Proofnets and context semantics for the additives

Harry G.Mairson   Xavier Rival

Computer Science Department
Brandeis University
Waltham, Massachusetts 02454
`mairson@cs.brandeis.edu`

École Normale Superieure
45 rue d'Ulm
75005 Paris
`rival@di.ens.fr`

Abstract. We provide a *context semantics* for Multiplicative-Additive Linear Logic (MALL), together with *proofnets* whose reduction preserves semantics, where proofnet reduction is equated with cut-elimination on MALL sequents. The results extend the program of Gonthier, Abadi, and Lévy, who provided a "geometry of optimal λ-reduction" (context semantics) for λ-calculus and Multiplicative-Exponential Linear Logic (MELL). We integrate three features: a semantics that uses *buses* to implement slicing; a proofnet technology that allows *multidimensional boxes* and *generalized garbage*, preserving the linearity of additive reduction; and finally, a *read-back procedure* that computes a cut-free proof from the semantics, which is closely related to full abstraction theorems.

Peut-être que la logique se trompe.
—Yves Lafont

Linear Logic [4, 7] appeared in 1987 and turned out quickly to be an interesting tool to model programming languages, specifically reasoning that is sensitive to the notion of *consumable resources.* Indeed the *multiplicative* fragment of Linear Logic ($\otimes$, $\parr$) allows linear products (pairing and unpairing), implementing functions: a *context* pairs a continuation and an argument, a *function* unpairs and connects the two. The *additive* fragment ($\oplus$, $\&$) allows linear sums (injection and case dispatch), implementing features of *processes* in the style of CSP or CCS [3, 17, 12]. The crucial difference between these two components is the way they take care with consumption of resources. The *exponential* fragment implements sharing of resources: arguments, control contexts. We can then implement, for example, graph reduction technology for λ-calculus with control operators (`call/cc`, `abort`, jumps), and related mechanical proof systems for classical logic—taking care of the *sharing* and *copying*, implicit in these calculi [16, 18, 11].

Linear Logic was initially a sequent calculus, but this sequentialized structure was too strong and the design of a nice cut-elimination procedure was complicated. Therefore, proofnets were introduced as a more flexible representation of proofs [8, 13].

Further, Geometry of Interaction (GoI) developed the idea that the reduction of proofs can be seen as a local interaction process [5, 6]. Its intensional features provide a mediating Purgatory between the Heaven of denotational semantics, and the Hell of operational semantics. GoI was simplified in the "geometry of optimal λ-reduction" by Gonthier, Abadi and Lévy [9, 10] in the context of the MELL fragment. They reduced Hilbert spaces to simple data-structures, known as *context semantics*, and developed a proofnet technology which implemented the context semantics locally. Reduction on proofnets preserves the semantics, and Lamping's algorithm for optimal reduction of λ-terms [14] is a method of graph reduction. They further indicated how to *read back* any part of the Böhm tree (normal form) of a λ-term from its context semantics.

Can this program be carried out for full Linear Logic?

In this paper we extend these result to the MALL fragment (multiplicatives and additives): this may be a step towards a satisfactory proofnet syntax for full Linear Logic with a good characterization of proofs.

The MALL fragment is quite problematic since it does not have a nice cut-elimination procedure—unlike MLL, which enjoys a straightforward one. Part of the work will be to understand and improve the reduction procedure for MALL.

The main contributions of this paper are to provide an integrated development of (1) a context semantics for the MALL fragment; (2) a proofnet technology allowing normalization of MALL proofs, using the ideas of *multidimensional boxes* and *generalized garbage*; and (3) a read-back procedure that inputs a valid context semantics and outputs a normalized proofnet.

In Section 1, we recall some basic definitions on MALL, proofnets and Linear Logic. We introduce in Section **??** a form of context semantics for MALL proofs, and we derive from it a bus-notation based proofnet syntax in Section 3. Then, we envisage the main problems that come from the reduction on the additives in Section 4. The first difficulty is that the additive cut-elimination is not really linear, since an additive reduction step erases a whole part of the proof (this is also a problem for the locality of the reduction we would like to achieve). The second problem also stems from the additives: the way one should reduce a cut involving auxiliary formulas of two &-links is unclear. The solution to this problem will come from an extension of the MALL syntax. After this adaptation, we will be able to get a much more satisfactory cut-elimination procedure. In Section 5, we show how a normalized proof can be read-back from the context semantics of a proof, and we see how this result can be related to a form of full completeness. In Section 6, we compare our approach with other works.

1 MALL: Proofs, Nets, Reduction

Definition 1 (Formula). *The* formulas *of* MALL *are generated by the grammar* $F \longrightarrow V \mid V^{\perp} \mid F \otimes F \mid F \parr F \mid F \& F \mid F \oplus F$ *where* V *ranges over variables,* $\otimes$ *and* $\parr$ *are the conjunction and disjunction of the multiplicative component,* & *and* $\oplus$ *are the conjunction and disjunction of the additive component;* $(-)^{\perp}$ *is the involutive negation on literals.*

The atomic negation can be extended to the formulas as a defined involutive connector, using the De Morgan identities $(A \otimes B)^{\perp} = A^{\perp} \parr B^{\perp}$ and $(A \& B)^{\perp} =$

$A^\perp \oplus B^\perp$. We use right-handed sequents, where all sequent formulas play the same role.

Definition 2 (Sequent). *A* sequent *is a multiset of formulas* $\vdash F_0, \ldots, F_{n-1}$ *(also simply written* $F_0, \ldots, F_{n-1}$*).*

Figure 1 gives the MALL rules: *Ax* and *Cut* are the identity rules. The rules $\otimes$ and $\parr$ (resp. &, $\oplus_0$ and $\oplus_1$) are the multiplicative (resp. additive) rules.

$$\frac{}{\vdash A, A^\perp}\,\mathrm{Ax} \qquad \frac{\vdash \Gamma, A \quad \vdash \Delta, A^\perp}{\vdash \Gamma, \Delta}\,\mathrm{Cut} \qquad \frac{\vdash \Gamma, A \quad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \otimes B}\,\otimes$$

$$\frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B}\,\parr \qquad \frac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, A \& B}\,\& \qquad \frac{\vdash \Gamma, A}{\vdash \Gamma, A \oplus B}\,\oplus_0 \qquad \frac{\vdash \Gamma, B}{\vdash \Gamma, A \oplus B}\,\oplus_1$$

Fig. 1. The rules of the Multiplicative and additive fragment

Definition 3 (Prooftree). *A* prooftree *(or* MALL-prooftree*) is a tree whose leaves are sequents, linked by the rules showed in Figure 1.*

There is exactly one *introduction* rule for each additive or multiplicative connector (except $\oplus$); the *principal formula* of a link is the new formula introduced, and the other formulas are *auxiliary.* The *cut formulas* of a cut-link are the two hypotheses that are *eliminated* in the conclusion (A and $A^\perp$ in Figure 1). An *immediate cut* is a cut link whose cut formulas are the principal formulas of the two links above the cut. The *ports* of a prooftree are the formulas in the final proof link. Since full linear logic has a cut-elimination procedure, so does MALL (see [4]):

Theorem 1 (Cut-elimination). *There exists an algorithm which inputs a prooftree π of MALL sequent S, and outputs a prooftree π' of the same sequent S without any occurrence of the Cut-rule.*

A well-known interpretation of the connectives is *economic* [7]: the formulas of a sequent are the algebraic terms of a trade. Negation represents need, and involution ($A^{\perp\perp} = A$) means if you need to need, you have. A proof of $\vdash \Gamma, A^\perp$ means you only need (a proof of) A to produce (a proof of) Γ; thus the *Ax*-rule says that you need A to produce A. Similarly, the *Cut*-rule says if you need a proof of $\Gamma^\perp$ to produce a proof of A, and you need a proof of A to produce a proof of Δ, then transitively "trading" the need and production of A, you need a proof of $\Gamma^\perp$ to produce a proof of Δ— thus $\vdash \Gamma, \Delta$ is provable.

