université PARIS DIDEROT PARIS 7

Rapport de TRE

Master 1 Informatique
Spécialité Recherche

Encadrant : Alexis Saurin

Towards a term syntax for L-nets

Pablo DONATO

April 1, 2020

Abstract

We present a novel term syntax for boxed paraproof nets of multiplicative linear logic, allowing for a linear, uniform representation of paraproofs with partial sequentiality.

We start by explaining the relationship between maximally and minimally sequential representations of proofs, namely sequent calculus proofs and proof nets. It appears natural in this setting to generalize results to a wider class of objects that includes "erroneous" proofs, called paraproofs.

Next we introduce term syntaxes for both representations, inspired respectively by the c-designs of K. Terui, and the algebraic presentation of differential interaction nets of D. Mazza. We relate the two syntaxes through a two-step desequentialization procedure, where paraproofs are first translated into totally sequential paraproof nets sharing the same computational behavior. This is made possible by augmenting paraproof nets with a box construct similar to those used traditionally to encode exponentials. Desequentialization then proceeds by simply removing boxes, offering fine-grained control over the degree of sequentiality of the intermediate paraproof nets.

We conclude by sketching a natural generalization of our syntax to L-nets, a game model of concurrent computation that can itself be seen as a generalization of paraproof nets.

Contents

Introduction

Since its discovery in the 80s by J.-Y. Girard, linear logic has been rooted in the Curry-Howard isomorphism, which gives a precise correspondance between proofs (logic) and programs (computation). In his seminal paper [Gir87], Girard already noted that a certain notion of *parallel* computation can be found in the meaning of the connectives of linear logic, especially in the so-called *multiplicative* fragment.

More recently, L-nets have been introduced by C. Faggian and F. Maurel [FM05] as a game model of concurrent interaction, inspired by a close inspection of the computational behavior of two structures arising in the proof theory of linear logic: *proof nets* and *designs*.

Proof nets are a more economic presentation of the proofs of linear logic, abstracting away from "irrelevant" permutations of inferences that can be found in the standard sequent calculus presentation of proofs. One big advantage of proof nets is that their cut elimination procedure is *confluent*. From the computational perspective, it means that the "execution" of a proof net will always give the same result, regardless of the path of computation that has been choosen. However, the feature of proof nets that motivated the creation of L-nets is really the *parallelism* mentioned earlier, that results from forgetting the order of permutable inferences. A novelty of L-nets is that they allow to *partially* recover (or forget) this sequentiality, a behavior that will be mimicked by the box construct introduced in section 5.

Designs on the other hand are the main objects of *ludics*, a theory developed by J.-Y. Girard and intended as an *interactive* account of the foundations of logic. Designs can be seen as *abstract* proofs of linear logic, where formulae are replaced by their *occurrences*, retaining only the sub-formula relation. A fundamental aspect of ludics is the presence of the *daimon rule* $\maltese$, which has the static meaning of a generalized axiom that can prove any (set of occurrences of) formulae. Therefore it introduces many "incorrect" proofs (called *paraproofs*) if we still think in terms of formulae. The interactive nature of ludics then lies in the possibility of characterizing proofs of a formula A as those paraproofs that *react* the same way when confronted to paraproofs of $A^{\perp}$ (which exist thanks to $\maltese!$). This is the essence of the counter-proofs criterion introduced in section 3. The main purpose of L-nets is to bring the parallelism of proof nets to the abstract, interactive setting of ludics and designs.

The paper will proceed as follows: we first introduce in section 1 a focalized sequent calculus of multiplicative only linear logic, that captures the (correct) proofs which are

fully sequential[1]. Then we explain in section 2 how forgetting the order of permutable inferences leads to a graph-theoretical formulation of proof nets, that can precisely be characterized either as the desequentialization of sequent calculus proofs, or as those graph structures which can be sequentialized into sequent calculus proofs. We present in section 3 the *counter-proofs* criterion, one of many *correctness* criterions that give an alternative way of capturing the sequentializable proof structures. The appeal of this particular criterion is that it gives some insight into the interactive computation mechanism underlying L-nets, by generalizing proofs into paraproofs with the introduction of the daimon ✠.

Sections 4 and 5 introduce novel term syntaxes for (respectively) sequent calculus paraproofs and paraproof structures, together with computational apparatuses in the form of cut reductions. The first syntax is just a specialization of the *computational designs* of K. Terui [Ter11], making yet closer the connection with L-nets. The second one is strongly inspired by the algebraic presentation of differential interaction nets of D. Mazza [Maz16], which is quite close in spirit to process algebras such as the π-calculus. Section 6 defines a two-step desequentialization procedure relating the two syntaxes, where paraproofs are first translated into totally sequential paraproof nets sharing the same computational behavior. This is made possible by augmenting paraproof nets with a box construct similar to those used traditionally to encode exponentials. We then sketch a proof that the term syntaxes and desequentialization are correct, both from a static (with respect to the graphical syntaxes) and a dynamic (with respect to cut-elimination) point of view.

We conclude by explaining how our syntax could be extended to L-nets, notably by including the additive connectives of linear logic. We also give a list of future work that would be necessary for further ensuring the correctness of our syntax, as well as translating the results on L-nets in this new setting.

[1]It should be noted that the term *sequent* has (at least historically) no connection with the *sequentiality* of proofs we are interested in.

Sequential proofs

In this section, we present a fully sequential and focalized proof system for MLL, which is the multiplicative, unit-free fragment of linear logic.

1.1 A focalized sequent calculus for MLL

For the reader unacquainted with proof theory and sequent calculus, we start with a few standard definitions:

Definition 1. *We call* sequent *an expression* $\vdash \Gamma$*, where* Γ *is a finite sequence of formulae.*

Sequents were introduced by G. Gentzen (1934) in his sequent calculi for classical and intuitionistic logic, and are now the most standard way of formulating proof systems for many logics. To build them, we need a syntax for formulae, given here by the following grammar:

$$P ::= X \mid N \otimes N \mid \downarrow N$$
$$N ::= X^{\perp} \mid P \parr P \mid \uparrow P$$

Formulae built with the rules P and N are respectively qualified as *positive* and *negative*, hence we say that the syntax is *polarized*. We can see that they are built out of *propositional atoms* X and $X^{\perp}$[1], and put together with the binary connectives of *multiplicative conjunction* $\otimes$ (spelled "tensor") and *multiplicative disjunction* $\parr$ (spelled "parr"). Notice also how the grammar generates only formulae that are an alternation of positive and negative layers of sub-formulae: this is one way to enforce a *focalized* proof system, that is a system where proofs are made of an alternation of rules acting on positive and negative formulae. The unary connectives $\downarrow$ and $\uparrow$ are called *shifts*, and serve as a mean to change the polarity of a formula. Focalized systems have been proved to be equivalent to unfocalized ones in terms of provability (see e.g. [And92]), and will allow us to get closer to the syntax of designs and L-nets, which are also focalized.

[1] Where X denotes any atom in a given countable set.

Now that we have sequents, we want to prove them by composing *instances* of *inference rules*:

Definition 2. *An* inference rule *r is given by a set $\sigma_1, \dots, \sigma_n$ of sequents called* premisses*, and a sequent σ called* conclusion*. It has the following graphical presentation:*

$$\frac{\sigma_1 \quad \dots \quad \sigma_n}{\sigma}\ r$$

Sequents of a rule generally contain propositional metavariables *$P, P_1, P_2, \dots$ (resp. $N, N_1, N_2, \dots$) ranging over positive (resp. negative) formulae. An* instance *of the rule r is r where every occurrence of propositional metavariable has been replaced by an occurrence of concrete formula built out of atoms. We will often omit the "instance" part and confound rules with their instances.*

Before presenting the rules of our calculus, we need one last notion which is that of *negation* (or *dual*):

Definition 3. *The* negation *$P^\perp$ of a positive formula P is defined inductively as:*

$$\begin{aligned}(X)^\perp &= X^\perp \\ (N_1 \otimes N_2)^\perp &= N_1^\perp \parr N_2^\perp \\ (\downarrow N)^\perp &= \uparrow N^\perp\end{aligned}$$

with the negation $N^\perp$ of a negative formula N defined inductively as:

$$\begin{aligned}(X^\perp)^\perp &= X \\ (P_1 \parr P_2)^\perp &= P_1^\perp \otimes P_2^\perp \\ (\uparrow P)^\perp &= \downarrow P^\perp\end{aligned}$$

Negation in linear logic is most often presented as an involutive function on formulae that relates dual connectives through De Morgan equations, rather than as a separate connective. For our purpose, this will enforce the idea of interaction between (para)proofs of A and $A^\perp$ introduced in section 3.

