

# Taming Modal Impredicativity: Superlazy Reduction

Ugo Dal Lago\*, Luca Roversi† and Luca Vercelli‡

## Abstract

Pure, or type-free, Linear Logic proof nets are Turing complete once cut-elimination is considered as computation. We introduce *modal impredicativity* as a new form of impredicativity causing the complexity of cut-elimination to be problematic from a complexity point of view. Modal impredicativity occurs when, during reduction, the conclusion of a residual of a box  $b$  interacts with a node that belongs to the proof net *inside* another residual of  $b$ . Technically speaking, *superlazy reduction* is a new notion of reduction that allows to control modal impredicativity. More specifically, superlazy reduction replicates a box only when all its copies are opened. This makes the overall cost of reducing a proof net finite and predictable. Specifically, superlazy reduction applied to any pure proof nets takes primitive recursive time. Moreover, any primitive recursive function can be computed by a pure proof net via superlazy reduction.

**Keywords.** Linear logic, implicit computational complexity, proof theory.

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Pure Proof Nets</b>	<b>4</b>
2.1	Superlazy Reduction . . . . .	6
<b>3</b>	<b>Soundness</b>	<b>9</b>
<b>4</b>	<b>Completeness</b>	<b>10</b>
4.1	Recalling the Primitive Recursive Functions . . . . .	10
4.2	Representation of Integer Functions . . . . .	11
4.3	The Completeness theorem . . . . .	12
<b>5</b>	<b>Further Developments</b>	<b>14</b>
<b>6</b>	<b>Conclusions</b>	<b>15</b>

## 1 Introduction

Predicativity is a logical concept known from a century, starting from Russel's work. It has various technical meanings. All of them, however, refer implicitly or explicitly to some form of aciclicity (see [2] for an excellent survey). Impredicative definitions or logical rules in a logical system may lead to logical paradoxes. On the other hand, if logical systems are interpreted as programming

---

\*Dip. di Informatica, Univ. di Bologna, <http://www.cs.unibo.it/~dallago/>

†Dip. di Informatica, Univ. di Torino, <http://www.di.unito.it/~rover/>

‡Dip. di Matematica, Univ. di Torino, <http://www.di.unito.it/~vercelli/>

languages (via the Curry-Howard correspondence), impredicativity may lead to type systems and programming languages with high expressive power.

In this paper, we introduce the notion *modal impredicativity*. We start from Linear Logic which gives first-order status to structural rules (on the logical side) and to duplication and erasure (on the computational side). The very definition of modal impredicativity refers to *boxes*, i.e., those portions of proof nets, related to the modal meaning of formulae, that may be duplicated. During cut-elimination, a duplication occurs when a box interacts with a contraction node, which corresponds to an instance of the structural rule contraction in a logical derivation. Boxes allow to structure proofs into *layers*: any rule instance has a *level*, the number of boxes into which it is contained. Focusing our attention to boxes is the reason why our notion of impredicativity is dubbed as modal. Specifically, modal impredicativity occurs when, during reduction, the *root* of a residual of a box  $b$  interacts with a node that belongs to the proof net *inside* another residual of  $b$ . The paradigmatic example where an interaction of this kind occurs is in the (pure) proof net in Figure 1, which encodes the prototypical *non normalizing* lambda-term  $(\lambda x.xx)(\lambda x.xx)$ . Call  $b$  the (unique) box in Figure 1. After two reduction steps we get two copies  $b'$  and  $b''$  of  $b$ . Two further reduction steps plug the root of  $b'$  as premise of the node  $X$  belonging to the second copy  $b''$  of  $b$ . This is a basic form of one-step-long cycle, since the content of  $b$  interacts with the root of  $b$  itself. Compared to what happens classically, *self-copying* plays the rôle of *self-application* or *self-definition*. Notice that the cycles we are speaking about can have length

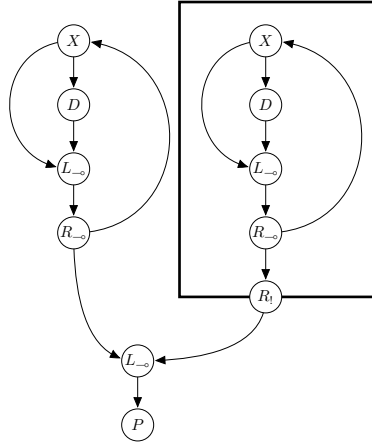


Figure 1: The pure proof net  $\Delta\Delta$ .

greater than one. As an example, consider the proof net corresponding to the lambda term  $(\lambda x.\lambda y.xy x)(\lambda x.\lambda y.xy x)(\lambda x.\lambda y.xy x)$ : it includes two boxes  $b_1$  and  $b_2$  where  $b_1$  copies  $b_2$  and  $b_2$  copies  $b_1$ .

Our long-term goal is to define proper restrictions on Linear Logic allowing to control modal impredicativity. This paper is just the first step towards this goal. What we define here is a new notion of *reduction* for Linear Logic proof nets which rules out the previously described cyclic phenomenon dynamically, i.e. at the level of the graph-theoretic rewriting relation which governs proof net reduction. This way, we break impredicative cycles, while keeping the freedom of *statically* compose pure proof nets.

**Light Logics and Modal Impredicativity.** We now recall how the known subsystems of Linear Logic, introduced as characterization of certain complexity classes, work. Let us call them light logics, for short. Proof-theoretically, they parsimoniously use the contraction rule. On the computational side, they control the duplication of structure. Technically, we currently know two ways of controlling the use of duplication. One is *stratification*. The other one is what we like to call *boundedness*.

*Stratification* is a structural constraint that, at the dynamic level, has the following meaning: one reduction step at level  $n$  can only increase the complexity of the underlying proof at levels (strictly) higher than  $n$ . This is achieved by dropping dereliction and digging as logical rules. The consequence is the control over the dimension of every single reduct, that reflects on the overall control of reduction time. The mechanism is implicit in the structural and combinatorial properties of proofs and is totally independent from its logical soundness. In stratified systems, the level of any node *cannot* change during reduction. As a consequence, any stratified system, by definition, cannot be modally impredicative because the nodes inside a box  $b$  cannot interact with the root of any copy of  $b$ . Elementary Linear Logic, Light Linear Logic [5] and their affine versions use *stratification*.

Concerning boundedness, recall that in ordinary Linear Logic,  $!A$  is semantically equivalent to  $\bigcup_{n \in \mathbb{N}} \underbrace{(A \otimes \dots \otimes A)}_n$ . *Boundedness* refers to various methodologies that, informally, put  $!A$  in correspondence to a *finite* subset of  $\bigcup_{n \in \mathbb{N}} \underbrace{(A \otimes \dots \otimes A)}_n$ . Computationally, this means the number

of copies of each box in a proof can somehow be statically or dynamically predicted, i.e. bounded. This way, we automatically get a system that cannot be modally impredicative, since in bounded systems  $!A$  cannot be equal to  $!A \otimes A$  (and this principle seems necessary to have self-copying). Soft Linear Logic [7] and Bounded Linear Logic [3] use *boundedness*.

**Superlazy Reduction and Modal Impredicativity.** *Superlazy reduction* is a new notion of reduction for Linear Logic (pure) proof nets. It is specifically designed to control modal impredicativity. Under superlazy reduction, any box  $b$  can interact with a tree of contraction, dereliction, digging and weakening nodes only if the global result of this interaction somehow reduces the overall complexity of the proof net, namely when it produces (possibly many) “open” copies of  $b$ . If this is not the case, reduction is blocked and cannot be performed. This way modal impredicativity is automatically ruled out, since whenever the *content* of  $b$  is copied,  $b$  as a box is destroyed and no residuals of  $b$  are produced. Technically, this is ensured by prescribing that reduction can happen only when the box is faced with a *derelicting tree of nodes*, a key notion introduced in Section 3.