The multiplicative and additive rules describe two related trading situations (see Figure 1). In the former, we need $\Gamma^\perp, \Delta^\perp$ to produce both A and B; in the latter, we need $\Gamma^\perp$ as a sufficient resource to prove either A or B.

1.1 Cut-elimination and proofnets

Cut-elimination for MALL is described by a collection of local rewriting rules that push *Cut*-links upwards and make them disappear; these rules appear in the proof of Theorem 1. The rules are not confluent, partly because the prooftree syntax introduces unnecessary sequentializations. For instance, if we start with the proof:

$$\dfrac{\dfrac{\dfrac{\pi_0}{\vdash \Gamma, A, B, F}}{\vdash \Gamma, A \parr B, F}\,\parr \qquad \dfrac{\dfrac{\pi_1}{\vdash \Delta, C, F^{\perp}}}{\vdash \Delta, C \oplus D, F^{\perp}}\,\oplus_0}{\vdash \Gamma, \Delta, A \parr B, C \oplus D}\,Cut$$

then we can rewrite it in two steps to a proof ending with one of the following sequences of links:

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \vdash \Gamma, \Delta, A, B, C\end{array}}{\vdash \Gamma, \Delta, A \parr B, C}\,\parr}{\vdash \Gamma, \Delta, A \parr B, C \oplus D}\,\oplus_0 \qquad \dfrac{\dfrac{\begin{array}{c}\vdots\\ \vdash \Gamma, \Delta, A, B, C\end{array}}{\vdash \Gamma, \Delta, A, B, C \oplus D}\,\oplus_0}{\vdash \Gamma, \Delta, A \parr B, C \oplus D}\,\parr$$

Proofnets [8, 13] eliminate such useless sequentializations: the two proofs above have the same meaning and should not be distinguished.

The &-connector is unique in its problematic *sharing* of the formulas Γ in the production of $\Gamma, A\&B$ from both Γ, A and Γ, B (see Figure 1). This non-linear phenomenon is represented in proofnet syntax either by drawing a *box* around the two subproofs (becoming the left and the right side of the box) above a &-link, or by adding a Boolean *eigenweight* to all the formulas in the proof. In the latter, the formulas on the left and right sides of the &-link get opposite boolean values, making their distinction possible. Each side is called a *slice*. These two approaches (boxes and weights) are equivalent. We define in the following a simple proofnet syntax, and we will provide a semantics that lets us distinguish slices.

Definition 4 (Proofnets with boxes). *The inductive encoding* [.] *of prooftrees into* proofnets *is shown in Figure 2. A* port *of the proofnet* $[\pi]$ *is a proofnet* wire *corresponding to a port of* π.

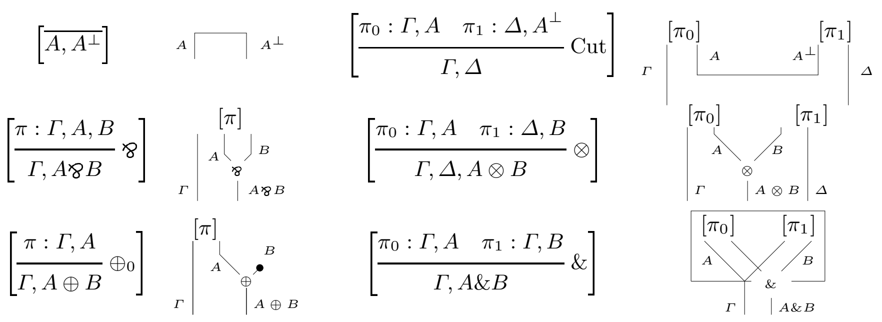

$\left[\overline{A, A^{\perp}}\right]$
A
$A^{\perp}$
$\left[\dfrac{\pi_0 : \Gamma, A \quad \pi_1 : \Delta, A^{\perp}}{\Gamma, \Delta}\,\text{Cut}\right]$
$[\pi_0]$
$[\pi_1]$
Γ
Δ
$\left[\dfrac{\pi : \Gamma, A, B}{\Gamma, A \parr B}\,\parr\right]$
$[\pi]$
$A \parr B$
$\left[\dfrac{\pi_0 : \Gamma, A \quad \pi_1 : \Delta, B}{\Gamma, \Delta, A \otimes B}\,\otimes\right]$
$A \otimes B$
$\left[\dfrac{\pi : \Gamma, A}{\Gamma, A \oplus B}\,\oplus_0\right]$
B
$A \oplus B$
$\left[\dfrac{\pi_0 : \Gamma, A \quad \pi_1 : \Gamma, B}{\Gamma, A\&B}\,\&\right]$
$\&$
$A\&B$

Fig. 2. Proofnets with boxes

The reduction of cuts in proofs annihilates reciprocal links (& and $\oplus$, or $\otimes$ and $\parr$). *Geometry of interaction* [5, 6] provides a mathematical framework for this phenomenon, where a semantics (see [9, 10, 15]) is defined that is meant to be preserved by reduction. We now define such a semantics, and use it as a starting point to define a proofnet syntax containing both the meaning of proofs, and the handling of cut-elimination.

2 Context semantics for MALL prooftrees

We define a context semantics for MALL prooftrees. This semantics could be defined on equivalent proofnets; we will give a proofnet syntax accomodating this semantics in Section **??**. The semantics is essentially described by *contexts* comprising *eigenweights* and *command strings*. Contexts relate the structure of formulas and proofs.

Definition 5 (Eigenweight, eigenvalue). *An* eigenweight *is a variable ranging over the booleans $\mathcal{B} = \{0, 1\}$. If W is a set of eigenweights, an* eigenvalue with base W *is a function $W \to \mathcal{B}$.*

Each eigenweight corresponds to a &-link (rule) in the proof. The value 0 (resp. 1) characterizes the left (resp. right) part of the subproof above the link.

Definition 6 (Context). *A* token *is one of the elements of the set $\{l, r, g, d\}$; l and r (*left *and* right*) are the* multiplicative tokens*, g and d (*gauche *and* droite*) are the* additive tokens*.*

The command strings $\mathcal{S}$ *are defined by $s \longrightarrow \diamond \mid t.s$ where $t \in \{l, r, g, d\}$.*

Given a set W of eigenweights, the contexts with base $\mathcal{C}_W$ *is the set pairs (s, ω) where s is a command string and ω is an eigenvalue with base W.*

Given W, we can inductively define the set C_F of valid command strings for a formula F as: $C_{A \otimes B} = C_{A \parr B} = l.C_A \cup r.C_B$ for the multiplicatives, $C_{A \& B} = C_{A \oplus B} = g.C_A \cup d.C_B$ for the additives, and $C_V = C_{V^\perp} = \mathcal{C}_W$ for variables.

Thus a command string describes a possible path in a formula, and eigenweights define slices in proofs. Since a context contains an eigenvalue and a command string, it defines a position in the additive structure of a proof (i.e., a slice and a path in formulas). In fact, this second component of a context will correspond to a path in proofs and not in formulas—but a path in a normal proof corresponds exactly to a path in a formula.