The rules of sequent calculi are generally divided into three groups: identity, structural and logical. Since we want our system to be focalized, we restrict contexts $\Gamma, \Delta, \Sigma, \dots$ to positive formulae, so that the following *focalization property* will hold:

Proposition 1 (Focalization Property)**.** *If a sequent is provable, then it contains at most one negative formula. Sequents with no (resp. one) negative formula are said to be* positive *(resp.* negative*).*

The *identity* group comprises the two following rules:

$$\frac{}{\vdash P^\perp, P}\ \text{ax} \qquad\qquad \frac{\vdash P^\perp, \Gamma \quad \vdash P, \Delta}{\vdash \Gamma, \Delta}\ \text{cut}$$

With these two rules, we can already build (trivial) proofs, which are just trees of rules whose root is the rule which has the proved sequent as conclusion, and whose nodes (resp. leaves) are instances of the cut (resp. ax) rule. As is standard with sequents, the ax rule will be the only axiom (that is the only kind of proof tree leaf) of the calculus, with other rules acting as nodes in the proof tree. The identity group is really the common kernel of every sequent calculus, with possibly some variants on the axiom, where for example one can restrict formulae to the atomic case[2]. However, we *do* want here to allow any formula, since it induces the existence of more proofs, and therefore more computational expressivity[3].

One specificity of linear logic is to be found in the absence of two usual *structural* rules, called *weakening* and *contraction*. We will not go into too much details here since these two rules are only of interest when studying the exponential fragment of linear logic. Hence our structural group will only comprise the following *exchange* rule, which just serves as a mean to abstract away from the order of formulae inside sequents:

$$\frac{\vdash \Gamma, A, B, \Delta}{\vdash \Gamma, B, A, \Delta}\ X$$

where at most one formula among A and B is negative.

Then finally comes the *logical* group, whose purpose is to build proofs of compound formulae from proofs of their direct sub-formulae. There is therefore one rule for each connective:

$$\frac{\vdash N_1, \Gamma \quad \vdash N_2, \Delta}{\vdash N_1 \otimes N_2, \Gamma, \Delta}\ \otimes \qquad \frac{\vdash P_1, P_2, \Gamma}{\vdash P_1 \parr P_2, \Gamma}\ \parr$$

$$\frac{\vdash N, \Gamma}{\vdash \downarrow N, \Gamma}\ \downarrow \qquad \frac{\vdash P, \Gamma}{\vdash \uparrow P, \Gamma}\ \uparrow$$

We can see in the rules for shifts how they only act as a way to change the polarity of a formula, so that any formula in an unpolarized syntax can be turned into a formula in the polarized syntax by introducing shifts in the right places, and then proved with these rules.

Remark 1. *For example, it is possible thanks to the $\parr$ rule to turn every sequent $\vdash A_1, \dots, A_n$ into a one-formula sequent $\vdash A'_1 \parr \dots \parr A'_n$ where A'_i is A_i if A_i is positive, $\downarrow A_i$ otherwise, and this without losing provability.*

1.2 Cut elimination

Through the Curry-Howard isomorphism, proof systems can be seen as formalisms to write well-typed programs. Formulae correspond to types, while inference rules

[2]This corresponds to the process of η-expansion in λ-calculi.

[3]This will really matter when switching to the more expressive setting of paraproofs/L-nets.

$$\dfrac{\dfrac{}{\vdash P^\perp, P}\,\text{ax} \qquad \vdots\ \pi \\ \vdash P, \Gamma}{\vdash P, \Gamma}\,\text{cut} \quad \rightsquigarrow_\pi \quad \begin{array}{c}\vdots\ \pi \\ \vdash P, \Gamma\end{array} \tag{1.1}$$

$$\dfrac{\dfrac{\begin{array}{c}\vdots\ \pi_1 \\ \vdash P_1, P_2, \Gamma\end{array}}{\vdash P_1 \parr P_2, \Gamma}\,\parr \qquad \dfrac{\begin{array}{c}\vdots\ \pi_2 \\ \vdash P_1^\perp, \Delta\end{array} \qquad \begin{array}{c}\vdots\ \pi_3 \\ \vdash P_2^\perp, \Sigma\end{array}}{\vdash P_1^\perp \otimes P_2^\perp, \Delta, \Sigma}\,\otimes}{\vdash \Gamma, \Delta, \Sigma}\,\text{cut} \quad \rightsquigarrow_\pi \quad \dfrac{\dfrac{\begin{array}{c}\vdots\ \pi_1 \\ \vdash P_1, P_2, \Gamma\end{array} \qquad \begin{array}{c}\vdots\ \pi_2 \\ \vdash P_1^\perp, \Delta\end{array}}{\vdash P_2, \Gamma, \Delta}\,\text{cut} \qquad \begin{array}{c}\vdots\ \pi_3 \\ \vdash P_2^\perp, \Sigma\end{array}}{\vdash \Gamma, \Delta, \Sigma}\,\text{cut} \tag{1.2}$$

$$\dfrac{\dfrac{\begin{array}{c}\vdots\ \pi_1 \\ \vdash P, \Gamma\end{array}}{\vdash \uparrow P, \Gamma}\,\uparrow \qquad \dfrac{\begin{array}{c}\vdots\ \pi_2 \\ \vdash P^\perp, \Delta\end{array}}{\vdash \downarrow P^\perp, \Delta}\,\downarrow}{\vdash \Gamma, \Delta}\,\text{cut} \quad \rightsquigarrow_\pi \quad \dfrac{\begin{array}{c}\vdots\ \pi_1 \\ \vdash P, \Gamma\end{array} \qquad \begin{array}{c}\vdots\ \pi_2 \\ \vdash P^\perp, \Delta\end{array}}{\vdash \Gamma, \Delta}\,\text{cut} \tag{1.3}$$

Figure 1.1: Key cases of cut reduction $\rightsquigarrow_\pi$ on sequent calculus proofs

correspond to typing rules specifying when program constructs are well-typed. More crucially, one should be able in this view to *compute* with proofs, in the same way that programs (well-typed or not) can be evaluated. In sequent calculus, the way to evaluate proofs is through the *cut elimination* procedure. The idea is to eliminate all instances of the cut rule, by iterating a set of rewriting steps on proofs called *cut reduction*. We can distinguish two different kinds of cut reduction steps (in the following, A denotes the active cut formula, that is the formula that disappears together with its dual from both premisses of the cut rule):

- The **key cases** (figure 1.1) occur either when one premiss of the cut is an axiom, or when the two premisses last rules introduce A and $A^\perp$. In the first case, we simply erase the cut and the axiom to only keep the proof of the remaining premiss (step 1.1). In the second case, the cut as well as the premisses last rules are replaced by cuts on the direct sub-formulae of A (steps 1.2, 1.3).
- The **commutative cases** occur when no key case matches the cut rule, that is when either one of the premisses does not introduce A or $A^\perp$. The idea is then to descend the cut into sub-proofs by "swapping" it with the last rule of the premiss at fault, which will eventually lead to a key case.

