Linear Logic Without Boxes

Georges Gonthier[*] Martín Abadi[†] Jean-Jacques Lévy[*]

[*] INRIA
Domaine de Voluceau
Rocquencourt, B.P. 105
78153 Le Chesnay Cedex, France

[†] Digital Equipment Corporation
Systems Research Center
130 Lytton Avenue
Palo Alto, CA 94301, USA

Abstract

Girard's original definition of proof nets for linear logic involves boxes. The box is the unit for erasing and duplicating fragments of proof nets. It imposes synchronization, limits sharing, and impedes a completely local view of computation. Here we describe an implementation of proof nets without boxes. Proof nets are translated into graphs of the sort used in optimal λ-calculus implementations; computation is performed by simple graph rewriting. This graph implementation helps in understanding optimal reductions in the λ-calculus and in the various programming languages inspired by linear logic.

1 Beyond the λ-calculus

The λ-calculus is not entirely explicit about the operations of erasing and duplicating arguments. These operations are important both in the theory of the λ-calculus and in its implementations, yet they are typically treated somewhat informally, implicitly. The proof nets of linear logic [1] provide a refinement of the λ-calculus where these operations become explicit; they are even reflected in the type system for proof nets (that is, in linear logic). Abramsky, Wadler, and others have suggested that this new expressiveness makes linear logic a good basis for principled and useful improvements in functional-programming systems (e.g., [2, 3]).

In some sense, however, linear logic could go further. The usual formulation of proof nets involves boxes. The box is the unit for discarding and copying fragments of proof nets. It works as a synchronization mark. The disappearance, reproduction, opening, and movement of boxes remain global operations; full boxes are handled at once, not incrementally, so for example it is not possible to copy a box gradually, in little pieces. As Girard points out, boxes are a bridle to parallelism. They are also an obstacle to sharing: the box formalism does not support some sophisticated mechanisms for "partial sharing" of common subexpressions available in λ-calculus implementations such as Lamping's [4] and Kathail's [5]. These sharing mechanisms are essential for optimality in reductions, and we believe that they can be of practical value. Moreover, boxes complicate the proof theory of linear logic; with boxes, linear logic falls short of giving a fully local account of computation.

In this paper we describe a translation of proof nets into a system of sharing graphs. Proof-net reduction is simulated with graph rewriting. Sharing graphs are interaction nets, in the sense of Lafont [6]; hence rewriting is obviously Church-Rosser, and a naive implementation is straightforward. Everything in the graph system is entirely local. In particular, there are no boxes. Instead, brackets are included as nodes in the system, and they represent the boundaries of boxes. These brackets can propagate and interact with other nodes independently of one another, so boxes can disintegrate. Partial copying and partial sharing become possible.

The main sources of this work are Girard's geometry of interaction [7, 8] and Lamping's optimal implementation of the λ-calculus. The geometry of interaction provides a semantics for linear logic. The semantics does not require any global notion for interpreting boxes. At a more syntactic level, Lamping invented graphs similar to ours and used them for implementing the λ-calculus. The geometry of interaction provides a useful foundation for these graphs, leading to various enhancements, while the graphs are a concrete implementation of the geometry of interaction; this was the subject of a previous paper [9]. In this work, we exploit the graphs defined in [9] for evaluating proof nets instead of λ-terms.

The graph system described is of some help in understanding Lamping's work on the λ-calculus and our previous work. The extant and new encodings of the λ-calculus can be obtained by encoding the λ-calculus in linear logic and then translating linear logic into graphs. In turn, a further change of formal system illuminates the encoding of linear logic: a variant of

Girard's unified logic [10] admits a simpler, more regular treatment than linear logic.

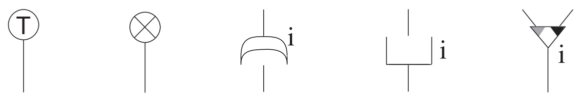
This graph system may also serve as a basis for efficient implementations of the various programming languages inspired by linear logic in recent years (e.g., [11, 12, 3]). In fact, work on the λ-calculus [13] suggests an optimality criterion for proof-net reduction; our system is optimal in this sense, and so one may even envision optimal implementations of those programming languages.

So far we have not succeeded in treating the full linear logic. We seem to have encountered new manifestations of the some of the same difficulties that have troubled Girard for some time. In particular, we cannot deal with the additive connectives (so far not covered by the geometry of interaction); also, the weakening rule is problematic in classical linear logic. For these reasons, we sometimes focus on the intuitionistic multiplicative-exponential fragments of logics. We hope that some of Girard's current progress on polarities [14] will help in extending this work to even richer logical systems. Some evidence in this direction is provided by a satisfying translation of polarities for unified logic.

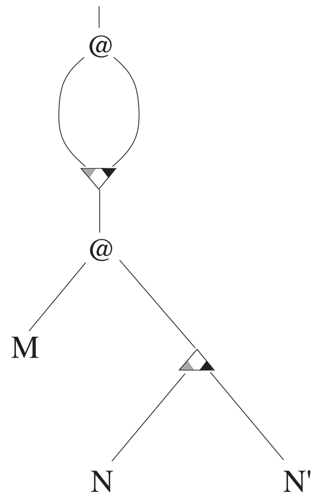
The next section is a review of sharing graphs and of the corresponding rewrite rules. Sections 3 and 4 then introduce the translation of multiplicative-exponential fragments of linear logic and of unified logic. Sections 5 and 6 discuss correctness and optimality. Finally, section 7 discusses the additive connectives and the weakening rule.

2 Sharing graphs (review)

We deal with undirected graphs built from the following nodes:



The fan nodes are the main nodes. Their function is to allow sharing. Two different parts of a proof net may share a common subproof; the two parts are connected to the top ports of a fan node and the common subproof is connected to the bottom port. The common subproof may include an inverted fan node, for "unsharing"; intuitively, then, the common subproof has a hole that is filled differently in different versions. It is easiest to give an example of unsharing with λ-terms: writing @ for application, the term $(MN)(MN')$ can be depicted



Here $(M\ \_)$ is shared by the function and the argument of the top application; the hole is filled with N for the function and with N' for the argument.