2.1 Semantics of a prooftree

Definition 7 (Semantics of a prooftree). *Let π be a prooftree of $\vdash \Gamma$ and W be a set of eigenweights, one for each &-rule in π. Let $\mathcal{F}_\pi$ be the set of occurrences of formulas in π. The* ports *of π are the formulas in Γ.*

For each eigenvalue ω (i.e., for every function $W \to \mathcal{B}$) we define a binary relation $\to_\omega$ on $\mathcal{F} \times \mathcal{S}$. This relation is the union of the contributions of all the links in the proof, the contribution of a rule being defined in Figure 3. Let $\leftrightarrow^\star_\omega$ be the reflexive transitive closure of $\to_\omega$.

The context semantics *of π is the function $[\![\pi]\!] : \Gamma \times \mathcal{S} \times (W \to \mathcal{B}) \to \Gamma \times \mathcal{S}$ such that $[\![\pi]\!](F, s, \omega) = (F', s')$ if and only if $(F, s) \leftrightarrow^\star_\omega (F', s')$*

$$\begin{array}{c}\dfrac{}{\vdash A, A^{\perp}}\,\mathrm{Ax}\\ (A,s) \to_{\omega} (A^{\perp},s)\\ (A^{\perp},s) \to_{\omega} (A,s)\end{array}\qquad \begin{array}{c}\dfrac{\vdash \Gamma^1, A \quad \vdash \Delta^1, A^{\perp}}{\vdash \Gamma^0, \Delta^0}\,\mathrm{Cut}\\ (F^0 \in \Gamma^0, s) \to_{\omega} (F^1 \in \Gamma^1, s)\\ (F^0 \in \Delta^0, s) \to_{\omega} (F^1 \in \Delta^1, s)\\ (A,s) \to_{\omega} (A^{\perp},s)\\ (A^{\perp},s) \to_{\omega} (A,s)\end{array}\qquad \begin{array}{c}\dfrac{\vdash \Gamma^1, A \quad \vdash \Delta^1, B}{\vdash \Gamma^0, \Delta^0, A \otimes B}\,\otimes\\ (F^0 \in \Gamma^0, s) \to_{\omega} (F^1 \in \Gamma^1, s)e\\ (F^0 \in \Delta^0, s) \to_{\omega} (F^1 \in \Delta^1, s)\\ (A \otimes B, l.s) \to_{\omega} (A,s)\\ (A \otimes B, r.s) \to_{\omega} (B,s)\end{array}$$

$$\begin{array}{c}\dfrac{\vdash \Gamma^1, A, B}{\vdash \Gamma^0, A ⅋ B}\,⅋\\ (F^0 \in \Gamma^0, s) \to_{\omega} (F^1 \in \Gamma^1, s)\\ (A ⅋ B, l.s) \to_{\omega} (A,s)\\ (A ⅋ B, r.s) \to_{\omega} (B,s)\end{array}\qquad \begin{array}{c}\dfrac{\vdash \Gamma^1, A \quad \vdash \Gamma^2, B}{\vdash \Gamma^0, A \& B}\,\&(w)\\ \text{if} \quad \omega(w_i) = 0, \quad \text{then:}\\ (F^0 \in \Gamma^0, s) \to_{\omega} (F^1 \in \Gamma^1, s)\\ (A \& B, g.s) \to_{\omega} (A,s)\\ \text{if} \quad \omega(w_i) = 1, \quad \text{then:}\\ (F^0 \in \Gamma^0, s) \to_{\omega} (F^1 \in \Gamma^2, s)\\ (A \& B, d.s) \to_{\omega} (B,s)\end{array}\qquad \begin{array}{c}\dfrac{\vdash \Gamma^1, A}{\vdash \Gamma^0, A \oplus B}\,\oplus_0\\ (F^0 \in \Gamma^0, s) \to_{\omega}\\ (F^1 \in \Gamma^1, s)\\ (A \oplus B, g.s) \to_{\omega}\\ (A,s)\end{array}$$

Fig. 3. Context semantics of prooftrees
(by convention we assign distinct marks to distinct occurrences of the same formula)

Intuitively, an eigenvalue defines a slice in the proof, that is a choice of a side (left or right) for each &-rule in the proof. Once a slice ω is chosen, the relation $\to_{\omega}$ defines the set of paths going up in the proof that is included in this slice.

The transitive closure $\to_{\omega}^{\star}$ of $\to_{\omega}$ defines upwards paths in the proof that are contained in the slice defined by ω (a maximal upwards path starts either at an hypothesis or at a cut formula and end at an axiom formula). The reflexive transitive closure $\leftrightarrow_{\omega}^{\star}$ of $\to_{\omega}$ defines the set of all the valid paths in the slice defined by ω.

The semantics collects the paths included in all the slices of the proof.

This semantics is similar to the "token machine" of Olivier Laurent [15]: the definition of contexts is related to his definition. The main difference is that the weight information has the same format everywhere in the proof in our settings: &-links do not pop or push weight tokens but just read them as his machine does. This is more convenient for the design of proofnets.

This semantics is compositional (the semantics of a prooftree can be computed inductively). The definition above could be adapted to the proofnets defined in Section 1 straightforwardly.

Example 1. Let us consider the proof π (with the convention that distinct occurrences of a same formula get distinct marks):

$$\dfrac{\dfrac{\dfrac{}{(A^{\perp})^2, A^1} \qquad \dfrac{}{(A^{\perp})^3, A^2}}{(A^{\perp})^1, A^0}\,Cut \qquad \dfrac{\dfrac{}{(A^{\perp})^5, A^3}}{(A^{\perp})^4, A \oplus B}\,\oplus_l}{(A^{\perp})^0, A \& (A \oplus B)}\,\&(w)$$

Then, if $\omega_i(w) = i$ (for $i = 0, 1$), we have:

$$((A^\perp)^0, \diamond) \to_{\omega_0} ((A^\perp)^1, \diamond) \to_{\omega_0} ((A^\perp)^2, \diamond)$$
$$(A^1, \diamond) \to_{\omega_0} ((A^\perp)^2, \diamond)$$
$$((A^\perp)^3, \diamond) \to_{\omega_0} (A^2, \diamond)$$
$$(A\&(A \oplus B), g.\diamond) \to_{\omega_0} (A^0, \diamond) \to_{\omega_0} (A^2, \diamond)$$

The formulas A^1 and $(A^\perp)^3$ are cut together therefore:

$$[\![\pi]\!]((A^\perp)^0, \diamond, \omega_0) = [\![\pi]\!](A\&(A \oplus B), g.\diamond)$$

Similarly, we have: $[\![\pi]\!](A\&(A \oplus B), d.\diamond, \omega_1) = ((A^\perp)^0, \diamond)$

We will address the problem of preservation of the context semantics by the reduction in Section 4.

3 Proofnets

We now have a semantics for MALL proofs. In this section, we give a proofnet syntax based on the bus notation (in the spirit of the notation used in [9]) that supports another much simpler definition of the semantics and local reductions.

3.1 Syntax and semantics

We have seen above that the semantics involves two complementary parts: an eigenvalue and a command string. Hence we will replace the simple wires of the proofnets we considered in Section 1 by buses of wires representing the two contributions of the semantics. Given a proof that contains n &-links (that are in bijection with n eigenweight variables $w_0, \ldots, w_{n-1}$), edges in the encoded proofnet will be composed of:

– n *weight wires*, one for each variable $w_0, \ldots, w_{n-1}$.
– one *command wire*

Weight wires will be drawn on the left, the command wire on the right. Intuitively, a weight wire carries the weight information for one eigenweight, the command wire carries the command information (command string).

