Linear Additives

Gianluca Curzi
University of Turin
Torino, Italy
curzi@di.unito.it

We introduce LAM, a subsystem of IMALL_2 with restricted additive rules able to manage duplication linearly, called *linear additive rules*. LAM is presented as the type assignment system for a calculus endowed with copy constructors, which deal with substitution in a linear fashion. As opposed to the standard additive rules, the linear additive rules do not affect the complexity of term reduction: typable terms of LAM enjoy linear strong normalization. Moreover, a mildly weakened version of cut-elimination for this system is proven which takes a cubic number of steps. Finally, we define a sound translation from LAM's proofs into IMLL_2's linear lambda terms, and we study its complexity.

1 Introduction

Linear Logic (LL) is a refinement of both classical and intuitionistic logic introduced by Girard in [13]. A central role in LL is played by the *exponential* modalities ! and ?, which give a logical status to the structural rules of weakening and contraction. The exponentials allow us to discriminate between *linear* resources, consumed exactly once, and *non-linear* resources, reusable at will. Moreover, since the uncontrolled use of the structural rules is forbidden, conjunction and disjunction come with two distinct presentations: the *multiplicative* version (resp. $\otimes$ and $⅋$) and the *additive* one (resp. & and $\oplus$).

The presence of multiple formulations of the same connective has prompted the analysis of specific *fragments* of Linear Logic, i.e. subsystems of LL that illustrate the behavior of a specific group of connectives. The simplest fragment of LL is MLL (*Multiplicative Linear Logic*), i.e. the modal-free subsystem of LL with inference rules for $\otimes$, $⅋$. Another one is MALL (*Multiplicative Additive Linear Logic*), obtained by extending MLL with *additive rules*, i.e. the inference rules for &, $\oplus$.

As pointed out in [19], according to the "computation-as-normalization" paradigm, the additive rules of LL can be used to express non-deterministic program executions. This intuition has been further investigated in the field of ICC (*Implicit Computational Complexity*), a branch of computational complexity devising calculi that abstract from machine models and characterize complexity classes without imposing "external" resource bounds. In this setting, several variants of the additive rules able to express non-deterministic computation more explicitly have been proposed to capture the class NP. Examples are [19, 11, 20], all based on *light logics*, i.e. subsystems of (second-order) LL with weaker exponential rules that induce a complexity bound on proof normalization.

Using variants of the additive rules to characterize the non-deterministic polytime problems raises some issues, because these inference rules affect the complexity of cut-elimination/normalization, which may require an exponential cost. A standard approach to circumvent this fact is to focus on a specific cut-elimination strategy called *lazy* [12], which "freezes" those commuting cut-elimination steps that involve duplication of (sub)proofs. A similar technique can be found in [11], where the type system STA_+ is introduced to capture the complexity class NP in the style of ICC. STA_+ extends *Soft Type Assignment* [10] with the *sum rule* below

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : A}{\Gamma \vdash M + N : A}\, \mathit{sum} \tag{1}$$

U. Dal Lago and V. de Paiva (Eds.) Second Joint International
Workshop on Linearity & Trends in Linear Logic and Applications.
EPTCS 353, 2021, pp. 74–93, doi:10.4204/EPTCS.353.4

and the non-deterministic reductions $M \leftarrow M+N \rightarrow N$ for the choice operator $+$. The sum rule is close to the additive rules, and suffers the same drawbacks. Consequently, to prove that STA_+ is sound for NP, a particular reduction strategy is needed to avoid exponential computations.

In this paper we present a different solution to the complexity-theoretical issues caused by the additive rules. We start focusing on the second-order intuitionistic formulation of MALL, i.e. IMALL_2. We shall look at the latter as a type system, essentially by considering formulas as types and by decorating logical derivations with λ-terms endowed with pairs and projections. The analysis of the non-linear features of the additive rules leads to the new type system LAM (*Linearly Additive Multiplicative Type Assignment*). This is a subsystem of IMALL_2 obtained by imposing some conditions on types and by replacing the standard additive rules with weaker versions, called *linear additive rules*. To some extent, LAM is more expressive than the "lazy" restriction of IMALL_2. Indeed, the linear additive rules allow some forms of duplication that lazy evaluation forbids. Nonetheless, these special additive rules are able to maintain control on the complexity of normalization, preventing exponential explosions and recovering *linear strong* normalization.

The cut-elimination rules for LAM are constrained to copy-cat the reduction rules on terms, making the cut rule no longer admissible. We then identify a class of types, called $\forall$-*lazy*, whose derivations can always be turned into cut-free ones in cubic time. This result is analogous to Girard's restricted (lazy) cut-elimination theorem for MALL in [12], but somehow more permissive.

Last, following essentially [7], we introduce a computationally sound translation mapping a derivation of LAM into a linear λ-term of IMLL_2 whose size can be exponentially bigger than the source derivation. The translation exploits the mechanisms of linear duplication and erasure studied by Mairson and Terui in [17, 18], and shows how LAM is able to express such mechanisms in a very compact and elegant way.

2 From Standard Additives to Linear Additives

In this section we briefly recall the $(\multimap, \&, \forall)$ fragment of IMALL_2 as a type system, and we show how nesting instances of the additive rules in a derivation produces an exponential blow up in normalization. To circumvent this issue we introduce *linear additives*, weaker versions of the standard additives permitting restricted forms of duplication without affecting the complexity of normalization.

2.1 The System IMALL_2

We present IMALL_2 as a type assignment system for the calculus $\Lambda_{\pi,\langle\rangle}$, whose terms are defined by the following grammar:

$$M := x \mid \lambda x.M \mid MM \mid \langle M,M\rangle \mid \pi_1(M) \mid \pi_2(M) \tag{2}$$

where x is taken from a denumerable set of variables. The set of free variables of a term M is written $FV(M)$, and the meta-level substitution for terms is denoted $M[N/x]$. The *size* $|M|$ of a term M is inductively defined as follows:

$$\begin{aligned} |x| &\triangleq 1 \\ |\lambda x.M| \triangleq |\pi_i(M)| &\triangleq |M|+1 \qquad\qquad i\in\{1,2\} \\ |MN| \triangleq |\langle M,N\rangle| &\triangleq |M|+|N|+1. \end{aligned} \tag{3}$$

The *one-step relation* $\rightarrow_\beta$ is the binary relation over $\Lambda_{\pi,\langle\rangle}$ defined by:

$$(\lambda x.M)N \rightarrow_\beta M[N/x] \qquad\qquad \pi_i\langle M_1, M_2\rangle \rightarrow_\beta M_i \qquad i \in \{1,2\} \tag{4}$$

Its reflexive and transitive closure is $\rightarrow^*_\beta$. If M β-reduces to N in exactly n steps we write $M \rightarrow^n_\beta N$. As usual, a λ-term is in (or is a) *normal form* whenever no reduction rule applies to it.

The set $\Theta_\&$ of types of IMALL_2 is generated by the following grammar:

$$A := \alpha \mid A \multimap A \mid A \,\&\, A \mid \forall\alpha.A \tag{5}$$

where α belongs to a denumerable set of type variables. The set of free type variables of a type A is written $FV(A)$, and the standard meta-level substitution for types is denoted $A\langle B/\alpha\rangle$. A type A is *closed* if $FV(A) = \emptyset$. The *size* $|A|$ of a type A is inductively defined as follows:

$$\begin{aligned} |\alpha| &\triangleq 1 \\ |A \multimap B| &\triangleq |A \,\&\, B| \triangleq |A| + |B| + 1 \\ |\forall\alpha.A| &\triangleq |A| + 1. \end{aligned} \tag{6}$$

We define the notions of *positive subtype* and of *negative subtype* of a type A by simultaneous induction on the structure of A:

- A is a positive subtype of itself;
- if $B \multimap C$ is a positive (resp. negative) subtype of A, then B is a negative (resp. positive) subtype of A, and C is a positive (resp. negative) subtype of A;
- if $B \,\&\, C$ is a positive (resp. negative) subtype of A, then so are B and C;
- if $\forall\alpha.B$ is a positive (resp. negative) subtype of A, then so is B.

We say that a type A has *positive (resp. negative) occurrences of* $\forall$ if there exists a positive (resp. negative) subtype of A with shape $\forall\alpha.B$. We define in a similar way positive and negative occurrences of $\multimap$ and &.

The system IMALL_2 (*Intuitionistic Second-Order Multiplicative Additive Linear Logic*) is displayed in Figure 1, where &R and &Li are the *additives rules*. It derives *judgements* with form $\Gamma \vdash M : A$, where $M \in \Lambda_{\pi,\langle\rangle}$, $A \in \Theta_\&$ and Γ is a *context*, i.e. a finite multiset of *assumptions* $x : A$. The system requires the *linear constraint* $FV(\Gamma) \cap FV(\Delta) = \emptyset$ in both *cut* and $\multimap$L, where $FV(\Gamma)$ denotes the set of all free type variables in Γ. With $\mathscr{D} \triangleleft \Gamma \vdash M : A$ we denote a derivation $\mathscr{D}$ of $\Gamma \vdash M : A$, and in this case we say that M is an *inhabitant* of A (or that A is *inhabited* by M). The *size* $|\Gamma|$ of a context $\Gamma = x_1 : A_1, \ldots, x_n : A_n$ is $\sum_{i=1}^n |A_i|$, and the *size* $|\mathscr{D}|$ of a derivation $\mathscr{D}$ is the number of its rules applications.

We recall that IMLL_2 (*Intuitionistic Second-Order Multiplicative Linear Logic*) is obtained from IMALL_2 by excluding the additive rules from Figure 1. It gives a type *exactly* to the class of *linear λ-terms* (see [14, 18]), i.e. those terms M from the standard λ-calculus such that:

(i) each free variable of M occurs in it exactly once, and

(ii) for each subterm $\lambda x.N$ of M, x occurs in N exactly once.

Tensors ($\otimes$) and units ($\mathbf{1}$) can be introduced in IMLL_2 (and hence in IMALL_2) by means of second-order definitions:

$$\begin{array}{ll} \mathbf{1} \triangleq \forall\alpha.(\alpha \multimap \alpha) & A \otimes B \triangleq \forall\alpha.(A \multimap B \multimap \alpha) \multimap \alpha \\ \mathbf{I} \triangleq \lambda x.x & M \otimes N \triangleq \lambda z.zMN \\ \texttt{let}\ M\ \texttt{be}\ \mathbf{I}\ \texttt{in}\ N \triangleq MN & \texttt{let}\ M\ \texttt{be}\ x \otimes y\ \texttt{in}\ N \triangleq M(\lambda x.\lambda y.N). \end{array} \tag{7}$$

$$\frac{}{x : A \vdash x : A}\ ax \qquad \frac{\Gamma \vdash N : A \qquad \Delta, x : A \vdash M : C}{\Gamma, \Delta \vdash M[N/x] : C}\ cut$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \multimap B}\ \multimap\mathrm{R} \qquad \frac{\Gamma \vdash N : A \qquad \Delta, x : B \vdash M : C}{\Gamma, y : A \multimap B, \Delta \vdash M[yN/x] : C}\ \multimap\mathrm{L}$$

$$\frac{\Gamma \vdash M_1 : A_1 \qquad \Gamma \vdash M_2 : A_2}{\Gamma \vdash \langle M_1, M_2 \rangle : A_1 \,\&\, A_2}\ \&\mathrm{R} \qquad \frac{\Gamma, x_i : A_i \vdash M : C \quad i \in \{1,2\}}{\Gamma, y : A_1 \,\&\, A_2 \vdash M[\pi_i(y)/x_i] : C}\ \&\mathrm{L}i$$

$$\frac{\Gamma \vdash M : A\langle \gamma/\alpha \rangle \quad \gamma \notin \mathrm{FV}(\Gamma)}{\Gamma \vdash M : \forall \alpha.A}\ \forall\mathrm{R} \qquad \frac{\Gamma, x : A\langle B/\alpha \rangle \vdash M : C}{\Gamma, x : \forall \alpha.A \vdash M : C}\ \forall\mathrm{L}$$

Figure 1: The system IMALL_2.