**Superlazy Reduction and Primitive Recursion.** The calculus we obtain by adopting superlazy reduction over pure proof nets is still powerful enough to characterize the class PR of primitive recursive functions. We show the characterization under a standard pattern. As for *soundness*, we prove that every pure proof net  $G$  can be rewritten to its normal form in time bounded by a primitive recursive function in the size of  $G$ . This is remarkable by itself, since the mere fact that superlazy reduction *computes something* is interesting by itself, considered the strong requirements superlazy reduction must satisfy. As for *completeness*, every function in PR can be represented as a pure proof net, even under superlazy reduction.

**Superlazy Reduction and Expressive Power.** We here want to make some observations about pure proof nets and superlazy reduction as a paradigmatic programming language. The set of terms we can program with are pure proof nets coming from Linear Logic. Namely, we can use every lambda-term as a program,  $\Delta\Delta$  (Figure 1) included. However, we do not have the standard unconstrained reduction steps, which, by simulating the usual beta-reduction, allow to embed pure, i.e. untyped, lambda-calculus into pure proof nets. In particular, the proof net  $\Delta\Delta$  is normal in our setting, the reason being that we can never reach a situation where the “amount” of open copies of the box it contains is known in advance.

**Related Work.** Several authors have used some notion of predicativity as a way to restrict the expressive power of programming languages or logical systems. We here recall some of them, without any hope of being exhaustive. In [10], as a first example, Danner and Leivant presented a

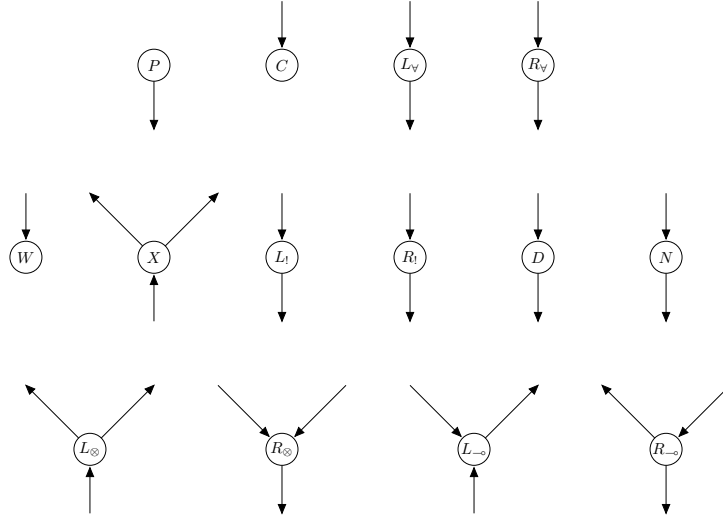


Figure 2: The nodes of the proof nets.

variant of the second order  $\lambda$ -calculus obtained imposing a restriction on the second-order quantification. Such a restriction has semantic flavor: all the *types* have a *rank*, an ordinal number; the type  $\forall R : \xi.\tau$  is a legal type where  $R$  can be replaced with every other type *of rank at most*  $\xi$ . Of course the formula  $\forall R : \xi.\tau$  has rank higher than  $\xi$ , so a form of predicativity is reached. However, this is not enough to limit the expressive power: the maximum possible rank must be limited *a priori*. This way they get a characterization of primitive recursion. Another, earlier, example is Leivant's predicative recursion [8]: if predicativity is imposed on ordinary primitive recursion (on any word algebra), one gets a characterization of polynomial time computable functions. Further work shows how other classes can be characterized with similar tools [9, 11]. A final example is Simmons' fine analysis of tiering [13]. All the cited proposals, however, share a property which make them fundamentally different from superlazy reduction: predicativity is enforced through static constraints, i.e., constraints on programs rather than on the underlying reduction relation. The first author has recently proposed a characterization of primitive recursion by a fragment of Gödel's  $\mathsf{T}$  [1].

On the other hand, restricted notions of reduction on Linear Logic proof nets have appeared in the literature. This includes, for example, Girard's closed reduction [4] or head linear reduction [12]. None of them, however, decreases the expressive power of the logical systems on top of which they are applied, as superlazy reduction does.

**Paper Outline.** Section 2 recalls pure, i.e., untyped, proof nets and defines what are derelicting trees and superlazy reduction. Section 3 proves the primitive recursive soundness of superlazy reduction on pure proof nets. Section 4 shows that even under superlazy reduction, pure proof nets remain expressive enough to represent all the primitive recursive functions. Section 5 presents some further developments on the ideas presented in the paper.

## 2 Pure Proof Nets

Pure proof nets are graph-like structures built using the nodes in Figure 2 and the inductive clauses in Figure 3(a), 3(b) and 4. Figure 3(a) says that a wire is a proof net. Given the two proof nets in 3(b) we can build those in Figure 4. The inductive rule at the end of Figure 3(a) introduces (modal) *boxes*.

Please notice that the proof nets introduced here are slightly different from the usual ones. In particular, there is not any explicit node playing the rôle of the cut rule or of axioms. Moreover,

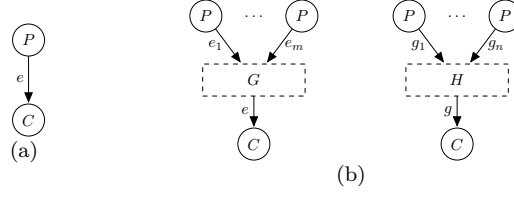


Figure 3: Base cases.

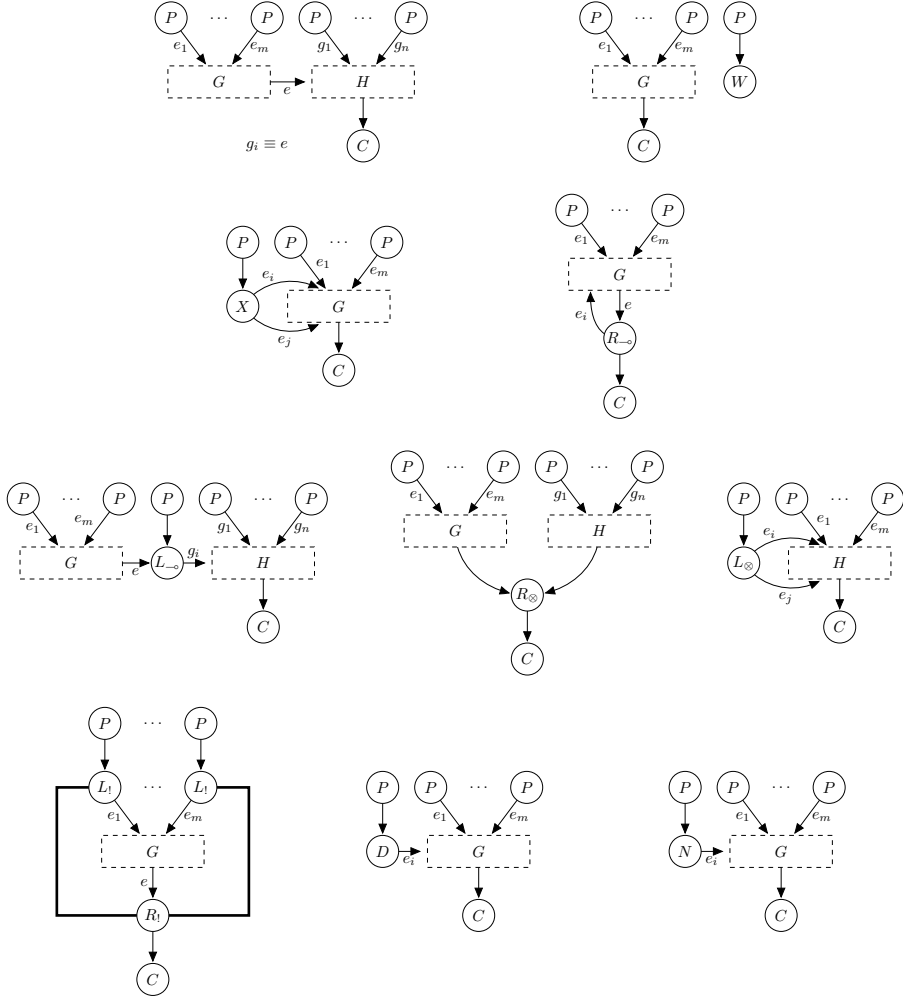


Figure 4: Inductive cases

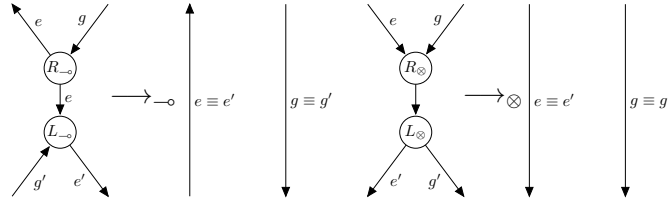


Figure 5: Linear graph rewriting rules.

proof net conclusions are partitioned into one proper conclusion and some premises. This way, proof nets get an intuitionistic flavor which makes the correspondence with lambda-terms more evident; this will be useful in Section 4.