There will be three types of ternary nodes: the multiplicative nodes, the additive nodes and the weight nodes. A node has two auxiliary ports (above the triangle) and one principal port (below the triangle). Note that multiplicative and additive nodes act on the command wire whereas a weight node acts on a weight wire (see Figure 4).

We suppose that each edge of the proofnet is labeled by a formula. Some wires do not end at a port of a node: they are either the ports of the proofnet or terminated by a plug.

We are now ready for the recursive encoding of prooftrees into proofnets.

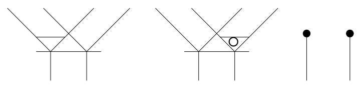


Fig. 4. weight node, command node (∘), plug

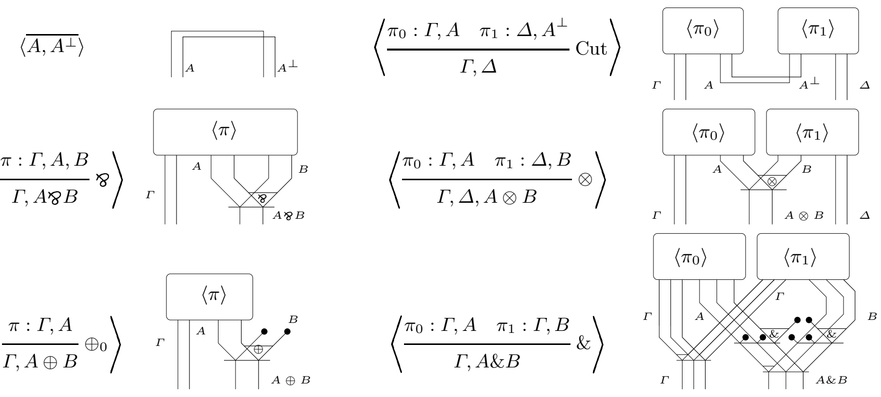

$\langle \overline{A, A^\perp} \rangle$
A
$A^\perp$
$\left\langle \dfrac{\pi_0 : \Gamma, A \quad \pi_1 : \Delta, A^\perp}{\Gamma, \Delta}\text{Cut} \right\rangle$
$\langle\pi_0\rangle$
$\langle\pi_1\rangle$
Γ
Δ
$\left\langle \dfrac{\pi : \Gamma, A, B}{\Gamma, A \parr B}\parr \right\rangle$
$\langle\pi\rangle$
B
$A \parr B$
$\left\langle \dfrac{\pi_0 : \Gamma, A \quad \pi_1 : \Delta, B}{\Gamma, \Delta, A \otimes B}\otimes \right\rangle$
$A \otimes B$
$\left\langle \dfrac{\pi : \Gamma, A}{\Gamma, A \oplus B}\oplus_0 \right\rangle$
$A \oplus B$
$\left\langle \dfrac{\pi_0 : \Gamma, A \quad \pi_1 : \Gamma, B}{\Gamma, A \& B}\& \right\rangle$
$A \& B$

Fig. 5. Proofnets with bus notation

Definition 8 (Proofnets with Bus-notation). *The recursive encoding of a prooftree π into a proofnet $\langle\pi\rangle$ is done according to the rules in Figure 5.*

In the following, "proofnet" by default always means "bus-notation based proofnet".

Note that an additive node is initially connected to one plug, a multiplicative node is connected to no plug.

The definition of the context semantics of a proofnet is similar to the definition in the case of prooftrees given in Section 2.1. Given an eigenvalue ω, we just replace the relation $\rightarrow_\omega$ by the relation $\hookrightarrow_\omega$ on $\mathcal{E} \times \mathcal{S}$ (where the set $\mathcal{F}$ of occurrences of formulas in the prooftree is replaced by the set of edges of the graph $\mathcal{E}$) defined as follows (we note a_l and a_r the left and right auxiliary ports and p the principal port of a node, a port being identified to the edge connected to it):

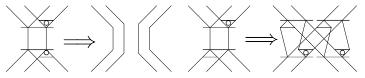
for an additive node:	for a multiplicative node:	for a weight node:
$(p, g.s) \hookrightarrow_\omega (a_l, s)$	$(p, l.s) \hookrightarrow_\omega (a_l, s)$	$\omega(w) = 0 \Rightarrow (p, s) \hookrightarrow_\omega (a_l, s)$
$(p, d.s) \hookrightarrow_\omega (a_r, s)$	$(p, r.s) \hookrightarrow_\omega (a_r, s)$	$\omega(w) = 1 \Rightarrow (p, s) \hookrightarrow_\omega (a_r, s)$

Nodes in proofnets act on contexts as *routers*: in this sense we can say that proofnets are a low-level encoding of context semantics.

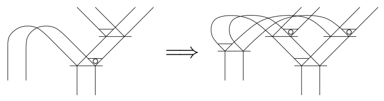
3.2 Local reduction rules

We will need the same reduction rules as in [9] plus some extra ones. We suppose we only have one eigenweight variable in the pictures that follows (all the rules are not represented in the pictures for brevity; the other rules can be deduced from those that are presented).

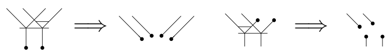
– The *Cut-rules* make two nodes interact on their principal port. If they are on the same wire and of the same type they disappear (this will be the case, for instance, when an immediate cut will be reduced), else they duplicate each other. This rule will be used primarily for box copying.



– The *η-rule* corresponds to the duplication of a node by a weight node (it play a part in duplication of boxes and in η-expansion of proofs in 5.5).



– The presence of plugs inside the proofnet means some nodes might be useless—for instance, a multiplicative node with a plug on one of its ports, or an additive node with plugs on two of its ports are meaningless. Thus we have *garbage collection* rules for deleting such nodes and propagate plugs.



All these rules are local. This is one of the advantages of this proofnet syntax.

The Cut and η rules straighforwardly preserve the semantics. The garbage collection rules "select" some part of the semantics and throw away some non relevant part of it.

4 Cut elimination

In this section we envisage some problems that happen during cut-elimination and the way we solve them. We will finally get a nice link between the reduction of the prooftree and the reduction of the proofnet.

4.1 Multiplicative immediate cuts

Let us consider a multiplicative cut:

$$\dfrac{\dfrac{\dfrac{\pi_0}{\Gamma, A} \quad \dfrac{\pi_1}{\Delta, B}}{\Gamma, \Delta, A \otimes B}\otimes \qquad \dfrac{\dfrac{\pi_2}{\Pi, A^\perp, B^\perp}}{\Pi, A^\perp ⅋ B^\perp}⅋}{\Gamma, \Delta, \Pi}\mathit{Cut}$$

In the encoded proofnet, we have two multiplicative nodes facing each other on their auxiliary port so the cut rule can be applied and make these two nodes disappear. The resulting proofnet is the encoding of the prooftree obtained by eliminating the cut in the above prooftree (i.e., with two cuts on A and B).

4.2 Additive immediate cuts and garbage

Let us consider the following additive cut-elimination step:

$$\dfrac{\dfrac{\dfrac{\pi_0}{\Gamma, A} \qquad \dfrac{\pi_1}{\Gamma, B}}{\Gamma, A\&B}\,\&(w) \qquad\qquad \dfrac{\dfrac{\pi_2}{\Delta, A^\perp}}{\Delta, A^\perp \oplus B^\perp}\,\oplus_0}{\Gamma, \Delta}\,Cut \qquad \Longrightarrow \qquad \dfrac{\dfrac{\pi_0}{\Gamma, A} \qquad \dfrac{\pi_2}{\Delta, A^\perp}}{\Gamma, \Delta}\,Cut$$

This step can somewhat be handled on the encoded proofnet —but in several steps: duplication of the additive node corresponding to the $\oplus$-link and of the proof $\langle \pi_2 \rangle$ by a w-weight node, annihilation of the two resulting $\oplus$ nodes by the & nodes (Cut rule), garbage collection of the proof $\langle \pi_1 \rangle$ and of the right duplicate of $\langle \pi_2 \rangle$ and then of the w-weight nodes.