Root nodes are attached at the important ends of graphs (for example, in λ-calculus encodings, at the ends that represent results and free variables). Void nodes are attached at the unimportant ends, those where information is discarded (so the value of a discarded subexpression can be fed to a void node). Croissants and brackets are all bracketing constructs, and they serve to simulate the edges of boxes.

The easiest way to understand the nodes is as context transformers. A context is a data structure that serves in navigating a graph, so that matching edges are taken out of opposing fan nodes (so that sharing and unsharing are performed correctly). For example, traversing a fan node from top to bottom adds to the context a record of which port was used for entry, while a traversal from bottom to top removes such a record to decide on the port of exit.

Several of the nodes are indexed with integers. The integers indicate the "depth" at which context operations take place. An alternative presentation of our system, given in [9], does not rely on indices but decomposes edges into bundles of parallel edges; this presentation is somewhat more primitive, but we use the one with indices for the sake of brevity.

The rules of rewriting are in Figure 1, where it is assumed that $0 \leq \mathtt{i} < \mathtt{j}$.

It is clear from these rules that each node has a preferred port of interaction. When two nodes meet on an edge facing one another through their ports of interaction, something may happen—or there may be a deadlock. Thus, our graphs are interaction nets in the sense of Lafont. Reduction in interaction nets is local and automatically Church-Rosser.

Additional garbage-collection rules are natural, and handy for eliminating useless parts of graphs. For example, we may add:

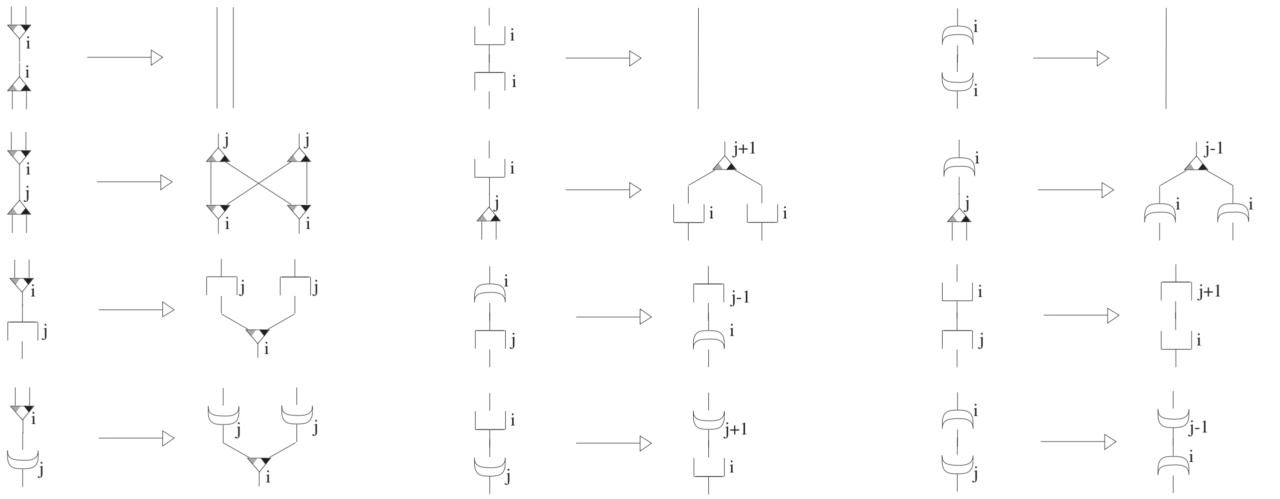


Figure 1: **Reduction rules**

We do not include these rules in the core system, because they do not remove all garbage and because they are not essential for correctness.

3 Implementing linear logic

Now we have the machinery necessary for defining a translation S from proofs in classical multiplicative-exponential linear logic (MELL) into graphs in our formalism. The logic treated has one-sided sequents, and includes the multiplicative connectives $\otimes$ and $⅋$, and the exponential connectives ! and ?. As usual, linear negation $(\ )^\perp$ is left in the metalanguage, by viewing it as primitive on atomic proposition symbols and using De Morgan laws.

We define the translation by induction on the length of proofs, rule by rule (omitting the technical details connected with the exchange rule). The translation involves an auxiliary function T, which yields graphs with unterminated edges. The translation $S(\pi)$ of a complete proof π is obtained by adding root nodes labelled with the conclusion formulas of π to the dangling edges of $T(\pi)$.

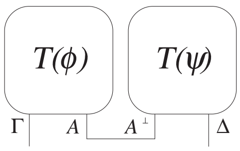
Axioms and the cut rule are somewhat dual. Their effect is to introduce links between complementary formulas, exactly as in proof nets.

- The axiom $\vdash A, A^\perp$ is translated:

- Suppose the proof π is obtained by applying the cut rule

$$\frac{\vdash A, \Gamma \quad \vdash A^\perp, \Delta}{\vdash \Gamma, \Delta}$$

to proofs ϕ and ψ, with conclusions $\vdash A, \Gamma$ and $\vdash A^\perp, \Delta$, respectively. Then $T(\pi)$ is



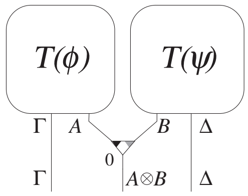
The propositions on dangling edges only serve to indicate the corresponding conclusion formulas of π, ϕ, or ψ—they are not part of the graph.

Both the rules for $\otimes$ and $⅋$ introduce fan nodes.

- Suppose the proof π is obtained by applying the $\otimes$ rule

$$\frac{\vdash A, \Gamma \quad \vdash B, \Delta}{\vdash A \otimes B, \Gamma, \Delta}$$

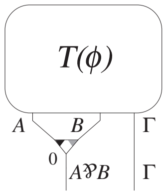
to proofs ϕ and ψ, with conclusions $\vdash A, \Gamma$ and $\vdash B, \Delta$, respectively. Then $T(\pi)$ is



- Suppose the proof π is obtained by applying the $⅋$ rule

$$\frac{\vdash A, B, \Gamma}{\vdash A ⅋ B, \Gamma}$$

to a proof ϕ with conclusion $\vdash A, B, \Gamma$. Then $T(\pi)$ is



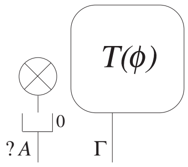
So both $\otimes$ and $\parr$ become fan nodes, but typically the fan nodes obtained are going to face in opposite directions. More precisely, in the translation of a correct proof, the fan nodes for $\otimes$ and $\parr$ connectives are typically going to meet, face to face, through the edges generated by the cut rule, and are going to cancel.