**Reduction Rules.** The reduction rules for the proof nets are in Figure 5, and 6. Call  $\longrightarrow$  the contextual closure of the rewriting steps  $\longrightarrow_{\multimap}, \longrightarrow_{\otimes}, \longrightarrow_D, \longrightarrow_X, \longrightarrow_W, \longrightarrow_N, \longrightarrow_M$ . The reflexive and transitive closure of  $\longrightarrow$  is  $\longrightarrow^*$ .

The reduction rules  $\longrightarrow_X$  and  $\longrightarrow_N$  are the only ones somehow increasing the size of the underlying proof-net: the first one copies a box, while the second one puts a box inside another box. Superlazy reduction, as we will see shortly, does not simply eliminate those rules, but rather forces them to be applicable only in certain contexts, i.e., only when those rules are part of a sequence of modal rewriting rules which have a globally predictable behavior.

## 2.1 Superlazy Reduction

We shall be able to prove a soundness result about the cost of the reduction of the proof nets relatively to a *superlazy* version of  $\longrightarrow$  that requires the notion of *derelicting tree*.

**Derelicting trees.** For every proof net  $G$ , let us assume that the *cost* of traversing any  $X$ -node of  $G$  is 0, any  $D$ -node is  $-1$ , any  $W$ -node is 0 and any  $N$ -node is  $+1$ . The cost of a path from node  $u$  to node  $v$  is the sum of the costs of nodes in the path *including*  $u$  and  $v$ . A *derelicting tree* in  $G$  is a subgraph  $t$  of  $G$  that satisfies the following four conditions:

1.  $t$  only contains nodes  $X, D, N, W$ ; so it must be a tree, and we call  $w$  its root;
2. the leaves of  $t$  are labelled either with  $D$  or with  $W$ ;
3. for every leaf  $v$  labelled with  $D$  in  $t$ , the cost of the path from  $w$  to  $v$  in  $t$  is  $-1$ ;
4. the cost of any other path in  $t$  starting from  $w$  is nonnegative.

Figure 7(a) shows an example of a derelicting tree. Conditions 1. and 2. are trivially satisfied. The cost of  $v_1v_2v_4v_7v_{10}$  is  $-1$  and the same for  $v_1v_3v_6$ . Finally, any other path starting from  $v_1$  has nonnegative cost. For example,  $v_1v_3v_5v_8$  has cost 1.

Figure 7(b) depicts a remarkable instance of derelicting tree: an  $n$ -bounded spine with  $n$  occurrences of  $X$  nodes that we shall represent as a dashed box with name  $nX$ .

**Superlazy Reduction Step and Rewriting System on Pure Proof Nets.** The *superlazy normalization step* is  $\rightarrow_{XNDW}$ , defined in Figure 8,  $\nabla t$  being any derelicting tree.  $\rightsquigarrow_{\multimap}, \rightsquigarrow_{\otimes}, \rightsquigarrow_M, \rightsquigarrow_{XNDW}$  denote the *surface* contextual closure of the rewriting steps  $\longrightarrow_{\multimap}, \longrightarrow_{\otimes}, \longrightarrow_M, \longrightarrow_{XNDW}$ . “Surface” means that we never apply a reduction inside a box.  $\rightsquigarrow$  is the union of  $\rightsquigarrow_{\multimap}, \rightsquigarrow_{\otimes}, \rightsquigarrow_M, \rightsquigarrow_{XNDW}$ . The reflexive and transitive closure of  $\rightsquigarrow$  is  $\rightsquigarrow^*$ .

Superlazy reduction is a *very* restricted notion of reduction. In particular, it is almost useless when applied to proof nets obtained from ordinary lambda-terms via the usual, uniform encoding. In particular:

- If terms are encoded via the so-called call-by-name encoding (i.e., the one induced by Girard’s correspondence  $(A \rightarrow B)^\circ \equiv !A^\circ \multimap B^\circ$ ), then any redex  $(\lambda x.M)N$  where  $M$  consists of an

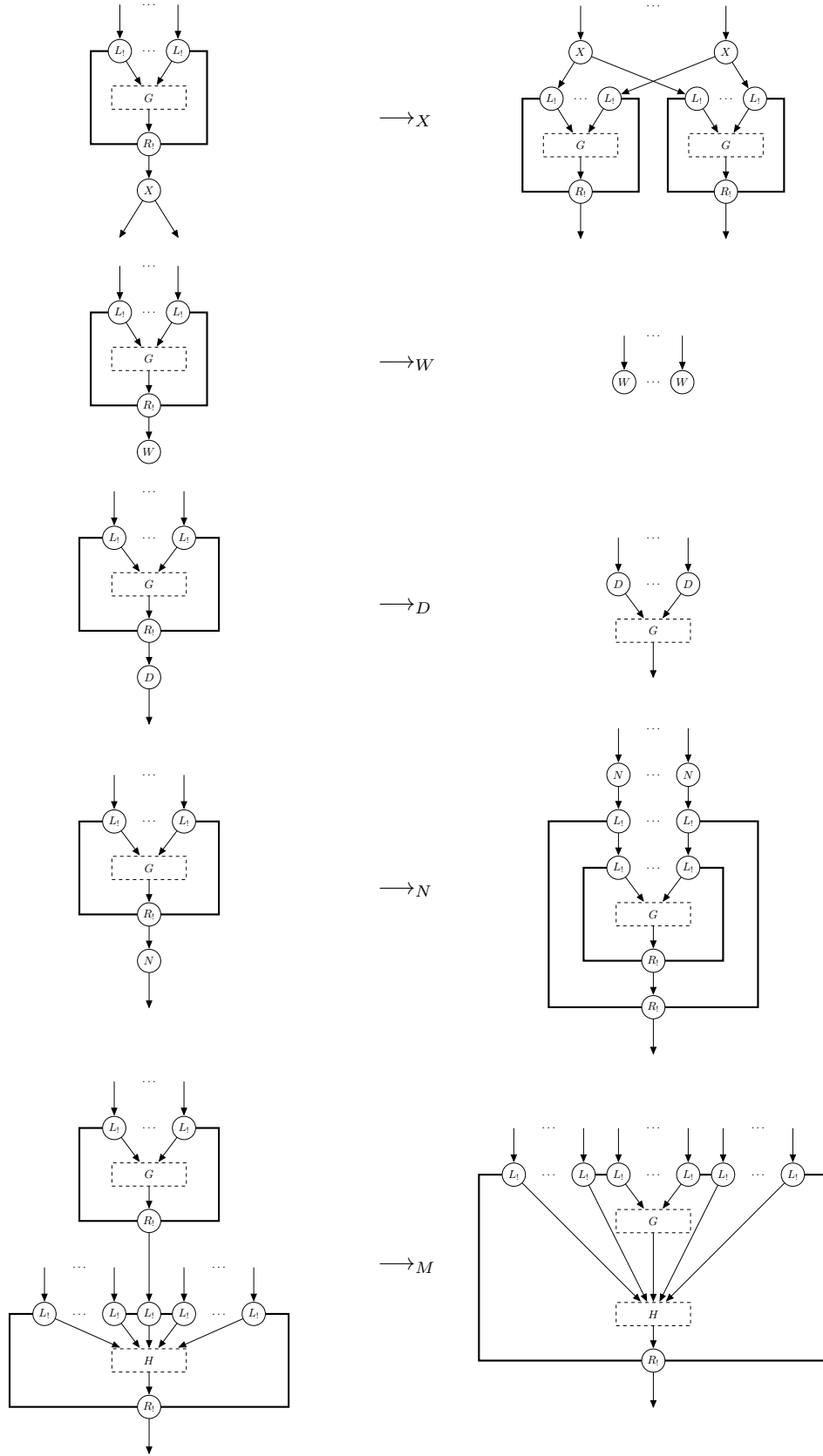


Figure 6: Modal graph rewriting rules.