With an adequate measure on proofs, it can be shown that every cut reduction step decreases the given measure, and therefore that cut elimination terminates. It means that every proof that contains cuts can be evaluated into a cut-free proof. However,

despite the simplicity of our calculus, cut elimination is not *confluent*. This can be observed with the following proofs[4]:

$$\tilde{\pi} = \left\{ \dfrac{\dfrac{\dfrac{}{\vdash A, A^\perp}\,\text{ax} \quad \dfrac{}{\vdash A, A^\perp}\,\text{ax}}{\vdash A \otimes A, A^\perp, A^\perp}\,\otimes \quad \dfrac{\dfrac{}{\vdash A, A^\perp}\,\text{ax} \quad \dfrac{}{\vdash A, A^\perp}\,\text{ax}}{\vdash A^\perp \otimes A^\perp, A, A}\,\otimes}{\vdash A \otimes A, A^\perp \otimes A^\perp, A^\perp, A}\,\text{cut} \right.$$

$$\pi = \left\{ \dfrac{\dfrac{\dfrac{}{\vdash A, A^\perp}\,\text{ax} \quad \dfrac{}{\vdash A, A^\perp}\,\text{ax}}{\vdash A^\perp \otimes A^\perp, A, A}\,\otimes \quad \dfrac{}{\vdash A, A^\perp}\,\text{ax}}{\vdash A \otimes A, A^\perp \otimes A^\perp, A^\perp, A}\,\otimes \right.$$

$$\pi' = \left\{ \dfrac{\dfrac{\dfrac{}{\vdash A, A^\perp}\,\text{ax} \quad \dfrac{}{\vdash A, A^\perp}\,\text{ax}}{\vdash A \otimes A, A^\perp, A^\perp}\,\otimes \quad \dfrac{}{\vdash A, A^\perp}\,\text{ax}}{\vdash A \otimes A, A^\perp \otimes A^\perp, A^\perp, A}\,\otimes \right.$$

Indeed, we have that $\tilde{\pi}$ can be evaluated both to π and π' depending on which cut reduction steps are choosen, and both π and π' cannot be reduced further since they are cut-free. Also, notice how π and π' only differ by permutation of the two $\otimes$ rules: this suggests that this kind of permutation might be the root of the non-confluence of cut elimination.

[4]To simplify the example, we forget here about polarities, shifts and focalization.

Parallel proofs

At the end of the previous section, we saw how the total sequentiality of proofs creates artificial orderings on some inferences, discriminating proofs that only differ up to permutation of these, even though they are equivalent from the computational perspective of cut elimination. To recover a kind of "canonicity" in the representation of proofs, we would therefore like to remove this artificial sequentiality. Since it cannot be done with fully sequential objects such as the proof *trees* of sequent calculus, a natural alternative that still has a nice graphical presentation would be *graphs*.

2.1 Proof structures

We now give the definition of *proof structures*, which are indeed a graph-theoretical "parallel syntax" for proofs:

Definition 4. *A* focalized proof structure *for* MLL *is a finite directed graph whose nodes are labelled either by logical connectives* $(\otimes, \parr, \downarrow, \uparrow)$, ax, cut, *or* c *(for* conclusion*), and whose edges are labelled by formulae. Furthermore, each label* l *has an associated arity* (p_l, c_l)*, meaning that every node labelled with* l *must have exactly* p_l *ingoing edges and* c_l *outgoing edges (the letters* p *and* c *stand respectively for* premisses *and* conclusions*). Arities are given by the following table:*

Label	ax	cut	$\otimes$	$\parr$	$\downarrow$	$\uparrow$	c
Arity	$(0,2)$	$(2,0)$	$(2,1)$	$(2,1)$	$(1,1)$	$(1,1)$	$(1,0)$

In particular, nodes labelled by connectives will have one conclusion, which must be labelled by the formula built out of their premisses with the given connective. Also, the two edges associated to ax *and* cut *nodes must be labelled by dual formulae. To enforce focalization, we add the two following conditions:*

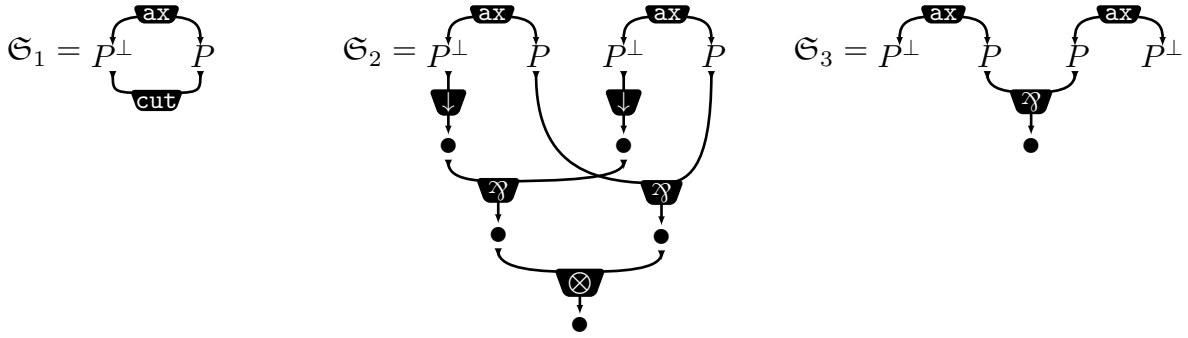
- *premisses of nodes labelled with a connective must be of the opposite polarity of that connective;*
- *the set of conclusion nodes must contain at most one node with a negative premiss.*

Since the textual definition is a bit heavy, we directly give the *desequentialization* function deseq_π (figure 2.1) that maps sequent calculus proofs to the associated proof structures. This should already give a good intuition of the kind of proof structures that can be constructed, and how they relate to sequential proofs.

2.2 Proof nets

Since the proof structures obtained by desequentialization come from sequent calculus proofs, there is no doubt that they are *correct* proofs. In fact, desequentialization is one way to define *proof nets*, which are precisely those proof structures that correspond to at least one sequent calculus proof. Our goal being to identify sequential proofs that differ only by non-significant permutations of inferences, there will generally be *many* such proofs associated to a single proof net.

Now, a natural question is : are all proof structures proof nets? The following examples show that it is *not* the case:



Indeed, they would correspond to the following "ill-formed" sequent calculus proofs:

$$\dfrac{\dfrac{}{\vdash P^\perp, P}\,\text{ax}}{\vdash}\,\text{cut} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash P^\perp, P}\,\text{ax}}{\vdash \downarrow P^\perp, P}\,\downarrow \quad \dfrac{\dfrac{}{\vdash P^\perp, P}\,\text{ax}}{\vdash \downarrow P^\perp, P}\,\downarrow}{\vdash \downarrow P^\perp \parr \downarrow P^\perp, P, P}\,\parr}{\vdash \downarrow P^\perp \parr \downarrow P^\perp, P \parr P}\,\parr}{\vdash (\downarrow P^\perp \parr \downarrow P^\perp) \otimes (P \parr P)}\,\otimes \qquad \dfrac{\dfrac{}{\vdash P^\perp, P}\,\text{ax} \quad \dfrac{}{\vdash P^\perp, P}\,\text{ax}}{\vdash P \parr P, P^\perp, P^\perp}\,\parr$$

$\mathfrak{S}_1$ is known as the "vicious circle", and exploits the fact that a proof structure can have no conclusion. This corresponds to proving the empty sequent, which is impossible since the only way to do this would be with a (well-formed) cut, which contradicts the fact that every sequent has a cut-free proof thanks to cut elimination. We also see through examples $\mathfrak{S}_2$ and $\mathfrak{S}_3$ how proof structures do not make any distinction between $\otimes$ and $\parr$ (apart from their polarity).