The rules for ? are weakening, dereliction, and contraction.

- Suppose the proof π is obtained by applying the weakening rule

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

to a proof ϕ with conclusion $\vdash \Gamma$. Then $T(\pi)$ is

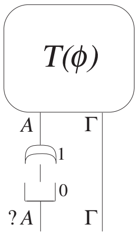


Thus, a void node serves to terminate a link to a conclusion $?A$ that gets discarded as we go up the proof.

- Suppose the proof π is obtained by applying the dereliction rule

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

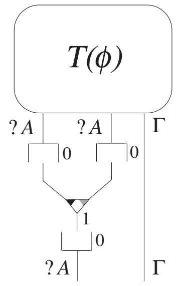
to a proof ϕ with conclusion $\vdash A, \Gamma$. Then $T(\pi)$ is



- Suppose the proof π is obtained by applying the contraction rule

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

to a proof ϕ with conclusion $\vdash ?A, ?A, \Gamma$. Then $T(\pi)$ is

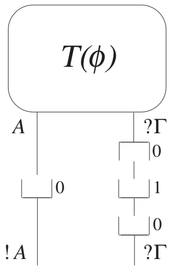


As in the rule for $\parr$, a fan node gives sharing.

Finally, we have the rule for !. Suppose the proof π is obtained by applying the ! rule

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

to a proof ϕ with conclusion $\vdash A, ?\Gamma$. Then $T(\pi)$ is



In proof nets, a box would be drawn around π. Here the box is simulated with a line of brackets.

As in the λ-calculus, computation in proof nets is cut elimination. Cut elimination involves the global manipulation of boxes. In the graph formalism, computation is performed with local rewriting steps. We explain the correspondence between these steps and the exponential cut-elimination steps in proof nets. First note that the edge implementing an exponential cut always connects a pair of facing index-0 brackets, which reduces to a single edge e. There are four cases:

1. The cut is between $\vdash !A, ?\Gamma$, obtained by the ! rule, and $\vdash ?(A^\perp), \Delta$, obtained by the weakening rule. In the proof-net formalism, the box with the proof of $\vdash !A, ?\Gamma$ is discarded all at once when the cut is eliminated. In our graphs, the edge e connects the box contents to a void node. With the garbage-collection rule this void node may propagate, and the proof of $\vdash !A, ?\Gamma$ is discarded gradually.

2. The cut is between $\vdash !A, ?\Gamma$, obtained by the ! rule, and $\vdash ?(A^\perp), \Delta$, obtained by the dereliction rule. In the proof-net formalism, the box around the

proof of $\vdash !A, ?\Gamma$ evaporates when this cut is eliminated and a cut appears between $\vdash A, ?\Gamma$ and $\vdash A^\perp, \Delta$. In our graphs, e connects the box contents to an index-1 croissant. This croissant may propagate over the proof of $\vdash !A, ?\Gamma$; this is the equivalent of opening a box gradually.

3. The cut is between $\vdash !A, ?\Gamma$, obtained by the ! rule, and $\vdash ?(A^\perp), \Delta$, obtained by the contraction rule. In the proof-net formalism, the box with the proof of $\vdash !A, ?\Gamma$ is copied all at once when this cut is eliminated; the result is two cuts with $\vdash ?(A^\perp), ?(A^\perp), \Delta$, one for each $?(A^\perp)$. In our graphs, e connects the box contents to an index-1 fan node. This fan node may propagate over the proof of $\vdash !A, ?\Gamma$, which is then duplicated node by node. The two index-0 brackets above the fan become the new main doors of the box copies.

4. The cut is between $\vdash !A, ?\Gamma$, obtained by the ! rule, and $\vdash ?(A^\perp), \Delta$, obtained also by the ! rule. In the proof-net formalism, the box with the proof of $\vdash !A, ?\Gamma$ moves to the inside of the box with the proof of $\vdash A^\perp, \Delta$ when this cut is eliminated. In our graphs, e connects the box contents to an index-1 bracket. This bracket may propagate over the proof of $\vdash !A, ?\Gamma$, changing indices in other nodes; this is the equivalent of lifting a box inside another box a node at a time. The remaining index-0 bracket becomes the new door of the lifted box.

Our translation, then, gives a local implementation of linear-logic proofs. It also clarifies and systematizes some aspects of Lamping's implementation of the λ-calculus and ours. Those implementations can be seen as resulting from encoding the λ-calculus in linear logic and then translating linear logic into graphs. They rely on the use of a recursive type D defined by $D = !(D \multimap D)$. However, the more traditional equation $D = (!D) \multimap D$ can be used instead, and another implementation of the λ-calculus results. This implementation gives call-by-name. It seems to be less surprising for orthodox logicians.

4 Implementing unified logic

Girard introduced unified logic as a synthesis of classical, intuitionistic, and linear logics. Each of these logics appears as a fragment of unified logic. The synthesis is achieved by distinguishing zones in sequents. Each zone is either classical or linear; the zones differ in what structural rules apply for them. Roughly, some of the exponential connectives of linear logic disappear in favor of structural information: formulas in the classical zones correspond to formulas with an exponential in linear logic, and are managed accordingly (so for example they can be duplicated).

Here we present a "lightweight" variant LW of Girard's unified logic LU and describe its implementation. One advantage of LW with respect to MELL is that it admits a simpler treatment of proof nets, with a more abstract approach to structural rules. This simplicity also eases the translation into graphs, and the related statements and proofs of correctness and optimality results.

4.1 LW and its implementation

In LW, sequents are not broken into zones, but some of the formulas in them are marked with a (?). These are the formulas that are managed classically. In the description below, Γ and Δ are sequences of formulas, possibly marked with (?); A and B are linear formulas (with no (?)); C and D are formulas, possibly marked with (?); and $(?)\Gamma$ is a sequence of formulas marked with (?).