Hence, the inference rules for $\otimes$ and $\mathbf{1}$, and the reduction rules

$$\begin{gathered} \mathtt{let}\ \mathbf{I}\ \mathtt{be}\ \mathbf{I}\ \mathtt{in}\ N \to_\beta N \\ \mathtt{let}\ M_1 \otimes M_2\ \mathtt{be}\ x_1 \otimes x_2\ \mathtt{in}\ N \to_\beta N[M_1/x_1, M_2/x_2] \end{gathered} \tag{8}$$

are derivable in IMLL_2.

2.2 Exponential Blowup and Lazy Cut-Elimination for IMALL_2

The additive rule &R in Figure 1 affects the complexity of normalization in IMALL_2, letting the size of typable terms and the number of their redexes grow exponentially during reduction. Definition 1 and Proposition 1 show an example.

Definition 1 (Nesting IMALL_2 terms)**.** Let $A \in \Theta_\&$ and $M \in \Lambda_{\pi,\langle\rangle}$. For all $n \in \mathbb{N}$, we define $A_{[n]}$ and $M_{[n]}$ as follows:

$$A_{[n]} \triangleq \begin{cases} A & n = 0 \\ A_{[n-1]} \,\&\, A_{[n-1]} & n > 0 \end{cases} \qquad M_{[n]} \triangleq \begin{cases} M & n = 0 \\ \langle M_{[n-1]}, M_{[n-1]} \rangle & n > 0 \end{cases} \tag{9}$$

Notice that $\langle M, M \rangle_{[n]} = M_{[n+1]}$. Moreover, for all $n \in \mathbb{N}$ we define $\mathtt{add}^x_n$ as follows:

$$\mathtt{add}^x_n \triangleq \begin{cases} x & n = 0 \\ (\lambda y.\mathtt{add}^y_{n-1})\langle x, x \rangle & n > 0 \end{cases} \tag{10}$$

Proposition 1 (Exponential blow up)**.** *Let $A \in \Theta_\&$, and let $M \in \Lambda_{\pi,\langle\rangle}$ be of type A. For all $n \in \mathbb{N}$:*

1. $\vdash \lambda x.\mathtt{add}^x_n : A \multimap A_{[n]}$ *is derivable in* IMALL_2*;*
2. $(\lambda x.\mathtt{add}^x_n)M \to^{n+1}_\beta M_{[n]}$*, where* $|(\lambda x.\mathtt{add}^x_n)M| = \mathscr{O}(n \cdot |M|)$ *and* $|M_{[n]}| = \mathscr{O}(2^{n \cdot |M|})$.

$$\dfrac{\begin{array}{c}\mathscr{D}\\ \Gamma \vdash M : A\end{array} \qquad \dfrac{\begin{array}{c}\vdots\\ \Delta, x : A \vdash N : B\end{array} \qquad \begin{array}{c}\vdots\\ \Delta, x : A \vdash P : C\end{array}}{\Delta, x : A \vdash \langle N, P\rangle : B \,\&\, C}\ \&\mathrm{R}}{\Gamma, \Delta \vdash \langle N[M/x], P[M/x]\rangle : B \,\&\, C}\ \mathit{cut}$$

$$\rightsquigarrow$$

$$\dfrac{\dfrac{\begin{array}{c}\mathscr{D}\\ \Gamma \vdash M : A\end{array} \qquad \begin{array}{c}\vdots\\ \Delta, x : A \vdash N : B\end{array}}{\Gamma, \Delta \vdash N[M/x] : B}\ \mathit{cut} \qquad \dfrac{\begin{array}{c}\mathscr{D}\\ \Gamma \vdash M : A\end{array} \qquad \begin{array}{c}\vdots\\ \Delta, x : A \vdash P : C\end{array}}{\Gamma, \Delta \vdash P[M/x] : C}\ \mathit{cut}}{\Gamma, \Delta \vdash \langle N[M/x], P[M/x]\rangle : B \,\&\, C}\ \&\mathrm{R}$$

Figure 2: Commuting cut-elimination step (with term annotation) involving duplication of $\mathscr{D}$.

Proof. Concerning point 1, we prove by induction on n that $\lambda x.\mathtt{add}^x_n$ has type $A_{[k]} \multimap A_{[k+n]}$, for all $k \in \mathbb{N}$. If $n = 0$ then $\mathtt{add}^x_0 = x$, so that $\lambda x.x$ has type $A_{[k]} \multimap A_{[k]}$. Let us consider the case $n > 0$. By induction hypothesis, $\lambda y.\mathtt{add}^y_{n-1}$ has type $A_{[k+1]} \multimap A_{[k+n]}$. If x has type $A_{[k]}$ then $\langle x, x\rangle$ has type $A_{[k+1]}$ and $(\lambda y.\mathtt{add}^y_{n-1})\langle x, x\rangle$ has type $A_{[k+n]}$. Therefore, $\lambda x.\mathtt{add}^x_n = \lambda x.((\lambda y.\mathtt{add}^y_{n-1})\langle x, x\rangle)$ has type $A_{[k]} \multimap A_{[k+n]}$. Concerning point 2, it suffices to prove by induction on n that $\mathtt{add}^x_n \rightarrow^n_\beta x_{[n]}$. The base case is trivial. If $n > 0$ then $\mathtt{add}^x_n = (\lambda y.\mathtt{add}^y_{n-1})\langle x, x\rangle \rightarrow_\beta \mathtt{add}^y_{n-1}[\langle x, x\rangle / y]$. Since by induction hypothesis $\mathtt{add}^y_{n-1} \rightarrow^{n-1}_\beta y_{[n-1]}$ then $\mathtt{add}^y_{n-1}[\langle x, x\rangle / x] \rightarrow^{n-1}_\beta y_{[n-1]}[\langle x, x\rangle / y] = \langle x, x\rangle_{[n-1]} = x_{[n]}$. Finally, we notice that, for all $n \in \mathbb{N}$, it holds that $|\mathtt{add}^x_n| = (5 \cdot n) + 1$ and $|M_{[n]}| = 2^{n \cdot |M|} + \sum_{i=0}^{n-1} 2^i$. $\square$

Proposition 1 is about the complexity of normalization for IMALL_2's typable terms, but it can be easily restated for cut-elimination. To prevent the problem of exponential computation, Girard introduced in [12] the so-called *lazy cut-elimination*, a rewriting procedure for both MALL's proof nets and their second-order versions that requires only a linear number of steps. Intuitively, in the sequent calculus presentation of IMALL_2, lazy cut-elimination "freezes" the commuting cut-elimination steps applied to those instances of *cut* whose right premise is the conclusion of &R, as they involve duplication of (sub)proofs (see Figure 2). Since lazy cut-elimination cannot perform certain cut-elimination steps, it may fail to produce a cut-free proof. Nonetheless, Girard showed that a cut-elimination theorem for MALL_2's proof nets holds if we stick to those receiving $(\&, \exists)$-free types (see [12]). Following [18], we call these special types "lazy", and we define their intuitionistic counterparts as follows:

Definition 2 (Lazy types)**.** A type $A \in \Theta_\&$ is *lazy* if it contains no negative occurrences of $\forall$ and no positive occurrences of &.

Note that the restriction on negative occurrences of $\forall$ in the above definition is just to forbid the hiding of & connectives by $\forall$L in IMALL_2.

Lazy cut-elimination for IMALL_2 can be reformulated as a reduction on terms by introducing some form of sharing, e.g. explicit substitution (see [1]). The idea is to replace the meta-notation $M[N/x]$ with a term constructor of the following shape:

$$M\langle\langle x := N\rangle\rangle \tag{11}$$

and to add reduction rules able to introduce and perform substitution stepwise, such as:

$$\begin{aligned}
(\lambda x.M)N &\to M\langle\langle x := N\rangle\rangle \\
x\langle\langle x := N\rangle\rangle &\to N \\
y\langle\langle x := N\rangle\rangle &\to y && y \neq x \\
(\lambda y.M)\langle\langle x := N\rangle\rangle &\to \lambda y.(M\langle\langle x := N\rangle\rangle) && y \neq x \text{ and } y \notin FV(N) \\
(MP)\langle\langle x := N\rangle\rangle &\to M\langle\langle x := N\rangle\rangle P\langle\langle x := N\rangle\rangle \\
\pi_i(M)\langle\langle x := N\rangle\rangle &\to \pi_i(M\langle\langle x := N\rangle\rangle) && i \in \{1,2\} \\
\langle M, P\rangle\langle\langle x := N\rangle\rangle &\to \langle M\langle\langle x := N\rangle\rangle, P\langle\langle x := N\rangle\rangle\rangle.
\end{aligned} \tag{12}$$

Lazy reduction is then obtained by forbidding substitutions of terms in a pair, i.e. by ruling out the last rewriting rule of (12), since it mimics the cut-elimination step in Figure 2. As an example, consider the following typable terms of IMALL_2:

$$\begin{aligned}
M &\triangleq (\lambda x.\pi_1\langle x, x\rangle)N \\
M' &\triangleq (\lambda x.\langle x, x\rangle)N.
\end{aligned} \tag{13}$$

If we applied standard β-reduction to M we would obtain $\pi_1\langle N, N\rangle$, thus creating a (useless) copy of N. By contrast, thanks to explicit substitution, lazy reduction enables a better control on the parameter-passing mechanism and reduces M without making new copies of N:

$$(\lambda x.\pi_1\langle x, x\rangle)N \to \pi_1\langle x, x\rangle\langle\langle x := N\rangle\rangle \to x\langle\langle x := N\rangle\rangle \to N.$$

The above reasoning does not apply to M', as the substitution $\langle x, x\rangle\langle\langle x := N\rangle\rangle$ would remain unperformed, so that lazy reduction may fail to produce a substitution-free term, in general. Again, as in the case of cut-elimination, when a term has lazy type in IMALL_2 each pair $\langle P, Q\rangle$ occurring in it eventually turns into a redex $\pi_i\langle P, Q\rangle$, so that all suspended substitutions are sooner or later carried out. Going back to the terms in (13), notice that M has lazy type in IMALL_2 whenever N has, while only (non-lazy) types with shape $A \& A$ can be assigned to M'.

Explicit substitution and other forms of sharing play a fundamental role in the study of reasonable cost models of the untyped λ-calculus, where β-reduction and meta-level substitution cause size explosions similar to Proposition 1 (see e.g. [2, 3, 16]). As an example, in [3] Accattoli and Dal Lago have shown that λ-terms with explicit substitutions can be managed in reasonable (i.e. polynomial) time, without having to unfold the sharing (that would re-introduce an exponential size blow up). Explicit substitutions have been introduced to cover the gap between the λ-calculus and concrete implementations, but they can also produce pathological behaviors in a typed setting, as shown by Melliès in [21].

2.3 Linear Additives

To prevent the exponential blow up discussed in the previous section we introduce weaker versions of the standard additive rules called *linear additive rules*, which give types to `copy` constructs. The linear additive rules are displayed in Figure 3, where V is a *value*, i.e. a closed and normal term free from instances of `copy` and projections, and the types A, A_1, A_2 are closed and $\forall$-*lazy*, according to the following definition:

Definition 3 ($\forall$-lazy types)**.** A type $A \in \Theta_\&$ is $\forall$-*lazy* if it contains no negative occurrences of $\forall$. We say that $x_1 : A_1, \ldots, x_n : A_n \vdash M : B$ is a $\forall$-*lazy judgement* if $A_1 \multimap \ldots \multimap A_n \multimap B$ is a $\forall$-lazy type. Finally, we say that $\mathscr{D} \triangleleft \Gamma \vdash M : A$ is a $\forall$-*lazy derivation* if $\Gamma \vdash M : A$ is a $\forall$-lazy judgement.

$$\frac{\vdash M_1 : A_1 \qquad \vdash M_2 : A_2}{\vdash \langle M_1, M_2\rangle : A_1 \,\&\, A_2}\ \&\mathrm{R0} \qquad \frac{\Gamma, x_i : A_i \vdash M : C \quad i \in \{1,2\}}{\Gamma, y : A_1 \,\&\, A_2 \vdash M[\pi_i(y)/x_i] : C}\ \&\mathrm{L}i$$

$$\frac{x_1 : A \vdash M_1 : A_1 \qquad x_2 : A \vdash M_2 : A_2 \qquad \vdash V : A}{x : A \vdash \mathtt{copy}^V x \ \mathtt{as}\ x_1, x_2 \ \mathtt{in}\ \langle M_1, M_2\rangle : A_1 \,\&\, A_2}\ \&\mathrm{R1}$$

Figure 3: Linear additive rules, where $V \in \mathscr{V}$ and A, A_1, A_2 are closed and $\forall$-lazy types.