This is not completely satisfactory because garbage collection modifies the semantics (by selecting its "meaningful" part). In the following, we will clearly distinguish the cut-elimination step and the garbage collection step: the first one preserves the semantics while the second selects the good part. In order to do so, we will have to introduce *garbage* explicitly in the proofs (i.e., parts that would disappear in the usual MALL prooftrees) and to remove it after the normalization. The partial additive cut-elimination step is described on figure 6.

This approach is related to the $\flat$-rules of [6, 15]. We will present it in 4.4.

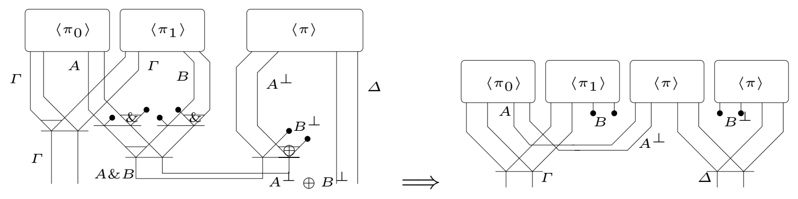


Fig. 6. Additive cut-elimination, generation of garbage

4.3 Cut of &-links on their auxiliary port

The only case of non immediate cuts that matters corresponds to cuts involving one auxiliary link of a box (encoding of a &-link). If the other link involved by the cut is not an auxiliary port of a box, then it is just copied. If both links above the cut are auxiliary ports of boxes, we have a proof like:

$$\dfrac{\dfrac{\dfrac{\pi_0}{\Gamma, A, F} \qquad \dfrac{\pi_1}{\Gamma, B, F}}{\Gamma, A\&B, F}\,\&(w_0) \qquad\qquad \dfrac{\dfrac{\pi_2}{\Delta, C, F^\perp} \qquad \dfrac{\pi_3}{\Delta, D, F^\perp}}{\Delta, C\&D, F^\perp}\,\&(w_1)}{\Gamma, \Delta, A\&B, C\&D}\,Cut$$

This non immediate cut can be reduced in two ways: we can push it upwards left and duplicate the right &-link or push it upwards right and duplicate the left &-link. The same happens for the encoded proofnet.

Proofnets were introduced to avoid useless sequentializations and we have to face here a case where they fail to achieve this goal: the two rewritings suggested above are completely symmetric and we have no reason to choose one instead of the other.

The solution we choose is to merge the "boxes" that correspond to each &-link into a *two dimensional box*. We will add a rule in the prooftree syntax corresponding to n &-links in parallel. As regards the proofnets, we will not modify the existing syntax. A n dimensional higher-order &-link (or "higher order box") will be encoded like a "normal" &-link: at each port of the subnet encoding the box, we will have a tree of weight nodes of height n instead of a single one.

The order of the weight nodes at the ports of the encoding of a higher-order box is left arbitrary. This is not problematic since the η-rule can permute them. A cut involving auxiliary ports of two boxes is shown in Figure 7.

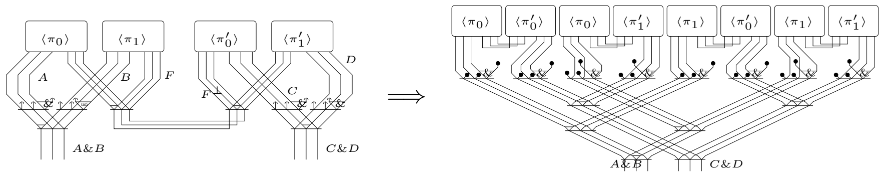


Fig. 7. Generation of a higher dimensional box (of dimension 2)

4.4 Towards a new sequent syntax

We have seen, in the last two subsections, we will have to make some changes to the prooftree syntax in order to include some garbage information in prooftrees and to parallelize several &-links. These two points will be handled by a single rule, generalizing the &-rule:

Definition 9 (Generalized &-rule). *The* $\&(n, k, g_0)$-rule *(where* $n, k \in \mathbb{N}$*, and* $g_0 \in \mathcal{B}^k$*) has*

- *a conclusion:* $\Gamma, A_0^1 \& A_1^1, \ldots, A_0^n \& A_1^n$
- 2^{n+k} *hypotheses* $\pi(b)$ *where* $b \in B^{n+k}$*:*
  - 2^n *proofs* $\pi(b, g_0) : \Gamma, A_{b_1}^1, \ldots, A_{b_n}^n$ *where* $b \in \mathcal{B}^n$
  - $2^n(2^k - 1)$ *garbage proofs* $\pi(b, g) : \Gamma, A_{b_1}^1, \ldots, A_{b_n}^n$ *where* $b \in \mathcal{B}^n$ *and* $g \in \mathcal{B}^k \setminus \{g_0\}$.

The $\bullet$ symbol, chosen to evoke a plug, marks garbage. These plugs guarantee that the garbage is disconnected; later we will see that detecting this disconnectedness in the semantics is decidable, which facilitates read-back.

Moreover, some rules will be useful to introduce and handle garbage:

$$\frac{\Gamma, X \quad \Delta, Y}{\Gamma, \Delta, \bullet}\,\mathbf{G} \qquad \frac{\Gamma, \bullet, \bullet}{\Gamma, \bullet}\,\bullet$$

The "usual" & rule now corresponds to the $\&(1,0)$-rule (one principal conclusion $A \& B$, no garbage, and two hypotheses).

The problem of adding garbage in prooftrees occured in Section 4.2 while considering the elimination of an immediate additive cut. The elimination of the additive $\&(1,0)(w)$ box below against an $\oplus_0$ link results in a $\&(0,1,(w \mapsto 0))$ box:

$$\dfrac{\dfrac{\dfrac{\pi_0}{\Gamma, A_0} \quad \dfrac{\pi_1}{\Gamma, A_1}}{\Gamma, A_0 \& A_1} \quad \dfrac{\dfrac{\pi_2}{\Delta, A_0^\perp}}{\Delta, A_0^\perp \oplus A_1^\perp}\oplus_0}{\Gamma, \Delta} \Longrightarrow \dfrac{\dfrac{\dfrac{\pi_0}{\Gamma, A_0} \quad \dfrac{\pi_2}{\Delta, A_0^\perp}}{\Gamma, \Delta}\mathit{Cut} \quad \dfrac{\dfrac{\pi_1}{\Gamma, A_1} \quad \dfrac{\pi_2}{\Delta, A_0^\perp}}{\Gamma, \Delta, \bullet}\mathbf{G}}{\Gamma, \Delta}$$

More generally, a $\&(n,0)$ rule corresponds to the parallelization of n &-links.

The cut-elimination on extended prooftrees is quite similar to the cut-elimination on usual prooftrees. The generalized &-link behaves as follows:

- a $\&(n_0, k_0, g_0)$-link cut against a $\&(n_1, k_1, g_1)$-link results in a $\&(n_0+n_1, k_0+k_1, (g_0, g_1))$-link. For instance in Section 4.3, two $\&(1,0)$-links are cut against each other and we get a $\&(2,0)$-link.
- a $\&(n, k, g)$-link cut on one of its principal ports F against a $\oplus_i$-link results in a $\&(n-1, k+1, g')$-link, where g' is obtained from g by assigning i to the eigenweight corresponding to the principal port F.