There are three groups of rules for LW: identity rules, logical rules, and structural rules. We present them and at the same time give their graph implementation, which is rather straightforward. The notation is as in section 3; in particular, in all cases, π is a proof obtained by following a proof ϕ with an application of the rule under consideration; we also reuse the notations S and T.

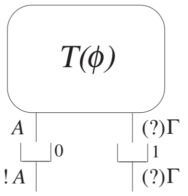
The identity rules are the usual axiom $\vdash A, A^\perp$ and the cut rule. Their implementation is as in MELL.

The logical rules are those for $\otimes$ and $⅋$, as in MELL, and two new rules for the exponentials:

- ! is introduced with

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

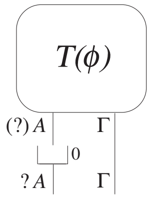
and $T(\pi)$ is



- ? is introduced with

$$\frac{\vdash (?)A, \Gamma}{\vdash ?A, \Gamma}$$

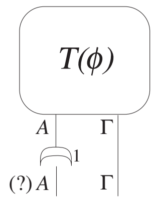
and $T(\pi)$ is



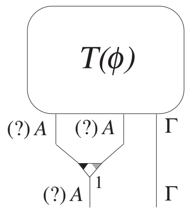
Thus, each of the logical rules is implemented by adding a node with index 0: a fan for the multiplicative connectives and a bracket for the exponential connectives. In addition, the introduction rule for ! involves some nodes with index 1; such nodes are typical in the treatment of structural rules, which we give next.

There is an uninteresting permutation rule, and three more structural rules for classical formulas:

- The rule
$$\frac{\vdash A, \Gamma}{(?)A, \Gamma}$$
is similar to the dereliction rule, and $T(\pi)$ is



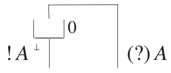
- The rule
$$\frac{\vdash (?)A, (?)A, \Gamma}{\vdash (?)A, \Gamma}$$
is similar to the contraction rule, and $T(\pi)$ is



- The rule
$$\frac{\vdash \Gamma}{\vdash (?)A, \Gamma}$$
is similar to the weakening rule, and has an analogous implementation.

For the sake of cut elimination, Girard adds a derivable rule for exponential cuts, which we would rephrase:
$$\frac{\vdash A, (?)\Gamma \quad \vdash (?)A^{\perp}, \Delta}{\vdash (?)\Gamma, \Delta}$$
We do not have a need for this rule.

On the other hand, it is useful to distinguish the following derivable axiom in the implementation:
$$\vdash (?)A, !(A^{\perp})$$
The sequent corresponds to the empty box in proof nets. Its implementation is:



(This implementation is somewhat optimized; a direct translation of an empty box from linear logic involves additional nodes.)

With this axiom, it is possible to show that LW is equivalent to MELL. The translation from LW to MELL replaces (?) with ?, and the other translation is the identity; both translations preserve provability. Dereliction and weakening in MELL are simulated in two steps in LW: first the corresponding structural rule introduces a $(?)A$ formula, then the ? introduction turns the (?) into a ?. Contraction and ! introduction require three steps: first a sequence of cuts with the empty-box axiom to change ?'s into (?)'s, then the corresponding LW rule, and finally ? introductions.

Now we have a two-phase implementation of MELL, via LW. This indirect procedure seems clearer than the direct one of section 3. There, some of the rules are implemented by adding many nodes, which here are accounted for one by one. Moreover, the graph implementations of linear-logic proofs in normal form are not always in normal form. This phenomenon is explained by looking at the translation from linear logic to unified logic, which can introduce cuts.

4.2 Lightweight proof nets

It is useful to introduce also a proof-net notation for LW. The proof nets of LW are very similar to those of linear logic, but more abstract. For the sake of brevity, we assume that the reader is familiar with Girard's definition of proof nets, and focus on the novelties of LW proof nets.

Like MELL proof nets, LW proof nets consist of a set of occurrences of MELL formulas connected by a set of *links*. Each link L has a set of *premises* and a set of *conclusions*; a premise or conclusion of a link is said to be *adjacent* to the link. In the LW net that denotes an LW proof P, there is a link for every logical or identity rule in P, *and* a link for each LW formula in the conclusion of P. There are no links for structural rules, as these affect only the arrangement of conclusions.

Links for identity rules (axioms $\vdash A, A^{\perp}$ and cuts) and multiplicative rules are just the same as for MELL. Conclusion links are new to LW nets. There are different kinds of conclusion links. The conclusion link for a linear (MELL) conclusion formula A has no conclusions and a single premise which must be an occurrence of A. The conclusion link for a classical conclusion formula $(?)A$ has no conclusions but can have an arbitrary number of premises A, possibly 0; this abstracts away all structural rules applied to that conclusion. (In MELL nets, only the linear case occurs, so a conclusion link can be identified with its only premise; but this identification does not work for classical conclusions. The LW approach is rather uniform, in that every formula is conclusion of exactly one link, and premise of exactly one link.)

The net implementation of the exponential rules in LW is quite different from that for MELL. The ? link has

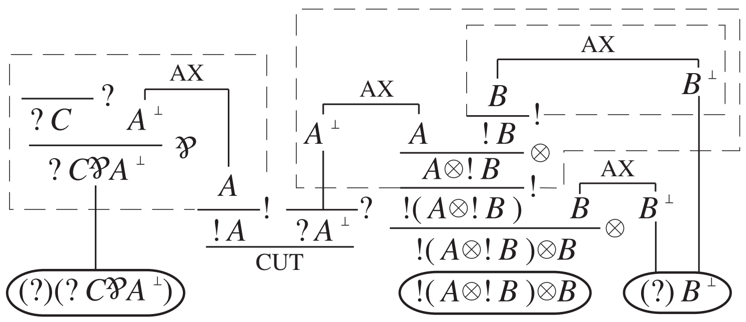


Figure 2: **An LW proof net**

a single conclusion conclusion $?A$, but can have any number of premises A; it abstracts over all previous structural rules, just as the (?) conclusion link. The ! link has a single conclusion $!A$ and a single premise A. In addition, for every ! link L there is a set of links and formulas called the *!-box contents of L*, or !-box for short. In contrast, in MELL the ! link has several premises and conclusions, one of them is distinguished as the "main door," and the box contents is defined by connectivity from the premises.