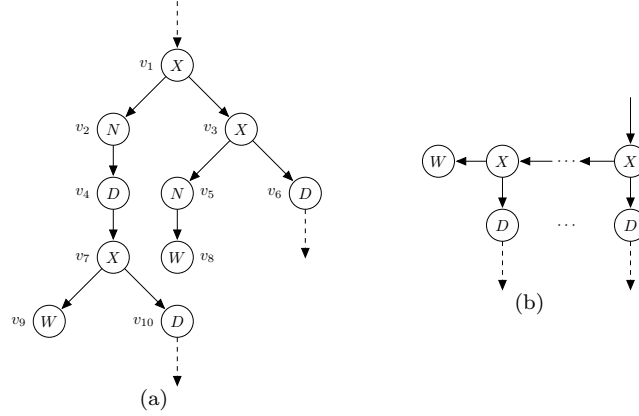


Figure 7: A generic derelicting tree (a) and a bounded spine (b)

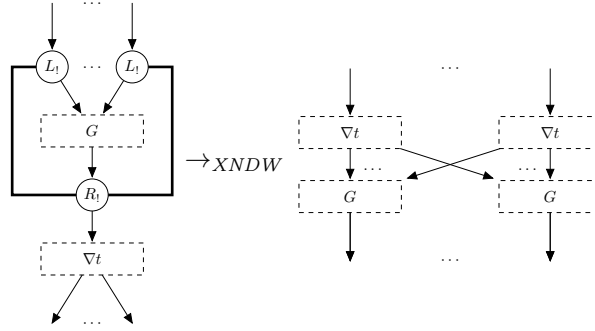


Figure 8: Superlazy cut elimination step

application  $LP$  and  $x$  appears free in  $P$ , cannot be reduced in the corresponding proof net: a box would be faced with something different from a derelicting tree.

- On the other hand, if terms are encoded via the call-by-value encoding (i.e. via the correspondence  $(A \rightarrow B)^\circ \equiv !(A^\circ \multimap B^\circ)$ ), then any redex  $(\lambda x.M)N$  where  $M$  consists itself of an abstraction  $\lambda y.L$  and  $x$  appears free in  $L$  cannot be reduced in the corresponding proof net.

Unfortunately, we do not even know any criteria allowing to guarantee that certain proof nets can be reduced to normal form (w.r.t. ordinary reduction) by way of superlazy reduction. Moreover, there currently isn't any result characterizing the class of normal forms w.r.t. superlazy reduction; this is in contrast, for example, to lambda calculus and call-by-value reduction, where the cbv normal form of any (closed) term  $M$  (if any) is always a value. This is why proving that proof nets are complete w.r.t. some given class of functions, under the superlazy reduction, is non-trivial. Nonetheless, we explicitly prove the completeness of superlazy reduction (see Section 4). The next section shows why both  $(A \rightarrow B)^\circ \equiv !(A^\circ \multimap B^\circ)$  and  $(A \rightarrow B)^\circ \equiv !(A^\circ \multimap B^\circ)$  do not work well here: superlazy reduction of any proof net always terminates in a time bounded by suitable primitive recursive functions.

Linear Logic with superlazy reduction can be seen as a generalization of the principles of Soft Linear Logic. Every time a box is replicated, that box is opened. According to this vision, a derelicting tree with  $m$  leaves is similar to a multiplexor node of rank  $m$ . However, the structure of SLL gives a further restriction: if  $k$  is the rank of the proof net  $G$ , that is the maximum rank of the multiplexers in  $G$ , we are sure that every box of  $G$  will be copied at most  $k$  times. Such a restriction leads to a polytime bound (see [7]).



### 3 Soundness

Let's begin with some definitions. Given a proof  $G$  and any reduction relation  $\rightarrow$ ,  $[G]_{\rightarrow}$  and  $\|G\|_{\rightarrow}$  denote the maximum length of a reduction sequence starting in  $G$  (under  $\rightarrow$ ) and the maximum size of any reduct of  $G$  (under  $\rightarrow$ ), respectively.  $|G|$  is the size of a proof net  $G$ . If  $G \rightsquigarrow^* F$  and  $b^*$  is a box in  $F$ , then  $b^*$  is the merge of the residuals of one or more boxes  $b'_1, \dots, b'_k$  in  $G$ ; in this case we say that  $b^*$  is the *residual\** of  $\{b'_1, \dots, b'_k\}$ . For shortness we say also that  $b^*$  is the residual\* of  $b'_1$ , meaning that  $b^*$  is the residual\* of  $b'_1$  plus some other boxes. If  $b$  is a box in  $G$ , we denote  $\mathcal{B}_G(b) = \{b\} \cup \{d \mid d \text{ is a box of } G \text{ contained in } b\}$ . A reduction sequence  $\sigma \equiv G \rightsquigarrow G' \rightsquigarrow \dots \rightsquigarrow H$  is said to be a  $(n, d)$ -box reduction if there are  $r \leq n$  boxes  $b_1, \dots, b_r$  between those at level 0 in  $G$  such that the following conditions hold:

- The nodes of the proof nets inside every box  $b_1, \dots, b_r$  are at most at depth  $d$ .
- The box  $d^*$  involved in any step  $\rightsquigarrow_{XNDW}$  of  $\sigma$  is a residual\* of some boxes  $\{d_1, \dots, d_k\}$  of  $G$  where for every  $i \leq k$  there exists  $j \leq r$  such that  $d_i \in \mathcal{B}_G(b_j)$ .
- $r$  is the least with the above properties.

**Remark 3.1** *A few observations about the above definition:*

1.  $G$  may contain more than  $n$  boxes at level 0, or it may contain boxes whose depth is greater than  $d$ ;
2.  $H$  is not necessarily a normal form. It may contain boxes at level  $d$ , or higher;
3. By definition, any node inside the boxes  $b_1, \dots, b_r$  must have depth at least 1. This means that if  $\sigma$  is a  $(n, 0)$ -box reduction it is, in fact, talking about at most  $n$  boxes that contain nodes at level 0. They cannot exist, so  $\sigma$  only uses the linear rewriting steps  $\rightarrow_{\circ}, \rightarrow_{\otimes}$ ;
4. By definition, every reduction starting at any net  $G$  is a  $(|G|, \partial(G))$ -box reduction because  $|G|$  necessarily bounds the number  $n$  of boxes, and  $\partial(G)$  bounds the value  $d$ ;
5. If  $b^*$  is a box involved in a  $\rightsquigarrow_{XNDW}$  step and it is a residual\* of some box  $b$  in  $G$  at level 0, then  $b = b_i$  for some  $i$  by definition. More important, also the converse holds: for each  $i$  there exists a residual\*  $b_i^*$  of  $b_i$  along the reduction  $G \rightsquigarrow^* H$  that is involved in a  $\rightsquigarrow_{XNDW}$  step; that is,  $b_i$  will eventually be opened.

Now we define a family of functions  $\{f_d : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}\}_{d \in \mathbb{N}}$  such that every  $f_d(n, m)$  bounds both the reduction cost and the size of the reducts, of every net  $G$  when performing a  $(n, d)$ -box reductions on it:

$$\begin{aligned} f_0(n, m) &= m \\ f_{d+1}(0, m) &= m \\ f_{d+1}(n+1, m) &= 1 + f_{d+1}(n, m) + f_d(2f_{d+1}(n, m)^2, 2f_{d+1}(n, m)^2) \end{aligned}$$

By definition, all the functions  $f_d$  are primitive recursive.