2.3 Correctness

Currently, the only way we have to check that a proof structure $\mathfrak{S}$ is indeed a proof net is by having an associated sequent calculus proof π, and by verifying that $\mathsf{deseq}_\pi(\pi) = \mathfrak{S}$. It would be much more satisfactory to have a way of checking the correctness of a proof structure that is completely independant from the sequential syntax of sequent calculus. This is what *correctness criterions* are for: they allow to characterize sequentializable proof structures on the basis of their geometrical/topological properties. In this section, we give an explanation of the first criterion that was devised for proof nets: the **DR criterion** (named after its inventors, Danos and Régnier). First we need to define the notion of *switching*:

Definition 5. *A* switching *s on a proof structure $\mathfrak{S}$ is a function from the set of $\parr$-nodes of $\mathfrak{S}$ to $\{0,1\}$. The* switching graph *induced by s is $\mathfrak{S}$ where every $\parr$-node n has one of its premisses erased: the left one if $s(n) = 0$, the right one otherwise.*

The DR criterion is then stated as follows:

Definition 6. *A paraproof structure $\mathfrak{S}$ satisfies the DR criterion if all of its switching graphs are trees, that is they are connected and acyclic.*

The criterion might seem quite surprising at first, especially since it has seemingly no connection with the desequentialization procedure. Its meaning will become more apparent when related to the counterproofs criterion, to be introduced in the next section. Let's try to apply it to the three examples above:

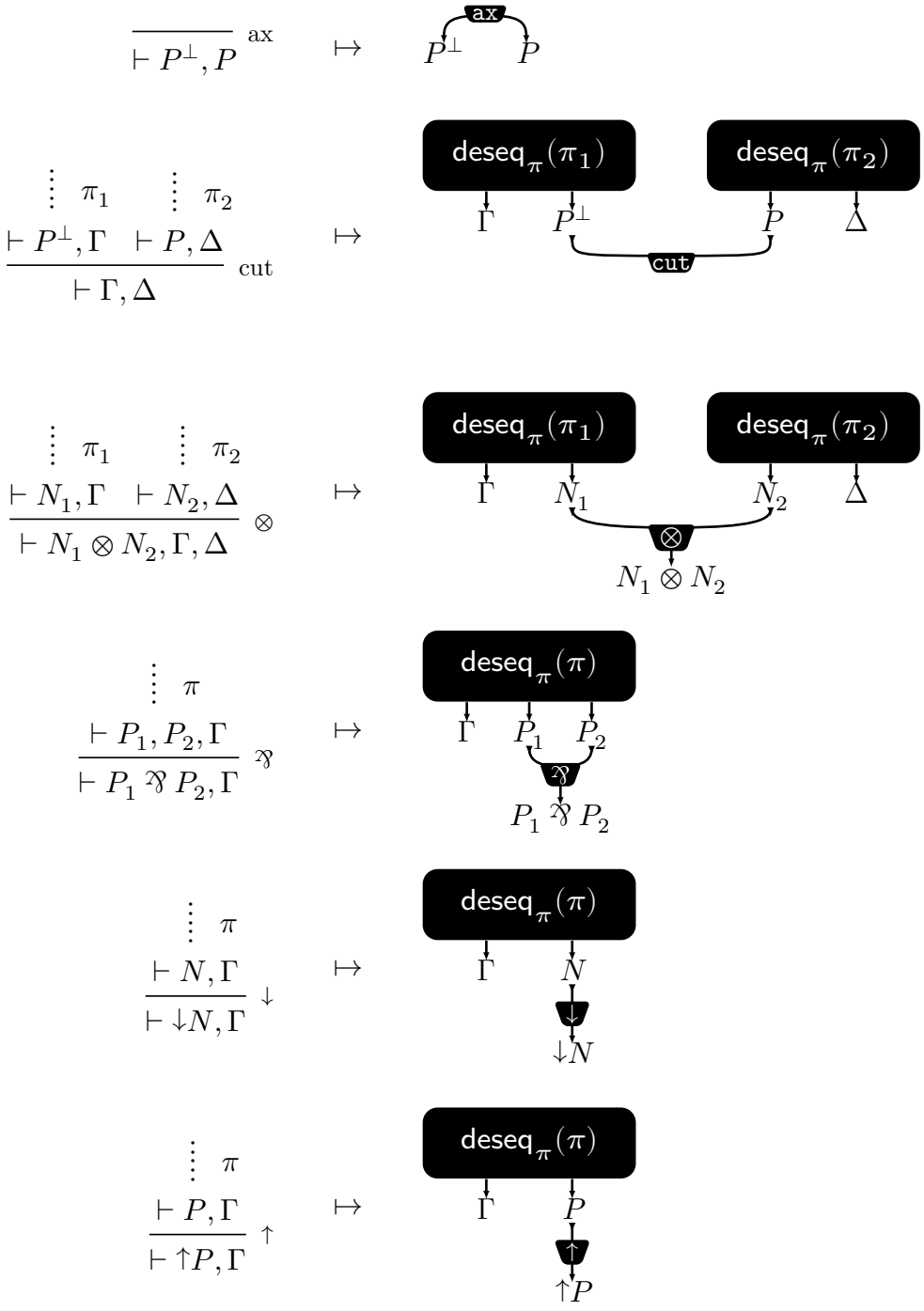
- $\mathfrak{S}_1$ is trivially incorrect, since it is the simplest cycle one can build, and it does not contain any $\parr$-node.
- $\mathfrak{S}_2$ is a more subtle example of violation of the acyclicity condition: indeed, all of the 4 switching graphs are connected, and only 2 of them actually contains a cycle (those in which both left/right premisses have been erased).
- $\mathfrak{S}_3$ shows how the connectedness condition is also important, and independant from the acyclicity condition. It can be shown that removing it amounts to accepting the *mix* rule in sequent calculus:

$$\frac{\vdash \Gamma \quad \vdash \Delta}{\vdash \Gamma, \Delta}\ \text{mix}$$

To ensure that the DR criterion is actually a correctness criterion, it is necessary to prove the following *sequentialization theorem*:

Theorem 1 (Sequentialization)**.** *A proof structure satisfies the DR criterion if and only if it is the image of a sequent calculus proof by the desequentialization function.*

Since the proof is quite technical, we do not give it here. It could have been however of interest if we had tried to export the criterion to the new syntax that we devise for (para)proof structures in section 5.



Inside proof structures, Γ, Δ denote sets of conclusion nodes with premisses labelled by the formulae in Γ, Δ. Also, we have not actually drawn conclusion nodes to improve readability.

Figure 2.1: Desequentialization deseq_π from MLL proofs to MLL proof structures

Paraproofs

3.1 The daimon ✠

In this section, we present another correctness criterion called the *counterproofs* criterion. The idea is to characterize proof nets of conclusion A as those proof structures who behave well when confronted to their counterproofs, that is proof structures of conclusion $A^\perp$. By saying that, we already notice a problem: if $\vdash A$ is provable, how can we have at the same time proofs of $\vdash A^\perp$? This would refute the principle of non-contradiction, and make our logic inconsistent. In fact, this is exactly what we are going to do by introducing a new rule/node, the *daimon* ✠:

$$\frac{}{\vdash P_1, \dots, P_n}\ ✠$$

The daimon acts statically as a generalized axiom, allowing to prove any sequent. Since we can now prove both A and $A^\perp$, we should not consider ourselves in a logical system in the traditional sense anymore. However, the new objects, that we call *paraproofs*, still have the fine structure of proofs, and will be of even more interest from a computational point of view.

3.2 Cut elimination

To formulate the counterproofs criterion, we need to define cut elimination for paraproof structures. Indeed, the way proofs will interact with their counterproofs is by *cutting* their dual conclusion, and evaluating the resulting paraproof structure. Cut reduction steps are defined in the same way than those for sequent calculus, except for the new ✠-node that we give in figure 3.1. While it acts statically as the axiom, the daimon has a very different computational behavior, in that it *absorbs* proofs that interact with it instead of letting them take its place. By the way, it can be shown that cut elimination terminates and is confluent for (para)proof structures, either with or without the daimon. This confirms the intuition about permutation of inferences being the cause of non-confluence in sequent calculus.