The !-box of the link corresponding to a ! introduction contains exactly the links corresponding to identity or logical rules above the rule application, and their adjacent formulas. Also, because of side condition of the ! introduction rule, only n-ary links ((?) conclusion or ? links) may reach inside a !-box for some of their premises.

Rather than draw LW nets for each of the cases of this definition, we give as an example in Figure 2 the net for a proof ending with the cut-rule

$$\frac{\vdash (?)(C⅋A^\perp), !A \quad \vdash ?A^\perp, !(A\otimes!B)\otimes B, (?)B^\perp}{\vdash (?)(C⅋A^\perp), !(A\otimes!B)\otimes B, (?)B^\perp}$$

Cut elimination for LW nets is simpler than usual since there is only one exponential cut-elimination rule. The rule simultaneously opens the !-box and copies its contents at any number of locations, possibly inside other boxes (see Figure 3, ignoring the labels).

LW net reduction can be simulated in MELL nets by lumping together exponential reductions, so that entire ? link trees are handled at once. Conversely, MELL net reduction can be simulated in LW nets, but only up to exponential cut elimination.

5 Correctness

In proving the correctness of the graph implementations, we face some of the same difficulties encountered in connecting linear logic with the geometry of interaction. As usual, it is not too hard to obtain a correctness theorem for "observables." Roughly, this theorem says that if a proof reduces to the representation of a boolean value then its translation reduces to a representation of the same value.

In some situations, it is important to have correctness criteria and theorems for graphs other than observables. For example, optimality was a central concern in our work on the λ-calculus, and optimality can be discussed only by considering intermediate results which are not observable. Unfortunately, it is hard to recover an appropriate proof from an arbitrary result of reducing the translation of a proof, in general. As in the λ-calculus, part of the problem is that copying and lifting may be done in several different but equivalent ways, so the corresponding context operations can be permuted; and part of the problem is that graph reductions do not include garbage collection, and so large portions of graphs may be dead (correspond to erased subnets or subterms).

The key to the correspondence between proof nets and sharing graphs is the semantics of graphs as context transformers. This context semantics keeps track of the subtle copying information needed to unwind the sharing in graphs. In order to "match" a net and a graph, we combine this information with an embedding of the net in the graph. The matching is established by the initial translation into sharing graphs, and, as we show, both net reduction and graph reduction preserve it.

In order to derive a procedure for reading back a proof net from a sharing graph, we need to find conditions under which the matching relation is injective. We characterize the live (non-garbage) parts of proof nets and find that injectivity holds on live parts. Finally, we obtain a full correctness theorem for ILW, an intuitionistic fragment of LW for which we can define a sound garbage-collection procedure.

5.1 The context semantics

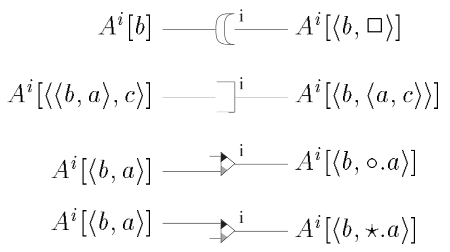
The basic semantics of sharing graphs is given by paths labelled with *context marks*, as in [9]. As the labelling rules there were explained for the "bus" presentation of the graphs, we reformulate them in terms of indexed nodes.

Context marks (contexts for short) are partial ordered binary trees with variables:

- $\Box$ is a context. It represents a tree leaf.
- Any v in a set $\mathcal{V}$ of variables is a context. It is a variable leaf.
- If a is a context then so are $\circ.a$ and $\star.a$. These terms represent the trees obtained by taking a node and putting a under it, either to the left or to the right. (The notations $\circ$ and $\star$ come from Lamping's work, where they are used instead of our grey and black marks, respectively.)
- If a and b are contexts then so is $\langle b, a\rangle$. This term represents the cons of b and a.

All the contexts c that we use to label paths have only binary nodes on the left branch, so they are of the shape $c = \langle \ldots \langle \Box, a_{n-1}\rangle, \ldots, a_0\rangle$. We say that n is the *width* of the context c. Now in the translation from logics to sharing graphs we assign a width to each edge, equal to 1 + the box depth of the edge if the edge bears a linear type in the translation, and 2 + the box depth if the edge bears a classical (?) type. (Here, by "box depth" we refer to the number of ! introduction rules in the sequent proof below the bottom formula of the edge.)

A *consistent* context-labelled path in a graph G is an undirected path in G where each edge is labelled with a context of same width, such that any consecutive pair of edges satisfies one of the following labelling constraints below. In this table $A^i[b]$ denotes a context $\langle \ldots \langle b, a_{i-1}\rangle, \ldots, a_0\rangle$ with b at "depth" i.



Among other things, these constraints forbid turnarounds in a consistent path.

Note that the constraints are consistent with the edge widths: if the left-hand context has width w, then the right hand context has width $w+1$ for a croissant, $w-1$ for a bracket, and w for a fan node. Conversely, these rules are enough to reconstruct the edge widths of the graph, given that the width of the edge to a root labelled with a linear type is 1, and the width of edges to a root with a classical type is 2. Moreover, the same rules give a unique and consistent width assignment for all reducts in Figure 1, so we can assume all edges in our graphs have widths satisfying these rules.

Closer inspection shows that the rules in Figure 1 preserve not only width consistency but also all consistent labelled paths:

Theorem 1 *If σ is a consistent labelled path in G and $G \twoheadrightarrow G'$ with a redex not involving the endpoints of σ, then there is a unique path σ' in G' identical to σ except for the redex nodes and the labelling of the redex edge.*

A special case is when σ is a root-to-root path: thus, the symmetric one-to-one relation between (root,context) pairs induced by such paths is preserved by reduction. This gives rise to the "context semantics" of [9].

5.2 Matching nets with graphs

As proof of correctness for the graph implementation of LW, we define a notion of matching between proof nets and sharing graphs, consistent with the LW translations and with reduction. In some sense, this is easier than for the λ-calculus because we seek to match two differents sorts of graphs, not a term and a graph.

The LW-to-graph translation gives the first component of this matching: conclusion links of a net correspond to roots of a graph, and logical links to index-0 node. Hence we define a *mapping* ϕ from an LW net N to a sharing graph G to be a function from non-identity links of N to nodes in G that sends

- conclusions of N of LW type C onto roots of G labelled with C,
- multiplicative links to index-0 fans of G, and
- exponential links to index-0 brackets of G.