**Lemma 3.2** *For every  $d, n, m \in \mathbb{N}$ ,  $f_{d+1}(n, m) \geq f_d(n, m)$  and  $f_d(n+1, m) \geq f_d(n, m)$ .*

The proof is by induction on  $n$ . We can prove the following:

**Proposition 3.3** *Let  $G$  be any proof net and let  $G \rightsquigarrow^k H$  be a  $(n, d)$ -box reduction with  $k$  steps. Then  $k, |H| \leq f_d(n, |G|)$ .*

**Proof.** We can proceed by induction on  $d$ :

- If  $d = 0$ , thanks to the *aspect* (iii) above, the reduction  $G \rightsquigarrow^k H$  only involves multiplicative reduction steps. They strictly reduce the size of the underlying net. So,  $|H|, k \leq |G| = f_0(n, |G|)$ .
- Suppose the thesis holds for  $d$  and suppose  $G \rightsquigarrow^k H$  is a  $(n, d+1)$ -box reduction. We proceed by another induction, this time on  $n$ :
  - If  $n = 0$ , then none of the boxes at level  $d+1$  can be reduced. As a consequence,  $G \rightsquigarrow^k H$  only involves multiplicative reduction steps and, again,

$$|H|, k \leq m = f_{d+1}(0, m).$$

- Suppose the thesis holds for  $n$  and suppose  $G \rightsquigarrow^k H$  is a  $(n+1, d+1)$ -box reduction sequence. By definition, there are  $r \leq n+1$  boxes  $b_1 \dots, b_r$  in  $G$  satisfying the definition above. If  $r = 0$ , the reduction is performed in linear time, so  $|H| \leq |G| \leq f_{d+1}(n+1, m)$ . Otherwise, we already noted that for each box  $b_i$  there is a residual\*  $b_i^*$  that will be involved in a  $\rightsquigarrow_{XWND}$  step. There is clearly one index  $t$ , where  $1 \leq t \leq r$ , such that  $b_t^*$  is the last box being copied *among the residuals of the  $b_i$ 's* in the sequence  $G \rightsquigarrow^k H$ . The reduction under consideration can be decomposed as follows:

$$G \rightsquigarrow^i F \rightsquigarrow_{XWND} J \rightsquigarrow^j H$$

where the step  $F \rightsquigarrow_{XWND} J$  is the reduction of  $b_t^*$ . Up to the  $F$ , the reduction sequence can be considered as a  $(n, d+1)$ -box reduction sequence: the witness list of boxes is  $b_1, \dots, b_{t-1}, b_{t+1}, \dots, b_r$ . After  $J$ , on the other hand, the reduction sequence can be considered as a  $(|J|, d)$ -box reduction sequence. Indeed, by definition of  $\sigma$ , all the boxes that will be involved in a  $\rightsquigarrow_{XWND}$  step are residual\* of some boxes  $\{d_1, \dots, d_k\}$  inside  $b_1, \dots, b_r$ ; but all the boxes  $b_i$  have been opened, so the residual of their content is less depth. Applying both inductive hypothesis, we get

$$\begin{aligned} i, |F| &\leq f_{d+1}(n, m) \\ |J| &\leq 2|F|^2 \quad \text{usual reduction bound} \\ j, |H| &\leq f_d(|J|, |J|) \end{aligned}$$

As a consequence:

$$\begin{aligned} |J| &\leq 2f_{d+1}(n, m)^2 \\ k &= i + 1 + r \leq f_{d+1}(n, m) + 1 + f_d(|J|, |J|) \\ &\leq f_{d+1}(n, m) + 1 + f_d(2f_{d+1}(n, m)^2, 2f_{d+1}(n, m)^2) \\ |H| &\leq f_d(|J|, |J|) < f_{d+1}(n, m) + 1 + f_d(|J|, |J|) \\ &\leq f_{d+1}(n, m) + 1 + f_d(2f_{d+1}(n, m)^2, 2f_{d+1}(n, m)^2) \end{aligned}$$

This concludes the proof.

**Corollary 3.4** *For every  $n \in \mathbb{N}$  there is a primitive recursive function  $g_n : \mathbb{N} \rightarrow \mathbb{N}$  such that for every proof net  $G$ ,  $[G]_{\rightsquigarrow}, ||G||_{\rightsquigarrow} \leq g_n(|G|)$ .*

**Proof.** Remember that every reduction sequence from  $\Pi$  is a  $(|\Pi|, \partial(\Pi))$ -reduction sequence. So, we can use Proposition 3.3 and choose  $g_d(n) = f_d(n, n)$ .

## 4 Completeness

The goal is to show the existence of an embedding of any primitive recursive function  $f$  into a pure proof net  $G_f$  such that  $G_f$  simulates  $f$  via superlazy reduction. The existence of such an embedding is what we mean by *completeness*.

### 4.1 Recalling the Primitive Recursive Functions

The primitive recursive functions (PR) are functions from tuples of natural numbers to a natural number. PR is the least set that both contains the *basic* primitive recursive functions and which is closed under a finite number of applications of *composition* and *primitive recursion*.

**Basic Primitive Functions.** The basic primitive functions are the 0-ary *constant* function 0, the 1-ary *successor* function  $s : \mathbb{N} \rightarrow \mathbb{N}$ , and the  $m$ -ary *projection* function  $\pi_i^m : \mathbb{N}^m \rightarrow \mathbb{N}$ , for every  $m \geq 1$  and  $1 \leq i \leq m$ . For every natural number  $n$ , the natural number  $s(n)$  is simply  $n+1$ , while  $\pi_i^m(n_1, \dots, n_m)$  is the  $i$ -th argument  $n_i$ , for every  $n_1, \dots, n_m \in \mathbb{N}$ .

**Composition.** Let  $f$  be an  $n$ -ary PR function and let  $g_1, \dots, g_n$  be  $m$ -ary PR functions. The *composition* of  $f$  with  $g_1, \dots, g_n$  is the  $m$ -ary function defined as follows:

$$\circ[f, g_1, \dots, g_n](n_1, \dots, n_m) = f(g_1(n_1, \dots, n_m), \dots, g_n(n_1, \dots, n_m)).$$

**Primitive Recursion.** Let  $f$  be an  $m$ -ary PR function and let  $g$  be an  $(m+2)$ -ary PR function. The function defined by *primitive recursion* on  $f$  and  $g$  is the  $(m+1)$ -ary function defined as follows:

$$\begin{aligned} \text{rec}[f, g](0, n_1, \dots, n_m) &= f(n_1, \dots, n_m) \\ \text{rec}[f, g](n+1, n_1, \dots, n_m) &= g(n, \text{rec}[f, g](n, n_1, \dots, n_m), n_1, \dots, n_m) \end{aligned} \quad (1)$$

## 4.2 Representation of Integer Functions

We represent (tuples) of natural numbers and functions from naturals to natural as proof nets.

**Clusters.** Figure 9 introduces clusters of, at least, one instance of the nodes  $L_{\rightarrow}, L_{\otimes}, R_{\otimes}$ .

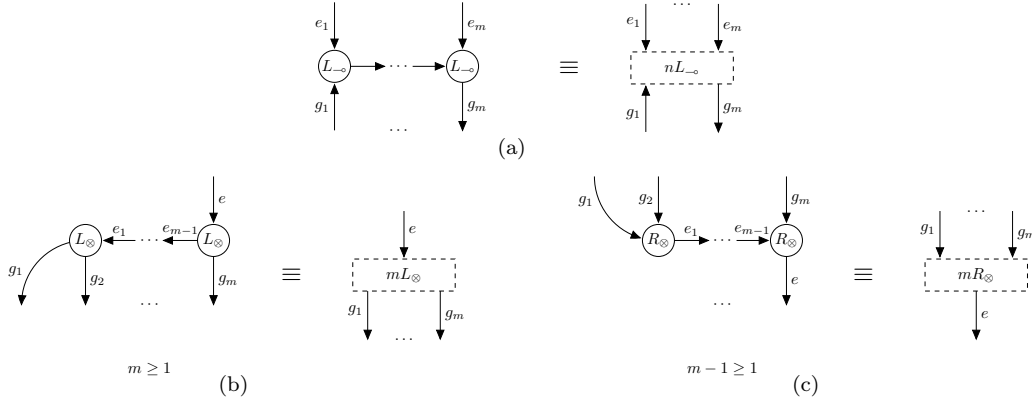


Figure 9: Clusters of nodes

**Numerals.** Figure 10 introduces the closed proof nets  $\bar{0}$  and  $\bar{n}$ , with  $n \geq 1$ .  $nX$  is the  $n$ -bounded spine in Figure 7(b). A basic observation to prove the completeness is that every  $\bar{n}$  contains a  $n$ -bounded spine.