An extended MALL proof can be translated back to a MALL proof (erasure of the garbage in the proof) so extended MALL is exactly as expressive as MALL.

4.5 Correctness of the semantics with respect to the reduction

We are now able to prove that our proofnets can handle additive cut-elimination and that reduction (e.g. cut- elimination) preserves the semantics.

Theorem 2 (Cut-elimination on proofnets). *Let π_0 be an extended MALL prooftree. If π_0 can be reduced to the prooftree π_1, then: $\langle \pi_0 \rangle \longrightarrow^\star_{Cut+\eta} \langle \pi_1 \rangle$ If π_1 is normal, $\langle \pi_1 \rangle$ is also normal (there is no local cut-redex in $\langle \pi_1 \rangle$).*

Proof. By induction on the length of the reduction.

The case of a multiplicative immediate cut is obvious.

The case of an additive immediate cut is treated above.

A non immediate cut is either two generalized &-links cut against each other on their auxiliary port (treated in Section 4.4) or a generalized &-link cut against another link: then the other link is duplicated and absorbed by the &-link.

Theorem 3 (Correctness). *With the notations of Theorem 2,* $[\![\pi_0]\!] = [\![\pi_1]\!]$ *Or equivalently:* $[\![\langle\pi_0\rangle]\!] = [\![\langle\pi_1\rangle]\!]$

Proof. By induction on the length of the reduction.

The case of a one step reduction is straightforward.

Moreover the garbage collection does the same work on the prooftree and on the proofnet. In both cases it does modify the semantics.

These two results can be summarized by the diagram:

$$\begin{tikzcd}
\pi_0 \arrow[r, "Cut"] \arrow[d] & \pi_1 \arrow[r, "GC"] \arrow[d] & \pi_2 \arrow[d] \\
\langle\pi_0\rangle \arrow[r, "Cut+\eta"] \arrow[d] & \langle\pi_1\rangle \arrow[r, "GC"] \arrow[d] & \langle\pi_2\rangle \\
[\![\pi_0]\!] \arrow[r, equal] & [\![\pi_1]\!] &
\end{tikzcd}$$

5 Read-back and completeness

5.1 Overview

The read-back of a proof consists in building a normal proof (i.e., without cut) from a valid semantics. This way, we can hope to get normal proofs without normalizing. This property corresponds to the following theorem:

Theorem 4 (Definition of read-back). *There exists an algorithm* $\mathcal{R}$ *that inputs the context semantics* S *of a proof* π_0 *and outputs a normal proofnet* π_1 *such that* $[\![\pi_1]\!] = S$.

The proof (in the three next subsections) is necessarily constructive, since we expect an algorithm, and not merely the existence of a normalized proofnet having some semantics. We design a top-down algorithm that determines what the external structure (i.e., closed to the ports) of the proofnet can be, and recursively reapplies itself on some "subcomponents" of the initial semantics. At the end of the process, a proofnet can be reconstructed provided the algorithm keeps track of the operations it carries out.

The three main steps of the recursive deconstruction phase are:

– determination of the meaningful slice of the outermost boxes (and at the same time, removing garbage of outermost boxes);
– determination of the paths to the principal ports of the outermost boxes and of the structure of the normal proofnet outside the outermost boxes;
– re-application of the algorithm to each slice of the outermost boxes, after having eliminated the totally useless eigenweights for each component.

Note that the semantics of a proofnet π is finitely representable. Indeed, we know a normal proofnet π_0 that has the same semantics (thanks to the normalization property proved in Theorem 2) and the depth of a normal proof is bounded by the depth of the formulas at its ports. This remark implies that the semantics of a proofnet is finitely representable, and that the properties of the semantics we will design, which characterize the *normal* proofnets having this semantics, are decidable.

Whenever we use the word "proofnet" in the following, it could be replaced by "extended prooftree". However, this would have a consequence on the proofs: the algorithm would not be deterministic any more because it would involve a stage of sequentialization of a proofnet into a prooftree (which is not deterministic). This is the reason why we only consider proofnets.

5.2 Outermost boxes and garbage

The first step of the algorithm consists in finding the outermost boxes, so we first enumerate some properties characterizing the garbage-slices of a toplevel box.

Definition 10 (ω-slice). *Let π a be proof and ω an eigenvalue. We write $\mathcal{P}$ for the set of ports of π. For all command string s, we note $\mid s \mid$ the command string obtained from s by erasing the additive information. The ω-*slice *$[\![\pi]\!]_\omega$ of π is defined by: $[\![\pi]\!](p, s) = (p', s') \iff [\![\pi]\!]_\omega(p, \mid s \mid) = (p', \mid s' \mid)$*

Proposition 1. *Given the semantics of a proofnet coding a MALL proof in normal form, each ω-slice gives the context semantics of a normalized MLL proofnet.*

Proposition 2. *Two different boxes that branch on the same eigenweight cannot be in the same slice—an important case being at the top level. As a consequence, if a box B is not contained in any other box, there is only one copy of B in the proofnet.*

Box absorption causes copying of &-boxes into different slices, so we specify *occurrences* of boxes by choosing values for some subset of eigenweights, as well as a path of $\otimes, \parr, \&$ and $\oplus_i$ nodes leading to the box.

If $[\![\pi]\!]$ is the context semantics of a top-level $\&(n, k)$-box occurrence, some eigenweights $w_1, \ldots, w_n$ are "good" eigenweights and give useful slices of the box; the other eigenweights $w_{n+1}, \ldots, w_{n+k}$ are "bad" eigenweights and indicate garbage. How can we tell the good from the bad?

Lemma 1. *Let $[\![\pi]\!]$ be the context semantics of a* top-level *$\&(n, k)$-box occurrence with eigenweights W. Then p is a bad eigenweight if there exists a (bad) setting of p to 0 or 1 such that, for any setting of eigenweights $W - p$, the defined ω-slice does not give a MLL proofnet.*

Proof. Take a MLL proofnet and attach plugs to some nonempty subset of the ports of the nodes: the semantics of the proofnet does not then give another MLL proofnet (proof by induction).

That the box be top-level is important in the above argument: two occurrences of the same box (in different slices, by Proposition 2), may be cut against $\oplus_0$ in one case and $\oplus_1$ in the other, so there is no unique bad setting to the eigenweight. For example, consider

$$\dfrac{\dfrac{\dfrac{C, \Gamma, A^{\perp}}{C, \Gamma, A^{\perp} \oplus B^{\perp}}\oplus_0 \qquad \dfrac{C, \Gamma, B^{\perp}}{C, \Gamma, A^{\perp} \oplus B^{\perp}}\oplus_1}{C \&_p D, \Gamma, A^{\perp} \oplus B^{\perp}}\&(p) \qquad \dfrac{\Delta, A \qquad \Delta, B}{\Delta, A \&_q B}\&(q)}{C \&_p D, \Gamma, \Delta}\text{Cut}$$

where we annotate the implicit boxes with eigenvariables: in the normal form, p may be at top level, but q is not. While q is a "bad" eigenweight in the semantics, we cannot tell whether $q = 0$ or $q = 1$ results in garbage: in slice $p = 0$, $q = 1$ gives garbage, and in slice $p = 1$, $q = 0$ gives garbage. This is, essentially, why all garbage cannot be determined at the top level of the read-back algorithm.

By iterating the use of this Lemma, we can detect each of the k bad eigenweights and its bad setting, and then project out the good part of the semantics that does not involve garbage.