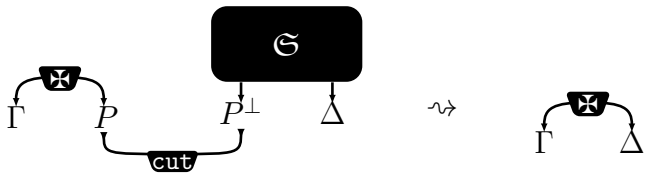


Figure 3.1: Cut reduction step for the ✠-node of paraproof structures

The idea of well-behaved interaction between two paraproofs can be captured by the notion of *orthogonality*:

Definition 7. *Two paraproof structures $\mathfrak{S}$ of conclusion A and $\mathfrak{S}'$ of conclusion $A^\perp$ are said to be* orthogonal *(written $\mathfrak{S} \perp \mathfrak{S}'$) when the paraproof structure obtained by adding a cut between A and $A^\perp$ evaluates to a ✠-node.*

We can now give a precise formulation of the counterproofs criterion in terms of orthogonality:

Definition 8. *A proof structure $\mathfrak{S}$ of conclusion A satisfies the* counterproofs criterion *if for every paraproof structure $\mathfrak{S}'$ of conclusion $A^\perp$, we have $\mathfrak{S} \perp \mathfrak{S}'$.*

Notice how the counterproofs criterion relies crucially on the computational device of cut elimination, but this time as an *interaction* between correct paraproofs and their incorrect counterproofs. This is really the essence of how the original designs of ludics, and therefore the more parallel L-nets, compute. Also, the fact that we have restricted the criterion to one-conclusion paraproof structures is not a problem, since every set of conclusions can be turned into a a single conclusion by introducing $\parr$ and shift nodes adequately (recall remark 1).

To conclude this section, we shall highlight an interesting similarity between the DR and counterproofs criterions, which is not immediately apparent. The trick is to remark that the switching graphs of a paraproof structure are isomorphic to its *extreme counterproofs*:

Definition 9. *The* extreme paraproofs *are the (sequent) paraproofs obtained by restricting $\otimes$-rules to the cases where one of the premisses has an empty context:*

$$\frac{\vdash N_1, \Gamma \quad \vdash N_2}{\vdash N_1 \otimes N_2, \Gamma}\ \otimes \qquad\qquad \frac{\vdash N_1 \quad \vdash N_2, \Gamma}{\vdash N_1 \otimes N_2, \Gamma}\ \otimes$$

The extreme counterproofs *of a paraproof structure of conclusion A are the paraproof structures obtained by desequentialization of the extreme paraproofs of conclusion $\vdash A^\perp$.*

The DR criterion can therefore be seen as a simplified formulation of the counterproofs criterion, which is topologic rather than computational.

Paraproof terms

In the first three sections, we have introduced the main concepts of the proof theory of multiplicative linear logic. We emphasized the relationship between sequential and parallel representations of proofs, as well as the concept of paraproofs which turns proofs into computational rather than logical objects. Our goal in the next three sections will be to adapt these concepts to a setting more familiar to most computer scientists, which is that of *term syntaxes*.

4.1 Multiplicative c-designs

In this section, we introduce sequential objects called *multiplicative c-designs*, which are really Terui's *c-designs* [Ter11] specialized to the connectives of multiplicative linear logic with shifts[1]. C-designs were introduced precisely as a "handy term syntax" for the sequential designs of ludics, with the notable difference that they incorporate explicit identities (axioms) and cuts, while the original designs are identity and cut-free. This allows for a lot more computational power: in fact, Terui shows that the original designs capture exactly the regular languages, while c-designs can express arbitrary sets of finite data. Multiplicative c-designs are defined *co-inductively* by the following grammar:

$$P ::= \Phi(\vec{x}) \mid \Omega \mid N_0 \parallel \otimes(N_1, N_2) \mid N_0 \parallel \downarrow(N_1)$$
$$N ::= x \mid \parr(x_1, x_2).P \mid \uparrow(x).P$$

Figure 4.1: Definition of multiplicative c-designs $\mathcal{D}$

Since the definition is co-inductive, it is important to notice that c-designs can be *infinitary*, as is the case with the designs of ludics. It entails that some (in fact many!) c-designs do not actually correspond to a paraproof. However, every paraproof can be expressed as a c-design, as shown by the translation D defined in figure 4.2. Proofs ending with a positive (resp. negative) rule are translated into positive (resp. negative) c-designs, where the notion of polarity for c-designs is (as for formulae) based on their grammar. Let's detail how the translation works:

[1] We will often drop the "multiplicative" qualifier since we do not deal with full c-designs.

- The base cases are pretty straight-forward: the ✠ rule has a corresponding ✠ term, and axioms are translated like in λ-calculus as variables. We have added to the original definition of c-designs the variables $\vec{x}$ captured by the ✠ term, which correspond to the occurences of positive formulae introduced by the ✠ rule[2]. Notice how every occurrence of positive formulae is forgotten and replaced by a variable: c-designs are indeed untyped objects, and the fact that logical connectives appear in their syntax is just to make the correspondance with proofs more visible.

- Negative proofs π are translated as terms $a(\vec{x}).P$, where P is the translation of π's direct subproof, a is the connective of π's last rule ($\parr$ or $\uparrow$), and the variables $\vec{x}$ correspond to the direct subformulae of the formula introduced by the rule. $a(\vec{x})$ is called a *negative action*, and acts as a *binder* for $\vec{x}$ in P, in the same way that $\lambda x.M$ would bind x in M in λ-calculus.

- The tricky part of the translation is the case of the cut rule. The intuition is that a cut between a negative proof π_0 and a positive proof π will be translated as a positive term $N_0 \parallel \overline{a}(\vec{N})$, where N_0 is the translation of π_0, $\overline{a}$ is the *positive action* associated to the connective of π's last rule ($\otimes$ or $\downarrow$), and $\vec{N}$ are the translations of π's direct subproofs. The translation actually does this in two steps: first π is translated as $x \parallel \overline{a}(\vec{N})$ (with x fresh), and then x is replaced by N_0 when translating the cut. Replacement is done with the *substitution function* defined in figure 4.4, which does what we want here because x occurs exactly once since it is fresh.

4.2 Cut elimination

We now need a cut-elimination procedure to compute with our multiplicative c-designs. The procedure should simulate that of MLL paraproofs, since multiplicative c-designs are intended as a term syntax for them. One advantage of having a term syntax is that cut reduction is much more succint to express, in the same way that λ-terms only need the β-rule to compute. We indeed only have the three following rules:

[2]The only purpose of these variables is to carry information about the conclusions of ✠ when translating c-designs to the term syntax that we devise for paraproof nets in section 5.

$$\frac{}{(\parr(x_1, x_2).P) \parallel \otimes(N_1, N_2) \rightsquigarrow_{\mathcal{D}} P[N_1 \,/\, x_1]\,[N_2 \,/\, x_2]}$$

$$\frac{}{(\uparrow(x).P) \parallel \downarrow(N) \rightsquigarrow_{\mathcal{D}} P[N \,/\, x]}$$

$$\frac{\overline{a} \neq b}{(a(\vec{x}).P) \parallel b(\vec{N}) \rightsquigarrow_{\mathcal{D}} \Omega}$$

Figure 4.3: Cut reduction $\rightsquigarrow_{\mathcal{D}}$ of multiplicative c-designs

The two first rules correspond to the two key cases for $\parr/\otimes$ and $\uparrow/\downarrow$ of cut reduction on paraproofs, and do have a striking similarity with the β-rule. The key case for the axiom (resp. daimon) is simulated by the base case of substitution on variables (resp. $\maltese$) (see figure 4.4). There is a little subtlety in the $\maltese$ case, since it should be able to absorb the conclusions of the paraproof with which it is cut: this is handled by capturing the free variables of the corresponding c-design, since the notion of free variables of a c-design matches exactly that of conclusions of a paraproof. Commutative cases are handled by the recursive cases of substitution, and this is where the term syntax brings most simplicity.