For each non-identity link L of N, such a ϕ induces a *local mapping* ϕ_L from formulas adjacent to L to edges incident to $\phi(L)$, as follows:

- If L is a multiplicative link, ϕ_L sends the conclusion (resp., left premise, right premise) of L to the edge incident to the unmarked (resp., grey, black) point of the fan $\phi(L)$.
- If L is an exponential link, ϕ_L sends the conclusion (resp., any premise) of L to the edge incident to the closed (resp., open) side of the bracket $\phi(L)$.
- If L is a conclusion link, ϕ_L sends any premise of L to the edge incident to the root $\phi(L)$.

Obviously, for a net to match a graph the mapping should also satisfy some connectivity properties. Namely, given two formulas adjacent to non-identity links and connected by a chain of identity links, it should be possible to connect the corresponding edges by a path in the graph that traverses only positive-index nodes.

This is much too weak a constraint, since the large amount of sharing and unsharing that goes on in graphs allows paths to merge and cross arbitrarily. We want paths to respect sharing and unsharing, so we restrict ourselves to consistently labelled paths. We can restrict the path labels to marks of the form $\langle b, \Box\rangle$, since the paths cross only positive-index nodes. Such paths are entirely determined by the left subtrees of the context marks labelling the end edges.

Thus the matching between an LW net and a sharing graph should provide those context marks. We define a *marking* μ of an LW net N to be a family of local marking functions μ_L, one for each non-identity link L of N; each μ_L sends formulas adjacent to L to context marks, subject to the restrictions detailed below. The local markings induce a marking of the links themselves, as follows:

- If L is a conclusion, then $\mu(L) = \Box$.
- If L is a logical link, and A its sole conclusion, then $\mu(L) = \mu_L(A)$.

The link markings completely characterize the sharing in the graph: in a matching, every link L should have a different $(\phi(L), \mu(L))$ pair.

The markings should be consistent with the type of nodes the links are mapped to, hence the following restrictions:

- If A is a premise of a linear conclusion or multiplicative link L, then $\mu_L(A) = \mu(L)$.
- If A is a premise of a (?) conclusion or exponential link L, then there is a context $\mu^L(A)$ such that $\mu_L(A) = \langle\mu(L), \mu^L(A)\rangle$.

The residual markings μ^L of n-ary nodes encode the arity of the node. Different premises can be shared, but not identified in the graph, so the μ^L should be injective. Dually, for a ! link, the unique μ^L should be compatible with the arity of any opposing ? link:

- If A and A' are different premises of an n-ary link, then $\mu^L(A)$ and $\mu^L(A')$ are non-unifiable.
- If A is the sole premise of a ! link L, then $\mu^L(A)$ is a context variable.

Finally, the $\mu^L(A)$ variable of a ! link should trace the contents of its box, so that μ contains all the box information of N:

- A non-identity link L' is in the !-box of L iff $\mu^L(A)$ appears in $\mu(L')$, and a formula A' adjacent to L' is in the !-box of L iff $\mu^L(A)$ appears in $\mu_{L'}(A')$.

Now we can define matchings precisely:

Definition 1 *A* matching *from an* LW *net N to a sharing graph G is a pair (ϕ, μ) of a mapping $\phi : N \to G$ and a marking μ of N, such that for any non-identity links L and L' of N:*

- *If $L \neq L'$ and $\phi(L) = \phi(L')$ then $\mu(L)$ and $\mu(L')$ are non-unifiable.*
- *If formulas A and A' respectively adjacent to L and L' are connected by a chain of identity links, and $A \neq A'$ if $L = L'$, then there is a consistent labelled path in G from $\phi(L)$ to $\phi(L')$, starting with $(\phi_L(A), \langle\mu_L(A), \Box\rangle)$, ending with $(\phi_{L'}(A'), \langle\mu_{L'}(A'), \Box\rangle)$, and traversing only positive-index nodes.*

We say net N matches *G when there exists a matching from N to G.*

The translation from sequent-calculus proofs to sharing graphs establishes a matching:

Proposition 1 *Let π be a proof in* LW*. Then its proof net translation N matches its sharing graph implementation G.*

The next two propositions establish the links between matching, graph reduction, and reduction in proof nets. Combined with the injectivity results in the next subsection, they give the readback and correctness theorems for the graph implementation. Combined with the redex labelling results in the next section, they imply that "matches" is a bisimulation between graph reduction and parallel labelled reduction in LW nets.

Proposition 2 *Let N be an* LW *net that matches a graph G, and assume $G \multimap\!\!\!\triangleright G'$ by one of the rules in Figure 1, then N reduces to a net N' that matches G'.*

(i) If $G \multimap\!\!\!\triangleright G'$ is not a index-0 fan elimination then $N \multimap\!\!\!\triangleright^* N'$ need not contain exponential cut eliminations.

(ii) If $G \multimap\!\!\!\triangleright G'$ is not a index-0 bracket elimination then $N \multimap\!\!\!\triangleright^* N'$ need not contain multiplicative cut eliminations.

(iii) If $G \multimap\!\!\!\triangleright G'$ is neither of these then we can take $N' = N$.

Proposition 3 *If the net N matches G, and $N \multimap\!\!\!\triangleright N'$, then G reduces to a graph G' and N' reduces to a net N'' that matches G'. Furthermore if $N \multimap\!\!\!\triangleright N'$ is a multiplicative cut elimination, then $G \multimap\!\!\!\triangleright^* G'$ need not involve any index-0 bracket nodes and only two index-0 fans, ending with their elimination, dually for exponential cuts, and we can take $G = G'$ for axiom cuts.*

5.3 Readback and garbage collection

The "matches" relation is actually almost a bijection between marked nets and graphs. It fails to be a function (i.e., a net can match several graphs) because graphs cannot emulate the box erasures performed by exponential cut elimination, since there are no garbage-collection rules in Figure 1. Hence, when a marked net matches a graph, there can be an arbitrary amount of garbage in the net that lies outside the codomain of the mapping.