The reduction rule corresponding to copy will be of the following form:

$$\mathtt{copy}^V U \ \mathtt{as}\ x_1, x_2 \ \mathtt{in}\ \langle M_1, M_2\rangle \to \langle M_1[U/x_1], M_2[U/x_2]\rangle \qquad U, V \in \mathscr{V} \tag{14}$$

where $\mathscr{V}$ is the set of all values. Notice that the inference rule &R0 in Figure 3 is introduced to let the above reduction rule preserve types, i.e. to assure Subject reduction. Intuitively, the operator copy behaves as a suspended substitution, quite like in lazy reduction discussed in Section 2.2: the crucial difference is that lazy reduction forbids *any* substitution of a term N in a pair $\langle M_1, M_2\rangle$, while copy allows it when N is turned into a value U.

Hence, some limited forms of duplication are permitted by the linear additive rules. Nonetheless, they do not affect the complexity of normalization. On the one hand, indeed, the reduction rule in (14) can only copy values, i.e. normal terms, so that no redex is duplicated. This allows linear time normalization. On the other hand, since any $\forall$-*lazy* type A is inhabited by finitely many values (see Proposition 3.2), by taking V in Figure 3 as the largest one of that type, we enable the size of $\mathtt{copy}^V$ to bound the size of the new copy of U produced by applying (14) (since U has type A). This lets reduction strictly decrease the size of terms, recovering linear space normalization.

As a final remark, let us observe that the linear additive rule &R1 introduces a seemingly severe restriction: context-sharing is permitted for premises having *exactly* one assumption. This constraint has no real impact on the algorithmic expressiveness of linear additives, since a general inference rule with premises sharing multiple assumptions can be easily derived by exploiting the definitions in (7). Indeed, tensors are able to turn a context with n assumptions $x_1 : A_1, \dots, x_n : A_n$ into one with single assumption $x : A_1 \otimes \dots \otimes A_n$. Narrowing context-sharing has its benefits: it avoids heavy notation produced by several occurrences of the construct copy, each one expressing the sharing of a single assumption.

3 A Type Assignment With Linear Additives

In this section we introduce *Linearly Additive Multiplicative Type Assignment*, LAM for short. It is a subsystem of IMALL_2 endowed with the linear additive rules in Figure 3. Thanks to the controlled use of substitution, these rules can be freely nested in LAM without incurring exponential normalization.

3.1 The System LAM

The following grammar generates *raw terms*:

$$M, N := x \mid \lambda x.M \mid MM \mid \langle M, M\rangle \mid \pi_i(M) \mid \mathtt{copy}^U M \ \mathtt{as}\ x, y \ \mathtt{in}\ \langle M, M\rangle \tag{15}$$

$$U,V := x \mid \lambda x.U \mid UU \mid \langle U,U \rangle \tag{16}$$

where $\mathtt{copy}^U M \mathtt{\ as\ } x,y \mathtt{\ in\ } \langle P,Q \rangle$ binds both x in P and y in Q. A *value* is any closed raw term generated by the grammar (16) that is *normal* with respect to the reduction step $(\lambda x.U)V \to U[V/x]$. The set of all values is denoted $\mathscr{V}$. A raw term is a *term* if any occurrence of the $\mathtt{copy}^U$ operator in it is such that $U \in \mathscr{V}$. The set of terms is denoted $\Lambda_{\mathtt{copy}}$. We extend the definition of $|M|$ in (3) to the new clause:

$$|\mathtt{copy}^U M \mathtt{\ as\ } x,y \mathtt{\ in\ } \langle P,Q \rangle| = |U| + |M| + |\langle P,Q \rangle| + 1$$

The *one-step reduction* $\to$ on $\Lambda_{\mathtt{copy}}$ extends $\to_\beta$ in (4) with the reduction rule in (14), and applies in any context. Its reflexive and transitive closure is $\to^*$. If M reduces to N in exactly n steps we write $M \to^n N$. A term is in (or is a) *normal form* if no reduction applies to it.

The system LAM is essentially obtained by replacing the standard additive rules of IMALL_2 with the linear additive rules in Figure 3, and by imposing a mild requirement on the inference rules $\multimap$L and $\forall$R that plays an crucial role to assure a weak form of cut-elimination.

Definition 4 (The system LAM)**.** The system LAM (*Linearly Additive Multiplicative Type Assignment*) is the type assignment for $\Lambda_{\mathtt{copy}}$ obtained by extending IMLL_2 with the *linear additive rules* in Figure 3, and by imposing the following *closure conditions*:

(i) no instance of $\multimap$L has conclusion $\Delta, y : A \multimap B, \Gamma \vdash M : C$ with $FV(B) = \emptyset$ and $FV(A) \neq \emptyset$;

(ii) no instance of $\forall$R has conclusion $\Gamma \vdash M : \forall\alpha.A$ with $FV(\forall\alpha.A) = \emptyset$ and $FV(\Gamma) \neq \emptyset$.

As it stands, LAM is able to linearly bound the number of steps required to normalize typable terms, essentially because we can only duplicate values, which are redex-free. However, the system has no control over the size of terms, which may grow exponentially during normalization. What we need is to bound the size of the new copies of values that are produced by applying the reduction rule (14). This is the goal of Proposition 3 and Remark 1.

Definition 5 (η-expansion)**.** Given $\mathscr{D} \lhd \Gamma \vdash M : A$ a $\forall$-lazy and *cut-free* derivation, we say that $\mathscr{D}$ is η-*expanded* if all its axioms are *atomic*, i.e. of the form $x : \alpha \vdash x : \alpha$ for some type variable α. In this case, M is called a η*-long normal form* (note that M must be a normal form).

We now state some basic properties about $\forall$-lazy and η-expanded derivations.

Proposition 2 ($\forall$-lazy derivations)**.**

1. *If a premise of one among the rules $\multimap$R, $\multimap$L and $\forall$R is not $\forall$-lazy, then neither is its conclusion. Moreover, the conclusions of $\&R1$, $\&Li$ and $\forall L$ are not $\forall$-lazy, while the conclusion of $\&R0$ is.*
2. *Let $\mathscr{D} \lhd \Gamma \vdash M : A$ be $\forall$-lazy and cut-free, then:*
   - *$\mathscr{D}$ contains no instances of $\&R1$, $\&Li$ and $\forall L$;*
   - *M is normal and contains no occurrences of $\mathtt{copy}$ and π_i;*
   - *if $\Gamma = \emptyset$ then $M \in \mathscr{V}$.*

Proposition 3 (η-expanded derivations)**.** *Let $\mathscr{D} \lhd \Gamma \vdash M : A$ be $\forall$-lazy and η-expanded. Then:*

1. *$|M| \leq |\Gamma| + |A| \leq 2 \cdot |\mathscr{D}|$;*
2. *if $\mathscr{D}' \lhd \Gamma \vdash N : A$ is cut-free for some $N \in \Lambda_{\mathtt{copy}}$, then both $|N| \leq |M|$ and $|\mathscr{D}'| \leq |\mathscr{D}|$.*

Proof. By Definition 5 and Proposition 2.2, $\mathscr{D}$ is cut-free and without instances of &R1, &Li, $\forall$L, with M normal and free from `copy` constructs. Point 1 is a straightforward induction on $\mathscr{D}$. Concerning point 2, we have $|\mathscr{D}'| \leq |\mathscr{D}|$, because $\mathscr{D}$ is η-expanded. Now, notice that $|N| + \forall(\Gamma, A) = |\mathscr{D}'| + \forall_{ax}(\mathscr{D}')$, where $\forall(\Gamma, A)$ denotes the number of occurrences of $\forall$ in Γ, A, and $\forall_{ax}(\mathscr{D}')$ denotes the number of occurrences of $\forall$ in the conclusions of the instances of ax in $\mathscr{D}'$. We have, $|N| + \forall(\Gamma, A) = |\mathscr{D}'| + \forall_{ax}(\mathscr{D}') \leq |\mathscr{D}| + \forall_{ax}(\mathscr{D}) = |M| + \forall(\Gamma, A)$, which implies $|N| \leq |M|$. $\square$

Remark 1. Given $\mathscr{D} \lhd \Gamma \vdash M : A$ a $\forall$-lazy and cut-free derivation, we can always construct a η-expanded derivation $\mathscr{D}^* \lhd \Gamma \vdash M^* : A$. Moreover, Proposition 3.2 implies both $|N| \leq |M^*|$ and $|\mathscr{D}'| \leq |\mathscr{D}^*|$, for any cut-free derivation $\mathscr{D}' \lhd \Gamma \vdash N : A$. Henceforth, w.l.o.g. we shall assume that the derivation of the premise $\vdash V : A$ of &R1 is η-expanded.

Remark 1 prevents the increase of size during normalization, since the size of V in &R1 is always bigger than then size of any value U of type A, and so the construct $\mathtt{copy}^V$ bounds the size of the new copy of U produced by applying the reduction rule in (14).

3.2 Linear Additives Prevent the Exponential Blowup

We can observe the benefits of moving from the standard additive rules to the linear additive rules as soon as we adapt the constructions in Definition 1 to LAM.

Definition 6 (Nesting LAM terms)**.** Let $V \in \mathscr{V}$ and $k \in \mathbb{N}$. For all $n \in \mathbb{N}$ we define $\mathtt{ladd}_n^{x,V_{[k]}}$ as follows:

$$\mathtt{ladd}_n^{x,V_{[k]}} \triangleq \begin{cases} x & n = 0 \\ (\lambda y.\mathtt{ladd}_{n-1}^{y,V_{[k+1]}})(\mathtt{copy}^{V_{[k]}} x \;\mathtt{as}\; x_1, x_2 \;\mathtt{in}\; \langle x_1, x_2 \rangle) & n > 0 \end{cases}$$

where $V_{[k]}$ is as in (9).

The following proposition states that nesting instances of &R1 in a derivation of LAM produces no exponential blow up.

Proposition 4 (Linearity of LAM)**.** *Let A be a closed and $\forall$-lazy type, and let $U, V \in \mathscr{V}$ be inhabitants of A, with V a η-long normal form. For all $n \in \mathbb{N}$:*

1. $\vdash \lambda x.\mathtt{ladd}_n^{x,V} : A \multimap A_{[n]}$ *is derivable in* LAM*;*
2. $(\lambda x.\mathtt{ladd}_n^{x,V})U \to^{2n+1} U_{[n]}$*, where* $|(\lambda x.\mathtt{ladd}_n^{x,V})U| > |U_{[n]}| > 2n+1$.

