As example, think of a behaviour such as $G_1 \otimes (G_2 \otimes G_3)$ with $\Phi = \{1, 2\}, \{1, 3\}$.

**Proof.** Let $D = \bigcap_{i \in I} (D^{(i)} \mid i)$. We want to show that for any $c \in G^\perp$, $\langle D, c \rangle = 1$. Let us take any material $c$.
Since $I \in \Phi (G)$, $c$ accepts the first action of $D$. Any other action involved in
the normalization is of the form $(\sigma, J, K)$, where $(\sigma, J) < (\sigma, K)$ has already been
used, and then belongs to both designs.

Let the normalization be on $(\sigma, J, K) \in c \in G$. Thus $(\sigma, J)^+ \subseteq D$, because it has
already been used, and is followed by a rule $(\sigma, M)$. This belongs to $D^{(i)} \mid i$ for
an opportune $i$. Since $c$ is material, there is a design $B \in G$ s.t. $(\sigma, J, K) \in B$.
Because $G$ is fully concordant, and $D^{(i)}$ is a material design in $G$, then $K \in M$.

Let the normalization be on $(\tau, J, K) \in D$. This action belongs to one of the
$D^{(i)}$, hence there is a material $\exists \in G$ that uses it. $(\tau, J, K) \in \exists$ and $G$ concordant
implies that $(\tau, J)^+ \leq (\tau, J, K)^-$ in $c$.

\qed
Remark 5.1.14 We request that if \((\xi, M)^-\) occurs in \(D\) and \((\xi, M')^-\) occurs in \(D'\) then \(M = M'\) for whatever positive action generates \(\xi\), not simply when \((\xi, I) <_D (\xi, M), (\xi, I) <_{D'} (\xi, M')\). We use this when \(\xi = 
abla\).

In fact, for any \(G = \bigoplus (\bigotimes G_i), (i, M)^- \in D, (i, M')^- \in D'\) forces \(M = M'\).

Remark 5.1.15 The following condition would be sufficient but not necessary to ensure a single step multiplicative decomposition. For any two designs \(B, B' \in G, G^\perp:\)

(i) \((\xi, I)^+ \leq (\xi, K) \in B, (\xi, I)^+ \in B', \) then \((\xi, I)^+ \leq (\xi, K) \in B'\)

(ii) \((i, M)^- \in B, (i, M')^- \in B'\) then \(M = M'\)

Proposition 5.1.11 allows us to state also that

Corollary 5.1.16 \(G \mid i = G_I \mid i\) for any \(i \in G\) s.t. \(i \in I\).

An immediate consequence is that

Proposition 5.1.17 \(G \mid i = (G \mid i)^{\perp\perp}.

Proof. The result is established in [Gir01b] for \(G\) connected, thus follows from \(G \mid i = G_I \mid i\). We recall the proof in proposition 5.1.19.

We can embed \(G_i\) into \(G\) and \((G \mid i)^\perp\) into \(G^\perp\). The result is established in [Gir01b] for the behaviours that are connected. From that, it follows immediately for any behaviour, since we can write any \(G\) as \(\bigoplus G_i\).

Definition 5.1.18 Let \(G\) be connected, and \(A\) a design in either \(G \mid i\) or \(|(G \mid i)^\perp|\).
In both cases there is a single first action, that is \((\nabla, i)\). \(\Phi_I(A)\) is the design obtained from \(A\) by substituting the first action \((\nabla, \{i\})\) with \((\nabla, I)\).

We refer to Chapter 1 for the design \(\text{Ram}\).

Proposition 5.1.19 (Girard) Let \(G\) be connected.

1. If \(D \in (G \mid i)\) then \(D' = D \oplus \text{Ram}_{\langle \nabla, I \setminus \{i\} \rangle} \in G\).
2. If \(E \in \{|(G \mid i)^\perp|\}\) then \(\Phi_I(E) \in G^\perp\).
3. \(G \mid i = (G \mid i)^{\perp\perp}.

Proof. 1. If \(B \in G\) then \(B \mid i \in (G \mid i)\). Since \([B, \Phi_I(E)] = [B \mid i, E] = 
\), we conclude that \(\Phi_I(E) \in G^\perp\) If \(D \in (G \mid i)\) then there is \(B \in G, B \mid i = D, \) and \(B \preceq D \oplus \text{Ram}_{\langle \nabla, I \setminus \{i\} \rangle}\).

2. For any \(\mathcal{F} \in G^\perp, \mathcal{F}_i = (\mathcal{F})\text{Ram}_{\langle \nabla, I \setminus \{i\} \rangle} \in (G \mid i)^\perp\). In fact, if \(B \in G\) then \(B \preceq (B \mid i) \oplus \text{Ram}_{\langle \nabla, I \setminus \{i\} \rangle}\), then \(\mathcal{F} \perp (B \mid i) \oplus \text{Ram}_{\langle \nabla, I \setminus \{i\} \rangle}\). By adjunction \(\mathcal{F}_i \perp (B \mid i), \) hence \(F \in (G \mid i)^\perp\).

If \(D \in G^{\perp\perp}\), in particular \(D \perp \mathcal{F}_i, \) for any \(\mathcal{F} \in G^\perp\). Since \([D \oplus \text{Ram}_{\langle \nabla, I \setminus \{i\} \rangle}, \mathcal{F}] = [D, \mathcal{F}_i], \) we conclude that \(D \oplus \text{Ram}_{\langle \nabla, I \setminus \{i\} \rangle} \in G^{\perp\perp} = G\) and hence \(D \in G \mid i\).
**Definition 5.1.20** If $p$ is a sequence of actions with first action $(<> , i)$ then $\Phi (p)$ is the same sequence, where the first action $(<> , i)$ has been substituted by $(<> , I)$.

**Proposition 5.1.21** 1. Let $\mathcal{D} , \mathcal{E}$ resp. in $G | i , (G | i)^{\perp}$. Let $\mathcal{D}' , \mathcal{E}'$ as above. If $[\mathcal{D} = \mathcal{E}] = p$ then $[\mathcal{D}' = \mathcal{E}'] = \Phi (p)$. More precisely, if $\text{Pull} (p) = \{ \mathcal{G} , \mathcal{I} \}$, then $\text{Pull} ([\mathcal{D}' = \mathcal{E}']) = \{ \Phi (\mathcal{G}) , \Phi (\mathcal{I}) \}$.

2. If $\mathcal{D}$ is material in $(G | i)$, then $\Phi (\mathcal{D}) \subseteq [\mathcal{D}']_{G}$. If $\mathcal{E}$ is a material design in $(G | i)^{\perp}$, then $\Phi (\mathcal{E})$ is a material design in $(G)^{\perp}$.

**Proof.** It is immediate that $\mathcal{A} \subseteq \mathcal{D}$ implies $\Phi (\mathcal{A}) \subseteq \Phi (\mathcal{D})$.

Let $p = [\mathcal{D} = \mathcal{E}] , \mathcal{G} = \text{Pull}^{+} (p) , \mathcal{I} = \text{Pull}^{-} (p)$. $\mathcal{G} \subseteq \mathcal{D}$ implies $\Phi (\mathcal{G}) \subseteq (\mathcal{D}) \subseteq \mathcal{D}' , \mathcal{I} \subseteq \mathcal{E}$ implies $\Phi (\mathcal{I}) \subseteq \Phi (\mathcal{E})$.

2. Immediate. □

We can generalize the embedding to any behaviour, using the fact that we can recover the incarnation of additive behaviours from the incarnation of the components and conversely.

**Remark 5.1.22** Remember that

$$|A^{\perp} \& B^{\perp}| = |A^{\perp}| \times |B^{\perp}| \quad \quad A \oplus B = |A| \cup |B|$$

**Proposition 5.1.23** Let $G = \bigoplus_{I} G_{I}$.  

1. If $\mathcal{D} \in (G | i)$ then $\mathcal{D}' = \Phi (\mathcal{D}) \circ \text{Ram}_{\langle > , I \rangle} \in G$. If $\mathcal{E} \in (G | i)^{\perp}$ then $\mathcal{E}' = \Phi (\mathcal{E}) \cup \text{Dir}_{\langle > , X \rangle} \in G^{\perp}$.

2. If $\mathcal{D}$ is material in $(G | i)$, then $\Phi (\mathcal{D}) \subseteq [\mathcal{D}']_{G}$. If $\mathcal{E}$ is material in $(G | i)$, then $\mathcal{E}'$ is material in $(G)^{\perp}$.

**Proof.** 1. If $\mathcal{D} \in (G | i)$ then $\mathcal{D}' = \Phi (\mathcal{D}) \circ \text{Ram}_{\langle > , I \rangle} \in G_{I}$, hence the result because $G = \bigcup G_{I}$.

Let $\mathcal{B} \in G$. If $\mathcal{B} \in G_{I}$, then $[\mathcal{B} , \mathcal{E}'] = [\mathcal{B} , \Phi (\mathcal{E})]$. If $\mathcal{B} \in G_{J} , J \neq I$ then $[\mathcal{B} , \mathcal{E}'] = \perp$.