Definition 11 (projection). *Let $[\![\pi]\!]$ be the context semantics of an $\&(n, k, g_0)$-box with bad eigenweights w_1, …, w_n; the only assignment to $w_1, \ldots, w_n$ that does not lead to garbage is $g_0(w_1), \ldots g_0(w_n)$. Then the* projection *of $[\![\pi]\!]$ is obtained by restricting it to eigenvalues g such that $\forall i, g(w_i) = g_0(w_i)$.*

The first step of the algorithm consists in determining the garbage-eigenweights of the outermost box together with their good setting (i.e., eliminate garbage) as described above.

Observe that Proposition 2 assures us that top-level boxes have not been copied: when this condition fails, we may not be able to recover garbage immediately.

5.3 External structure of the proofnet

We now determine the structure of the proofnet that is external to the outermost box. This is done in two steps: (1) localize the &-links that correspond to the principal ports of the outermost boxes, and (2) recover the proofnet structure external to these boxes via a kind of projection. This is done by considering paths in the main formulas that end at a &-node.

Identifying paths to principal ports of outermost boxes Let F be a formula at a port and $\underline{\&}$ an occurrence of a &-connective in F. We call $\underline{\&}$ *primary* in F if either $F = A\underline{\&}B$, or $F = A \circ B$ ($\circ \in \{\otimes, \parr, \oplus\}$) and $\underline{\&}$ is primary in A or B. How do we determine if a primary occurrence is a port of an outermost box?

Definition 12 (&-path). *A command string s codes a* &-path *to a primary &-connective of a formula F if one of the following is satisfied:*

– *$F = A\&B$ and $s = \diamond$;*
– *$F = A \oplus B$ and $s = g.s'$ and s' codes a &-path of A;*
– *$F = A \oplus B$ and $s = d.s'$ and s' codes a &-path of B;*
– *$F = A \otimes B$ or $F = A \parr B$ and $s = l.s'$ and s' codes a &-path of A;*
– *$F = A \otimes B$ or $F = A \parr B$ and $s = r.s'$ and s' codes a &-path of B.*

Proposition 3. *The eigenweight w is the eigenweight of a box with command string s if there exists a port p, command string s coding a &-path to a primary &-link, such that for any command string s' and weight distribution ω:*

$$\begin{array}{c}[\![\pi]\!](p, s.g.s'', \omega) = (p'', c''') \Longrightarrow \omega(w) = 0 \\ [\![\pi]\!](p, s.d.s'', \omega) = (p'', c''') \Longrightarrow \omega(w) = 1\end{array}$$

Informally, Proposition 3 just says that w is a good eigenweight if whenever we take a path from a proofnet port to a primary &-node, w is always 0 when we go left and 1 when we go right. (Recall that the branching at a &-node is done both by the associated eigenweights *and* by the command string.)

If a top-level box has a &-connective at its port, the connective is primary, though it is not clear that every primary &-connective is at the port of a top-level box.

Recovering proofnet structure external to outermost boxes The identification of primary &-connectives and the &-paths to them uncovers a forest of trees, where each tree is located at a different external port of the proofnet, and constructed from the &-paths. By simultaneously examining the logical formula at a port, we can also recover the logical connectives along the path. Binary links in the trees only occur at $\otimes$ and $\parr$ nodes.

Definition 13 ($\circ$-removal). *Let p be a port with formula $A \circ B$ ($\circ \in \{\otimes, \parr\}$). A* $\circ$-removal *is a modification of the semantics that splits p into ports p' with formula A and p'' with formula B. (An $\oplus_i$-removal is defined similarly but p is then just replaced by another port instead of two.)*

Definition 14 (partition). *A* partition *is a minimal sequence of removals of nodes $n_1, \ldots, n_k$ where n_i ($i < k$) are $\parr$ or $\oplus_i$ nodes and n_k is a $\otimes$ node, where after the successive removals of $n_1, \ldots, n_{k-1}$, the removal of n_k divides the semantics into two disjoint sets S and S' of paths, such that the set of ports referenced in S are disjoint from those referenced in S'.*

Lemma 2. *If the semantics does not have a partition, there is at most one top-level box, and every primary &-node is a port.*

Our read-back procedure iterates the search for partitions. It keeps track of the nodes removed from each partition which makes the reconstruction of the proofnet possible at the end. Each *component* of the partition is guaranteed to have at most one top-level box.

5.4 Completely useless eigenweights

The last step divided the proofnet (the semantics) into several components. Each component corresponds to a top-level box, or is empty and then will not be considered any more. Before reapplying the algorithm on a slice of one component, it is useful to get rid of useless eigenweights corresponding to boxes of the other components.

An eigenweight w is *totally useless* if and only if the paths in the proofnet does not depend on it, which can be decided looking at the semantics $[\![\pi]\!]$ of a component, as it is equivalent to:

$$\forall F_0, F_1, s_0, s_1, \forall \omega, \begin{cases} [\![\pi]\!](F_0, s_0, \omega \mid_{w=0}) = [\![\pi]\!](F_1, s_1) \\ \Downarrow \\ [\![\pi]\!](F_0, s_0, \omega \mid_{w=1}) = [\![\pi]\!](F_1, s_1) \end{cases}$$

This deconstruction step corresponds in the reconstruction stage to the relamination of the components of the proofnets once they have been re-computed from their semantics (The *lamination* of a proofnet corresponds to adding one or several weight channels to it, that do not play any role. This is equivalent to adding some eigenweights to the base that are not matched by any &-link in the proof).

5.5 Correctness of read-back, consequences

The previous subsections show the existence of the algorithm $\mathcal{R}$. We prove here its correctness:

Theorem 5 (Correctness of read-back). *Read-back is correct: if π_0 is a MALL proofnet and if read-back applied to π_0 outputs a proofnet π_2 then:*

$$\pi_0 \rightarrow_{Cut+\eta} \pi_1 \rightarrow_{GC+Cut+\eta} \pi_2$$

where the first arrow corresponds to normalization and the second to full η-expansion (i.e., nodes connected to auxiliary ports of boxes are absorbed).

Proof (sketch). Reduction and η-expansion preserve the semantics. The read-back algorithm produces a deterministic result so we just have to look at the Cut-normal forms.

The *η-expansion* related above comes from the fact that the context semantics cannot distinguish proofs that are equivalent modulo the absorption and the duplication of a link by a box. This property of the semantics is absolutely essential to ensure correctness of the semantics with respect to the reduction since reduction of non immediate cuts since they involve absorptions and duplications.

Example 2 (η-equivalence of prooftrees). For instance the next two proofs have the same context semantics:

$$\dfrac{\dfrac{\overline{F, F^\perp} \qquad \overline{F, F^\perp}}{F, F^\perp \& F^\perp}\,\&(w)}{F \oplus G, F^\perp \& F^\perp}\,\oplus_0 \qquad \dfrac{\dfrac{\overline{F, F^\perp}}{F \oplus G, F^\perp}\,\oplus_0 \qquad \dfrac{\overline{F, F^\perp}}{F \oplus G, F^\perp}\,\oplus_0}{F \oplus G, F^\perp \& F^\perp}\,\&(w)$$

We note $\omega_i(w) = i$.