As for the third reduction rule, it has no counterpart in paraproofs since it involves a new c-design called Ω. It comes from ludics, and as noticed by Curien [Cur05], has the meaning of non-termination of evaluation. Whereas this rule is not necessary for the original c-designs because of the additive superimposition $\sum_a a(\vec{x_a}).P_a$, we do need it here to handle "ill-formed" cuts of non-dual actions.

$$\frac{}{\vdash x_1 : P_1, \dots, x_n : P_n}\ \maltese \qquad \mapsto \qquad \maltese(x_1, \dots, x_n)$$

$$\frac{}{\vdash P^\perp, x : P}\ \mathrm{ax} \qquad \mapsto \qquad x$$

$$\frac{\begin{array}{cc}\vdots\ \pi_1 & \vdots\ \pi_2 \\ \vdash P^\perp, \Gamma & \vdash x : P, \Delta\end{array}}{\vdash \Gamma, \Delta}\ \mathrm{cut} \qquad \mapsto \qquad \mathsf{D}(\pi_2)\,[\mathsf{D}(\pi_1)\ /\ x]$$

$$\frac{\begin{array}{c}\vdots\ \pi \\ \vdash x : P, \Gamma\end{array}}{\vdash \uparrow P, \Gamma}\ \uparrow \qquad \mapsto \qquad \uparrow(x).\mathsf{D}(\pi)$$

$$\frac{\begin{array}{c}\vdots\ \pi \\ \vdash N, \Gamma\end{array}}{\vdash x : \downarrow N, \Gamma}\ \downarrow \qquad \mapsto \qquad x \parallel \downarrow(\mathsf{D}(\pi))$$

$$\frac{\begin{array}{c}\vdots\ \pi \\ \vdash x_1 : P_1, x_2 : P_2, \Gamma\end{array}}{\vdash P_1 \parr P_2, \Gamma}\ \parr \qquad \mapsto \qquad \parr(x_1, x_2).\mathsf{D}(\pi)$$

$$\frac{\begin{array}{cc}\vdots\ \pi_1 & \vdots\ \pi_2 \\ \vdash N_1, \Gamma & \vdash N_2, \Delta\end{array}}{\vdash x : N_1 \otimes N_2, \Gamma, \Delta}\ \otimes \qquad \mapsto \qquad x \parallel \otimes(\mathsf{D}(\pi_1), \mathsf{D}(\pi_2))$$

To ease the translation, we work with MLL paraproofs augmented with unique variables associated to each occurrence of positive formula. These can be added with a simple procedure that starts by associating fresh variables to the positive formulae in the conclusion, and keeps track of those associations when descending into subproofs, associating fresh variables to new occurrences encountered along the way.

Figure 4.2: Translation D from MLL paraproofs to multiplicative c-designs $\mathcal{D}$

$$
\begin{aligned}
&\mathsf{fv}(\Phi(\vec{x})) = \vec{x} \\
&\mathsf{fv}(N_0 \parallel \overline{a}(N_1, \ldots, N_n)) = \bigcup_{i=0}^{n} \mathsf{fv}(N_i) \\
&\mathsf{fv}(x) = x \\
&\mathsf{fv}(a(\vec{x}).P) = \mathsf{fv}(P) \setminus \vec{x}
\end{aligned}
$$

$$
\begin{aligned}
\Phi(\vec{x})\,[N \,/\, y] &= \begin{cases} \Phi(\vec{x} \setminus y, \mathsf{fv}(N)) & \text{if } y \in \vec{x} \\ \Phi(\vec{x}) & \text{otherwise} \end{cases} \\
N_0 \parallel \overline{a}(N_1, \ldots, N_n)\,[N \,/\, y] &= N_0\,[N \,/\, y] \parallel \overline{a}(N_1\,[N \,/\, y], \ldots, N_n\,[N \,/\, y]) \\
x\,[N \,/\, y] &= \begin{cases} N & \text{if } x = y \\ x & \text{otherwise} \end{cases} \\
(a(\vec{x}).P)\,[N \,/\, y] &= \begin{cases} a(\vec{x}).P[N \,/\, y] & \text{if } y \notin \vec{x} \text{ and } \vec{x} \cap \mathsf{fv}(N) = \emptyset \\ a(\vec{x}).P & \text{otherwise} \end{cases}
\end{aligned}
$$

Figure 4.4: Free variables and substitution of multiplicative c-designs

Paraproof net terms

5.1 Paranets

In this section, we introduce partially sequential objects called *paranets*, which provide a term syntax in the style of the π-calculus for paraproof structures. We take heavy inspiration from the differential interaction nets of D. Mazza [Maz16], to which we add the daimon in order to simulate paraproof structures, as well as a box construct which allows us to add partial sequentiality, so that we get as close as possible to the behavior of L-nets. We give here our variant of the definition of nets by Mazza, which only changes in the cells available to build paranets, and in the finiteness condition:

Definition 10 (Paranet)**.** *A* cell *is an expression of one of the following forms:*

daimon: $\text{Ⱶ}(\vec{x})$, $\mathrm{gc}(\vec{x};\vec{y})$
box: $\mathrm{box}(\vec{x};\vec{x}')$
multiplicative cells: $\otimes(x;y,z)$, $⅋(x;y,z)$
shift cells: $\downarrow(x;y)$, $\uparrow(x;y)$

where x, y, z range over a denumerably infinite set of ports*, and $\vec{x}, \vec{y}, \vec{z}$ are vectors of ports. Ports on the left of ; are the cell's* conclusions*, and those on the right are its* premisses*.*

A wire *is a multiset containing exactly 2 (not necessarily distinct) ports x, y, which we denote by $x \leftrightarrow y$ (or $y \leftrightarrow x$).*

A net *is a (possibly infinite) multiset of cells and wires in which every port appears at most twice. The set of* free ports *of a net μ, denoted by $\mathsf{fp}(\mu)$, is the set of ports appearing exactly once in μ. The ports appearing twice in a net are called* bound*. We identify any two nets which may be obtained one from the other by an injective renaming of their bound ports (this is α-*equivalence*).*

Given two nets μ, ν, we denote by $\mu \mid \nu$ the net obtained by renaming (using α-equivalence) the bound ports of μ and ν so that the two nets have no bound name in common, and by taking then the standard multiset union. The operation $\mid$, called parallel composition*, is obviously commutative and has the empty net, denoted by $*$, as neutral element. It is not associative in general; however, for $\mu \mid (\nu \mid \rho)$ and $(\mu \mid \nu) \mid \rho$ to be equal, it is enough to suppose that $\mathsf{fp}(\mu) \cap \mathsf{fp}(\nu) \cap \mathsf{fp}(\rho) = \emptyset$. More generally, if $\mu_1, \ldots, \mu_n$ are such that, for any pairwise distinct i, j, k, $\mathsf{fp}(\mu_i) \cap \mathsf{fp}(\mu_j) \cap \mathsf{fp}(\mu_k) = \emptyset$, then the expression $\mu_1 \mid \ldots \mid \mu_n$ is not ambiguous. In the sequel, we shall always assume this to be the case.*

As for multiplicative c-designs, we give a translation N (figure 5.1) that maps paraproof nets to paranets, which can give the reader a better intuition of what kind of paranets can be constructed. Of course in the same way that some c-designs do not correspond to any paraproof, some paranets do not correspond to any paraproof structure, one of the reasons being that they can also be infinite[1].

The idea of the translation is quite simple: every node is translated as the corresponding cell, except for axioms and cuts which are represented by bound ports: a port that appears in two premisses (resp. conclusions) is an axiom (resp. a cut). Since a proof net might consist only of axioms, they are represented as wires, which can be erased later in cut elimination. Links between nodes are represented by bound ports that appear in one premiss of a cell and one conclusion of another cell.

5.2 Cut elimination

Once again we need to define a cut elimination, that will this time simulate paraproof structures. The formulation is a bit more complex than with c-designs, and relies crucially on an extended notion of *structural reduction/congruence* $\rightarrow_\omega$ in the style of process calculi, defined in figure 5.2.