2. Immediate by previous results and the properties of incarnation for With and Plus. Just remember that $\Phi (\mathcal{E}) \in |G_{I}^{\perp}|$ and $\text{Dir}_{\langle > , J \rangle} \in |G_{J}^{\perp}|$. □

**Proposition 5.1.24** If $G , G^{\perp}$ are fully concordant, then $G | i , G | i^{\perp}$ are fully concordant. Moreover $\text{Height} (G | i) \leq \text{Height} (G)$ and $\text{Height} (G | i^{\perp}) \leq \text{Height} (G^{\perp})$

**Proof.** By appropriate embedding, the fact that two material designs in $G | i$ (resp. $(G | i)^{\perp}$) are concordant follows from the same property in $G$ (resp. $G^{\perp}$). □
5.1.3 G admits MA decomposition

Lemma 5.1.25 Let D be a material design of base S = Ξ ⊨ Λ in the sequent of behaviours F_S, where for any address \( ξ \in S \), \( F_ξ \) admits MA decomposition of finite depth. If the last rule of D is

\[
\frac{S}{S_1 \ldots S_n}
\]

then the subdesign D_{S_j} induced by any of the premises is a material design in F_{S_j}

Proof. The proof is on the same lines as the proof of completeness, in the easy constant-only case.

Positive base Let D \( \vdash P_ξ, G_Λ \), and \((ξ, I)\) be the last rule. For any \( ε_σ \in G_Λ^⊥ \), \( [D, ε_σ] \in P \), having \((ξ, I)\) as last rule.

Because of the additive decomposition, \( P = \bigoplus P_J \). Accordingly to the ramification \( I \) fixed by the last rule, \( [D, ε_σ] \in P_I \). Hence D \( \vdash P_I, G_Λ \).

Now \( P_I = \bigotimes P_I \). D can be written as \( \bigotimes D_I \). From \([D, ε_σ] \in P_I\), we conclude \([D_I, ε_σ] \in P_I \), hence \( D_I \vdash P_I, Λ \).

If \( P = \downarrow Q_\downarrow \), then \( D_I = \downarrow D' \) that is the subdesign on the desired base.

Negative base. Let D \( \vdash P \vdash G_Λ \). If \( P = \bigoplus P_I \), for any ramification \( I \) we have

\( D_I \in P_I \vdash G_Λ \).

From \( P_I = \bigotimes \downarrow Q_\downarrow \), we obtain \( D'_I \in P \downarrow G_Λ \).

\[\square\]

Proposition 5.1.26 Let F be a behaviour that admits MA decomposition of finite depth (think in particular of the interpretation of a constant-only MALL formula). All material designs in F, F^⊥ are concordant.

Proof. Let D, C be material designs in F, \((ξ_i, N) \in D \) and \((ξ_i, M) \in C \). By the previous Lemma we can build the material designs D', C' \in P, where P is the component of F located in \( ξ_i \). Materiality makes it immediate that \( N = M \). If \( K \in M \), since C' is material in P, there is a counter-design in P^⊥ that has \((ξ_i, K)\) as first action. Hence, \( K \in N \), because D' \in P.

\[\square\]

5.2 Discussion

To characterize the behaviours that can be decomposed, and thus in particular the behaviours which are interpretation of formulas, is an important open question in the theory.
In this chapter we work with finite material designs. This is enough to deal with constant-only MALL, which is already an expressive system. Moreover, for applications one is likely to be interested in finite objects, or in some sort of finite syntax.

However, we stress that in this way we do not deal with propositional variables. The difficulty towards a more general result is that it is hard to work with unknown behaviours (propositional variables) without the help of uniformity (cf Appendix A). To say that \( \mathcal{C} \in \mathbf{G}^\perp \) we need to check that \( \mathcal{C} \perp \mathcal{D} \) for enough designs \( \mathcal{D} \in \mathbf{E} \) to have \( \mathbf{E}^{\perp \perp} = \mathbf{G} \). Unfortunately uniform material designs are not enough to generate a behaviour.
Chapter 6

From Disputes to Designs: Ludics and Games

In this chapter\(^1\) we want to describing a design \(\mathcal{D}\) as the collection of its possible interactions: \(Disp \mathcal{D} = \{ \mathcal{D} = \mathcal{E}, \mathcal{E} \perp \mathcal{D} \} \). This corresponds to an early idea in Ludics, which was already explored by Girard in an unpublished manuscript.

The first step will be to characterize the sequences of actions that correspond to a dispute (Section 6.1).

We will then need to characterize the set of disputes which correspond to interactions of the same design, and verify that we have all of them. We therefore need:

(i) a “coherence condition” to guarantee that a set of disputes is compatible, meaning that all the disputes are paths on the same design, and

(ii) a “saturation condition” to guarantee we have all the possible paths.

The approach we follow establishes a bridge with the notions of Game Semantics. In fact, we are going to make precise the following correspondence:

- action – move
- dispute – play
- chronicle – view
- design – innocent strategy

The crucial correspondence is

\[ \text{“view - chronicle - sequent calculus branch”} \]

\(^{1}\)The results presented in this chapter are outcome of discussions with Martin Hyland.
This correspondence is the key to move between Ludics and Games Semantics. To keep all notions concrete, always remember that a chronicle is a branch in a sequent calculus derivation, a design being the “skeleton” of a sequent calculus derivation. Conditions on views, as conditions on chronicles, can easily be understood as conditions on the branches of a sequent calculus derivation.

A knowledge of Games Semantics is not necessary to read the chapter, because we redefine all notion we need. However, we review the basic definitions of Games Semantics in the Appendix ??.

Notation and Conventions

In this chapter:
\( \kappa, \alpha, b, c \) indicate actions (moves); \( p, q, s, t \) indicate sequences of actions (disputes, plays); \( \xi, \delta \) indicate chronicles. \( \mathcal{D}, \mathcal{E} \) indicate designs (sets of chronicles) and \( \mathcal{S}, \mathcal{T} \) indicate strategies (set of plays).

Remember that the sequent \( \Xi \vdash \lambda_1, \ldots, \lambda_n \) identifies the closed base \( \vdash \Xi \vdash \Xi \vdash \lambda_1, \ldots, \lambda_n \lambda_1 \vdash \). In particular, the closed net on \( \vdash<> \) is the closed net of base \( \vdash<> <> \vdash \).

6.1 Arenas, players and legal positions

To simplify the presentation, in this section we only consider designs (more precisely cut-nets) on the empty base \( <> \). The associated “dependency tree” is the universal Arena. The generalization to the base \( \Xi \vdash \Lambda \) is straightforward.

Players: We exploit parity to split the universe into two players: one owning the even-length addresses (Even), the other owning the odd-length addresses (Odd). As soon as we fix a point of view, one will be called Proponent (P), the other Opponent (O). We stress that in Ludics there is no difference in role between the two players: they obey the same rules.

Arena: We recall that an arena is given by a set of moves, a labelling function from the moves to \( \{P, O\} \), and an enabling relation. The Universal Arena is the forest of actions induced by the sub-address relation.

Definition 6.1.1 (Universal Arena) The universal arena is given by a set of moves, a labelling function and an enabling relation, as follows:

Moves: The moves are all the actions \( (\xi, J) \), where \( \xi \) is an address and \( J \in \wp_{\text{fin}}(\mathbb{N}) \).

Both labelling and enabling are already coded in the action:

Labels: The labelling is implicit in the address: all even-length addresses are attributed to one Player, all odd-length addresses are attributed to the other.
Enabling relation:
We say that \((\xi, I)\) justifies \((\xi_i, J)\), if \(i \in I\). We call initial move an action which is not justified (the actions whose focus is \(<\)).

The universal arena \(A\) can be reoriented to any initial address \(\xi\). As usual, the moves of \(\xi(A)\) are those of \(A\) with the renaming \(\xi(\sigma, I) = (\xi_\sigma, I)\).

We call such a structure an atomic arena. An atomic arena is identified by an atomic base. The universal arena has base \(\vdash <\).

**Parity.** There is a complete symmetry between the two players. In particular, to any play we will associate both a strategy and a counter-strategy.

Since it is convenient to fix a point of view, let us fix Proponent the one who starts (the player owning the initial move) and Opponent the other. Observe that this is opposite to Games Semantics tradition, but more natural in Ludics, which takes the positive point of view: we prefer to say that a design of base \(\vdash \xi\) belongs to Player.

**Polarity.** Remember that the polarity is relative to the Player: a move is positive for a player if it belongs to that player, negative if it belongs to the other. Positive means same parity ("mine"), negative means opposite parity ("yours").

\(P\)-move ("move belonging to P") = \(P\)-positive ("move positive for P") = \(O\)-negative ("move negative for O").

\(O\)-move = \(O\)-positive = \(P\)-negative.

**Notation 6.1.2** When we need to specify a player but do not wish to take a point of view, we will use the variables \(X\) where \(X \in \{P, O\}\) and \(\overline{X}\) for its dual.

To make explicit if a move \(\kappa\) is \(P, O\), positive or negative we use the notation: \(\kappa^P, \kappa^O, \kappa^+, \kappa^-\).

### 6.1 Plays

**Definition 6.1.3 (Linear positions)** A sequence of actions \(s\) is a linear position, if it satisfies the following conditions:

- **Parity** Parity alternates
- **Justification** Each move is either initial or is justified by an earlier move.
- **Linearity** Any address appears at most once.

Each position belongs to one of the players, according to the last move. The following definition takes care of the case where no move has been played yet.

**Definition 6.1.4 (X-position)** We call \(P\)-Position a position that expects an action by Opponent. Typically, a position whose last move is \(P\). An \(O\)-Position is
a position that expects an action by Proponent. Observe that if we choose to call
Proponent the player who starts, \( e \) is an O-position.

A P-position is a positive position for P, and a negative position for O. We make
explicit if a position is P, O, positive or negative using the notation \( p^P, p^O, p^+, p^- \).

**Remark 6.1.5** Observe that as a player identifies a polarization of the actions, so
a polarization (the explicitation of the polarity of the actions) identifies a player.