Proof. Concerning point 1, we prove by induction on n that $\lambda x.\mathtt{ladd}_n^{x,V_{[k]}}$ has type $A_{[k]} \multimap A_{[k+n]}$, for all $k \in \mathbb{N}$. If $n = 0$ then $\mathtt{ladd}_n^{x,V_{[k]}} = x$, so that $\lambda x.x$ has type $A_{[k]} \multimap A_{[k]}$. Let us consider the case $n > 0$. By induction hypothesis, $\lambda y.\mathtt{ladd}_{n-1}^{y,V_{[k+1]}}$ has type $A_{[k+1]} \multimap A_{[k+n]}$. If x, x_1, x_2 have type $A_{[k]}$ then $\mathtt{copy}^{V_{[k]}} x \;\mathtt{as}\; x_1, x_2 \;\mathtt{in}\; \langle x_1, x_2 \rangle$ has type $A_{[k+1]}$ and $(\lambda y.\mathtt{ladd}_{n-1}^{y,V_{[k+1]}})(\mathtt{copy}^{V_{[k]}} x \;\mathtt{as}\; x_1, x_2 \;\mathtt{in}\; \langle x_1, x_2 \rangle)$ has type $A_{[k+n]}$. Therefore, $\lambda x.\mathtt{ladd}_n^{x,V_{[k]}}$ has type $A_{[k]} \multimap A_{[k+n]}$. Concerning point 2, we show by induction on n that $(\mathtt{ladd}_n^{x,V_{[k]}})[U_{[k]}/x] \to^{2n} U_{[n+k]}$, for all $k \in \mathbb{N}$. The base case is trivial. If $n > 0$ then, since $U_{[k]} \in \mathscr{V}$, we have:

$$\begin{aligned} (\mathtt{ladd}_n^{x,V_{[k]}})[U_{[k]}/x] &= (\lambda y.\mathtt{ladd}_{n-1}^{y,V_{[k+1]}})(\mathtt{copy}^{V_{[k]}} U_{[k]} \mathtt{as}\; x_1, x_2 \;\mathtt{in}\; \langle x_1, x_2 \rangle) \\ &\to (\lambda y.\mathtt{ladd}_{n-1}^{y,V_{[k+1]}})\langle U_{[k]}, U_{[k]} \rangle \\ &\to \mathtt{ladd}_{n-1}^{y,V_{[k+1]}}[\langle U_{[k]}, U_{[k]} \rangle / y] = \mathtt{ladd}_{n-1}^{y,V_{[k+1]}}[U_{[k+1]}/y] \end{aligned}$$

which reduces in $2(n-1)$ steps to $U_{[k+n]}$ by induction hypothesis. Finally, we notice that $|\mathtt{ladd}_n^{x,V_{[k]}}| = 7n + \sum_{i=0}^{n-1} |V_{[k+i]}| + 1$ and $|U_{[n]}| = 2^{n\cdot|U|} + \sum_{i=0}^{n-1} 2^i$, for all $n \in \mathbb{N}$. Since V is a η-long normal form of A, we have $|V| \geq |U|$ by Proposition 3.2, and the conclusion follows. $\square$

4 Computational and Proof-Theoretical Properties of LAM

In Section 3.2 we have shown with the help of a key example how linear additives prevent exponential explosions in normalization. We now investigate further this point by proving that LAM enjoys both linear strong normalization (Theorem 8) and a mildly weakened cubic cut-elimination property (Theorem 13).

4.1 Subject Reduction and Linear Strong Normalization

Linear strong normalization for LAM is achieved by proving that reduction preserves types and shrinks the size of typable terms, i.e. a slightly stronger version of Subject reduction. First, we need some straightforward preliminary lemmas:

Lemma 5 (Linearity)**.** *If $\mathscr{D} \lhd \Gamma, x : A \vdash M : C$ then x occurs exactly once in M.*

Lemma 6 (Generation)**.**

1. *If $\mathscr{D} \lhd \Gamma \vdash \lambda x.M : A$ then $A = \forall\alpha_1 \ldots \forall\alpha_n.(B \multimap C)$ and, by permuting some rules of $\mathscr{D}$, we obtain a derivation $\mathscr{D}'$ of $\Gamma, x : B \vdash M : C$, followed by an instance of $\multimap R$ and a sequence of $\forall R$.*
2. *If $\mathscr{D} \lhd \Gamma, x : \forall\alpha.A \vdash M[xN/y] : B$ then $\mathscr{D}$ contains an instance of $\forall L$ that introduces $x : \forall\alpha.A$.*
3. *If $\mathscr{D} \lhd \Gamma, x : A \multimap B \vdash M[xN/y] : C$ then $\mathscr{D}$ contains an instance of $\multimap L$ that introduces $x : A \multimap B$.*
4. *If $\mathscr{D} \lhd \Gamma \vdash \langle M_1, M_2\rangle : A$ then $\Gamma = \emptyset$, $A = B_1 \,\&\, B_2$, and the last rule of $\mathscr{D}$ is $\&R0$.*
5. *If $\mathscr{D} \lhd \Gamma, x : A_1 \,\&\, A_2 \vdash M[\pi_i(x)/x_i] : A$ then $\mathscr{D}$ contains an instance of $\&Li$ that introduces $x : A_1 \,\&\, A_2$.*
6. *If $\mathscr{D} \lhd \Gamma, x : B \vdash M[\mathtt{copy}^V x \ \mathtt{as}\ x_1, x_2 \ \mathtt{in}\ \langle N_1, N_2\rangle/y] : A$, then $\mathscr{D}$ contains an instance of $\&R1$ that introduces $x : B$.*

Theorem 7 (Subject reduction)**.** *If $\mathscr{D} \lhd \Gamma \vdash M_1 : A$ and $M_1 \to M_2$ then:*

- *$|M_2| < |M_1|$, and*
- *$\mathscr{D}^* \lhd \Gamma \vdash M_2 : A$, for some $\mathscr{D}^*$.*

Proof. We proceed by structural induction on $\mathscr{D}$. The crucial case is when the last rule of $\mathscr{D}$ is an instance of *cut* introducing the redex in M_1 that has been fired by the reduction step $M_1 \to M_2$. We just consider the case where $M_1 = P[\mathtt{copy}^U V \ \mathtt{as}\ x_1, x_2 \ \mathtt{in}\ \langle N_1, N_2\rangle/y]$ and $M_2 = P[\langle N_1[V/x_1], N_2[V/x_2]\rangle/y]$. We have:

$$\dfrac{\mathscr{D}_1 \lhd \vdash V : B \qquad \mathscr{D}_2 \lhd \Delta, x : B \vdash P[\mathtt{copy}^U x \ \mathtt{as}\ x_1, x_2 \ \mathtt{in}\ \langle N_1, N_2\rangle/y] : A}{\mathscr{D} \lhd \Delta \vdash P[\mathtt{copy}^U V \ \mathtt{as}\ x_1, x_2 \ \mathtt{in}\ \langle N_1, N_2\rangle/y] : A}\ cut$$

By applying Lemma 6.6, $\mathscr{D}_2$ must be of the following form:

$$\begin{array}{c}\dfrac{\mathscr{D}' \lhd x_1 : B \vdash N_1 : B_1 \qquad \mathscr{D}'' \lhd x_2 : B \vdash N_2 : B_2 \qquad \mathscr{D}''' \lhd \vdash U : B}{x : B \vdash \mathtt{copy}^U x \ \mathtt{as}\ x_1, x_2 \ \mathtt{in}\ \langle N_1, N_2\rangle : B_1 \,\&\, B_2}\ \&R1 \\ \vdots\,\gamma \\ \mathscr{D}_2 \lhd \Delta, x : B \vdash P[\mathtt{copy}^U x \ \mathtt{as}\ x_1, x_2 \ \mathtt{in}\ \langle N_1, N_2\rangle/y] : A\end{array}$$

where γ is a sequence of rules. We construct $\mathscr{D}^*$ as the following derivation:

$$\begin{array}{c}\dfrac{\dfrac{\mathscr{D}_1 \lhd \vdash V : B \quad \mathscr{D}' \lhd x_1 : B \vdash N_1 : B_1}{\vdash N_1[V/x_1] : B_1}\,cut \quad \dfrac{\mathscr{D}_1 \lhd \vdash V : B \quad \mathscr{D}'' \lhd x_2 : B \vdash N_2 : B_2}{\vdash N_2[V/x_2] : B_2}\,cut}{\vdash \langle N_1[V/x_1], N_2[V/x_2]\rangle : B_1 \,\&\, B_2}\,\&\mathrm{R0} \\ \vdots\,\gamma \\ \Delta \vdash P[\langle N_1[V/x_1], N_2[V/x_2]\rangle / y] : A\end{array}$$

By Remark 1, U is a η-long normal form, so that $|V| \leq |U|$ by Proposition 3.2. By Lemma 5, x_1 and x_2 occur exactly once in N_1 and N_2, respectively. Hence, $|N_i[V/x_i]| = |N_i| + |V| - 1$, for $i = 1, 2$. We have:

$$\begin{aligned}|\langle N_1[V/x_1], N_2[V/x_2]\rangle| &= |N_1[V/x_1]| + |N_2[V/x_2]| + 1 = 2 \cdot |V| + |N_1| + |N_2| - 1 \\ &\leq |U| + |V| + |N_1| + |N_2| - 1 < |\mathtt{copy}^U V \;\mathtt{as}\; x_1, x_2 \;\mathtt{in}\; \langle N_1, N_2\rangle|\end{aligned}$$

and this implies $|M_2| < |M_1|$. □

Subject reduction entails linear strong normalization.

Theorem 8 (Linear strong normalization)**.** *If $\mathscr{D} \lhd \Gamma \vdash M : A$ then M reduces to a normal form in at most $|M|$ reduction steps.*

Remark 2. By Newman's Lemma (see [4]), confluence of $\rightarrow$ for typable terms holds as a consequence of Theorem 8 and weak confluence, the latter being easy to establish. Therefore, each typable term reduces to a *unique* normal form.

4.2 Cubic $\forall$-Lazy Cut-Elimination

LAM is a subsystem of IMALL_2. Hence, from a purely proof-theoretical viewpoint, the former inherits the cut-elimination rules of the latter (see for example [6]). However, these proof rewriting rules for LAM would be more permissive than the reduction rules for $\Lambda_{\mathtt{copy}}$, essentially because the $\mathtt{copy}$ construct can only duplicate values. The next examples illustrate this point.

Example 1. Let us consider the following derivation of LAM:

$$\dfrac{y : \mathbf{1} \vdash y\mathbf{I} : \mathbf{1} \qquad \dfrac{x_1 : \mathbf{1} \vdash x_1 : \mathbf{1} \qquad x_2 : \mathbf{1} \vdash x_2 : \mathbf{1} \qquad \vdash \mathbf{I} : \mathbf{1}}{x : \mathbf{1} \vdash \mathtt{copy}^{\mathbf{I}} x \;\mathtt{as}\; x_1, x_2 \;\mathtt{in}\; \langle x_1, x_2\rangle : \mathbf{1} \,\&\, \mathbf{1}}\,\&\mathrm{R1}}{y : \mathbf{1} \vdash \mathtt{copy}^{\mathbf{I}} y\mathbf{I} \;\mathtt{as}\; x_1, x_2 \;\mathtt{in}\; \langle x_1, x_2\rangle : \mathbf{1} \,\&\, \mathbf{1}}\,cut \tag{17}$$

and let us apply the cut-elimination rule of IMALL_2 moving the *cut* upward. We get a derivation of $y : \mathbf{1} \vdash M : \mathbf{1} \,\&\, \mathbf{1}$, where M is $\mathtt{copy}^{\mathbf{I}} y \;\mathtt{as}\; y_1, y_2 \;\mathtt{in}\; \langle y_1\mathbf{I}, y_2\mathbf{I}\rangle$. But $\mathtt{copy}^{\mathbf{I}} y\mathbf{I} \;\mathtt{as}\; x_1, x_2 \;\mathtt{in}\; \langle x_1, x_2\rangle \not\rightarrow^* M$.

Example 2. Let us consider the following derivation of LAM:

$$\dfrac{\dfrac{x_1 : \mathbf{1} \vdash x_1 : \mathbf{1} \qquad x_2 : \mathbf{1} \vdash x_2 : \mathbf{1} \qquad \vdash \mathbf{I} : \mathbf{1}}{x : \mathbf{1} \vdash \mathtt{copy}^{\mathbf{I}} x \;\mathtt{as}\; x_1, x_2 \;\mathtt{in}\; \langle x_1, x_2\rangle : \mathbf{1} \,\&\, \mathbf{1}}\,\&\mathrm{R1} \qquad \dfrac{z : \mathbf{1} \vdash z : \mathbf{1}}{y : \mathbf{1} \,\&\, \mathbf{1} \vdash \pi_1(y) : \mathbf{1}}\,\&\mathrm{L}i}{x : \mathbf{1} \vdash \pi_1(\mathtt{copy}^{\mathbf{I}} x \;\mathtt{as}\; x_1, x_2 \;\mathtt{in}\; \langle x_1, x_2\rangle) : \mathbf{1}}\,cut \tag{18}$$

and let us apply the principal cut-elimination rule for & in IMALL_2. We get a *cut* with premises $x_1 : \mathbf{1} \vdash x_1 : \mathbf{1}$ and $z : \mathbf{1} \vdash z : \mathbf{1}$. However, no reduction rule rewrites $\pi_1(\mathtt{copy}^{\mathbf{I}} x \;\mathtt{as}\; x_1, x_2 \;\mathtt{in}\; \langle x_1, x_2\rangle)$ into x_1.

	negative $\forall$	*positive* &
lazy types	X	X
$\forall$*-lazy types*	X	✓

Figure 4: Lazy types vs $\forall$-lazy types.