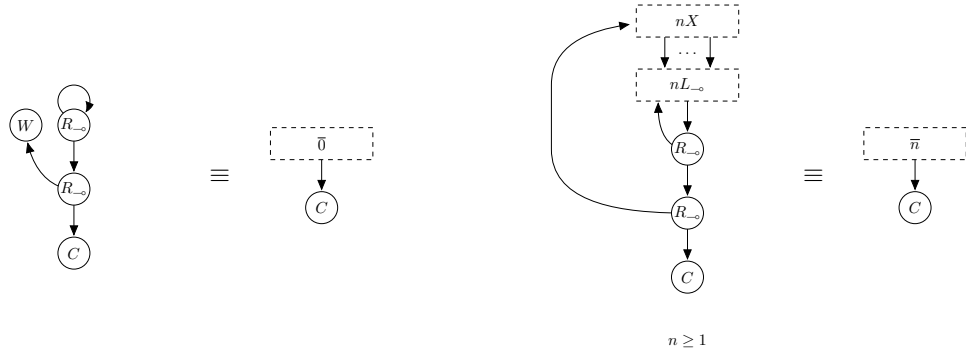


Figure 10: Church numerals as proof nets

**Tuples of numerals.** The proof nets  $\langle \overline{n_1}, \dots, \overline{n_m} \rangle$  are obtained as follows. If  $m = 1$ , then  $\langle \overline{n_1} \rangle$  coincides to the net  $\overline{n_1}$ . Otherwise, if  $m > 1$ , then  $\langle \overline{n_1}, \dots, \overline{n_m} \rangle$  is obtained by composing the  $m$  conclusions of every proof net  $\overline{n_1}, \dots, \overline{n_m}$  with one of the premises of  $mR_\otimes$ .

**Integer functions.** Let  $G$  be a proof net with a single premise and a single conclusion. We write  $G \smile \langle \overline{n_1}, \dots, \overline{n_m} \rangle$  to mean the closed proof net with a single conclusion obtained by plugging the conclusion of the net  $\langle \overline{n_1}, \dots, \overline{n_m} \rangle$  into the assumption of  $G$ .

For every function  $f : \mathbb{N}^m \rightarrow \mathbb{N}$ , with arity  $m \geq 0$ , we shall say that a proof net  $G_f$  with one premise *represents*  $f$  iff  $G_f \smile \langle \overline{n_1}, \dots, \overline{n_m} \rangle$  reduces to  $\overline{f(n_1, \dots, n_m)}$ , for every  $n_1, \dots, n_m \in \mathbb{N}$ .

### 4.3 The Completeness theorem

Now we can state formally the Completeness theorem:

**Theorem 4.1 (Completeness)** *Every  $f$  in PR is represented by a proof net  $G_f$ .*

To prove it we define how to map every  $f \in \text{PR}$  into a corresponding  $G_f$ . Then we show that superlazy reduction allows to rewrite any  $G_f \smile \langle \overline{n_1}, \dots, \overline{n_m} \rangle$  to  $\overline{f(n_1, \dots, n_m)}$ . To map a function into a net requires to program with the proof nets as we do here below.

**The Successor.** Figure 11 defines the proof net  $G_s$  that represents the successor function.  $G_s \smile \langle \overline{n} \rangle$  yields  $\overline{n+1}$  because  $G_s$  extends the bounded spine  $nX$  and  $nL_{\multimap}$  of  $\overline{n}$  into  $(n+1)X$  and  $(n+1)L_{\multimap}$ , the core of  $\overline{n+1}$ .

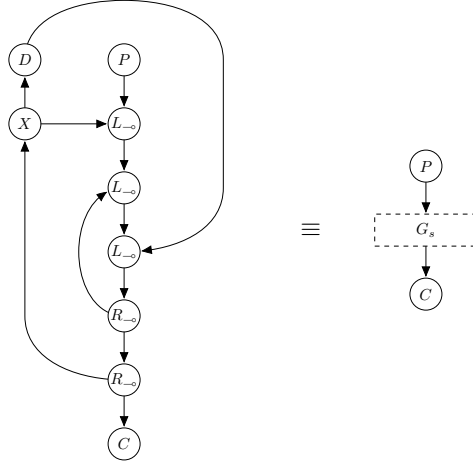
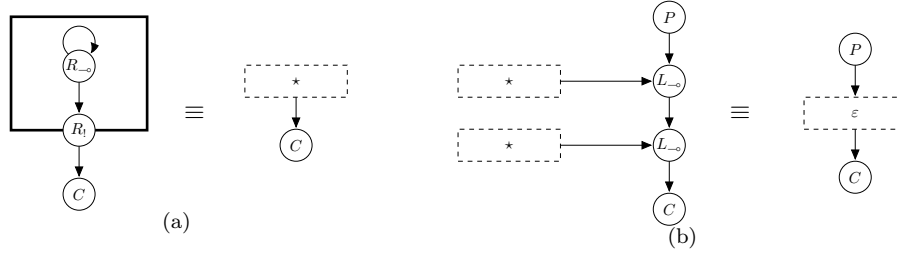
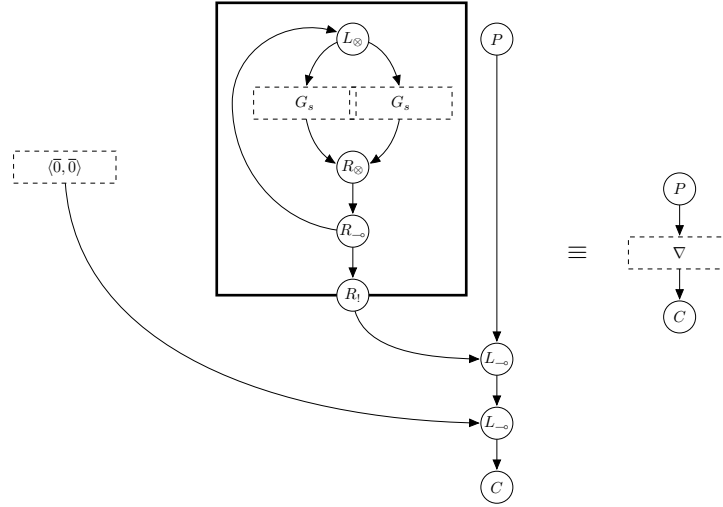


Figure 11: The proof net  $G_s$  (successor)

**Unity and Erasure.** Figure 12(a) introduces the closed proof net Unity that we identify as  $\star$ . Figure 12(b) introduces the proof net  $\varepsilon$  with two occurrences of  $\star$  in it.  $\varepsilon \smile \langle \overline{n} \rangle$  reduces to  $\star$ , because (i) the conclusion of the box of the topmost  $\star$  is plugged into the root of the bounded spine of  $\langle \overline{n} \rangle \equiv \overline{n}$ , and (ii) by means of  $\sim_{XWND}$ , we get  $n$  instances of the identity, “iteratively” applied to the lower occurrence of  $\star$ , which, after some reduction transform to  $\star$ .

**Diagonal.** Figure 13 introduces the proof net  $\nabla$  that duplicates a given unary string.  $\nabla \smile \langle \overline{n} \rangle$  reduces to  $\langle \overline{n}, \overline{n} \rangle$  because the conclusion of the box in the definition of  $\nabla$  is plugged into the root of the bounded spine in  $\langle \overline{n} \rangle \equiv \overline{n}$ .