W.r.t. HO games, observe that linearity makes the pointers unnecessary (unambiguous).

Notice that

**Fact 6.1.6** Linearity of \( p \) implies that for any \( s, t \subseteq p \), if \( s = s'(\xi, I) \) and \( t = t'(\xi, J) \) then \( s' = t' \) and \( I = J \).

The key notion is that of view, that we have already defined in chapter 2 and
recall below.

**Definition 6.1.7 (Views)** Let \( q \) be a linear position and \( X \in \{ O, P \} \) a player. Its
view \( _qX \) of \( q \) is inductively defined as follows. As we have already said, when there
is no ambiguity on the player, we simply write \( _q \) for \( _qX \). Below, positive and
negative is relative to \( X \).

- \( _e = e \);
- \( _s^+ = _s^- \kappa^+ \);
- \( _s^- = \kappa^- \) if \( \kappa \) is initial;
- \( _s^+ \kappa^- = _s^- \kappa^+ \), if \( \kappa = (\xi, J) \) and \( \kappa' = (\xi, I)^+ \).

We denote Opponent view by \( _qO \) and Proponent view by \( _qP \). Moreover, by
\( _qK^- \) we mean the view of the player for which \( \kappa \) is positive. If \( \kappa \) belongs to \( X \),
then \( _qK^- = _qX \) and \( _qK^- = _q\overline{X} \).

The following (standard) definition of play allows us to characterize the sequence
of actions that correspond to a dispute.

**Definition 6.1.8 (Legal positions/Plays)** We say that a linear position \( p \) is legal,
or a play, if it satisfies the following condition:

Visibility If \( tK \subseteq p \) where \( \kappa \) is non initial, then the justifier of \( \kappa \) occurs in \( _tK^+ \). According
to to our convention, this means that if \( \kappa \) is a P-move, its justifier occurs in
\( _tK^P \), and therefore in \( _t^P \), if \( \kappa \) is an O-move, its justifier occurs in \( _t^O \).

It is immediate that
Fact 6.1.9 1. If p is a legal position, then \( r_p \gamma^X \) is a legal position 2. \( rr_p \gamma^X \gamma^X = r_p \gamma^X \)

For any path \( p \) we trace on a design (cf. 3.1.1) parity alternates and the justification condition is satisfied. Visibility guarantees that the path can be produced by a counter-design. It exactly says that \( Pull^+(p) \), \( Pull^-(p) \) are designs: all views are chronicles because they satisfy the sub-address condition.

Remember that by \( \vdash < > \) we mean the closed base \( \vdash < > \) \( < > \vdash \).

Proposition 6.1.10 (Disputes as plays) Any dispute \( p \) on a closed net of base \( \vdash < > \) is a legal position on the universal arena.

Proof. Parity and justification are obvious. Visibility. Let \( tk \subseteq \kappa \), say, a P-move. \( r_{tk} \gamma^P = \epsilon \kappa \) is a chronicle. Then, by definition of chronicle, the justifier belong to \( \epsilon \). We conclude observing that \( r_{tk} \gamma^P = r_{t^P} \kappa \), and thus \( r_{t^P} = \epsilon \). The same is true if \( \kappa \) is an O-move. \( \square \)

6.1.2 Extracting a design and a counter-design from a play

Conversely, we shall show that, given a legal position \( p \), we can extract a design \( G \) and a counter-design \( T \) s.t. \( [G = T] = p \).

\( \{G, T\} \) is exactly the pull-back associated to \( p \).

To move from disputes to design we need to deal with the daimon. Our choice is to deal with it implicitly, as we shall explain, and retrieve it when we need. However, we also add \( \dagger \) to the arena as a special move. We use this as an intermediate step which allows us more compact definitions.

Definition 6.1.11 (Daimon) We extend the universal arena with a formal action \( \dagger \). \( \dagger \) is fixed positive for any player; it does not justifies and is not justified by any other action.
Given a collection of legal positions, we define an operation of (positive) closure w.r.t. either of the player: we complete all maximal negative plays with a daimon.

**Definition 6.1.12 (dai-closure)** Let $S$ be a collection of plays on the universal arena. We define its positive closure w.r.t. the player $X$ as follows:

$$p^{1x} = p * \dagger \text{ if } p \text{ is } X\text{-negative, } p^{1x} = p \text{ otherwise.}$$

$$S^{1x} = \{p^{1x}, p \in S\}$$

**Definition 6.1.13** Let $p$ be a legal position.

$$\text{Ch}^P(p) = \{r_q^{\neg P} : q \subseteq p, q \neq \epsilon\}.$$  

$$\text{Ch}^O(p) = \{r_q^{\neg O} : q \subseteq p, q \neq \epsilon\}.$$  

**Proposition 6.1.14 (Pull-back)** Let $p$ be a play.

$$\text{Ch}^P(p^{1r})$$ is a $P$-slice, $\text{Ch}^O(p^{1r})$ is an $O$-slice.

**Proof.** Let us fix a point of view, either $O$ or $P$ and check that $\text{Ch}^X(p)$ is a slice.

(i) $s \in \text{Ch}(p)$ is a chronicle.

**Alternation.** obvious. **Daimon.** obvious. **Negative focus.** Immediate by the definition of view on a negative action. **Positive focus.** This exactly corresponds to the visibility condition. **Destruction of foci:** imposed by linearity.

(ii) $\text{Ch}(p)$ is a design, in fact a slice.

Closure under prefix. Let $c_k^lq = r q_k^\gamma$, $q_k \subseteq p$. If $\kappa$ is positive: $c_k^{l'} = r q^{l'}$ and $q^{l'} \subseteq p$. If $\kappa$ is negative then $c_k^{l'} = s k^{l'} q^{\gamma}$ and $s k^{l'} \subseteq q^{l'}$.

**Coherence.** $c_1, c_2 \in \text{Ch}(p)$ are coherent. If $c_1, c_2$ are incomparable, let us consider $c_1 \wedge c_2 = c_k$. If $c_1 \sqsupseteq c_k k_1$ and $c_2 \sqsubseteq c_k k_2$, with $k_1 \neq k_2$, then $k$ is positive. Otherwise $k_1, k_2$ would be positive. Then $c_k k_1 = s_1 k k_1^{\gamma}$, $c_k k_2 = s_2 k k_2^{\gamma}$, and since linearity forces $s_1 = s_2$, then $k_1 = k_2$.

$\text{Ch}(p)$ is a Slice: propagation. Both propagation and the fact that $\text{Ch}(p)$ is a Slice are forced by linearity. If $c(\xi, I) = r_q(\xi, I)^\gamma$, $c'(\xi, I') = r_{q'}(\xi, I')^\gamma$, then by linearity $q(\xi, I) = q'(\xi, I')$ and $c(\xi, I) = c'(\xi, I)$.

\[ \square \]

**Example 6.1.15** On the empty base:

$$\text{Ch}^P(\epsilon^{1r}) = \{\epsilon\}, \text{ which corresponds to the derivation } \vdash<=>\dagger.$$  

$$\text{Ch}^O(\epsilon^{1r}) = \emptyset, \text{ which corresponds to the derivation } <=>\vdash (<>\vdash, \emptyset)$$

It is immediate that

**Remark 6.1.16** If $q^X \sqsubseteq p$ then $\text{Ch}^X(q) \subseteq \text{Ch}^X(p)$.  

\[ \]
Proposition 6.17 (Plays as disputes) To each play \( p \) on the universal arena we can associate a pair of slices \( \mathcal{S}, \mathcal{T} \) on the base \( \vdash < > < > \vdash \) s.t. \( \mathcal{S} = \mathcal{T} = p; \) \( \{ \mathcal{S}, \mathcal{T} \} \) is the pull-back associated to \( p \).

Proof. Let \( \mathcal{S} = Ch^p(p) \) and \( \mathcal{T} = Ch^o(p) \). We need to check that \( \mathcal{S} = \mathcal{T} = p \). This is immediate by the procedure of normalization on designs (cf. LAM). If \( \mathcal{S} = \mathcal{T} = t \) we show that for any prefix \( t_n \) of length \( n, t_n \subseteq p \).

If \( p = \epsilon \) the result is immediate. Step 1. The first action in the cut-net is the same as the first move of \( p \).

Let \( t'\kappa_n = t_n = p_n \). To perform the \( n+1 \)-ary step of normalisation on \( \mathcal{S}, \mathcal{T} \), we look in the slice where \( \kappa_n \) is negative, that is the chronicle \( \kappa_n^- \). Let \( \kappa = Succ(\kappa_n^-) \); such an action exists, either proper or improper, because \( \kappa_n \) is negative. We know that the chronicle \( \kappa_n^- \) is the view of a prefix of \( p \). \( \tau p'\kappa_n\kappa^- X, p'\kappa_n \kappa^- \subseteq p \). Linearity implies that \( p'\kappa_n^- = p_n \). Hence either \( p = p_n = t_n \) and \( \kappa_n+1 = \downarrow \) or \( p_n \kappa_n+1 \subseteq p \) and \( \kappa = \kappa_n+1 \). This also guarantees the existence of a chronicle \( \kappa_{n+1}^- \), thus \( t_n \kappa_n = p_n \kappa_n \).

\( \square \)

6.2 Strategies

We have a characterization of the disputes. Now we want to describe a design as the collection of its possible interactions:

Definition 6.2.1 (Disp (\( \mathcal{D} \)) ) Let \( \mathcal{D} \) be a design. \( \text{Disp} \mathcal{D} = \{ [\mathcal{D} \models \mathcal{E}] : \mathcal{E} \perp \mathcal{D} \} \).

