A type-assignment of linear erasure and duplication

A type-assignment of linear erasure and duplication

Gianluca Curzi[a], Luca Roversi[a]

[a]*Dipartimento di Informatica – Università di Torino*

Abstract

We introduce LEM, a type-assignment system for the linear λ-calculus that extends second-order IMLL_2, i.e., intuitionistic multiplicative Linear Logic, by means of logical rules that weaken and contract assumptions, but in a purely linear setting. LEM enjoys both a mildly weakened cut-elimination, whose computational cost is cubic, and Subject reduction. A translation of LEM into IMLL_2 exists such that the derivations of the former can exponentially compress the dimension of the derivations in the latter. LEM allows for a modular and compact representation of boolean circuits, directly encoding the fan-out nodes, by contraction, and disposing garbage, by weakening. It can also represent natural numbers with terms very close to standard Church numerals which, moreover, apply to Hereditarily Finite Permutations, i.e. a group structure that exists inside the linear λ-calculus.

Keywords: Second-Order Multiplicative Linear Logic, Type-assignment, Linear λ-calculus, Cut-elimination (cost), Boolean Circuits, Numerals, Hereditarily Finite Permutations

1. Introduction

Girard introduces *Linear Logic* (LL) in [9] as a refinement of both classical and intuitionistic logic. LL decomposes the intuitionistic implication "$\Rightarrow$" into the more primitive linear implication "$\multimap$" and modality "!" (of course), the latter giving a logical status to weakening and contraction by means of the so-called *exponential rules*. According to the *Curry-Howard correspondence*, this decomposition allows to identify a strictly linear component of the functional computations that interacts with the non-linear one, in which duplication and erasure are allowed.

This work focuses on IMLL_2, i.e. second-order intuitionistic *multiplicative* Linear Logic which, we recall, is free of any kind of exponential rules. The Curry-

Email addresses: `curzi@di.unito.it, gianluca.curzi@unito.it` (Gianluca Curzi), `roversi@di.unito.it, luca.roversi@unito.it` (Luca Roversi)

URL: `http://www.di.unito.it/~curzi` (Gianluca Curzi), `http://www.di.unito.it/~rover` (Luca Roversi)

Howard correspondence tightly relates IMLL_2 and the linear λ-calculus, a sub-language of the standard λ-calculus without explicit erasure and duplication.

Interesting works exist on the expressiveness of both the untyped and the typed linear λ-calculus.

Alves et al. [1] recover the full computational power of Gödel System T by adding booleans, natural numbers, and a linear iterator to the linear λ-calculus, the non-linear features coming specifically from the iterator and the numerals.

Matsuoka investigates the discriminating power of linear λ-terms with types in IMLL, i.e. intuitionistic multiplicative Linear Logic, proving typed variants of Böhm Theorem [17]. We remark that, in this setting, discriminating among linear λ-terms relies on *a specific form of weakening* already inside IMLL.

Another work that exploits the built-in erasure and copying mechanisms of the linear λ-calculus is by Mairson [15]. With no new constructors, Mairson encodes boolean circuits in the linear λ-calculus. Moreover, Mairson&Terui reformulate Mairson's results inside IMLL_2 and prove bounds on the complexity of the cut-elimination in sub-systems of LL [16].

Contributions. Starting from Mairson&Terui's [16], this work investigates a structural proof-theory, and the related Curry-Howard correspondence, of IMLL_2 extended with inference rules for contraction and weakening.

1. We introduce the *Linearly Exponential and Multiplicative* system LEM, giving a logical status to the erasure and the duplication that [16] identifies inside the linear λ-calculus. LEM is a type-assignment for a *linear* λ-calculus endowed with constructs for weakening and contraction, and it is obtained by extending IMLL_2 with rules on modal formulas "$\downarrow A$". LEM can be seen as a sub-system of LL with a restricted form "$\downarrow$" of "!".

2. We consider a mildly weakened cut-elimination, called "lazy", that faithfully represents the mechanism of linear erasure and duplication discussed in [16], and we identify a set of derivations in LEM that rewrite to cut-free ones under that lazy cut-elimination in a cubic number of steps (Section 5.1). Moreover, we show the Subject reduction of LEM (Section 5.2).

3. We prove that the cut-elimination of IMLL_2 can simulate the one of LEM at a cost which can be exponential in the size of the given derivation of LEM (Section 6). So, LEM can speed up the cut-elimination of IMLL_2, meaning that it compresses in smaller derivations what can be algorithmically expressed in IMLL_2.

4. Hence, we explore the algorithmic expressiveness of LEM (Section 7):

   (a) Both LEM and IMLL_2 can represent boolean circuits. However, the copying mechanism, directly available in LEM, makes the encoding of the fan-out of the nodes of the circuit essentially natural, facilitating the modularity and the readability of the encoding itself. Moreover, the erasure in LEM avoids to accumulate garbage when evaluating a circuit represented by a derivation of LEM, unlike in other proposals.