There is an obvious generalization  $m\nabla$  of  $\nabla$ , with  $m \geq 2$ , necessary to represent the composition of functions in PR.  $m\nabla$  has a single premise and a single conclusion, and it multiplies a given argument  $m$  times.

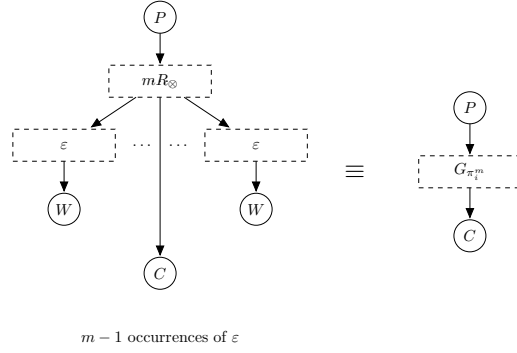
Figure 12: The unity proof net ( $\star$ ) and the erasure proof net ( $\varepsilon$ )Figure 13: The proof net  $\nabla$  (duplicator)

**Projections.** Figure 14 defines the proof net  $G_{\pi_i^m}$  that implements the basic primitive recursive function  $\pi_i^m$ , for any  $m \geq 1$ , with  $1 \leq i \leq m$ . The proof net has the expected behavior thanks to the correct behavior of  $\varepsilon$ .

**Composition.** Figure 15 introduces the proof net  $G_{\circ[f, g_1, \dots, g_n]}$  that represents the function  $h : \mathbb{N}^m \rightarrow \mathbb{N}$  obtained by composition from  $f : \mathbb{N}^n \rightarrow \mathbb{N}$  and  $g_1, \dots, g_n : \mathbb{N}^m \rightarrow \mathbb{N}$ . Of course the proof nets  $G_f, G_g$  must represent the functions  $f, g$ .

**Primitive Recursion.** We assume that the function  $f : \mathbb{N}^m \rightarrow \mathbb{N}$  can be represented by the proof net  $G_f$ , and that  $g : \mathbb{N}^{m+2} \rightarrow \mathbb{N}$  can be represented by the proof net  $G_g$ . Figure 16 introduces the proof net  $G_{\text{rec}[f, g]}$ . We shall show that  $G_{\text{rec}[f, g]}$  represents the  $(m+1)$ -ary primitive recursion  $\text{rec}[f, g]$ , as defined by equations (1).  $G_{\text{rec}[f, g]}$  is built using two more proof nets  $G_1$  and  $G_2$ . The base function  $G_1$  of  $G_{\text{rec}[f, g]}$  is in Figure 16.  $G_1$  is a closed net that, after an application, yields a function  $f'$  that, taking a tuple  $\vec{n}$  of  $m$  integers, gives a pair of naturals. A little bit more formally,  $f'(\vec{n}) = \langle 0, f(\vec{n}) \rangle$ . Instead, the net  $G_2$  in Figure 17 is the *iterable step function* of  $G_{\text{rec}[f, g]}$ . Iterable means that  $G_2$  can be used as the first argument of a Church numeral.  $G_2$  maps a net that behaves as a function with an ideal type  $\mathbb{N}^m \rightarrow \mathbb{N}^2$  to a net with the same type. More precisely,  $g'(\langle n_0, h \rangle, \vec{n}) = \langle n_0 + 1, g(n_0, h(\vec{n}), \vec{n}) \rangle$ . In particular, the iteration starts from the result of  $G_1$ , morally having the right type.

Now, we want to prove that  $G_{\text{rec}[f, g]} \curvearrowright \langle \overline{n_0}, \overline{n_1}, \dots, \overline{n_m} \rangle$  reduces to the expected result  $\overline{\text{rec}[f, g](n_0, \vec{n})}$ . The keypoint is the application of  $G_2$ , inside  $G_{\text{rec}[f, g]}$ , as an argument of  $n_0$ . If  $n_0 = 0$ , then the

Figure 14: The proof net  $G_{\pi_i^m}$ 

whole box around  $G_2$  is erased. This returns the identity. But, for every  $n_0 > 0$ ,  $G_2$  is replicated  $n_0$  times by the bounded spine  $n_0 X$  of  $\overline{n_0}$ . This iterates  $n_0$  times the function  $g'$ , leading to a new function  $g'^{(n_0)}$ . Now, the behavior of the net becomes obvious.  $g'^{(n_0)}$  takes  $\langle 0, f \rangle$  and  $\vec{n}$  as its arguments. Then, it iterates the step function. Finally, the result has form  $\langle n_0, g'^{(n)}(\langle n_0, f \rangle, \vec{n}) \rangle$ . An application of the right projection gives the result.

**Proof of Theorem 4.1.** By induction on the structure of  $f$ . The possible structures to be considered are:  $0$ ,  $s$ ,  $\pi_i^m$ ,  $\circ[h, g_1, \dots, g_n]$  and  $\text{rec}[h, g]$ . If  $f \equiv 0$  (base case), the statement holds by the reflexivity of  $\sim^*$ . If  $f \equiv s$  or  $\pi_i^m$ , we already know that these functions can be represented. If  $f \equiv \circ[h, g_1, \dots, g_n]$ , by inductive hypothesis the functions  $G_h$  and  $G_{g_i}$  can be represented. And the composition is representable. Similarly, if  $f \equiv \text{rec}[h, g]$ , by inductive hypothesis the functions  $G_h$  and  $G_g$  can be represented. So their primitive recursion  $\text{rec}[h, g]$  is representable too.

## 5 Further Developments

As we explained in the Introduction, this paper is just the first step in a long-term study about how to control modal impredicativity.

There are at least two distinct research directions the authors are following at the time of writing. We recall the obtained results here, pointing to further work for additional details and proofs.

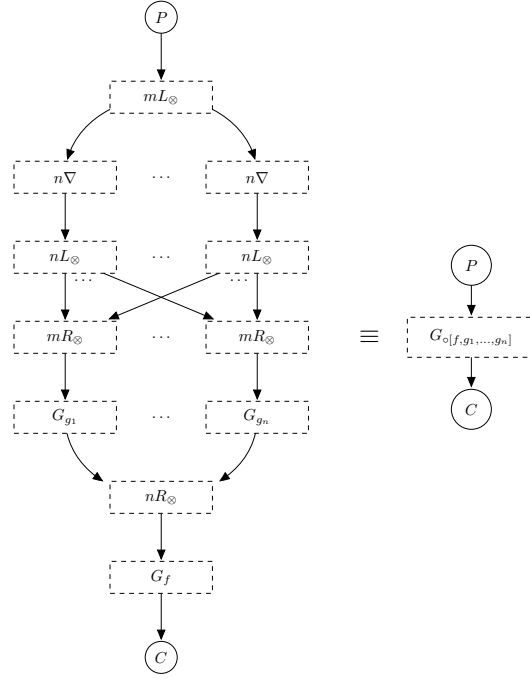
First of all, the way we proved completeness of proof net reduction w.r.t. primitive recursion suggests a way of capturing Hofmann's non-size increasing computation [6] by way of a proper, further restriction to superlazy reduction. We basically need two constraints:

- Boxes can only be copied when they are closed, i.e., when they have no premises.
- Any box  $b$  can only be copied if the proof net contained in  $b$  only contains *one* node  $X$  at level 0.