Definition 6.2.2 A chronicle \( \mathcal{c} \) is positive (negative) if its last action is positive (negative). Let us denote by \( \mathcal{D}^+ \) the subset of positive chronicles of \( \mathcal{D} \). We call it the positive part of \( \mathcal{D} \).

Observe that:

1. From \( \mathcal{D} \) point of view, a dispute always terminates on a positive action: either \( \downarrow \) is in \( \mathcal{D} \) (therefore normalisation terminates on the chronicle \( \tau p \gamma X \downarrow \)) or \( \uparrow \) is in \( \mathcal{E} \), therefore the last action of \( p \) is negative in \( \mathcal{E} \) and positive in \( \mathcal{D} \).

2. Moreover:

\[ \mathcal{D}^+ \subseteq \text{Disp} \mathcal{D} \]

because for any \( \mathcal{c}^+ \in \mathcal{D}^+ \), \( [\mathcal{D} \models Opp_c] = \mathcal{c} \).

6.2.1 Generating \( \text{Disp} \mathcal{D} \)

If we want to calculate all possible disputes on a given design \( \mathcal{D} \) we would not generate all possible designs of opposite base and execute the normalization... What we can do is to trace all possible paths (cf. 3.1.1), and verify that they correspond to a counter-slice. As we have just seen, this amounts to verify that the path is a
dispute/play. This guarantees that the tree of actions of opposite polarity (built as in 3.1.1) satisfies the sub-address condition, which is the sense of the visibility condition for plays. Actually, calculating the paths which are disputes is also the easiest way to calculate the orthogonal of a design.

Let us write a procedure to do this. A completed dispute will always stop on a positive action (possibly $\dagger$). To do an exhaustive research, we trace all possible positive paths of a given length, starting from the minimal ones.

**Definition 6.2.3 (Plays($D$))** Let $D$ be a design of base $X$. We define:

$$P_0(D) = \{ \epsilon \in D^+ : \epsilon \text{ is minimal for } \subseteq \}$$

$$P_{n+1}(D) = \{ pab \text{ s.t. } p \in P_n(D) \text{ and } \exists cab \in D^+ : pa \text{ is a legal play and } \gamma p a = c a \}$$

$$\text{Plays}(D) = \bigcup_n P_n(D), \text{ augmented of } \{ \epsilon \} \text{ if } D \text{ has negative base.}$$

This procedure describes all disputes on $D$. It is easy to understand the step $P_{n+1}$ if one has in mind the token in LAM normalization. Suppose that $p = [D \Rightarrow \mathcal{E}_p]$. If $\gamma p a \in D$ is not followed by $\dagger$ then $\gamma p a = \partial \in \mathcal{E}$, which continues with $\dagger$.

We now look for a design $\mathcal{E}_p a b \not\subseteq \mathcal{E}_p$, such that $[D \Rightarrow \mathcal{E}_p] = p ab$. To do so, we substitute $\dagger$ with an action $a$ that (i) will converge against $D$ and (ii) $\partial a$ is a chronicle (it must satisfy the sub-address condition). Therefore we need an action $a$ s.t. (i) $\gamma p a = c a$ is in $D$ and (ii) $p a$ is a play. Since $\gamma p a = \partial a$, visibility implies that $a$ is justified by an action in $D$.

Now normalization will proceed with $\mathcal{E}_p$ until the token is on the last action of $\partial$. From here it moves on to $a$, and then to $a$ in $\gamma p a = c a \in D$. There is a unique action $b$ which completes this chronicle. Either $b$ is $\dagger$, and we are done, or we add $b$ to $\mathcal{E}$ (we have seen several times that this is always possible) and complete the new chronicle with $\dagger$. Therefore we have built the design $\mathcal{E}_p a b$ we wanted.

**Proposition 6.2.4** Disp $D = \text{Plays}(D)$.

**Proof.** Let $p ab^+ \in \text{Disp } D$. Being a dispute, $p a \subseteq p ab$ is a legal position. For the induction, assume $p \in \text{Plays}(D)$. There is an $i$ such that $p \in P_i(D)$. Moreover, $\gamma q a b = c a b \in D^+$ and $\gamma p a = c a$. Hence, $p ab \in P_{i+1}$.

Let $p \in \text{Plays} D$. If $p \in D^+$, then $p \in \text{Disp } D$. Since $p$ is a play, we already know that $\text{Pull}(p)$ is a cut-net and that $\text{Pull}^X(p) \vdash \text{Pull}^X(p) = p$. We check that $\text{Pull}^X(p) \subseteq D$, by induction on the length of $p$. Assume $p = q ab^+$ and $\text{Pull}^X(q) \subseteq D$. There is an $i$ for which $q a b \in P_i$, $\gamma q a = c a$, and $c a b \in D^+$. Therefore $\gamma q a = c a$ and $\gamma q a b = c a b$ are both chronicles of $D$, hence $\text{Pull}^X(q a b) \subseteq D$.

$\square$
6.2.2 Strategies

Now we want to describe designs as collections of the possible interactions, forgetting the notion of design. What we need is a “coherence” condition on disputes (characterizing disputes on the same design) and a “saturation” condition that guarantees we have all such possible interactions. To do so let us first define a coherent collection of disputes, a strategy.

Definition 6.2.5 (X-Strategy) A P-strategy (O-strategy) $S$ on the universal arena $\triangledown \triangledown$ is a collection of plays (on that arena) which is closed under positive prefix and such that:

Coherence. If $p \neq q \in S$ then $p \land q$ is a positive position (a $P$-position for a P-Strategy, an $O$-position for an O-strategy).

Remark 6.2.6 Maximal position (and only maximal position) can be negative. This means that the last action is followed by $\dagger$ in the design we are describing.

Fact 6.2.7 It is immediate that the above definition is equivalent to the following one, in line with the most standard Games Semantics definition:

$S^1$ is a collection of plays such that

$s0$. $S^1$ is closed under positive prefix;

$s1$. $p \in S^1$ then $p$ is positive;

$s2$. determinism: $sb^X, sc^X \in S^1$ then $b = c$.

Definition 6.2.8 (Ch($\mathcal{D}$)) To an X-strategy $S$ we associate a collection of chronicles $Ch^X(S) = \bigcup_{p \in S^1 \times} Ch^X(p)$.

Remark 6.2.9 $Ch(S)$ can be seen as the super-imposition of the slices associated to the plays in $S$.

It is easy to see that $Disp \mathcal{D}$ is a strategy. However, a strategy does not necessarily correspond to any construct in Ludics.

Example 6.2.10 Let us consider the strategy $S$ on $\triangledown \triangledown$ given by the closure under prefix of

$p_1 = ((\triangledown, \{0, 1, 2\}), (0, I_0), (01, I_01), (1, J))$ and

$p_2 = ((\triangledown, \{0, 1, 2\}), ((0, I_0), (02, I_02), (020, I_{020}), (01, I_{01}), (2, K)))
$S$ is an $O$-strategy. $Ch^O(p_1)$ and $Ch^O(p_2)$ respectively produce the slices:

![Diagram]

The two slices cannot co-exist in the same design because they differ in the way they complete the chronicle $< >, 0, 01^-$. The two resulting chronicles are not coherent.

If two different disputes cover the same design, when they reach to the same negative action they continue in the same way. To arrive at the same action means they are on the same chronicle, that is the two path have the same view.

The notion we need to express this condition is exactly that of innocent strategy.

**Definition 6.2.11 (Innocent strategy)** An $X$-strategy $S$ is innocent if $S^\dagger$ satisfies:

$$sab^+ \in S^\dagger, \ p^+ \in S^\dagger, \ pa \ is \ a \ legal \ position, \ r_{pa}^X = r_{sa}^X \Rightarrow pab \in S^\dagger \ (\ast)$$

Innocence plays two roles:

1. It assures the uniqueness of the move that follows a negative action (cf. Fact 6.2.18);
2. It is a condition of “saturation.” It guarantees that all the possible disputes on a design are taken into account.

**Remark 6.2.12** Observe that Section 6.2.1 allows us to read “innocence” in a very concrete, procedural way. This is immediate as soon as we formulate innocence as in 6.2.16.

The following facts are well known in Games Semantics:

1. The collection of views of an innocent strategy generates, by innocence, the complete strategy.
2. The collections of views of an innocent strategy $S$ is contained in $S$.

The same applies to designs: a design can be seen as the collection of views of an innocent strategy. From the views we can recover the strategy, from the strategy we can extract the views.

Section 6.2.3 works out the details of this, in a general setting. Section 6.2.4 comes back to designs.
6.2.3 Innocent strategies: Views and Plays

The fact that an innocent strategy can be presented either as a set of plays or as a set of views is well known. In this section we work out the details of facts which apply to any innocent strategy. Notice that our constructions Views(−) and Plays(−) correspond to similar operations Fun(−) and Traces(−) defined in [McC96] and [Har99].

To simplify notations, all along this section we consider w.l.o.g. strategies whose plays are all positive. W.r.t our previous definitions, this means that we always work with the closure of our strategies

\[ S = S^\dagger. \]

**Definition 6.2.13 (Views(S))** Let S be an X-strategy. We define

\[ \text{Views}(S) = \{ r.p^X, p \in S \} \]

We say that a set of position \( V \) is stable under view if \( r.p = p \) for all \( p \in V \).