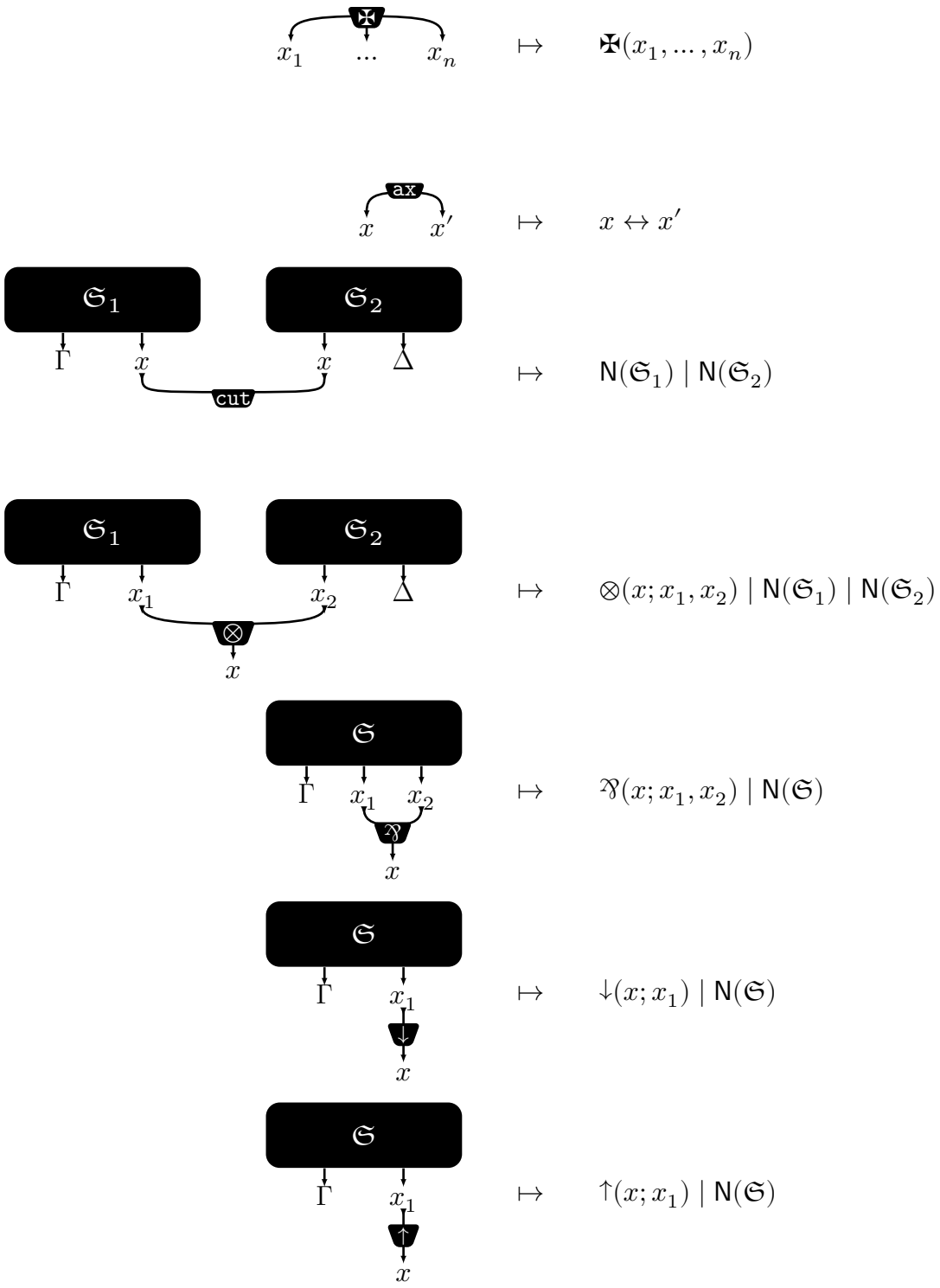
The equivalent of the key cases for connectives find their formulations in two rules given in the definition of cut reduction on paranets $\rightsquigarrow_{\mathcal{B}}$ (figure 5.3). The idea is to replace cut cells by wires between their premisses, wires being handled by the first rule of structural reduction $\rightarrow_\omega$. This rule has for sole purpose to execute (linear) *substitutions*, and thus plays the same role as the recursive cases and the base case on variables of substitution on c-designs. Since axioms are directly represented by wires, they are also handled there if needed.

A trickier part of cut elimination is the handling of the daimon. There is also one rule for it in $\rightsquigarrow_{\mathcal{B}}$, but it does not quite work like the one for paraproof structures. Indeed, it was possible in the latter to take a global look on the whole paraproof structure and to determine which subparaproof structure was interacting with the daimon, making it easy to erase it. In a paranet it is way harder to look at the structure globally, in fact we need to follow bound ports locally to find our way in the structure. It is therefore preferable to erase the interacting paranet step-by-step: this is what the two other rules of structural reduction do with the help of the gc node (as in *garbage collection*). Although this technique is already used in the paper introducing L-nets [FM05], we found it independently for paranets.

One last reduction rule in $\rightsquigarrow_{\mathcal{B}}$ is the one for *boxes*. Boxes can be seen as synchronization points, or just as a way to encode sequents into paranets. A box of the form $\mathrm{box}(\vec{x};\vec{x}')$ will allow one to gather all conclusions of a paranet in its premisses $\vec{x}'$, and to output each $\vec{x}'_i$ as a conclusion $\vec{x}_i$. What the rule does is simply connecting $\vec{x}_i$ and $\vec{x}'_i$ with a wire outside the box whenever $\vec{x}_i$ is cut with another cell.

All the above base cases of cut reduction are then closed by parallel composition $|$ and structural congruence $\equiv$.

[1]The other reason is that as with multiplicative c-designs, it is possible to place a cut between two non-dual cells.



To ease the translation, every formula occurrence in the proof net has been replaced by a unique port, except for cut premisses which share the same port. Γ, Δ denote sets of conclusion nodes with free ports as premisses.

Figure 5.1: Translation N from MLL paraproof nets to paranets

$$\frac{}{x \leftrightarrow y \mid \mu \to_\omega \mu\,[y \,/\, x]}$$

$$\frac{}{\mathrm{gc}(\vec{x};\emptyset) \mid \mu \to_\omega \text{✠}(\vec{x}) \mid \mu} \qquad \frac{\vec{y} \cap \vec{y}' \neq \emptyset}{\mathrm{gc}(\vec{x};\vec{y}) \mid C(\vec{y}';\vec{z}) \mid \mu \to_\omega \mathrm{gc}(\vec{x}, \vec{y}' \setminus \vec{y}; \vec{y} \setminus \vec{y}', \vec{z}) \mid \mu}$$

C denotes any cell constructor, and $\mu\,[y \,/\, x]$ is μ where the only occurrence of x has been replaced by y.

Figure 5.2: Structural reduction $\to_\omega$ of paranets

$$\frac{\vec{x} \cap \vec{y} = x}{\text{✠}(\vec{x}) \mid C(\vec{y};\vec{z}) \rightsquigarrow_{\mathcal{B}} \mathrm{gc}(\vec{x} \setminus x, \vec{y} \setminus x; \vec{z})}$$

$$\frac{\vec{x} \cap \vec{y} = \vec{x}_i}{\mathrm{box}(\vec{x};\vec{x}') \mid C(\vec{y};\vec{z}) \rightsquigarrow_{\mathcal{B}} \vec{x}'_i \leftrightarrow \vec{x}_i \mid \mathrm{box}(\vec{x} \setminus \vec{x}_i; \vec{x}' \setminus \vec{x}'_i) \mid C(\vec{y};\vec{z})}$$

$$\frac{}{\otimes(x;y_1,y_2) \mid \parr(x;z_1,z_2) \rightsquigarrow_{\mathcal{B}} y_1 \leftrightarrow z_1 \mid y_2 \leftrightarrow z_2}$$

$$\frac{}{\downarrow(x;y) \mid \uparrow(x;z) \rightsquigarrow_{\mathcal{B}} y \leftrightarrow z}$$

$$\frac{\mu \rightsquigarrow_{\mathcal{B}} \mu'}{\mu \mid \nu \rightsquigarrow_{\mathcal{B}} \mu' \mid \nu} \qquad \frac{\mu \equiv \mu' \quad \mu' \rightsquigarrow_{\mathcal{B}} \nu' \quad \nu' \equiv \nu}{\mu \rightsquigarrow_{\mathcal{B}} \nu}$$

C denotes any cell constructor, and $\equiv$ is the reflexive, symmetric and transitive closure of $\to_\omega$.

Figure 5.3: Cut reduction $\rightsquigarrow_{\mathcal{B}}$ of paranets

Desequentialization of terms

In this last section, we define a two-step *desequentialization* function, that first translates multiplicative c-designs into boxed paranets with the same computational behavior, and then does the desequentialization by simply removing boxes.