(b) We show that numerals, structurally related to Church ones, exist in LEM. Their type is $(\downarrow\forall\alpha.(\alpha \multimap \alpha)) \multimap \forall\alpha.(\alpha \multimap \alpha)$ that forbids iterations longer than the complexity of the lazy cut-elimination. Remarkably, the numerals in LEM admit successor and addition that work as expected, thanks to the Subject reduction.

(c) Finally, we show that Hereditarily Finite Permutations, which form a group inside the linear λ-calculus, inhabit a simple generalization of the here above type of numerals, so possibly connecting LEM with reversible computations.

The above contributions follow from a fully detailed, and not at all obvious, technical reworking of Mairson&Terui's [16] work. We propose it as a solid base to further investigations concerning duplication and erasure in a purely linear setting.

Section 2 is about (formal) preliminaries. Section 3 introduces the motivating background and Section 4 formally defines LEM.

Acknowledgments. We are indebted to the anonymous reviewers for their patience and constructive attitude with which they red and commented on previous versions of this work.

2. Preliminaries

2.1. The linear λ-calculus

We assume the reader to be familiar with standard λ-calculus and related concepts like: (i) the set $FV(M)$ of the free variables of the λ-term M, (ii) the meta-level substitution $M[N/x]$ that replaces the λ-term N for every free occurrence of the variable x in M, (iii) the contexts $\mathcal{C}[\,]$, i.e. λ-terms with a place-holder (hole) $[\,]$ that may capture free variables of a λ-term plugged into $[\,]$, (iv) the α-equivalence ($=_\alpha$), (v) the β-reduction $(\lambda x.M)N \to_\beta M[N/x]$, (vi) the η-reduction $\lambda x.Mx \to_\eta M$ that can apply if x is not free in M. Both $\to_\beta$ and $\to_\eta$ are considered contextually closed.

By $\to^*_\beta$ we denote the reflexive and transitive closure of the β-reduction, and by $=_\beta$ its reflexive, symmetric and transitive closure.

Also, by $\to^*_\eta$ we denote the reflexive and transitive closure of the η-reduction, and by $=_\eta$ its reflexive, symmetric and transitive closure.

Finally, by $\to_{\beta\eta}$ we denote $\to_\beta \cup \to_\eta$, and by $\to^*_{\beta\eta}$ we denote its reflexive and transitive closure.

A λ-term is in β-*normal form*, or simply (β-)*normal*, whenever no β-reduction applies to it. A λ-term is in η-*normal form*, or simply η-*normal* if no η-reduction applies to it. Finally, a λ-term is in $\beta\eta$-*normal form*, or simply $\beta\eta$-*normal*, whenever no $\beta\eta$-reduction applies to it.

A λ-term is *closed* if $FV(M) = \varnothing$.

The *size* $|M|$ of M is the number of nodes in its syntax tree.

The *linear* λ-calculus is the λ-calculus restricted to *linear* λ-terms:

Definition 1 (Linear λ-terms)**.** A λ-term M is *linear* if all of its free variables occur once in it and every proper sub-term $\lambda x.M'$ of M is such that x occurs in M' and M' is linear. $\square$

For example, $I \triangleq \lambda x.x$ and $C \triangleq \lambda x.\lambda y.\lambda z.xzy$ are linear, while $K \triangleq \lambda x.\lambda y.x$ and $S \triangleq \lambda x.\lambda y.\lambda z.xz(yz)$ are not.

To our purposes, we shall adopt the following notion of value:

Definition 2 (Values)**.** A *value* is every linear λ-term which is both (β-)*normal* and *closed*. $\square$

We shall generally use V and U to range over values.

Fact 1 (Stability)**.** *Linear λ-terms are stable under β-reduction, i.e. M linear and $M \to_\beta N$ imply N is linear. Analogously, linear λ-terms are stable under η-reduction, i.e. M linear and $M \to_\eta N$ imply N is linear. In both cases, $FV(N) = FV(M)$.* $\square$

Finally, we shall write $M \circ N$ in place of $\lambda z.M(Nz)$.

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

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

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

Figure 1: IMLL_2 as a type-assignment system.

2.2. The systems IMLL_2 *and* IMLL

We assume familiarity with basic proof-theoretical notions and with Linear Logic (see [10, 27].) *Second-order Intuitionistic Multiplicative Linear Logic* (IMLL_2), seen as a type-assignment for the linear λ-calculus, is in Figure 1, where, we remark, the only logical operators are the universal quantifier "$\forall$" and the linear implication "$\multimap$". IMLL_2 derives *judgments* $\Gamma \vdash M : A$, i.e. a *type* A for the linear λ-term M from the *context* Γ. A *type* is a (type) variable α, or an *implication* $A \multimap B$, or a *universal quantification* $\forall\alpha.A$, where A and B are types. The set of free type variables of A is $FV(A)$. If $FV(A) = \varnothing$, then A is *closed*. If $FV(A) = \{\alpha_1, \ldots, \alpha_n\}$, then *a closure* $\overline{A}$ of A is $\forall\alpha_1.\cdots.\forall\alpha_n.A$, not necessarily linked to a specific order of $\alpha_1, \ldots, \alpha_n$. The standard meta-level substitution of a type B for every free occurrence of α in A is $A\langle B/\alpha\rangle$. The *size*