**Definition 6.2.14 (Plays(\( \mathcal{V} \)))** Let \( \mathcal{V} \) be an X-strategy such that Views(\( \mathcal{V} \)) = \( \mathcal{V} \). We define:

\[ P_0(\mathcal{V}) = \{ p \in \mathcal{V} : p \text{ is minimal for } \sqsubseteq \} \]

\[ P_{n+1}(\mathcal{V}) = \{ pab \text{ s.t. } p \in P_n(\mathcal{V}), \exists p' \in \mathcal{V}, r.p^\gamma = p' \text{ and } p' \text{ is a legal position} \} \]

\[ \text{Plays}(\mathcal{V}) = \bigcup_n P_n(\mathcal{V}) \]

**Fact 6.2.15** Plays(\( \mathcal{V} \)) satisfies the property:

If \( p \in \text{Plays}(\mathcal{V}) \) and \( \exists p' \in \mathcal{V} \text{ s.t. } r.p^\gamma = p' \text{ and } p' \text{ is a legal position} \), then \( pab \in \text{Plays}(\mathcal{V}) \).

**Fact 6.2.16 (Innocence by views)** If S is an X-strategy and Views(S) \( \subseteq S \), then the property (1) in Definition 6.2.11 is equivalent to the following one:

\[ cab^+ \in \text{Views}(S), \quad p \in S, \quad p \text{ is a legal position}, \quad r.p^\gamma = c \text{ then } pab \in S \quad (**\)

**Remark 6.2.17** We indicate \( r.p^\gamma \) as c on purpose, to remember we can think of it as a chronicle.

**Some properties**

Observe that determinism together with innocence implies in particular that

**Fact 6.2.18 (Determinism under view)** Let S be an innocent X-strategy. If \( pab \in S, qac \in S, r.p^\gamma = r.qa^\gamma \) then \( b = c \).
This in particular solves example 6.2.10. It also means that View$(S)$ satisfies itself
determinism (cf. Definition 6.2.5) and thus it is a strategy.

**Proposition 6.2.19 (Closure under view)** If $S$ is an innocent X-strategy then
View$(S) \subseteq S$.

**Proof.** The proof is by induction on the length of $q \in S$. If $q = \epsilon, q = \langle a^+ \rangle$ or
$q = \langle a^-, b^+ \rangle$ the result is immediate since $\gamma^\gamma = q$.
Let $q = sc^+ta^-b$ where $c$ justifies $a$. Therefore $\gamma q^\gamma = \gamma sc^\gamma ab$. By induction,
$\gamma sc^+ \gamma \in S$. Innocence then implies that $\gamma sc^\gamma ab \in S$, because $\gamma sc^\gamma a$ is a legal
position, and $\gamma \gamma sc^+ \gamma a^\gamma = \gamma s^\gamma c^+a^\gamma = \gamma sc^+ta^-\gamma$. □

**Proposition 6.2.20 (Saturation)** Let $T$ be any strategy and $S$ an innocent strat-
egy. If View$(T) \subseteq S$ then $T \subseteq S$.

**Proof.** Proof by induction on the length of $q \in T$. Let $q = sab^+$. Let $\gamma sa^-b^+\gamma =
\gamma ca^-b^+$. By hypothesis, $\gamma ca^-b^+ \in S$ (then $\epsilon \in S$). By inductive hypothesis, $s \in S$.
Since $\gamma sa^\gamma = \gamma ca^\gamma$ we have by innocence that $sab \in S$. □

**Plays vs. Views**

**Proposition 6.2.21** Let $S$ be an innocent X-strategy.

View$(S)$ is an X-strategy, stable under view.

**Proof.** We already observed that determinism is implied by Proposition 6.2.18. Closure under prefix follows from the closure under prefix of $S$. □

**Proposition 6.2.22** Let $V$ be an X-strategy stable under view.

View$(\text{Plays}(V)) = V$

**Proof.** We need to show that View$(\text{Plays}(V)) \subseteq V$. Consider $p \in \text{Plays}(V)$.
Let $i$ be the smallest index such that $p \in P_i$. If $p \in P_0$ then $\gamma p^\gamma \in V$. Otherwise,
p = $sab, \gamma sa^\gamma = \gamma ca, cab \in V$, which concludes the proof because $\gamma p^\gamma = \gamma sa^\gamma b = \gamma cab$. □

**Proposition 6.2.23** Let $V$ be an X-strategy stable under view.

Plays$(V)$ is the smallest innocent strategy which contains $V$. 
Proof. \( \text{Plays}(\mathcal{V}) \) is a strategy. It is deterministic because \( \mathcal{V} \) is. Indeed, if \( sa, sb \in \text{Plays}(\mathcal{V}) \) then \( s a^n = r \) and \( s b^n = r \) are in \( \mathcal{V} \), hence \( a = b \).

Moreover \( \text{Plays}(\mathcal{V}) \) is innocent because by construction it satisfies the condition (***) of 6.2.16.

If \( S \) is an innocent strategy and \( \mathcal{V} \subseteq S \) then \( \text{Plays}(\mathcal{V}) \subseteq S \), because \( \text{Views}(\text{Plays}(\mathcal{V})) = \mathcal{V} \) and Proposition 6.2.20. \qed

**Proposition 6.2.24** Let \( S \) be an innocent X-strategy.

\[ \text{Plays}(\text{Views}(S)) = S \]

**Proof.** Let \( \text{Views}(S) = \mathcal{V} \). \( \text{Views}(S) \subseteq S \) implies \( \text{Plays}(\mathcal{V}) \subseteq S \).

We show \( S \subseteq \text{Plays}(\text{Views}(S)) \) by induction on the length of \( p \in S \). Assume \( tab \in S \). Observe that \( \tau tab^n = \tau ta^n b = tab \in \mathcal{V} \). By induction, \( t \in \text{Plays}(\mathcal{V}) \), \( \tau ta^n = ca \), hence \( tab \in \text{Plays}(\mathcal{V}) \). \qed

### 6.2.4 Designs and innocent strategies

**Definition 6.2.25** Let us indicate by \( \text{Views}^*(S) \) the set \( \text{Views}(S) \setminus \{\epsilon\} \). Let us indicate by \( \overline{S} \) the closure under non empty negative prefix of \( S \).

**Fact 6.2.26** \( \text{Ch}(S)^+ = \text{Views}^*(S^+) \).

\[ \overline{S} = \overline{S}^+ \text{ and } \text{Ch}(S) = \text{Views}^*(S^+) \]

**Proposition 6.2.27** Let \( \mathcal{D} \) be a design of base \( X \). Then

\[ \mathcal{D}^+ \text{ is an X-strategy stable under view.} \]

Unfortunately, a strategy stable under view is not a design, in that it does not satisfy the condition of linearity ("propagation"). To guarantee that all slices in a design are linear, it is not enough that each single play is linear. This phenomenon is of the same nature as similar phenomena we observed when studying interactive observability. Since this deserves a separate analysis, it will be discussed in Section 6.3.2.

Here we simply translate the condition of propagation from chronicles to views.

**Definition 6.2.28 (Propagation)** A strategy \( S \) satisfies the propagation condition if

If \( \tau \kappa, \tau \kappa' \in \text{Views}(S) \) and \( t = c \ast (\xi, I) \ast \overline{\kappa}, t' = c \ast (\xi', I') \ast \overline{\kappa'} \) then \( \xi = \xi' \).

**Proposition 6.2.29** Let \( \mathcal{V} \) be an X-strategy which is stable under view and which also satisfies propagation, then
\[ \mathcal{V} \text{ is the positive part of a design } \mathfrak{D}: \mathcal{V} = \mathfrak{D}^+ \]

Using previous proposition and the fact (cf. 6.2.1) that \( \text{Disp}(\mathfrak{D}) = \text{Plays}(\mathfrak{D}) = \text{Plays}(\mathfrak{D}^+) \) we have that:

**Proposition 6.2.30** (i) Let \( \mathfrak{D} \) be a design. Then:

\[ \text{Disp}(\mathfrak{D}) \text{ is an innocent strategy,} \]

the smallest innocent strategy which contains \( \mathfrak{D}^+ \).

(ii) Let \( S \) be an innocent strategy which satisfies propagation. Then:

\[ \text{Ch}(S) \text{ is a design.} \]

As for Views and Plays,

**Proposition 6.2.31**

\[ \text{Disp Ch}(S) = S \text{ and } \text{Ch}(\text{Disp } \mathfrak{D}) = \mathfrak{D} \]

**Proof.** \( \text{Disp Ch}(S) = \text{PlaysCh}(S)^+ = \text{PlaysViews}(S) = S. \)

\[ \text{Ch}(\text{Disp } \mathfrak{D}) = \overline{\text{ViewsPlays}(\mathfrak{D}^+)} = \overline{\mathfrak{D}^+} = \mathfrak{D} \]

\[ \square \]

6.3 Issues on Linearity

6.3.1 Extracting strategies from a play

In Section 6.1.2 we show that to a play \( p \) we can associate both a design and a counter-design. Observe that to do so it is essential that \( p \) is linear.

For example, to the play \( \langle \alpha, a0, \alpha \rangle \) we can associate a design, but not a counter-design. In other words, this play belongs to an innocent strategy, but not to an innocent counter-strategy.

6.3.2 Propagation and Linear Plays

As we have seen, there is only a delicate point to establish a correspondence between designs and innocent strategies, namely that it is not enough to consider linear plays. We need to explicitly ask the condition of propagation. This is due to the same phenomenon we observed in 3.2.1.

If we consider linear plays, without imposing propagation on strategies, what do we have in ludics? This is not the same as abolishing the condition of propagation in ludics, that would destroy the whole structure. It rather corresponds to making
linearity local. *W.r.t. computation*, a design that satisfies linearity locally (on strong slices) will behave as a design that satisfies linearity globally (on any slice).