The obtained reduction relation is said to be the *non-size increasing superlazy reduction* and is denoted with  $\Rightarrow$ . With these constraints, non-size increasing polytime computation can be simulated by pure proof nets. Moreover:

**Theorem 5.1** *For every  $n \in \mathbb{N}$  there is a polynomial  $p_n : \mathbb{N} \rightarrow \mathbb{N}$  such that for every proof net  $\Pi$ ,  $[G]_{\Rightarrow}, ||G||_{\Rightarrow} \leq p_{\partial(G)}(|G|)$*

The second direction we are considering concerns methods to keep modal impredicativity under control by more traditional static methods in the spirit of light logics. Consider the equivalence between  $!A$  and  $!A \otimes A$ . As already pointed out, this equivalence is somehow necessary to get modal impredicativity. So, controlling it means controlling modal impredicativity. Now, suppose that the above equivalence holds *but*  $A$  only contains instances of the  $!$  operator which are *intrinsically different* from the top-level one in  $!A$ . Morally, this would imply that even if a contraction node is “in”  $A$ , it cannot communicate with the box in  $!A$ , because they are of a different nature. This way

Figure 15: The proof net  $G_{\circ[f, g_1, \dots, g_n]}$ 

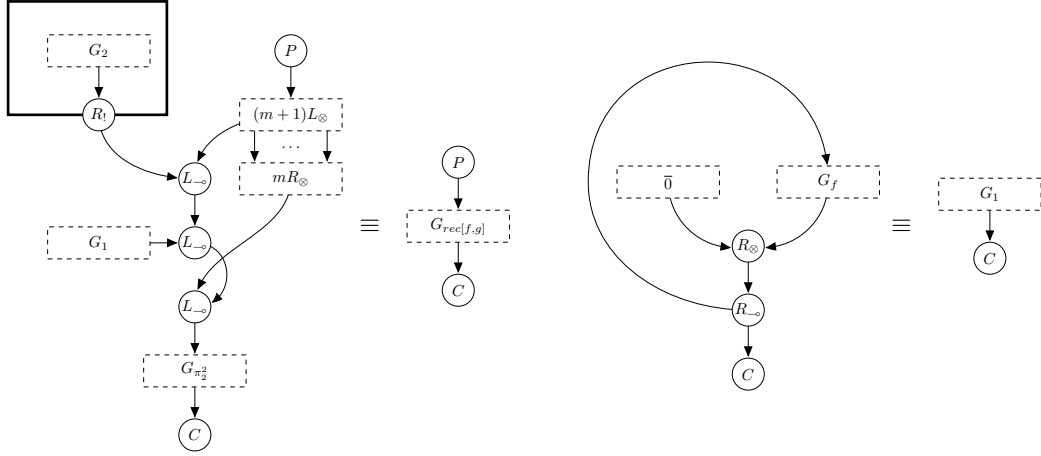
modal impredicativity would be under control. But the question is: how to distinguish different instances of  $!$  from each other? One (naive) answer is the following: consider a generalization of (multiplicative and exponential) Linear Logic where syntactically different copies  $!_{a_1}, !_{a_2}, \dots$  of the modal operator  $!$  are present. As an example, take the set  $\{!_n\}_{n \in \mathbb{N}}$ . Then, impose the following constraint:  $!_a A$  is a *legal* formula only if the operators  $!_{b_1}, \dots, !_{b_n}$  appearing in  $A$  are all different from  $a$ , i.e., if  $a \notin \{b_1, \dots, b_n\}$ . We strongly believe that this way a system enjoying properties similar to those of predicative recurrence schemes [8] can be obtained.

## 6 Conclusions

We described modal impredicativity and a concrete tool — superlazy reduction — that controls it. Superlazy reduction on pure proof nets greatly influences the expressive power of the underlying computational model: from a Turing complete model we go down to first-order primitive recursive functions.

In a sentence, we learn that the expressive power of a programming language system can be controlled by acting on the dynamics (i.e., the underlying reduction relation) without touching the statics (i.e., the language into which programs are written). To get the complete picture, however, we still need tools to predict which set of programs will be useful from a computational point of view.

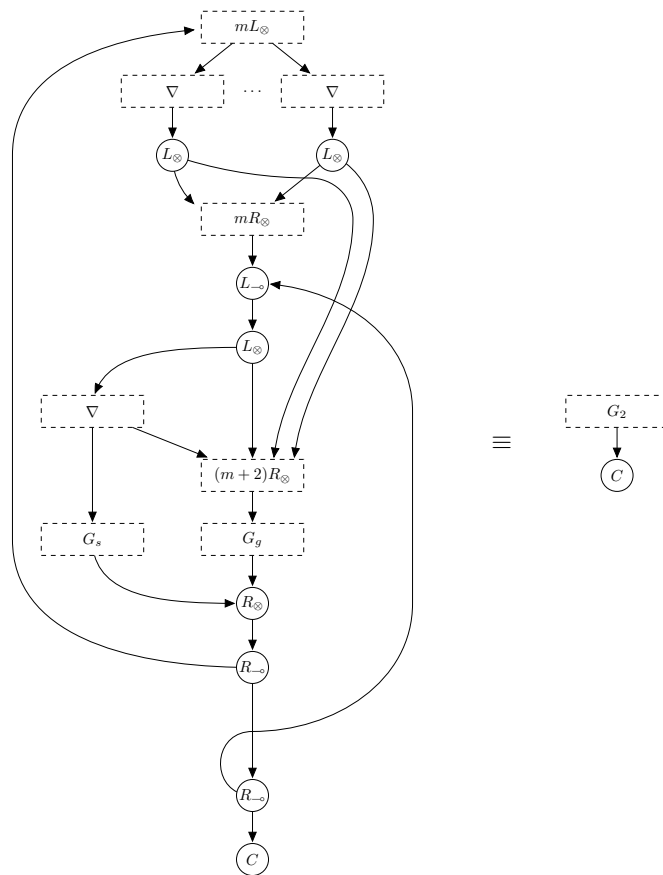
This could have potential applications in the field of implicit computational complexity, where one aims at designing programming languages and logical systems corresponding to complexity classes. Indeed, the impact of ICC in applications crucially depends on the intensional expressivity of the proposed systems: one should be able to write programs naturally.

Figure 16: The proof nets  $G_{\text{rec}[f,g]}$  and  $G_1$ 

## References

- [1] Ugo Dal Lago. The geometry of linear higher-order recursion. In *Proc. 20th Annual Symposium on Logic in Computer Science*, pages 366–375. IEEE Computer Society, 2005.
- [2] Solomon Feferman. Predicativity. In S. Shapiro, editor, *The Oxford Handbook of the Philosophy of Mathematics and Logic*, pages 590–624. Oxford University Press, 2005.
- [3] J.-Y. Girard, A. Scedrov, and P. Scott. Bounded linear logic: A modular approach to polynomial time computability. *Theoretical Computer Science*, 97:1–66, 1992.
- [4] Jean-Yves Girard. Geometry of interaction 1: interpretation of system F. In *Proc. Logic Colloquium '88*, pages 221–260, 1989.
- [5] Jean-Yves Girard. Light linear logic. *Information and Computation*, 143(2):175–204, 1998.
- [6] Martin Hofmann. Linear types and non-size-increasing polynomial time computation. In *Proceedings of the 14th IEEE Symposium on Logic in Computer Science*, pages 464–473, 1999.
- [7] Yves Lafont. Soft linear logic and polynomial time. *Theoretical Computer Science*, 318:163–180, 2004.
- [8] Daniel Leivant. Stratified functional programs and computational complexity. In *Proceedings of 20th ACM Symposium on Principles of Programming Languages*, pages 325–333, 1993.
- [9] Daniel Leivant. Ramified recurrence and computational complexity iii: Higher type recurrence and elementary complexity. *Ann. Pure Appl. Logic*, 96(1-3):209–229, 1999.
- [10] Daniel Leivant and Norman Danner. Stratified polymorphism and primitive recursion. *Math. Struct. in Comp. Science*, 9:507 — 522, 1999.
- [11] Daniel Leivant and Jean-Yves Marion. A characterization of alternating log time by ramified recurrence. *Theor. Comput. Sci.*, 236(1-2):193–208, 2000.
- [12] Gianfranco Mascari and Marco Pedicini. Head linear reduction and pure proof net extraction. *Theor. Comput. Sci.*, 135(1):111–137, 1994.
- [13] Harold Simmons. Tiering as a recursion technique. *Bulletin of Symbolic Logic*, 11(3):321–350, 2005.



Figure 17: The proof net  $G_2$