To circumvent the above mismatch between proof rewriting and term reduction, we introduce the $\forall$-*lazy cut-elimination rules*. They never eliminate instances of *cut* like (17) and (18), so that cut-elimination fails in general. We then show that, by defining a special $\forall$-lazy cut-elimination strategy with cubic cost, cut-elimination can be recovered in the restricted case of derivations of $\forall$-lazy types (Theorem 13). This result is analogous to the restricted (lazy) cut-elimination property for derivations of lazy types discussed in Section 2.2. The crucial difference is that, while Girard's lazy types rule out both negative occurrences of $\forall$ and positive occurrences of &, the $\forall$-lazy types only require the absence of negative $\forall$, as illustrated in Figure 4. This allows us to nest instances of &R1 in a derivation of LAM without incurring exponential proof normalization, as shown in Section 3.2 in the case of term reduction.

Definition 7 ($\forall$-lazy cut-elimination rules)**.**

- With (X,Y) we denote a *cut* whose left (resp. right) premise is the conclusion of an instance of the inference rule X (resp. Y). Cuts are divided into four classes: the *symmetric cuts* are $(\multimap \mathrm{R}, \multimap \mathrm{L})$, $(\&\mathrm{R0}, \&\mathrm{L}i)$, $(\forall\mathrm{R}, \forall\mathrm{L})$ and those of the form (X, ax) or (ax, Y), for some X and Y; the *copy-first cuts* have form $(\&R1, \&\mathrm{L}i)$; the *critical cuts* have form $(X, \&\mathrm{R1})$, for some rule X; finally, the *commuting cuts* are all the remaining instances of *cut*.
- Let the following be a critical cut:

$$\dfrac{\dfrac{\mathscr{D}}{\Gamma \vdash M : A} \qquad \dfrac{x_1 : A \vdash N_1 : B \qquad x_2 : A \vdash N_2 : C \qquad \vdash V : A}{x : A \vdash \mathtt{copy}^V\, x \;\mathtt{as}\; x_1, x_2 \;\mathtt{in}\; \langle N_1, N_2 \rangle : B \,\&\, C}\,\&\mathrm{R1}}{\Gamma \vdash \mathtt{copy}^V\, M \;\mathtt{as}\; x_1, x_2 \;\mathtt{in}\; \langle N_1, N_2 \rangle : B \,\&\, C}\,cut$$

  It is called *safe* if $\Gamma = \emptyset$, and *deadlock* otherwise. Also, it is called *ready* if it is safe and $\mathscr{D}$ is cut-free. In this case, since A is $\forall$-lazy, Proposition 2.2 implies $M \in \mathscr{V}$.
- The $\forall$*-lazy cut-elimination rules* are defined as follows:
  - they correspond to the standard cut-elimination rules of LL for commuting cuts and for the symmetric cuts $(\multimap \mathrm{R}, \multimap \mathrm{L})$, $(\forall\mathrm{R}, \forall\mathrm{L})$, (X, ax) and (ax, Y) (see e.g. [6]);
  - they are displayed in Figure 5 for the symmetric cut $(\&\mathrm{R0}, \&\mathrm{L}i)$ and for those critical cuts $(X, \&\mathrm{R1})$ which are *ready*;
  - there is no $\forall$-lazy cut-elimination rule for copy-first cuts and the remaining critical cuts.

  If $\mathscr{D}$ rewrites to $\mathscr{D}'$ by a $\forall$-lazy cut-elimination rule, we write $\mathscr{D} \rightsquigarrow \mathscr{D}'$. The reflexive and transitive closure of $\rightsquigarrow$ is $\rightsquigarrow^*$.

The $\forall$-lazy cut-elimination rules prevent duplication of terms that are not values, restoring a match between proof rewriting and term reduction. What we are going to show is that any $\forall$-lazy derivation $\mathscr{D}$ can be rewritten into a cut-free one by a specific strategy of $\forall$-lazy cut-elimination steps. This implies that any instance of critical cut in $\mathscr{D}$ (e.g. the deadlock in (17)) is eventually turned into a ready cut, and that all instances of copy-first cuts like (18) are eventually turned into $(\&\mathrm{R0}, \&\mathrm{L}i)$ cuts. The latter is due to the replacement of &R1 with &R0 given by the $\forall$-lazy cut elimination of ready cuts in Figure 5.

The proposition below exploits the *closure* conditions introduced in Definition 4.

$$(\&\mathrm{R0},\&\mathrm{L}i)\quad \dfrac{\dfrac{\vdash N_1 : A_1 \quad \vdash N_2 : A_2}{\vdash \langle N_1, N_2\rangle : A_1 \,\&\, A_2} \quad \dfrac{\Gamma, x_i : A_i \vdash M : B \quad i \in \{1,2\}}{\Gamma, x : A_1 \,\&\, A_2 \vdash M[\pi_i(x)/x_i] : B}}{\Gamma \vdash M[\pi_i\langle N_1, N_2\rangle/x_i] : B}\,\mathit{cut} \rightsquigarrow \dfrac{\vdash N_i : A_i \quad \Gamma, x_i : A_i \vdash M : B}{\Gamma \vdash M[N_i/x_i] : B}\,\mathit{cut}$$

$$(X,\&\mathrm{R1})\quad \dfrac{\dfrac{\mathscr{D}^\dagger}{\vdash V : A} \quad \dfrac{x_1 : A \vdash N_1 : B_1 \quad x_2 : A \vdash N_2 : B_2 \quad \vdash U : A}{x : A \vdash \mathtt{copy}^U x \;\mathtt{as}\; x_1, x_2 \;\mathtt{in}\; \langle N_1, N_2\rangle : B_1 \,\&\, B_2}}{\vdash \mathtt{copy}^U V \;\mathtt{as}\; x_1, x_2 \;\mathtt{in}\; \langle N_1, N_2\rangle : B_1 \,\&\, B_2}\,\mathit{cut} \rightsquigarrow \dfrac{\dfrac{\dfrac{\mathscr{D}^\dagger}{\vdash V : A} \quad x_1 : A \vdash N_1 : B_1}{\vdash N_1[V/x_1] : B_1}\,\mathit{cut} \quad \dfrac{\dfrac{\mathscr{D}^\dagger}{\vdash V : A} \quad x_2 : A \vdash N_2 : B_1}{\vdash N_2[V/x_2] : B_2}\,\mathit{cut}}{\vdash \langle N_1[V/x_1], N_2[V/x_2]\rangle : B_1 \,\&\, B_2}\,\&\mathrm{R0}$$

Figure 5: $\forall$-lazy cut-elimination rules for $(\&\mathrm{R0}, \&\mathrm{L}i)$ and for ready cuts, where $\mathscr{D}^\dagger$ is cut-free.

Proposition 9. *If $\mathscr{D} \triangleleft \Gamma \vdash M : A$ and $FV(A) = \emptyset$ then $FV(\Gamma) = \emptyset$.*

Proof. By induction on $\mathscr{D}$ using the closure conditions (i)-(ii) in Definition 4 and the conditions on linear additives. We only consider some interesting cases. Suppose $\mathscr{D}$ ends with an instance of $\multimap$L with premises $\Gamma' \vdash N : B$ and $\Delta, x : C \vdash M' : A$. If $FV(A) = \emptyset$ then, by induction hypothesis, $FV(\Delta) = FV(C) = \emptyset$. By applying the closure condition (i), we have $FV(B) = \emptyset$. By applying, again, the induction hypothesis we have $FV(\Gamma') = \emptyset$, and hence $FV(\Gamma', y : B \multimap C, \Delta) = \emptyset$. Let us now consider the case where $\mathscr{D}$ ends with an instance of &Li with premise $\Gamma', x_i : B_i \vdash M' : A$ and conclusion $\Gamma', y : B_1 \,\&\, B_2 \vdash M'[\pi_i(y)/x_i] : A$. If $FV(A) = \emptyset$ then $FV(\Gamma') = \emptyset$ by induction hypothesis. Moreover, since $B_1 \,\&\, B_2$ is a closed $\forall$-lazy type, we conclude $FV(\Gamma', y : B_1 \,\&\, B_2) = \emptyset$. □

The next lemmas are essential to ensure the restricted cut-elimination result for $\forall$-lazy types.

Lemma 10. *Let $\mathscr{D} \triangleleft \Gamma \vdash M : A$ be a derivation whose only cuts are either deadlocks or copy-first. If one of those cuts exists in $\mathscr{D}$, then $\mathscr{D}$ is not $\forall$-lazy.*

Proof. First, we show that both the conclusion of a deadlock and the conclusion of a copy-first cut cannot be $\forall$-lazy. It suffices to find a closed type in the context of these judgments, since closed types must contain at least a $\forall$ in positive position. Let

$$\dfrac{\Delta \vdash N : B \qquad x : B \vdash \mathtt{copy}^V x \;\mathtt{as}\; x_1, x_2 \;\mathtt{in}\; \langle P, Q\rangle : C \,\&\, D}{\Delta \vdash \mathtt{copy}^V N \;\mathtt{as}\; x_1, x_2 \;\mathtt{in}\; \langle P, Q\rangle : C \,\&\, D}\,\mathit{cut}$$

be a deadlock. By definition, we have $\Delta \neq \emptyset$. Since $x : B \vdash \mathtt{copy}^V x \;\mathtt{as}\; x_1, x_2 \;\mathtt{in}\; \langle N_1, N_2\rangle : C \,\&\, D$ is the conclusion of &R1, we have $FV(B) = \emptyset$. Hence $FV(\Delta) = \emptyset$, by Proposition 9. Moreover, let

$$\dfrac{x : B \vdash \mathtt{copy}^V x \;\mathtt{as}\; x_1, x_2 \;\mathtt{in}\; \langle N_1, N_2\rangle : B_1 \,\&\, B_2 \qquad \Delta, x : B_1 \,\&\, B_2 \vdash M[\pi_i(x)/x_i] : C}{\Delta, x : B \vdash M[\pi_i(\mathtt{copy}^V x \;\mathtt{as}\; x_1, x_2 \;\mathtt{in}\; \langle N_1, N_2\rangle)/x_i] : C}\,\mathit{cut}$$

be a copy-first cut. Its leftmost premise is the conclusion of &R1, so B must be closed by definition.

Suppose now that $\mathscr{D}$ contains some cuts. Then it contains at least a deadlock or a copy-first cut. In both cases, $\mathscr{D}$ contains a judgment that is not $\forall$-lazy. Let $R_1, \ldots R_n$ be the sequence of rule instances from $\Gamma \vdash M : A$ up to this judgment. We prove by induction on n that $\Gamma \vdash M : A$ cannot be $\forall$-lazy. The case $n = 0$ is trivial. If $n > 0$, then we have two cases depending on R_n. If R_n is a cut, then it is either a deadlock or a copy-first cut, and its conclusion cannot be $\forall$-lazy, so we apply the induction hypothesis. If R_n is not a cut, we apply Proposition 2.1 and the induction hypothesis. □

Lemma 11 (Existence of a safe cut)**.** *Let $\mathscr{D} \triangleleft \Gamma \vdash M : A$ be a $\forall$-lazy derivation whose only cuts are either critical or copy-first. Then:*

1. *if $\mathscr{D}$ has critical cuts, then it has safe cuts;*
2. *if $\mathscr{D}$ is free from critical cuts, then it is free from copy-first cuts.*

Proof. Both points follow from Lemma 10. □

Definition 8 (Height and weight)**.** Let $\mathscr{D} \triangleleft \Gamma \vdash M : A$ be a derivation of LAM:

- The *weight of* $\mathscr{D}$, written $\&(\mathscr{D})$, is the number of instances of the rule &R1 in $\mathscr{D}$.
- Given a rule instance R in $\mathscr{D}$, the *height of* R, written $h(R)$, is the number of rule instances from the conclusion $\Gamma \vdash M : A$ of $\mathscr{D}$ upward to the conclusion of R. The *height of* $\mathscr{D}$, written $h(\mathscr{D})$, is the largest $h(R)$ among its rule instances.