We may compare the situation to that in lambda calculus, where non-linearity of a term may be serious, or not (think for example of the expression “if then else”).

**Propagation.** The condition of propagation (cf. 1.2.4) is a way to explicitly demand the separation of the contexts. This is exactly what it describes. It is immediate that we can reformulate propagation as:

“In each slice, any address only appears once”.

If we abolish this condition, we need to radically change the theory. In particular, we lose linearity of the chronicles. For example a slice of the form

\[
\xi_{10} \quad \alpha_0 \quad \alpha_0 \\
\alpha \quad \alpha \\
\xi_1 \quad \xi_2 \\
\xi
\]

may normalize into

\[
\alpha_0 \\
\alpha \\
\alpha_0 \\
\alpha
\]
At this point, we do not know how to deal with separation.

**Linear plays.** Let us consider an innocent strategy of linear plays. We can associate to it a collection of chronicles. For example, to the innocent strategy \( \{ (\xi, \xi_1, \alpha), (\xi, \xi_2, \alpha) \} \) we would associate

\[
\begin{array}{c}
\alpha \\
\downarrow \\
\xi_1 \quad \xi_2 \\
\downarrow \\
\xi \\
\end{array}
\]

The objects associated to innocent strategies of linear plays do not satisfy propagation, but a weaker property:

"In each slice included in a strong slice, any address only appears once"

This means in particular that no sub-address of \( \xi \) appears after any of the two occurrences of \( \alpha \). As a consequence, if a path visits one of the two occurrences of \( \alpha \), then it cannot visits the other one, because of Proposition 3.3.4 and of the closure principle for normalization paths (Proposition 2.5.7).

The objects described by innocent linear strategies are therefore linear for all computational purpose.

Observe however that working in this way we would not reach a full completeness result for MALL. Typically, we would find a proof such as the following one. This is exactly the proof associated to the design above, inside the behaviour which interprets the conclusion of the derivation.

\[
\frac{0 \vdash A \quad 0 \vdash B}{\downarrow A \vdash \downarrow T, A \quad \downarrow B \vdash \downarrow T} \quad \frac{\downarrow A \vdash \downarrow T \quad \downarrow B \vdash \downarrow T}{\vdash (\downarrow A) \otimes (\downarrow \mathcal{B}), \downarrow T}
\]

Inside the theory, there is no way to *interactively* observe that \( \downarrow T \) belongs to both premises of the tensor.

Observe that if we do not have constants, and therefore all winning designs terminate by a fax, the notion of linear play entails propagation, because we can always find a strong slice that contains any given slice. We just need to perform some \( \eta \)-expansions to have enough space to allow all player to reach all addresses.
6.4 Discussion

6.4.1 Comments

Plays vs. Views

As we have shown, to present a design as a set of chronicles correspond to presenting an innocent strategy as the set of its views.

Both traditions (plays presentation and views presentation) are present in the literature, and different authors have done different choices. [ARM00], [HO00], [Nic94] prefer the play presentation, [CH98] adopt the view presentation.

The issue of lifting a play to a strategy (not a counter-strategy) is addressed by Danos Herbelin and Regnier in [DHR96].

Errors and daemons

In [Chr00], Chroboczek defines a notion of error. He deals with the errors “implicitly” (maximal plays in a strategy may be negative). Here we follow that choice.

6.4.2 Further work: Behaviours and Games

Let us sketch the way one would follow to go further.

It is immediate that two strategies belonging to opposite players are orthogonal if they intersect in a play.

Definition 6.4.1 (Orthogonality) \( S \perp \mathcal{T} \) if \( S \cap \mathcal{T} = \emptyset \).

One can then proceed as in [Gir01b]. In particular, one can define a type (a game, a behaviour) in an internal way, that is without setting special rules for each type:

Definition 6.4.2 (Behaviours / Games) A game \( G \) on the arena \( \vdash \langle \rangle \) is a set of innocent strategies on the same arena equal to its biorthogonal.

One can retrieve a more standard definition of game when looking at the incarnation.

Definition 6.4.3 (Incarnation) The incarnation \( |S| \) of \( S \) is the set of disputes which occur both in \( S \) and a strategy of \( G^\perp \).

A strategy is incarnated or material when \( S = |S| \). We define the incarnation \( |G| \) of \( G \) as the set of its material designs

It is immediate that we have again pleasant phenomenons such as that \( A \& B = A \cap B \) and \( |A \& B| = |A| \times |B| \).

The “standard” definition of games would correspond to a direct definition of the incarnation.
Appendix A

An introduction to Uniformity and Completeness

The research on full completeness for logic and λ-calculus has motivated a large amount of work, in particular over the past 10 years. J-Y. Girard has presented in [Gir01b] and in [Gir01a] a new approach in a framework called Ludics. The central notion in Ludics is that of design. A design corresponds to a proof, regarded under all possible points of view (syntactical proof, λ-term, function, clique, ...). Its orthogonal (counter-proof, anti-clique, ...) is also a design. Both proofs and counter-proofs are objects of the same nature. Then we will work with a set of designs equal to its biorthogonal, the behaviour (logical formula, type, coherent space,...). The main novelty of this approach is to overcome the duality syntax/semantics: the way for full completeness is then open.

In Ludics there are two levels of completeness: (i) the internal completeness and (ii) the full correspondence between ludic objects and logic. The internal completeness is about the decomposition of a connective (in some sense a ludical counter part of the sub-formula property). This is an essential part but not the whole story of full completeness, that can be stated as:

If $\mathcal{D}$ is a “good” design in the behaviour $\mathbf{A}$ associated to a closed $\Pi^1$ formula $\mathbf{A}$ then there is a $\text{MALL}_2$-proof of $\mathbf{A}$ which is interpreted by $\mathcal{D}$. ($\text{MALL}_2$ is the second order linear propositional calculus presented in [Gir01b]).

In order to obtain this result, the notion of behaviour has to be enriched, becoming that of “bihaviour,” where the central role is played by the property of uniformity.

---

1This note has been written in collaboration with Marie-Renee Fleurry-Donnadieu and Myriam Quatrini and will appear in the volume “Linear Logic in Computer Science,” Ehrhard, Girard, Ruet, Scott eds.
Here we are interested in this notion. Our aim is to help understanding the uniformity property. Why is it necessary? How does it work in the completeness result?

In this note, we will omit some details. The calculus we refer to is MALL_2, the second order linear propositional calculus introduced in [Gir01b]. Anyway, for the purpose of our examples, the reader can think of a standard two-sided calculus for multiplicative-additive linear logic, augmented with the connective \( \downarrow \), the “shift,” which simply changes the polarity of a formula.

### A.1 Why uniformity: examples

The full completeness theorem of Ludics is based on two ingredients: internal completeness and uniformity. The former allows us to decompose compound behaviours, while the latter takes care of the atoms, in a sense we shall make precise.

A behaviour \( \mathcal{G} \) is a set of designs equal to its biorthogonal \( \mathcal{G}^{\perp\perp} \). Internal completeness for the connectives of Ludics means that “there is no biorthogonal,” in the sense that any compound behaviour is born equal to its biorthogonal. Since the biorthogonal does not introduce new objects, we have a complete description of all the designs in the behaviour. For examples, \( A \otimes B \) is defined as \( (A \otimes B)^{\perp\perp} \) where \( A \otimes B = \{ a \otimes b, a \in A, b \in B \} \). Thus, for any \( \mathcal{D} \in A \otimes B \) we know we can decompose it as \( \mathcal{D}_1 \otimes \mathcal{D}_2 \), with \( \mathcal{D}_1 \in A \) and \( \mathcal{D}_2 \in B \).

To any closed \( \Pi_1 \) formula \( F \) of MALL_2, one can associate a behaviour \( \mathcal{F} \) which is its interpretation. One would like to say that any material, “winning” design in \( \mathcal{F} \) corresponds to a proof of \( F \) in MALL_2 (full completeness). As it is well explained by Girard, the proof of completeness can be resumed into a slogan: “find the last rule”. Concretely, given a design \( \mathcal{D} \), the whole process consists in “producing a last syntactical rule.”

If \( \mathcal{D} \) belongs to the interpretation of the formula \( F \), to decompose it is always immediate for the negative formulae, and exploits the “internal completeness” for the positive formulae. In this way, we are able to find the premises of the rule, represented by the designs \( \mathcal{D}_i \in F_i \), where \( F_i \) correspond to the subformulae of \( F \). The fact that each \( \mathcal{D}_i \) is again material and winning, allows us to carry on the induction.

In practice one works with sequents of behaviours rather than single formulae. We recall that the sequent of behaviours \( \vdash G \) is equal to \( G \), \( G \vdash \) is equal to \( G^{\perp} \) and:

\[
\begin{align*}
\exists \xi \in H \vdash \Delta \iff & \text{ for all } \xi \in H, \ [\xi, \xi] \not\vdash \Delta, \\
\exists \xi \not\vdash H, \Delta \iff & \text{ for all } \xi \in H^{\perp}, \ [\xi, \xi] \not\vdash \Delta.
\end{align*}
\]

The “closure principle,” which exploits associativity and separation, allows us to reduce the problem to the decomposition of a single formula at a time. As an
example, consider a design \( \mathcal{G} \) in \( \vdash P \oplus Q, R \). Assuming \( \mathcal{G} \) first focus on (the address of) \( P \oplus Q \), we cut \( \mathcal{G} \) against a design \( \mathcal{E} \in R \) and produce a design \( [\mathcal{G}, \mathcal{E}] \) which belongs to \( P \oplus Q \). We are then able to find a design \( \mathcal{D} \) either in \( P \) or in \( Q \). Suppose it is in \( P \), we can retrieve the wanted premise as the design \( \mathcal{G} \) in \( \vdash P, R \) such that \( \mathcal{D} = [\mathcal{G}, \mathcal{E}] \).