The semantics S of the two proofs above is defined by:

$$\begin{array}{ll} S(F \oplus G, g.s, \omega_0) = (F\&G, g.s) & S(F\&G, g.s, \omega_0) = (F \oplus G, g.s) \\ S(F \oplus G, g.s, \omega_1) = (F\&G, d.s) & S(F\&G, d.s, \omega_1) = (F \oplus G, g.s) \end{array}$$

Theorem 4 also means that context semantics characterize what a MALL proof is. Indeed, if Γ is sequent, W a set of eigenweights and $S \in \Gamma \times \mathcal{S} \times (W \to \mathcal{B}) \to \Gamma \times \mathcal{S}$ a finitely representable function, then either $\mathcal{R}(S)$ is a normalized proofnet π such that $\llbracket \pi \rrbracket = S$, or $\mathcal{R}(S)$ is undefined. In the second case, S is not the semantics of any proofnet: on the contrary there would exist a proofnet π' such that $\llbracket \pi' \rrbracket = S$ and by Theorem 5, $\pi' \to_{Cut+\eta+GC} \pi$.

Therefore, Theorems 4 and 5 relate a form of *full completeness* of the context semantics.

Theorems 4 and 5 can be summarized by the diagram:

$$\begin{tikzcd}
\pi_0 \arrow[r, "Cut"] \arrow[d] & \pi_1 \arrow[r] \arrow[d] & \pi_2 \arrow[d] \\
\langle \pi_0 \rangle \arrow[r, "Cut+\eta"] \arrow[dr] & \langle \pi_1 \rangle \arrow[r, "Cut+GC+\eta"] \arrow[d] & \langle \pi_2 \rangle \\
& \llbracket \pi_0 \rrbracket \arrow[ur, bend right, "\mathcal{R}"'] &
\end{tikzcd}$$

6 Related work

Among the works on proofnets for MALL, we can cite the paper by Girard [8]. He provides a syntax with weights, a sequentialization procedure and a cut-elimination procedure (restricted to so called *ready cuts*, i.e., cuts that are not in boxes). Tortora de Falco provides a more complete study of the reduction of the proofnets in [20]. His syntax involves boxes, but the problems he encounters are related to ours. He proves a restricted confluence property that should be extendable to our settings without any problem.

Following our work, we discovered that the notion of multiboxes appears in a manuscript of Tortora De Falco [19]. Our notion of generalized boxes is in the same spirit as his multiboxes; however, we have integrated a reduction-preserving semantics akin to Laurent [15], together with a more generalized notion of garbage that preserves the linearity of additive cut-elimination.

The reasons why we chose to design proofnets with bus notation was mainly that it encodes precisely (and locally) the context semantics, and it allows a fine study of the reduction process. This way, we could see exactly which step endangers the preservation of the semantics under reduction and postpone it. We hope to exploit this locality to say something interesting about optimal reduction in the context of the additives.

On the semantics point of view, our work is related to the Geometry of Interaction and to all its simplified versions [6, 10, 15]. The semantics exposed in this paper is quite close to the token machine of [15]. In this setting, Laurent proves essentially correctness of the semantics with respect to prooftree normalisation. Our proofnet syntax gives a sort of low-level implementation of this semantics. We extend the introduction of garbage (corresponding to $\flat$-rules in [15] and [6]) to the generalized &-connector, and hence to multidimensional garbage.

Among the proofs of full abstraction for MALL, we wish to mention the work of Abramsky and Melliès [2], based on concurrent games. We would like to know what, if anything, our constructions have to do with concurrent games.

7 Conclusion and future work

This work started with the search for a context semantics for the Multiplicative-Additive Linear Logic and for a convenient proofnet syntax for this part of Linear Logic as a first step before extending it to a larger fragment.

In proving a correspondence between normalization of sequents and proofnets, we were forced to slightly modify the rules for MALL, introducing at the same time garbage and parallelization in the syntax of proofs. Our presentation integrates various techniques (garbage, generalized boxes) and provides a complete understanding of the dynamic of MALL proofs.

In return, read-back involves the detection of garbage and parallelization. Read-back is not only a clean-up phase: it can also replace completely the reduction. The existence of read-back corresponds to a form of completeness of the semantics. The η-equivalence and the choice of a η-expanded form seem to be the price for this nice property.

Among the continuations of this work, the first one is its extension to full Linear Logic. The case of the units should not be too hard. The extension to the exponentials is probably more challenging since the rule ! act on a large bunch of formulas (by checking that all these formulas are of the form $?F$):

$$\frac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}\,!$$

This might be problematic, especially to get a local (optimal) reduction.

Another further work would be to determine to what extent our proofnet technology is expressive enough to encode languages for concurrency, like CSP, CCS, π-calculus, etc., following [3].

Last, the correctness of read back expresses that the context semantics enjoys some completeness property. The read-back algorithm could probably reformulated in the game semantics framework, which might be the starting point to some comparisons between the concrete insight given by the context semantics and some more theoretical games constructions [1, 2].

References

1. S. Abramsky and G. McCusker. Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions (extended abstract). In

Proceedings of 1996 Workshop on Linear Logic, volume 3 of *Electronic notes in Theoretical Computer Science*. Elsevier, 1996.

2. S. Abramsky and P.-A. Melliès. Concurrent games and full completeness. In *LICS'99*, pages 431–442. IEEE, July 1999.
3. G. Bellin and P. J. Scott. On the π-calculus and linear logic. *Theoretical Computer Science*, 135(1):11–65, Dec. 1994.
4. J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
5. J.-Y. Girard. Geometry of interaction I: Interpretation of system F. In *Logic Colloquium '88*, pages 221–260. North-Holland, 1989.
6. J.-Y. Girard. Geometry of interaction III: The general case. In *Advances in Linear Logic*, pages 329–389. Cambridge University Press, 1995. Proceedings of the 1993 Workshop on Linear Logic, Cornell Univesity, Ithaca.
7. J.-Y. Girard. Linear logic: its syntax and semantics. In *Advances in Linear Logic*, pages 1–42. Cambridge University Press, 1995. Proceedings of the 1993 Workshop on Linear Logic, Cornell Univesity, Ithaca.
8. J.-Y. Girard. Proof-nets: The parallel syntax for proof-theory. In *Logic and Algebra*. Marcel Dekker, 1996.
9. G. Gonthier, M. Abadi, and J.-J. Lévy. The geometry of optmnal lambda reduction. In *POPL'92*, pages 15–26, Albuquerque, Jan. 1992. ACM Press.
10. G. Gonthier, M. Abadi, and J.-J. Lévy. Linear logic without boxes. In *LICS'92*, pages 223–34. IEEE, Los Alamitos, 1992.
11. T. G. Griffin. The formulae-as-types notion of control. In *POPL'90*, pages 47–57. ACM Press, New York, 1990.
12. C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, Englewood Cliffs, NJ, 1985. & 0-13-153289-8.
13. Y. Lafont. From proof nets to interaction nets. In *Advances in Linear Logic*, pages 225–247. Cambridge University Press, 1995. Proceedings of the 1993 Workshop on Linear Logic, Cornell Univesity, Ithaca.
14. J. Lamping. An algorithm for optimal lambda-calculus reductions. In *POPL'90*, pages 16–30. ACM Press, Jan. 1990.
15. O. Laurent. A token machine for full geometry of interaction (extended abstract). In *TLCA'01*, volume 2044, pages 283–297. LNCS, Springer-Verlag, May 2001.
16. J. L. Lawall and H. G. Mairson. Sharing continuations: proofnets for languages with explicit control. In *ESOP'2000*, volume 1782. LNCS, Springer-Verlag, 2000.
17. R. Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, May 1999.
18. C. R. Murthy. Extracting constructive content from classical proofs. Technical Report TR90-1151, Cornell University, Computer Science Department, Aug. 1990.
19. L. Tortora de Falco. The additive multiboxes. *Annals of Pure and Applied Logic*. To appear.
20. L. Tortora de Falco. Additives of linear logic and normalization- part 1: a (restricted) church-rosser property. *Theoretical Computer Science*.