In [9] this problem was solved for the λ-calculus by determining which part of a net could be accessed by a λ-term mapping. Unfortunately, the problem is much worse for LW because the garbage in a graph can look just like part of a different net, so the "matches" relation is not injective. Injectivity can hold only for the subgraph of a net N that cannot possibly map to garbage, which we define as the *live* part of N below.

Definition 2 *The* live part *of an* LW *net N is the smallest subset N' of N such that:*

- *All conclusion links of N belong to N'.*
- *If any A adjacent to a link L belongs to N', so do L and all conclusions of L.*
- *If a non-n-ary link L belongs to N, then so do all premises of L.*

We call the complement of the live part the garbage part. *We say a link or formula of N is live (resp., garbage) when it belongs to the live (resp., garbage) part; a net is live when it equals its live part.*

The garbage part of a net is clearly stable by cut elimination: if a link becomes garbage, then it remains garbage after cut elimination.

Proposition 4 *If two* LW *nets N and N' match the same sharing graph G, and both N and N' are cut free, then the live parts of N and N' are identical, and the respective matchings coincide on the live parts up to a renaming of the box variables.*

Proof: Let (ϕ, μ) and (ϕ', μ') be the mappings from N and N' to G, respectively. Consider the maximal subset M of live non-axiom links of N such that there is a function ψ from M to N' making the matchings coincide, up to a renaming of box variables. M is unique and ψ is one-to-one because of the non-unifiability condition on matchings. Since all conclusions have the same mark $\square$, both ϕ and ϕ' are bijections from conclusions to roots, so M contains all conclusion links of N. We show that M is downwards closed. Let A be the premise of a link L that is the conclusion of a logical link K in M, then the conclusion A' of $K' = \psi(K)$ is the premise of a link L' in N'. Both N and N' match G, so there must be consistent labelled paths from $\phi(K) = \phi'(K')$ to $\phi(L)$ and $\phi(L')$. By definition of M both paths start with the same labelled edge, so the constraints on paths force them to end with the same labelled edge, hence the matchings coincide on L and L' and $\psi(L) = L'$. Since both nets are cut-free this forces ψ to respect link types, and especially to map non-n-ary links to non-n-ary links. A similar argument on labelled paths shows that if a premise of a non-n-ary link L is connected, possibly through an axiom, to a formula adjacent to some non-axiom link L', then $L' \in M$ and the congruent premise of $\psi(L)$ is similarly connected to a formula similarly adjacent to $\psi(L')$. It follows that M contains all the live non-axiom links in N, and that ψ can be extended to an isomorphism $\hat{\psi}$ from the live part of N to the live part of N', that makes the matchings coincide. $\square$

Proposition 4 defines a readback for the live part M of a normal proof net. The subset $\mathcal{C}_{\text{acc}}$ of the context semantics generated by paths inside $\phi(M)$ can be constructed from the topology of M plus the residual markings μ^L; conversely $\mathcal{C}_{\text{acc}}$ determines both of these. (By introducing *shunts* and *access paths* as in [9], $\mathcal{C}_{\text{acc}}$ could also be characterized directly, and then global correctness results follow.)

In general, it is impossible to read back nets with cuts, because arbitrary types may be hidden by cuts with a proof containing a weakening, and the definition of the live part depends on the distinction between ! and ? types. So, in order to define a general readback procedure, we consider *typed sharing graphs*, which are sharing graphs where every index-0 node is labelled with a linear type. Graph reduction ignores the labels, but preserves them when a positive-index node crosses an index-0 node. A net matches a typed sharing graph only if the mapping sends a logical link with conclusion A to a node labelled A. Propositions 1–3 go through unchanged for typed graphs, and we have:

Proposition 5 *If two* LW *nets N and N' match the same typed sharing graph G, and both N and N' are axiom-cut free, then the live parts of N and N' are identical, and the respective matchings coincide on the live parts up to a renaming of the box variables.*

Proposition 5 provides us with a natural definition for readbacks; and its proof gives a procedure for constructing them: we propagate breadth-first known contexts from the conclusions up, generating links as we go, stopping at the closed side of ? brackets and upon reaching a node with the same context twice.

Definition 3 *The readback $R(G)$ of a typed sharing graph G is, when it exists, the unique live and axiom-cut free* LW *net N that matches G.*

The obvious way to derive a correctness theorem from the results above is to define an operation $GC(N)$