However, one need to be careful when dealing with the negative case, such as \( \mathcal{G} \in R \vdash P \oplus Q \), where normalization entails a choice of the premises. \( R \) depends on the interpretation of the atoms, which can be whatever behaviour we like. When we cut \( \mathcal{G} \) against a design in \( R \), the result also may depend on the choice of interpretation. Think simply of \( R = X \). Even if we are able to decompose the design in \( P \oplus Q \) for any interpretation of \( X \), we could not be able to put things together again, as we shall illustrate. Internal completeness is no longer enough. The key point is that the premises need to be uniquely defined.

Next examples are paradigmatic of the situations which involve uniformity, making it necessary to move from behaviours to behaviours. In Section A.3 we will discuss the "identity axioms."

A.1.1 Example: a matter of focalization

Let us consider a behaviour \( X \vdash P, Q \), where \( X \) is the interpretation of an atom and \( P, Q \) are compound behaviours, respectively located in \( \sigma \) and \( \tau \). Consider a design \( \mathcal{G} \) of the form:

\[
\vdash \xi, I, \sigma, \tau \quad \vdash \xi, J, \sigma, \tau
\]

\[
\xi \vdash \sigma, \tau
\]

In order to decompose \( P \) or \( Q \) we first need to cut \( \mathcal{G} \) with a design in \( X \); for any choice we should have the same proof. Anyway, let us take \( X = \{D_I \cup D_J\}^\perp, \)

where

\[
D_I : \\
\vdash \xi, (I, I) \\
\vdash \xi, (J, I) \\
D_J : \\
\vdash \xi, (I, J)
\]

The normal form \([\mathcal{G}, D_I]\) is

\[
\vdash \sigma, \delta \vdash \tau \quad \vdash (\sigma, I)
\]

\[
\vdash \sigma, \tau
\]
which first focalizes on \( \sigma \), the address of \( P \).

The normal form \( [\bar{\gamma}, \mathcal{D}_J] \) is instead

\[
\vdots \\
\cdots \quad \tau, j \vdash \sigma \\
\vdash \sigma, \tau \\
(\tau, J)
\]

which first focalizes on \( \tau \).

The “last rule” is then not uniquely defined.

### A.1.2 Example: decomposing \( P \oplus Q \)

Let \( \bar{\gamma} \) be a design of base \( \xi \vdash \sigma \) belonging to the behaviour \( X \vdash P \oplus Q \); \( X \) is located in \( \xi \), \( P \) in \( \sigma.1 \) and \( Q \) in \( \sigma.2 \).

\[
\vdots \\
\cdots \quad \sigma.1 \vdash \xi_1 \\
\vdash \xi_1, \sigma \\
\sigma.2 \vdash \xi_2 \\
\vdash \xi_2, \sigma \\
(\xi, P_\mathcal{J}(N))
\]

We interpret the atom as in the previous example. In order to decompose \( P \oplus Q \), we cut \( \bar{\gamma} \) with a design \( \mathcal{D} \) in \( X \), to obtain a design in \( \vdash P \oplus Q \).

If we cut \( \bar{\gamma} \) with \( \mathcal{D}_I \), we obtain a design of the form

\[
\vdots \\
\vdash \sigma.1 (\sigma.1)
\]

which is in the component \( P \).

If instead we cut with \( \mathcal{D}_J \), we obtain a design in the component \( Q \):

\[
\vdots \\
\vdash \sigma.2 (\sigma.2)
\]

Once again, it is impossible to associate \( \bar{\gamma} \) with a derivation of \( X \vdash P \oplus Q \). Depending on the design we choose in the atom interpretation, we obtain once a design in \( X \vdash P \) and once a design in \( X \vdash Q \).

\(^2\) We are making a slight simplification here, in fact \( P \) stands for \( \downarrow P \) and \( Q \) for \( \downarrow Q \).
A.2 From proofs to uniformity

All the designs in the previous examples are incarnated and daimon-free, but we cannot associate a proof to them. The premises of the last rule depend on the interpretation of the atom: different choices lead to different rules. There is something “non-uniform” in this; we are going to make explicit this intuition.

Consider as working example for the discussion a design $\mathcal{F}$ of base $\xi \vdash \sigma$ belonging to $\mathcal{X} \vdash \mathcal{P}$ for any interpretation of the atom. The first rule is necessarily $(\xi, \mathcal{P}_{\mathcal{F}(\mathbb{N})})$, because $\mathcal{X}$ could be any behaviour, and $[\mathcal{F}, \mathcal{D}]$ must converge for any possible $\mathcal{D}$. The content of uniformity is that whatever premise the normalization with $\mathcal{D}$ selects, the proof should “continue in the same way”. We need to separate the designs according to how they interact with the orthogonal. Unfortunately, by definition of the orthogonal, convergence does not allow us any discrimination. To have a finer distinction, we need to consider a larger universe. The central role played by the premises of $\xi \vdash \sigma$ leads to the notion of partial design of a behaviour $\Gamma$.

A partial design $\mathcal{D}'$ is a “part of a design” $\mathcal{D} \in \mathcal{G}$: a subtree that has the same base, but where some of the premises may be missing. A typical example of partial design is a slice of a design $\mathcal{D}$: a subtree of $\mathcal{D}$ obtained selecting in all negative rules at most one premise. An extreme example of partial designs are the empty ones ($\mathcal{F}id$, $Skunk$). $\mathcal{G}^p$ denotes the set of all designs (total and partial) of a behaviour $\mathcal{G}$.

Now we can express the fact that:

all the partial designs $\vdash \xi \vdash \sigma (\xi; \{I\})$ included in $\vdash \xi \vdash \sigma (\xi; \mathcal{P}_{\mathcal{F}(\mathbb{N})})$ lead essentially to the same proof.

It becomes natural to introduce on $\mathcal{G}^p$ a partial equivalence relation (i.e. asymmetric and transitive relation) $\cong$ which separates the partial designs with respect to normalization. The key is that the equivalence relation identifies the closed nets normalizing into $\mathcal{D}ai$ and those normalizing into $\mathcal{F}id$.

Since designs and counter-designs (proofs and counter-proofs) have the same status, we need to consider the partial equivalence on $\mathcal{G}^{p\perp}$ induced by normalization, and come back by bi-orthogonal... Two partial designs are in the same class, if their reactions against equivalent partial counter-designs are the same.

As an example, the trivial equivalence relation is the one that identifies all proper\(^3\) designs of $\mathcal{G}^p$ ($\cong \perp$ distinguishes all proper designs of $\mathcal{G}^{p\perp}$)

In a sequent of behaviours $\mathcal{X} \vdash \mathcal{P}$, we want again that two partial equivalent designs (saying $\mathcal{D}$ and $\mathcal{D}'$) react in the same way against two equivalent designs of $\mathcal{X}$ (saying $\mathcal{E}$ and $\mathcal{E}'$); that means that $[\mathcal{D}, \mathcal{E}]$ and $[\mathcal{D}', \mathcal{E}']$ produce two equivalent

\(^3\)Any positive design distinct from $\mathcal{D}ai$ and $\mathcal{F}id$. 
designs in $P$.

On compound behaviours, the equivalence relations we are interested in must conserve the properties of the connective. For example, the behaviour $P \oplus Q$ is the union of the two distinct behaviours $P, Q$. The equivalence relation keeps distinct designs coming from distinct behaviours, while two designs are equivalent if they are equivalent either in $P$ or $Q$.

A design $D$ candidate to be a proof has to be equivalent to itself; it is called uniform.

Let us summarize a few definitions:

1. **A behaviour** is a couple $(G, \cong)$ equal to its biorthogonal.

2. **Sequents of behaviours**. Consider the sequent $G_0 \vdash G_1, \cdots, G_n$. We obtain a behaviour by considering the partial equivalence defined by: $G \cong_G G_0 \iff \forall D_0 \in G_0^p \forall D_i \in G_i^p \exists D' \in [G, D_0, D_1, \cdots, D_n] = [G', D_0', D_1', \cdots, D_n']$.

3. **Compound behaviours**. Let $(G_1, \cong_1)$ and $(G_2, \cong_2)$ be two disjoint behaviours on the same base.

   $\cong = \cong_1 \otimes \cong_2$ is defined by: for all $D, D' \in (G_1 \otimes G_2)^p$

   $D \cong G_1 \otimes G_2 D' \iff \exists i \in \{1, 2\}$ such that $D \in C_i$ and $D' \in C_i$.

4. **Uniform designs**. Let $D$ be a partial design in the behaviour $(G, \cong)$.

   $D$ is uniform iff $D \cong D$.

**A.2.1 Back to the examples.**

We are now able to make precise the intuition that the designs in the starting examples are not uniform.