Lemma 12 (Eliminating a ready cut)**.** *Let $\mathscr{D} \triangleleft \Gamma \vdash M : A$ be a $\forall$-lazy derivation whose only cuts are either critical or copy-first. The following statements hold:*

1. *if $\mathscr{D}$ has critical cuts, then it has ready cuts;*
2. *if $\mathscr{D}^*$ is obtained by eliminating a ready cut in $\mathscr{D}$, then $|\mathscr{D}^*| + 2 \cdot \&(\mathscr{D}^*) < |\mathscr{D}| + 2 \cdot \&(\mathscr{D})$.*

Proof. Concerning point 1, by Lemma 11.1 $\mathscr{D}$ contains at least one safe cut. Let R be the one with maximal height, we display as follows:

$$\dfrac{\begin{array}{c}\mathscr{D}' \\ \vdash N : B\end{array} \qquad \dfrac{\begin{array}{c}\mathscr{D}_1 \\ x_1 : B \vdash M_1 : B_1\end{array} \qquad \begin{array}{c}\mathscr{D}_2 \\ x_2 : B \vdash M_2 : B_2\end{array} \qquad \begin{array}{c}\mathscr{D}'' \\ \vdash U : B\end{array}}{x : B \vdash \mathtt{copy}^U x \;\mathtt{as}\; x_1, x_2 \;\mathtt{in}\; \langle M_1, M_2\rangle : B_1 \& B_2}\,\&\mathrm{R1}}{\vdash \mathtt{copy}^U N \;\mathtt{as}\; x_1, x_2 \;\mathtt{in}\; \langle M_1, M_2\rangle : B_1 \& B_2}\,\mathit{cut} \tag{19}$$

Since B is a $\forall$-lazy type, $\mathscr{D}'$ is a $\forall$-lazy derivation. By Lemma 11.1 and maximality of $h(R)$, $\mathscr{D}'$ has no critical cut, hence $\mathscr{D}'$ is cut-free by Lemma 11.2. Therefore, R is a ready cut.

As for point 2, let $\mathscr{D}^*$ be the derivation obtained by eliminating a ready cut like (19) in $\mathscr{D}$ (see Figure 5). By Remark 1, $\mathscr{D}''$ is η-expanded, hence cut-free by definition. Since both $\mathscr{D}'$ and $\mathscr{D}''$ are $\forall$-lazy and cut-free, they have no instances of &R1 by Proposition 2.2, so that $\&(\mathscr{D}^*) = \&(\mathscr{D}) - 1$. Moreover, $|\mathscr{D}'| \leq |\mathscr{D}''|$ by Proposition 3.2. We have: $2 \cdot |\mathscr{D}'| + |\mathscr{D}_1| + |\mathscr{D}_2| + 3 + 2 \cdot \&(\mathscr{D}^*) < |\mathscr{D}| + 2 \cdot \&(\mathscr{D}^*) + 2 = |\mathscr{D}| + 2 \cdot (\&(\mathscr{D}^*) + 1)$. Therefore, $|\mathscr{D}^*| + 2 \cdot \&(\mathscr{D}^*) < |\mathscr{D}| + 2 \cdot \&(\mathscr{D})$. □

Theorem 13 (Cubic $\forall$-lazy cut-elimination)**.** *Let $\mathscr{D} \triangleleft \Gamma \vdash M : A$ be a $\forall$-lazy derivation. Then, the $\forall$-lazy cut-elimination reduces $\mathscr{D}$ to a cut-free $\mathscr{D}^\dagger \triangleleft \Gamma \vdash M^\dagger : A$ in $\mathscr{O}(|\mathscr{D}|^3)$ steps.*

Proof. Let us define a $\forall$-*lazy cut-elimination strategy* divided into *rounds*. At each round:

{1} we eliminate all the commuting instances of *cut*;

{2} if a symmetric instance of *cut* exists, we eliminate it; otherwise, all instances of *cut* are either critical or copy-first, and we eliminate a *ready* one, if any.

We now show that the above $\forall$-lazy cut-elimination strategy terminates with a cut-free derivation. We proceed by induction on the lexicographical order of the pairs $\langle |\mathscr{D}| + 2 \cdot \&(\mathscr{D}), H(\mathscr{D})\rangle$, where $H(\mathscr{D})$ is the sum of the heights $h(\mathscr{D}')$ of all subderivations $\mathscr{D}'$ of $\mathscr{D}$ whose conclusion is an instance of *cut*. During {1}, every commuting $\forall$-lazy cut-elimination step moves an instance of *cut* upward, strictly decreasing $H(\mathscr{D})$ and leaving $|\mathscr{D}| + 2 \cdot \&(\mathscr{D})$ unaltered. During {2}, every symmetric $\forall$-lazy cut-elimination step shrinks $|\mathscr{D}|$. If only critical and copy-first cuts are in $\mathscr{D}$ then, by Lemma 11.2, either $\mathscr{D}$ has critical cuts or it is cut-free. In the former case, by Lemma 12.1 a ready cut exists. By Lemma 12.2, if we apply the corresponding $\forall$-lazy cut-elimination step $\mathscr{D} \rightsquigarrow \mathscr{D}^*$, we have $|\mathscr{D}^*| + 2 \cdot \&(\mathscr{D}^*) < |\mathscr{D}| + 2 \cdot \&(\mathscr{D})$.

We now exhibit a bound on the number of $\forall$-lazy cut-elimination steps from $\mathscr{D}$ to $\mathscr{D}^\dagger$. Generally speaking, we can represent a $\forall$-lazy cut-elimination strategy as:

$$\mathscr{D} = \mathscr{D}_0 \underset{cc_0}{\rightsquigarrow^*} \mathscr{D}'_0 \underset{src_0}{\rightsquigarrow} \mathscr{D}_1 \cdots \underset{cc_i}{\rightsquigarrow^*} \mathscr{D}'_i \underset{src_i}{\rightsquigarrow} \mathscr{D}_{i+1} \underset{cc_{i+1}}{\rightsquigarrow^*} \cdots \mathscr{D}'_{n-1} \underset{src_{n-1}}{\rightsquigarrow} \mathscr{D}_n \underset{cc_n}{\rightsquigarrow^*} \mathscr{D}'_n = \mathscr{D}^* \tag{20}$$

where, for $0 \leq i \leq n$ and $0 \leq j \leq n-1$, cc_i denotes a sequence of $\forall$-cut elimination steps applied to commuting cuts, while src_j denotes a $\forall$-cut elimination step applied to either a symmetric or a ready cut. A bound on the length of the sequence cc_i is $|\mathscr{D}_i|^2$ because every instance of rule in $\mathscr{D}_i$ can, in principle, be commuted with every other. Moreover, $|\mathscr{D}_{j+1}| + 2 \cdot \&(\mathscr{D}_{j+1}) < |\mathscr{D}'_j| + 2 \cdot \&(\mathscr{D}'_j)$, for every $0 \leq j \leq n-1$. Finally, since $|\mathscr{D}_i| = |\mathscr{D}'_i|$ for all $0 \leq i \leq n$, we have $n \leq |\mathscr{D}| + 2 \cdot \&(\mathscr{D}) \leq 3 \cdot |\mathscr{D}|$. Therefore, the total number of $\forall$-lazy cut-elimination steps in (20) is $O(|\mathscr{D}| \cdot |\mathscr{D}|^2)$. □

Recalling that the normal form of a typable term in LAM exists by Theorem 8 and is unique by Remark 2, we have the following straightforward corollary.

Corollary 14. *Let $\mathscr{D} \lhd \Gamma \vdash M : A$ be a $\forall$-lazy derivation, and let $M^\dagger$ be the normal form of M. Then:*

- *$M^\dagger$ is free from instances of* `copy` *and π_i;*
- *if $\Gamma = \emptyset$ then $M^\dagger \in \mathscr{V}$.*

Proof. It suffices to observe that, if a $\forall$-lazy derivation $\mathscr{D} \lhd \Gamma \vdash M : A$ rewrites to $\mathscr{D}' \lhd \Gamma \vdash M' : A$ by a $\forall$-lazy cut-elimination step, then $M \to^* M'$ (we consider terms up to α-equivalence). By Theorem 13 a cut-free $\mathscr{D}^\dagger$ exists such that $\mathscr{D}^\dagger \lhd \Gamma \vdash M^\dagger : A$, for some $M^\dagger$. We conclude by Proposition 2.2. □

As we already observed in Section 2.3, a `copy` construct behaves quite like a suspended substitution. So, a normal form with shape $\texttt{copy}^V N \texttt{ as } x_1, x_2 \texttt{ in } \langle P, Q \rangle$ represents a substitution that cannot be performed. Corollary 14 states that, whenever a term has $\forall$-lazy type, its normal form is always free from these "unevaluated" expressions. This result is analogous to Girard's linear normalization by lazy evaluation for terms having lazy type in IMALL_2 (see Section 2.2). However, the above corollary allows us to gain something more. On the one hand, indeed, LAM's typable terms enjoy linear *strong* normalization (Theorem 8). Therefore, as opposed to IMALL_2, the system LAM does not require specific evaluation strategies to avoid exponential reductions. On the other hand, as already remarked, the $\forall$-lazy types are more expressive than the lazy ones (see Figure 4).

5 Comparing LAM and IMLL_2

Following [7], we exploit the mechanisms of linear erasure and duplication studied by Mairson and Terui [17, 18] to define a sound translation of LAM into IMLL_2 (Theorem 17). A fundamental result of this section is Theorem 18, stating that derivations of LAM may exponentially compress linear λ-terms of IMLL_2. On the one hand, these results witness that the former system is not algorithmically more expressive than the latter. On the other hand, in a way similar to [7], they show that LAM is able to compactly represent Mairson and Terui's linear erasure and duplication.

5.1 Linear Erasure and Duplication in IMLL_2

Mairson has shown in [17] that IMLL is expressive enough to encode boolean circuits. This result was later reformulated by Mairson and Terui in IMLL_2 to prove results about the complexity of cut-elimination [18], where the advantage of working with IMLL_2 is to assign uniform types to structurally related linear λ-terms. In the latter encoding, the boolean values "true" and "false" are represented by $\mathtt{tt} \triangleq \lambda x.\lambda y.x \otimes y$ and $\mathtt{ff} \triangleq \lambda x.\lambda y.y \otimes x$ respectively, with type $\mathbf{B} \triangleq \forall \alpha.\alpha \multimap \alpha \multimap \alpha \otimes \alpha$. The key step of the encoding is the existence of an "eraser" $\mathtt{E}_{\mathbf{B}}$ and a "duplicator" $\mathtt{D}_{\mathbf{B}}$ for the Boolean data type $\mathbf{B}$:

$$\mathtt{E}_{\mathbf{B}} \triangleq \lambda z.\mathtt{let}\ z\mathbf{I}\mathbf{I}\ \mathtt{be}\ x \otimes y\ \mathtt{in}\ (\mathtt{let}\ y\ \mathtt{be}\ \mathbf{I}\ \mathtt{in}\ x) : \mathbf{B} \multimap \mathbf{1} \tag{21}$$

$$\mathtt{D}_{\mathbf{B}} \triangleq \lambda z.\mathtt{proj}_1(z(\mathtt{tt} \otimes \mathtt{tt})(\mathtt{ff} \otimes \mathtt{ff})) : \mathbf{B} \multimap \mathbf{B} \otimes \mathbf{B} \tag{22}$$

$$\mathtt{proj}_1 \triangleq \lambda z.\mathtt{let}\ z\ \mathtt{be}\ x \otimes y\ \mathtt{in}\ (\mathtt{let}\ \mathtt{E}_{\mathbf{B}}\, y\ \mathtt{be}\ \mathbf{I}\ \mathtt{in}\ x) : (\mathbf{B} \otimes \mathbf{B}) \multimap \mathbf{B} \tag{23}$$

where $\mathtt{proj}_1$ is the linear λ-term projecting the first element of a pair. For $M \in \{\mathtt{tt}, \mathtt{ff}\}$, we have $\mathtt{E}_{\mathbf{B}}\, M \to^*_\beta \mathbf{I}$ and $\mathtt{D}_{\mathbf{B}}\, M \to^*_\beta M \otimes M$. In other words, linear erasure involves a stepwise "data consumption" process, while linear duplication works "by selection and erasure": it contains *both* possible outcomes of duplication $\mathtt{tt} \otimes \mathtt{tt}$ and $\mathtt{ff} \otimes \mathtt{ff}$, and it selects the desired pair by linearly erasing the other one.