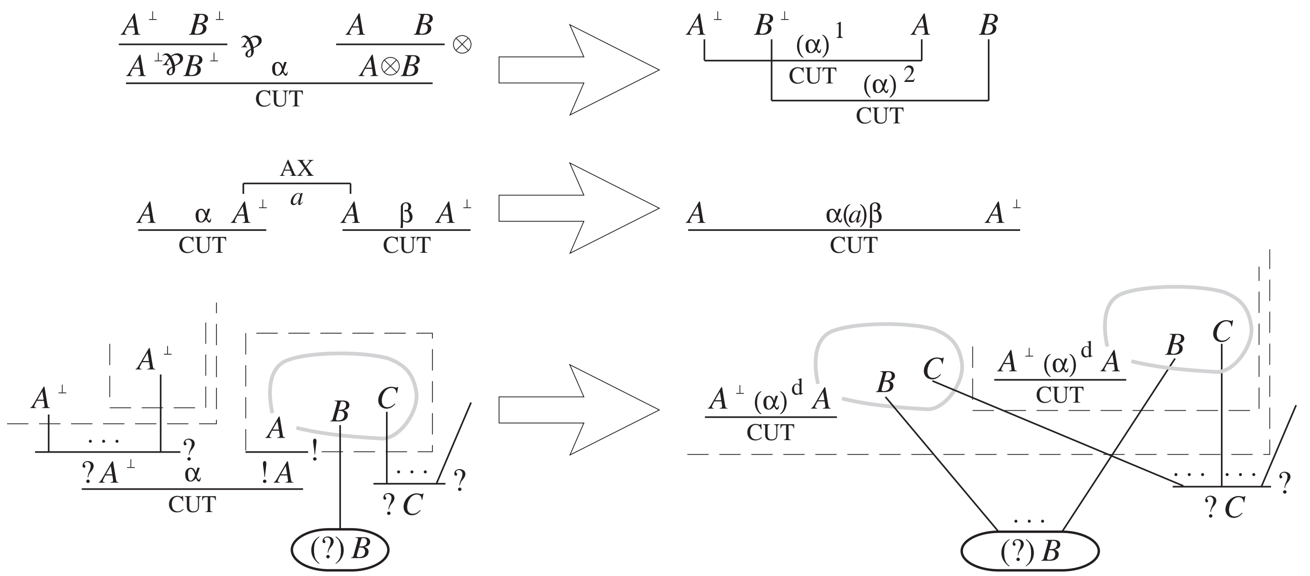


Figure 3: **Labelled LW cut elimination**

that restricts a net N to its live part, by removing all garbage from the premises of live n-ary links and of boxes of live !-links. Unfortunately, GC is an unsound operation for full LW, e.g., when all the conclusions of N are classical, $GC(N)$ has only empty conclusion links. However, GC is sound for the intuitionistic fragment ILW of LW (the analogue of ILL, the intuitionistic fragment of linear logic [2]).

Theorem 2 *Let N and G be the proof net and sharing graph translations of an* ILW *proof, respectively. If $G \twoheadrightarrow^* G'$, then $R(G')$ exists and N reduces by cut elimination to some N' such that $GC(N') = R(G')$. Furthermore, one can take $N' = N$ if $G' = G$, and N'' to be the normal form of N if G' is the normal form of G.*

It can be shown that all nets obtained as translation of typed λ-terms are live, hence that in this case the provisions for GC in Theorem 2 above can be omitted. This fact was used implicitly in the treatment of the pure λ-calculus in [9], and it also extends to system F proofs, the original domain for Girard's geometry of interaction [7].

6 Optimality

In the λ-calculus, there is a formal criterion for saying whether a reduction strategy is optimal, in the sense that it performs as few β steps as possible [13]. Optimality can also be defined for other formal systems, such as proof nets, where it is about as hard to attain.

As for the λ-calculus, a labelled calculus with labelled proof nets may be defined. In this calculus, it greatly simplifies matters to put labels only on identity links, and not on formulas. In order to capture all the sharing power of a given proof net, it is necessary to start from a proof net where every formula is either premise or conclusion of an identity link. Adding intermediate axiom cuts when necessary achieves this. Similarly, we take only a weak version of the axiom cut; this does not change the behavior of reduction, since asymmetric axiom cut can always be postponed. The reductions rules for labelled nets are defined by Figure 3.

Clearly, residuals of cut links keep the same label. Hence, a theory of duplication and families of cut links may be developed as for the λ-calculus [13]. It is easy to show that two cut links have a same label iff they are in a same family. For sharing graphs, it is proved as in [9] that different redex edges never bear the same label; this means that sharing graphs capture exactly the family notion for cut links.

A *shared reduction* step in a labelled net consists in reducing simultaneously all the cuts in a family. Sharing graphs thus implement shared reduction steps with a single index-0 node elimination.

A redex is *needed* if it is accessible from a root of the graph through a consistent access path. (Access paths are defined much as in [9].) A call-by-need reduction strategy in graphs will contract only needed redexes.

Theorem 3 *Shared call-by-need reductions are optimal for computing the live part of normal forms.*

7 The additives and weakening

We close with a brief discussion of the difficulties remaining in treating the full linear logic.

Our translation excludes the additive connectives of linear logic. In the standard formulation of proof nets, these connectives also require boxes. We have not succeeded in removing these boxes, in part because of the well-known problems in breaking symmetries during cut elimination with additives. Girard's polarities are a promising way to break this symmetry, and one may hope to use them for extending our translation.

Some of the results of section 5 not only exclude the additive connectives but also restrict the uses of the weakening rule. They apply only to intuitionistic logics, where the weakening rule is least devastating. It might be possible to recover the full power of classical logics by encodings.

Acknowledgements

We are grateful to Pierre-Louis Curien, Jean-Yves Girard, Yves Lafont, and John Lamping for fruitful discussions.

References

[1] J.-Y. Girard, "Linear logic," *Theoretical Computer Science*, vol. 50, pp. 1–102, 1987.

[2] S. Abramsky, "Computational interpretations of linear logic," *Theoretical Computer Science*, 1990. Special Issue on the 1990 MFPS Workshop, to appear.

[3] P. Wadler, "Linear types can change the world!," in *IFIP TC 2 Working Conference on Programming Concepts and Methods*, April 1990.

[4] J. Lamping, "An algorithm for optimal lambda calculus reduction," in *Seventeenth Annual ACM Symposium on Principles of Programming Languages*, pp. 16–30, ACM, Jan. 1990.

[5] V. Kathail, *Optimal interpreters for lambda-calculus based functional languages*. PhD thesis, MIT, May 1990.

[6] Y. Lafont, "Interaction nets," in *Seventeenth Annual ACM Symposium on Principles of Programming Languages*, pp. 95–108, ACM, Jan. 1990.

[7] J.-Y. Girard, "Geometry of interaction I: Interpretation of system F," in *Logic Colloquium '88* (Ferro, Bonotto, Valentini, and Zanardo, eds.), pp. 221–260, Elsevier Science Publishers B.V. (North Holland), 1989.

[8] J.-Y. Girard, "Geometry of interaction II: Deadlock-free algorithms.".

[9] G. Gonthier, M. Abadi, and J.-J. Lévy, "The geometry of optimal lambda reduction," in *Nineteenth Annual ACM Symposium on Principles of Programming Languages*, pp. 15–26, ACM, Jan. 1992.

[10] J.-Y. Girard, "On the unity of logic," tech. rep., June 1991. INRIA Report 1467.

[11] J. Chirimar, C. Gunter, and J. Riecke, "Linear ML." 1991.

[12] P. Lincoln and J. Mitchell, "Operational aspects of linear lambda calculus," in *Seventh Annual Symposium on Logic in Computer Science*, IEEE, June 1992.

[13] J.-J. Lévy, "Optimal reductions in the lambda-calculus," in *To H.B. Curry: Essays in Combinatory Logic, Lambda Calculus and Formalism* (J. Seldin and J. Hindley, eds.), pp. 159–191, Academic Press, 1980.

[14] J.-Y. Girard, "A new constructive logic: Classical logic," tech. rep., June 1991. INRIA Report 1443.