The translation $\mathcal{B}$ from c-designs to boxed paranets is given in figure 6.2. The idea is simply to put a box at the end of the translation of each subterm, the latter being done by associating each action to its corresponding cell. The definition looks a bit complicated because of the bureaucratic management of ports, which must satisfy freshness, order and arity conditions.

As for the actual desequentialization through removal of boxes, we give two variants in figure 6.1: the first one, deseq^n_μ, simply removes entire boxes in one go; while the second one, deseq^1_μ, is more fine-grained in that it proceeds wire by wire. We could consider these respectively as "big-step" and "small-step" variants on desequentialization, and their properties with respect to other rewriting relations such as cut reduction might constitute an interesting open research area.

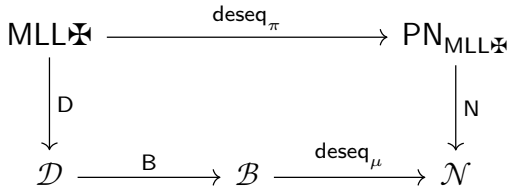
$$\mathsf{deseq}^n_\mu(\mathrm{box}(x_1, \dots, x_n; x'_1, \dots, x'_n) \mid \mu) = \mu\,[x_1 \,/\, x'_1] \dots [x_n \,/\, x'_n]$$

$$\mathsf{deseq}^1_\mu(\mathrm{box}(x_1, \dots, x_n; x'_1, \dots, x'_n) \mid \mu) = \begin{cases} \mu\,[x_1 \,/\, x'_1] & \text{if } n = 1 \\ \mathrm{box}(x_2, \dots, x_n; x'_2, \dots, x'_n) \mid \mu\,[x_1 \,/\, x'_1] & \text{if } n > 1 \end{cases}$$

Figure 6.1: Desequentialization of boxed paranets

6.1 Static correction

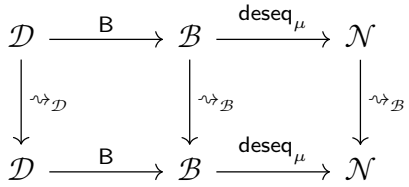
To ensure the *static correctness* of our desequentialization procedure, we would like to prove that the following diagram commutes:



In the diagram, MLL✠, PN$_{\text{MLL}✠}$ and $\mathcal{N}$ denote respectively the sets of sequent calculus paraproofs, paraproof nets, and paranets without boxes. Commutativity of the diagram would ensure that our term syntaxes and their desequentialization simulate correctly the graphical syntaxes for paraproofs, as well as their desequentialization.

6.2 Dynamic correction

To ensure the *dynamic correctness* of our desequentialization procedure, we would like to prove that the following diagram commutes:



Commutativity of the diagram would ensure that desequentialization does not interfere with cut elimination, and vice-versa.

$\Phi(\vec{x})$	$\mapsto$	$\mathrm{box}(\vec{x};\vec{x}') \mid \Phi(\vec{x}')$	with $\vec{x}' = \mathsf{fresh}(\lvert\vec{x}\rvert)$
$N_0 \parallel \downarrow(N_1)$	$\mapsto$	$\mathrm{box}(\vec{x_0},\vec{x_1};$ $\mathsf{fp}(\mu_0) \setminus x_0,$ $\mathsf{fp}(\mu_1) \setminus x_1) \mid$ $\mu_0 \mid \downarrow(x_0;x_1) \mid \mu_1$	with $\mu_0 = B(N_0, x_0),$ $\mu_1 = B(N_1, x_1),$ $\vec{x_0} = \mathsf{fresh}(\lvert\mathsf{fp}(\mu_0)\rvert - 1),$ $\vec{x_1} = \mathsf{fresh}(\lvert\mathsf{fp}(\mu_1)\rvert - 1),$ x_0, x_1 fresh
$N_0 \parallel \otimes(N_1, N_2)$	$\mapsto$	$\mathrm{box}(\vec{x_0},\vec{x_1},\vec{x_2};$ $\mathsf{fp}(\mu_0) \setminus x_0,$ $\mathsf{fp}(\mu_1) \setminus x_1,$ $\mathsf{fp}(\mu_2) \setminus x_2) \mid$ $\mu_0 \mid \otimes(x_0;x_1,x_2) \mid \mu_1 \mid \mu_2$	with $\mu_0 = B(N_0, x_0),$ $\mu_1 = B(N_1, x_1),$ $\mu_2 = B(N_2, x_2),$ $\vec{x_0} = \mathsf{fresh}(\lvert\mathsf{fp}(\mu_0)\rvert - 1),$ $\vec{x_1} = \mathsf{fresh}(\lvert\mathsf{fp}(\mu_1)\rvert - 1),$ $\vec{x_2} = \mathsf{fresh}(\lvert\mathsf{fp}(\mu_2)\rvert - 1),$ x_0, x_1, x_2 fresh
$x \ , \ z$	$\mapsto$	$\mathrm{box}(x, z; z', z')$	with z' fresh
$\uparrow(x).P \ , \ z$	$\mapsto$	$\mathrm{box}(z, \vec{x}; z', \mathsf{fp}(\mu) \setminus x) \mid$ $\uparrow(z';x) \mid \mu$	with $\mu = B(P),$ $\vec{x} = \mathsf{fresh}(\lvert\mathsf{fp}(\mu)\rvert - 1),$ z' fresh
$\Im(x_1,x_2).P \ , \ z$	$\mapsto$	$\mathrm{box}(z, \vec{x}; z', \mathsf{fp}(\mu) \setminus \{x_1, x_2\}) \mid$ $\Im(z';x_1,x_2) \mid \mu$	with $\mu = B(P),$ $\vec{x} = \mathsf{fresh}(\lvert\mathsf{fp}(\mu)\rvert - 2),$ z' fresh

$\mathsf{fresh}(n)$ denotes a set of n fresh ports, with ports being considered fresh when they occur neither as variables in the term being translated, nor in the translations of its subterms.

Figure 6.2: Translation $\mathsf{B}(P)/\mathsf{B}(N,z)$ from multiplicative c-designs $\mathcal{D}$ to boxed paranets $\mathcal{B}$

Conclusion

In this article, we have introduced and compared many different proof systems of multiplicative linear logic, each having a different take on sequentiality and interactivity. Thanks to the Curry-Howard isomorphism, it is indeed possible to study these systems for their computational, rather than logical properties. The ultimate goal was to provide a term syntax for L-nets, a game model of concurrent computation which showcases at the same time interactive and non-sequential features.

This goal was in fact quite ambitious, and even though we restricted ourselves to the multiplicative fragment of linear logic, we could not afford to make rigorous proofs ensuring the correctness of the definitions we introduced for our two syntaxes. So a first line of future research would be in actually checking the correctness of these definitions.

Once we have made sure that our syntaxes formalize correctly the models of computation that we want to study, a second line of research would be in importing definitions and theorems from the "graphical" setting to our syntaxes. Typically it would be interesting to see if the correctness criterions of paraproof structures can be expressed in the setting of paranets.

Finally, the natural continuation of our work would be to extend the syntaxes to the additive fragment of linear logic, which is present in L-nets. This extension should not be too difficult, in that paranets are already partially sequential and interactive in nature: this would only be a matter of abstracting away from logical rules, and embracing the fully abstract nature of L-nets.

Bibliography

[And92] Jean-Marc Andreoli, *Logic programming with focusing proofs in linear logic.*

[Cur05] Pierre-Louis Curien, *Introduction to linear logic and ludics, part II*, no. 5, 513–544.

[FM05] Claudia Faggian and François Maurel, *Ludics nets, a game model of concurrent interaction*, 07 2005, pp. 376–385.

[Gir87] Jean-Yves Girard, *Linear logic*, Theoretical Computer Science **50** (1987), no. 1, pp. 1–102.

[Maz16] Damiano Mazza, *The true concurrency of differential interaction nets*, Mathematical Structures in Computer Science **28** (2016), 1097–1125.

[Ter11] Kazushige Terui, *Computational ludics*, Theoretical Computer Science **412** (2011), pp. 2048–2071.