**On example A.1.1 (Focalization)**

Let us consider the example of Section A.1.1. Both $[\mathfrak{H}, D_f]$ and $[\mathfrak{H}, D_j]$ are located on the base $\vdash \sigma, \tau$. We close each of these nets, cutting with $\eta\sigma^{-}$ on the base $\sigma \vdash$ and with $\mathcal{G}k$ on the base $\tau \vdash$. We know that such two partial designs belong to any negative (partial) behaviour.

$$\eta\sigma^{-} = \cdots \vdash \sigma, \eta \vdash \cdots (\sigma, \mathcal{P}(\mathbb{N})) \quad \mathcal{G}k = \tau \vdash (\tau, \emptyset)$$
Since $[[\mathcal{A}, \mathcal{D}_I]]$ first focus on $\sigma$, while $[[\mathcal{A}, \mathcal{D}_J]]$ first focus on $\tau$, it is immediate that $[[\mathcal{A}, \mathcal{D}_I], \mathcal{Dai}_x, \mathcal{E}t_x] = \mathcal{D}_I$, while $[[\mathcal{A}, \mathcal{D}_J], \mathcal{Dai}_x, \mathcal{E}t_x] = \mathcal{D}_I$. We thus have that $[[\mathcal{A}, \mathcal{D}_I]] \not= e[[\mathcal{A}, \mathcal{D}_J]]$. On the other hand, if we take on $X$ the trivial relation that quotients all proper designs, we have $\mathcal{D}_I \equiv \mathcal{D}_J$. The definition of sequent of bhiaviours implies that $\mathcal{A} \not= \mathcal{A}$: $\mathcal{A}$ is not uniform.

**On example A.1.2 (Plus)**

Let us look at the example of section A.1.2. The two premises do not behave in the same way: one works in the left component of $P \oplus Q$, the other works with the right one. We already observed that $[[\mathcal{A}, \mathcal{D}_I]] \in P$ and $[[\mathcal{A}, \mathcal{D}_J]] \in Q$. The equivalence relation on a disjoint union of bhiaviours ($P \oplus Q$) distinguishes the element coming from distinct components. Hence $[[\mathcal{A}, \mathcal{D}_I]] \not= e[[\mathcal{A}, \mathcal{D}_J]]$. As in the previous example, it is enough to consider on $X$ the trivial equivalence to realize that $\mathcal{A}$ is not uniform.

To show an uniform design in the same behaviour, let assume that $P = \Phi(X)$ and $Q = \Psi(X)$ are distinct delocations of $X^4$. Consider $e =:

\[
\begin{align*}
\mathcal{A}x
\vdots
\sigma.1 \vdash \xi.I \\
\forall I \in P_f(\mathbb{N}) \quad \vdash \xi.I, \sigma \\
\xi \vdash \sigma
\end{align*}
\]

(\xi, P_f(\mathbb{N}))

For any design $\mathcal{D}$, $[e, \mathcal{D}] = \Phi(\mathcal{D})^5$. Hence, as soon as $\mathcal{D}, \mathcal{D}^I \in X$ are equivalent, so are $[e, \mathcal{D}]$ and $[e, \mathcal{D}^I]$.

**A.3 Uniformity and Fax**

The proof of completeness goes on decomposing the positive formulas of the sequent, and accumulating atoms on the left-hand side. This process stops when it reaches a positive atom on the right-hand side: $P \vdash X, \Delta$. Ideally, we should have reached (the interpretation of) an "identity axiom" $X \vdash X$. The only good inhabitant should be the design that interpret the identity, or rather the infinite $n$-expansion of it: the $\mathcal{A}x$.

In the case of $X \vdash X$, $\mathcal{A}x$ is the only incarnated design which does not make use of daimon. We do not need anything else to prove it. Anyway, to deal with the general case, uniformity become necessary to establish the following central result ([Gir01b]):

---

4To be precise, we should write $P = \downarrow \Phi(X)$ and $Q = \downarrow \Psi(X)$.

5Precisely, $\uparrow \Phi(D)$. 

---
Proposition A.3.1 (Polymorphic Lemma) If \( \tilde{\gamma} \in \Gamma \vdash X, \Delta \) is uniform, incarnated and daimon-free then \( X \in \Gamma \) and \( \tilde{\gamma} \) (essentially) behaves as a \( \tilde{\gamma}ax \).

The essential case is \( X, X \vdash X \).

**Designs in** \( X \vdash X \)**

Uniformity is not required to prove that

\[
\tilde{\gamma}ax \text{ is the only design } \tilde{\gamma} \in X \vdash X \text{ which is stubborn and incarnated.}
\]

Proof: Let \( \tilde{\gamma} \in X \vdash X \) be a design based on \( 1 \vdash \sigma \). We first observe that the first negative rule must be \((1, P_f(\mathbb{N}))\), to allow \( \tilde{\gamma} \) to converge with any possible design.

Now we fix a ramification \( I \) and observe that the rule above \( \vdash 1 \ast I, \sigma \) cannot focalize on a \( 1 \ast i \); only \( \sigma \) is available as focus, as one can check choosing a convenient designs \( D \) which makes the cut \([\tilde{\gamma}, D]\) fail when the condition is not realized. Moreover, for the same argument, the only possible rule is \((\sigma, I)\).

The last step is to check that the repartition of the addresses is one-to-one: \( \cdots \sigma \ast i \vdash 1 \ast i \cdots (\sigma, I) \)

This also can be checked choosing a convenient \( D \) and applying combinatory arguments.

**Designs in** \( X, X \vdash X \)

Let now make explicate the two uniform designs in \( X, X \vdash X \): \( \tilde{\gamma}ax_1 \) and \( \tilde{\gamma}ax_2 \). We can think of \( \tilde{\gamma}ax_1 \) as the first projection, i.e. that \( \tilde{\gamma}ax_1 \) maps a pair of designs of \( X \) (saying \( D_1 \odot D_2 \)) on a delocation of \( D_1 \).

Suppose that the behaviours \( X \) in the left side of the sequent are located on \( 1 \) and that they are disjoint (say we applied two delocations, the first mapping all the bias on even biases, the second mapping the biases on odd biases). The behaviour \( X \) in the right side of the sequent is located on \( \sigma \) by the delocation \( \theta \). Let \( \tilde{\gamma}ax_1 = \)

\[
\begin{align*}
\cdots \sigma \ast i \vdash 1 \ast i & \cdots (\sigma, I) \\
\cdots 1 \vdash (I \cup J), \sigma & \cdots (1, \{I \cup J : I, J \text{ as in } (*) \text{ below}\}) \\
1 \vdash \sigma &
\end{align*}
\]

(*): \( I \in P_f(2\mathbb{N}) \) and \( J \in P_f(2\mathbb{N} + 1) \).

\( \tilde{\gamma}ax_2 \) is obtained by exchanging \((\sigma, I)\) with \((\sigma, J)\).

Observe that \([\tilde{\gamma}ax_1, D_1 \odot D_2] = \theta(D_1)\) and \([\tilde{\gamma}ax_2, D_1 \odot D_2] = \theta(D_2)\).

The core of the Polymorphic Lemma really consists in showing that
\[ \mathfrak{ax}_1 \text{ and } \mathfrak{ax}_2 \text{ are the only uniform incarnated and daimon-free designs in } \mathbf{X}, \mathbf{X} \vdash \mathbf{X}. \]

The argument relies on the following points:
- the normalization between uniform designs produces an uniform design.
- the only uniform incarnated design in the behaviour \( (\mathcal{D} = \{ \emptyset \}^\perp) \) is \( \mathcal{D} \) itself.

To sketch a case, if \( \mathcal{D}_1 \neq \mathcal{D}_2 \) one considers the bihaviour \( \mathbf{X} = \mathcal{D}_1 \oplus \mathcal{D}_2 \), and since \( [\mathfrak{g}, \mathcal{D}_1 \otimes \mathcal{D}_2] \in \mathbf{X} \), its incarnation must be either \( \mathcal{D}_1 \) or \( \mathcal{D}_2 \). Monotonicity of normalization then allows one to complete the argument.

Observe that there are also plenty of non uniform design in \( \mathbf{X}, \mathbf{X} \vdash \mathbf{X} \). To have one we can build an \( \mathfrak{g} \) such that the premises above \( \vdash 1 * (I \cup J), \sigma \) depend on \( I \cup J \).

For example we set that above \( I_0 \cup J_0 \) we have the same rules as in \( \mathfrak{ax}_2 \) and for all the others \( I \cup J \) we have the same rules as in \( \mathfrak{ax}_1 \). We then obtain the following design, where again \( I \) contains only even bias, and \( J \) only odd bias.

\[
\begin{align*}
\mathfrak{ax}_{\sigma(i),1} & \vdash 1 * 1 \quad & \mathfrak{ax}_{\sigma(j),1} & \vdash 1 * 1 \\
\vdash 1 * \{I \cup J\}, \sigma & \quad & \vdash 1 * \{I_0 \cup J_0\}, \sigma \\
\vdash 1 & \vdash \sigma
\end{align*}
\]

Observe that \( [\mathfrak{g}, \mathcal{D}_1 \otimes \mathcal{D}_2] = \theta(\mathcal{D}_1) \text{ or } \theta(\mathcal{D}_2) \) depending on the first actions in \( \mathcal{D}_1 \) and \( \mathcal{D}_2 \).

Consider now any bihaviour \( \mathbf{X} \) containing four designs \( \mathcal{D}_1 \) and \( \mathcal{D}_i' (i = 1, 2) \) such that \( \mathcal{D}_i \cong \mathcal{D}_i' \) but \( \mathcal{D}_1 \neq \mathcal{D}_2 \). We then have \( \mathcal{D}_1 \otimes \mathcal{D}_2 \cong \mathcal{D}_1' \otimes \mathcal{D}_2' \), but \( [\mathfrak{g}, \mathcal{D}_1 \otimes \mathcal{D}_2] = \mathcal{D}_1 \neq \mathcal{D}_2' = [\mathfrak{g}, \mathcal{D}_1' \otimes \mathcal{D}_2'] \). Then \( \mathfrak{g} \) is not uniform.
Bibliography