In [18], Mairson and Terui generalize the above mechanism of linear erasure and duplication to the class of closed Π_1 types:

Definition 9 (Π_1 types [18])**.** A type of IMLL_2 is a Π_1 type if it contains no negative occurrences of $\forall$.

Closed Π_1 types represent *finite* data types, because they admit only finitely many closed and normal inhabitants. An example is $\mathbf{B}$, representing the Boolean data type.

The fundamental result about closed Π_1 types is the following:

Theorem 15 (Erasure and duplication [18])**.**

1. *For any closed Π_1 type A there is a linear λ-term $\mathtt{E}_A$ of type $A \multimap \mathbf{1}$ such that, for all closed and normal inhabitant M of A, $\mathtt{E}_A\, M \to^*_\beta \mathbf{I}$.*
2. *For any closed and* inhabited *Π_1 type A there is a linear λ-term $\mathtt{D}_A$ of type $A \multimap A \otimes A$ such that, for all closed and normal inhabitant M of A, $\mathtt{D}_A\, M \to^*_{\beta\eta} M \otimes M$.*

We call $\mathtt{E}_A$ *eraser* and $\mathtt{D}_A$ *duplicator* of A. Intuitively, by taking as input a closed and normal inhabitant M of closed Π_1 type A, $\mathtt{D}_A$ implements the following three main operations:

(1) "expand" M to an η-long normal form of A, let us say M';

(2) compile M' to a linear λ-term $\lceil M' \rceil$ which encodes M' as a boolean tuple;

(3) copy and decode $\lceil M' \rceil$ obtaining $M' \otimes M'$, which η-reduces to $M \otimes M$.

Point (3) implements Mairson and Terui's "duplication by selection and erasure" discussed for the type $\mathbf{B}$, and requires a nested series of `if-then-else` playing the role of a look-up table that stores all pairs of closed and normal inhabitants of A (which are always finite, as already observed). Each pair represents a possible outcome of duplication. Given a boolean tuple $\lceil M' \rceil$ as input, the nested `if-then-else` select the corresponding pair $M' \otimes M'$, erasing all the remaining "candidates". The inhabitation condition for A stated in Theorem 15.2 assures the existence of a default pair $N \otimes N$, a sort of "exception" that we "throw" if the boolean tuple in input encodes no closed normal inhabitant of A.

Point 2 of Theorem 15 was only sketched in [18]. A detailed proof of the construction of $\mathtt{D}_A$ is in [7], which also estimates the complexity of duplicators and erasers:

Proposition 16 (Size of $\mathtt{E}_A$ and $\mathtt{D}_A$ [7])**.** *If A is a closed Π_1 type, then $|\mathtt{E}_A| \in \mathscr{O}(|A|)$. Moreover, if A is inhabited, then $|\mathtt{D}_A| \in \mathscr{O}(2^{|A|^2})$.*

5.2 A Translation of LAM Into IMLL_2 and Exponential Compression

Following [7], we define a translation $(\_)^\bullet$ from derivations of LAM into linear λ-terms with type in IMLL_2. It maps closed $\forall$-lazy types into closed Π_1 types, and instances of the inference rules &R1 and &L*i* into, respectively, duplicators and erasers of closed Π_1 types. We prove that the translation is sound and the linear λ-term $\mathscr{D}^\bullet$ associated with a derivation $\mathscr{D}$ of LAM has size that can be exponential with respect to the size of $\mathscr{D}$.

Definition 10 (From LAM to IMLL_2)**.** We define a map $(\_)^\bullet$ translating a derivation $\mathscr{D} \lhd \Gamma \vdash M : A$ of LAM into a linear λ-term $\mathscr{D}^\bullet$ such that $\Gamma^\bullet \vdash \mathscr{D}^\bullet : A^\bullet$ is derivable in IMLL_2.

1. The map $(\_)^\bullet$ is defined on types of $\Theta_\&$ by induction on their structure:

$$\alpha^\bullet \triangleq \alpha \qquad\qquad (A \,\&\, B)^\bullet \triangleq A^\bullet \otimes B^\bullet$$
$$(A \multimap B)^\bullet \triangleq A^\bullet \multimap B^\bullet \qquad\qquad (\forall\alpha.A)^\bullet \triangleq \forall\alpha.A^\bullet \ .$$

   Notice that $(A\langle B/\alpha\rangle)^\bullet = A^\bullet\langle B^\bullet/\alpha\rangle$. If $\Gamma = x_1 : A_1, \ldots, x_n : A_n$, we set $\Gamma^\bullet \triangleq x_1 : A_1^\bullet, \ldots, x_n : A_n^\bullet$.

2. The map $(\_)^\bullet$ is defined on derivations $\mathscr{D} \lhd \Gamma \vdash M : A$ of LAM by induction on the last rule:
   (a) if $\mathscr{D}$ is *ax* with conclusion $x : A \vdash x : A$, then $\mathscr{D}^\bullet \triangleq x$;
   (b) if $\mathscr{D}$ has last rule *cut* with premises $\mathscr{D}_1 \lhd \Delta \vdash N : B$ and $\mathscr{D}_2 \lhd \Sigma, x : B \vdash P : A$, where $M = P[N/x]$, then $\mathscr{D}^\bullet \triangleq \mathscr{D}_2^\bullet[\mathscr{D}_1^\bullet/x]$;
   (c) if $\mathscr{D}$ has last rule $\multimap$R with premise $\mathscr{D}_1 \lhd \Gamma, x : B \vdash N : C$ and $M = \lambda x.N$, then $\mathscr{D}^\bullet \triangleq \lambda x.\mathscr{D}_1^\bullet$;
   (d) if $\mathscr{D}$ has last rule $\multimap$L with premises $\mathscr{D}_1 \lhd \Delta \vdash N : B$ and $\mathscr{D}_2 \lhd \Sigma, x : C \vdash P : A$, where $\Gamma = \Delta, \Sigma, y : B \multimap C$, then $\mathscr{D}^\bullet \triangleq \mathscr{D}_2^\bullet[y\mathscr{D}_1^\bullet/x]$;
   (e) if $\mathscr{D}$ has last rule &R0 with premises $\mathscr{D}_1 \lhd \vdash N_1 : B_1$ and $\mathscr{D}_2 \lhd \vdash N_2 : B_2$ then $\mathscr{D}^\bullet \triangleq \mathscr{D}_1^\bullet \otimes \mathscr{D}_2^\bullet$;
   (f) if $\mathscr{D}$ has last rule &L*i* with premise $\mathscr{D}_1 \lhd \Delta, x_i : B_i \vdash N : A$, where $\Gamma = \Delta, x : B_1 \,\&\, B_2$, then $\mathscr{D}^\bullet \triangleq \mathtt{let}\ x\ \mathtt{be}\ x_1 \otimes x_2\ \mathtt{in}\ (\mathtt{let}\ \mathtt{E}_{B^\bullet_{3-i}} x_{3-i}\ \mathtt{be}\ \mathbf{I}\ \mathtt{in}\ \mathscr{D}_1^\bullet)$, where $\mathtt{E}_{B^\bullet_{3-i}}$ is the eraser of $B^\bullet_{3-i}$;
   (g) if $\mathscr{D}$ ends with &R1 with premises $\mathscr{D}_1 \lhd x_1 : B \vdash N_1 : B_1$, $\mathscr{D}_2 \lhd x_2 : B \vdash N_2 : B_2$, and $\mathscr{D}_3 \lhd \vdash V^\bullet : B^\bullet$ then $\mathscr{D}^\bullet \triangleq\ \mathtt{let}\ \mathtt{D}_{B^\bullet} x\ \mathtt{be}\ x_1 \otimes x_2\ \mathtt{in}\ \mathscr{D}_1^\bullet \otimes \mathscr{D}_2^\bullet$, where $\mathtt{D}_{B^\bullet}$ is the duplicator of $B^\bullet$;
   (h) if $\mathscr{D}$ has last rule $\forall$R with premise $\mathscr{D}_1 \lhd \Gamma \vdash M : B\langle\gamma/\alpha\rangle$ then $\mathscr{D}^\bullet \triangleq \mathscr{D}_1^\bullet$;
   (i) if $\mathscr{D}$ has last rule $\forall$L with premise $\mathscr{D}_1 \lhd \Delta, x : B\langle C/\alpha\rangle \vdash M : A$, where $\Gamma = \Delta, x : \forall\alpha.B$, then $\mathscr{D}^\bullet \triangleq \mathscr{D}_1^\bullet$.

Remark 3. Points 2(f)-(g) are well-defined. Indeed, since B, B_1, B_2 in both points are closed and $\forall$-lazy, the types $B^\bullet, B_1^\bullet, B_2^\bullet$ are closed Π_1, so that Theorem 15.1 assures the existence of $\mathtt{E}_{B^\bullet_{3-i}}$. Moreover, the closed Π_1 type $B^\bullet$ in point 2(g) is inhabited by the closed linear λ-term $\mathscr{D}_3^\bullet$. The latter is also normal, since by Remark 1 $\mathscr{D}_3$ is η-expanded (hence cut-free). Therefore, Theorem 15.2 assures that $\mathtt{D}_{B^\bullet}$ exists.

We now show that every $\forall$-lazy cut-elimination step applied to a derivation $\mathscr{D} \lhd \Gamma \vdash M : A$ of LAM can be simulated by a sequence of $\beta\eta$-reduction steps applied to $\mathscr{D}^\bullet$.

Theorem 17 (Soundness of $(\_)^\bullet$)**.** *Let $\mathscr{D}$ be a derivation of* LAM*. If $\mathscr{D} \rightsquigarrow \mathscr{D}'$ then $\mathscr{D}^\bullet \rightarrow^*_{\beta\eta} \mathscr{D}'^\bullet$.*

Proof. W.l.o.g. it suffices to consider the case where the last rule of $\mathscr{D}$ is the instance of *cut* the $\forall$-lazy cut-elimination rule $\mathscr{D} \rightsquigarrow \mathscr{D}'$ is applied to. The only interesting cases are the $\forall$-lazy cut-elimination rules in Figure 5. So, suppose that $\mathscr{D}$ ends with a cut (&R0, &L*i*), where the premises of &R0 are $\mathscr{D}_1 \lhd \vdash N_1 : A_1$ and $\mathscr{D}_2 \lhd \vdash N_2 : A_2$, and the premise of &L*i* is $\mathscr{D}_3 \lhd \Gamma, x_i : A_i \vdash M : B$. Since $\mathscr{D}^\bullet_{3-i}$ is a closed linear λ-term of closed Π_1 type $A^\bullet_{3-i}$, by applying Theorem 15.1 and the reduction rules in (8) we have:

$$\mathscr{D}^\bullet = \ \mathtt{let}\ \mathscr{D}_1^\bullet \otimes \mathscr{D}_2^\bullet\ \mathtt{be}\ x_1 \otimes x_2\ \mathtt{in}\ (\mathtt{let}\ \mathtt{E}_{A^\bullet_{3-i}} x_{3-i}\ \mathtt{be}\ \mathbf{I}\ \mathtt{in}\ \mathscr{D}_3^\bullet)$$

$$\begin{aligned}
&\rightarrow_\beta \texttt{let E}_{A_{3-i}}\mathscr{D}^\bullet_{3-i}\ \texttt{be}\ \mathbf{I}\ \texttt{in}\ \mathscr{D}^\bullet_3[\mathscr{D}^\bullet_i/x_i]\\
&\rightarrow^*_\beta \texttt{let}\ \mathbf{I}\ \texttt{be}\ \mathbf{I}\ \texttt{in}\ \mathscr{D}^\bullet_3[\mathscr{D}^\bullet_i/x_i]\\
&\rightarrow_\beta \mathscr{D}^\bullet_3[\mathscr{D}^\bullet_i/x_i] = \mathscr{D}'^\bullet.
\end{aligned}$$

Finally, suppose that $\mathscr{D}$ ends with a *ready* cut $(X, \&\mathrm{R1})$, for some X, where the left premises of the *cut* is $\mathscr{D}_1 \triangleleft \vdash V : A$ and the premises of &R1 are $\mathscr{D}_2 \triangleleft x_1 : A \vdash N_1 : B_1$, $\mathscr{D}_3 \triangleleft x_2 : A \vdash N_2 : B_2$ and $\mathscr{D}_4 \triangleleft \vdash U : A$. Since the cut is ready, $\mathscr{D}_1$ must be cut-free, and hence $\mathscr{D}^\bullet_1$ is a closed and normal linear λ-term of closed Π_1 type $A^\bullet$. Therefore, by applying Theorem 15.2 and the reduction rules in (8), we have:

$$\begin{aligned}
\mathscr{D}^\bullet = {}& \texttt{let}\ \texttt{D}_{A^\bullet}\mathscr{D}^\bullet_1\ \texttt{be}\ x_1 \otimes x_2\ \texttt{in}\ \mathscr{D}^\bullet_2 \otimes \mathscr{D}^\bullet_3\\
&\rightarrow^*_{\beta\eta}\ \texttt{let}\ \mathscr{D}^\bullet_1 \otimes \mathscr{D}^\bullet_1\ \texttt{be}\ x_1 \otimes x_2\ \texttt{in}\ \mathscr{D}^\bullet_2 \otimes \mathscr{D}^\bullet_3\\
&\rightarrow_\beta \mathscr{D}^\bullet_2[\mathscr{D}^\bullet_1/x_1] \otimes \mathscr{D}^\bullet_3[\mathscr{D}^\bullet_1/x_2] = \mathscr{D}'^\bullet
\end{aligned}$$

this concludes the proof. □

The above result stresses a fundamental advantage of LAM over IMLL_2: as shown in both [18] and [7], the latter type system requires an extremely complex linear λ-term to encode linear duplication (see Theorem 16), while the former can compactly represent it by means of typed terms with shape:

$$\lambda x.\texttt{copy}^V x\ \texttt{as}\ x_1, x_2\ \texttt{in}\ \langle x_1, x_2\rangle : A \multimap A \,\&\, A \tag{24}$$

Moreover, linear erasure is expressed by the following simple typed term:

$$\lambda x.\pi_2(\langle x, \mathbf{I}\rangle) : A \multimap \mathbf{1} \tag{25}$$

This crucial aspect of LAM can be made apparent by estimating the impact of the translation $(\_)^\bullet$ on the size of derivations, and hence the cost of "unpacking" the inference rules &R1 and &Li.

Theorem 18 (Exponential compression for LAM)**.** *Let $\mathscr{D}$ be a derivation in* LAM*. Then,* $|\mathscr{D}^\bullet| = \mathscr{O}(2^{|\mathscr{D}|^k})$*, for some $k \geq 1$.*

Proof. By structural induction on $\mathscr{D}$. The only interesting case is when $\mathscr{D}$ ends with &R1 with premises $\mathscr{D}_1 \triangleleft x_1 : A \vdash N_1 : B_1$, $\mathscr{D}_2 \triangleleft x_2 : A \vdash N_2 : B_2$ and $\mathscr{D}_3 \triangleleft \vdash U : A$. By Definition 10, $\mathscr{D}^\bullet = \texttt{let}\ \texttt{D}_{A^\bullet}x\ \texttt{be}\ x_1 \otimes x_2\ \texttt{in}\ \mathscr{D}^\bullet_1 \otimes \mathscr{D}^\bullet_2$. By Remark 1, $\mathscr{D}_3$ is η-expanded, so that $|A| \leq 2 \cdot |\mathscr{D}_3|$ by Proposition 3.1. Hence, $|\texttt{D}_A| \in \mathscr{O}(2^{(2\cdot|\mathscr{D}_3|)^2})$ by Proposition 16. We apply the induction hypothesis on $\mathscr{D}_1$ and $\mathscr{D}_2$ and conclude. □

6 Conclusions

We introduce LAM, a type assignment system endowed with a weaker version of the Linear Logic additive rules &R and &L, called *linear additive rules*. We prove both linear strong normalization and a restricted cut-elimination theorem. Also, we present a sound translation of LAM into IMLL_2, and we study its complexity.

A future direction is to find linear additive rules based on the additive connective $\oplus$, and to prove results similar to Theorem 8 and Theorem 13. This goal turns out to be harder, due to the "classical flavor" of the inference rule $\oplus$L, displayed below:

$$\frac{\Gamma, x : A \vdash M : C \qquad \Gamma, y : B \vdash N : C}{\Gamma, z : A \oplus B \vdash \texttt{case}\ z\ \texttt{of}\ [\texttt{inj}_1(z) \rightarrow M \mid \texttt{inj}_2(z) \rightarrow N] : C}$$

Let us discuss this point. The linear additive rule &R1 prevents exponential normalization by carefully controlling context-sharing, which involves hidden contractions and is responsible for unrestricted duplication. Finding a linear additive rule corresponding to $\oplus$L means controlling the sharing of types in the right-hand side of the turnstile. But this sharing hides a co-contraction, i.e. $C \otimes C \multimap C$, which requires fairly different techniques to be tamed.

Interesting applications of linear additives are in the field of ICC, and indeed they motivate the tools developed in the present paper. As already discussed in the Introduction, variants of the additive rules expressing non-determinism explicitly have been used to capture NP [19, 11, 20]. To the best of our knowledge, all these characterizations of NP crucially depend on the choice of a special evaluation strategy able to avoid the exponential blow up described in Section 2.2. Linear additives can refine [19, 11, 20], because they do not affect the complexity of normalization, and so they allow for natural cost models that can be implemented with a negligible overhead. A possible future work could be then to extend Soft Type Assignment (STA), a type system capturing PTIME [10], with a non-deterministic variant of linear additives, and to show that *Strong* Non-deterministic Polytime Soundness holds for the resulting system. This would allow us to characterize NP in a "wider" sense, i.e. independently of the reduction strategy considered.

In a probabilistic setting, similar goals have already been achieved. In [8] Curzi and Roversi studied the type system PSTA, an extension of STA with a non-deterministic variant of the linear additive rules obtained by replacing &Li with the following:

$$\frac{\Gamma, x : A \vdash M : C}{\Gamma, y : A \,\&\, A \vdash M[\pi(y)/x] : C}$$

and by considering the non-deterministic reduction rule $M \leftarrow \pi(\langle M, N\rangle) \rightarrow N$ in place of $\pi_i(\langle M_1, M_2\rangle) \rightarrow M_i$. It is shown that, when PSTA is endowed with a probabilistic big-step reduction relation, it is able to capture the probabilistic polytime functions and problems independently of the reduction strategy.

Acknowledgments

I would like to thank L. Roversi for the precious discussions about the topic, and the anonymous reviewers for useful comments and suggestions. This work was supported by a UKRI Future Leaders Fellowship, 'Structure vs Invariants in Proofs', project reference MR/S035540/1.

References

[1] Martín Abadi, Luca Cardelli, Pierre-Louis Curien & Jean-Jacques Lévy (1990): *Explicit Substitutions*. In Frances E. Allen, editor: *Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages, San Francisco, California, USA, January 1990*, ACM Press, pp. 31–46, doi:10.1145/96709.96712.

[2] Beniamino Accattoli (2017): *(In)Efficiency and Reasonable Cost Models*. In Sandra Alves & Renata Wasserman, editors: *12th Workshop on Logical and Semantic Frameworks, with Applications, LSFA 2017, Brasília, Brazil, September 23-24, 2017*, *Electronic Notes in Theoretical Computer Science* 338, Elsevier, pp. 23–43, doi:10.1016/j.entcs.2018.10.003.

[3] Beniamino Accattoli & Ugo Dal Lago (2012): *On the Invariance of the Unitary Cost Model for Head Reduction*. In Ashish Tiwari, editor: *23rd International Conference on Rewriting Techniques and Applications (RTA'12), RTA 2012, May 28 - June 2, 2012, Nagoya, Japan*, *LIPIcs* 15, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 22–37, doi:10.4230/LIPIcs.RTA.2012.22.

[4] Henk Barendregt, Wil Dekkers & Richard Statman (2013): *Lambda calculus with types*. Cambridge University Press, doi:10.1017/CBO9781139032636.

[5] Roel Bloo & Kristoffer H. Rose (1995): *Preservation of Strong Normalisation in Named Lambda Calculi with Explicit Substitution and Garbage Collection*. In: *IN CSN-95: COMPUTER SCIENCE IN THE NETHERLANDS*, pp. 62–72.

[6] Torben Braüner & Valeria De Paiva (1996): *Cut-elimination for full intuitionistic linear logic*. 10, University of Cambridge, Computer Laboratory, doi:10.7146/brics.v3i10.19973.

[7] Gianluca Curzi & Luca Roversi (2020): *A type-assignment of linear erasure and duplication*. *Theoretical Computer Science* 837, pp. 26–53, doi:10.1016/j.tcs.2020.05.001.

[8] Gianluca Curzi & Luca Roversi (2020): *Probabilistic Soft Type Assignment*. *arXiv preprint arXiv:2007.01733*.

[9] Alejandro Díaz-Caro & Gilles Dowek (2013): *Non determinism through type isomorphism*. *arXiv preprint arXiv:1303.7334*.

[10] Marco Gaboardi & Simona Ronchi Della Rocca (2009): *From light logics to type assignments: a case study*. *Logic Journal of the IGPL* 17(5), pp. 499–530, doi:10.1093/jigpal/jzp019.

[11] Marco Gaboardi, Jean-Yves Marion & Simona Ronchi Della Rocca (2008): *Soft linear logic and polynomial complexity classes*. *Electronic Notes in Theoretical Computer Science* 205, pp. 67–87, doi:10.1016/j.entcs.2008.03.066.

[12] Jean-Yves Girard (2017): *Proof-nets: the parallel syntax for proof-theory*. In: *Logic and Algebra*, Routledge, pp. 97–124, doi:10.1201/9780203748671-4.

[13] Jean-Yves Girard & Yves Lafont (1987): *Linear logic and lazy computation*. In: *International Joint Conference on Theory and Practice of Software Development*, Springer, pp. 52–66, doi:10.1007/BFb0014972.

[14] J Roger Hindley (1989): *BCK-combinators and linear λ-terms have types*. *Theoretical Computer Science* 64(1), pp. 97–105, doi:10.1016/0304-3975(89)90100-X.

[15] Ross Horne (2019): *The sub-additives: A proof theory for probabilistic choice extending linear logic*. In: *4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, doi:10.4230/LIPIcs.FSCD.2019.23.

[16] Ugo Dal Lago & Simone Martini (2006): *An Invariant Cost Model for the Lambda Calculus*. In Arnold Beckmann, Ulrich Berger, Benedikt Löwe & John V. Tucker, editors: *Logical Approaches to Computational Barriers, Second Conference on Computability in Europe, CiE 2006, Swansea, UK, June 30-July 5, 2006, Proceedings*, *Lecture Notes in Computer Science* 3988, Springer, pp. 105–114, doi:10.1007/11780342_11.

[17] Harry G. Mairson (2004): *Linear Lambda Calculus and PTIME-completeness*. *J. Funct. Program.* 14(6), pp. 623–633, doi:10.1017/S0956796804005131.

[18] Harry G. Mairson & Kazushige Terui (2003): *On the Computational Complexity of Cut-Elimination in Linear Logic*. In Carlo Blundo & Cosimo Laneve, editors: *Theoretical Computer Science*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 23–36, doi:10.1007/978-3-540-45208-9_4.

[19] Satoshi Matsuoka (2004): *Nondeterministic Linear Logic*. *arXiv preprint cs/0410029*.

[20] François Maurel (2003): *Nondeterministic light logics and NP-time*. In: *International Conference on Typed Lambda Calculi and Applications*, Springer, pp. 241–255, doi:10.1007/3-540-44904-3_17.

[21] Paul-André Melliès (1995): *Typed Lambda-Calculi with Explicit Substitutions May Not Terminate*. In: *Proceedings of the Second International Conference on Typed Lambda Calculi and Applications*, TLCA '95, Springer-Verlag, Berlin, Heidelberg, p. 328–334, doi:10.1007/BFb0014062.