$|A|$ of the type A is the number of nodes in its syntax tree. A *context* Γ has form $x_1 : A_1, \ldots, x_n : A_n$, with $n \geq 0$, i.e. it is a finite multiset of *assumptions* $x : A$, where x is a λ-variable. The *domain* $\mathrm{dom}(\Gamma)$ of Γ is $\{x_1, \ldots, x_n\}$ and its range $\mathrm{rng}(\Gamma)$ is $\{A_1, \ldots, A_n\}$. The size $|\Gamma|$ of Γ is $\sum_{i=1}^{n} |A_i|$. Typically, names for contexts are Γ, Δ or Σ.

Since IMLL_2 gives types to linear λ-terms, $\multimap$L is necessarily subject to the *linearity constraint* $\mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Delta) = \varnothing$. We range over the derivations of IMLL_2 by $\mathcal{D}$. The *size* $|\mathcal{D}|$ of $\mathcal{D}$ is the number of the rule instances that $\mathcal{D}$ contains. We say that $\Gamma \vdash M : B$ is *derivable* if a derivation $\mathcal{D}$ exists that concludes with the judgment $\Gamma \vdash M : B$, and we also say that $\mathcal{D}$ is a derivation of $\Gamma \vdash M : B$. In that case we write $\mathcal{D} \lhd \Gamma \vdash M : B$ saying that M is an *inhabitant* of B or that B is *inhabited* by M from Γ. The cut-elimination steps for IMLL_2 are standard and both cut-elimination and confluence hold for it [27].

Propositional Intuitionistic Multiplicative Linear Logic (IMLL) is IMLL_2 without $\forall$R and $\forall$L. From Hindley [11], we recall that IMLL, thus IMLL_2, gives a type to every linear λ-term. The converse holds as well, due to the above *linearity constraint* on $\multimap$L, so the class of linear λ-terms is exactly the one of all typable λ-terms in IMLL_2. It follows that second-order does not allow to type more terms but it is nevertheless useful to assign uniform types to structurally related λ-terms.

We conclude by recalling standard definitions of types in IMLL_2:

Definition 3 (Basic datatypes)**.** The *unity* type is $\mathbf{1} \triangleq \forall\alpha.(\alpha \multimap \alpha)$ with constructor $I \triangleq \lambda x.x$, i.e. the identity, and destructor $\texttt{let}\ M\ \texttt{be}\ I\ \texttt{in}\ N \triangleq MN$;

The *tensor product* type $A \otimes B \triangleq \forall\alpha.(A \multimap B \multimap \alpha) \multimap \alpha$ with constructor $\langle M, N\rangle \triangleq \lambda z.z\,M\,N$ and destructor $\texttt{let}\ M\ \texttt{be}\ x,y\ \texttt{in}\ N \triangleq M(\lambda x.\lambda y.N)$.

Both binary tensor product and pair extend to their obvious n-ary versions $A^n = \underbrace{A \otimes \ldots \otimes A}_{n}$ and $M^n \triangleq \langle\underbrace{M, \ldots, M}_{n}\rangle$. □

Remark 1. Every occurrence of unity, (n-ary) tensor and n-tuple in the coming sections will be taken from Definition 3.

Finally, Definition 3 talks about *datatypes* because, by introducing a specific syntax for constructors and destructors, we implicitly adopt a pattern matching mechanism to operate on λ-terms typed with those types.

3. Duplication and erasure for the linear λ-calculus

As a motivational background we discuss erasure and duplication in the linear λ-calculus both in an untyped and in a type-assignment setting.

3.1. The untyped setting

The linear λ-calculus forbids any form of *direct* duplication of λ-terms, by means of multiple occurrences of the same variable, or of erasure, by omitting occurrences of bound variables in a λ-term. Nevertheless, erasure and duplication can be simulated. Concerning the former, a first approach has been developed by Klop [12], and can be called "erasure by garbage collection". It consists on accumulating unwanted data during computation in place of erasing it. For example, $K' = \lambda xy.\langle x, y\rangle$ represents the classical $K = \lambda xy.x$, the second component of $\langle x, y\rangle$ being garbage. Another approach is by Mackie, and can be called "erasure by data consumption" [14]. It involves a step-wise erasure process that proceeds by β-reduction, according to the following definition:

Definition 4 (Erasability)**.** A linear λ-term M is *erasable* if $\mathcal{C}[M] \rightarrow^*_\beta I$, for some context $\mathcal{C}[]$ such that $\mathcal{C}[M]$ is linear. □

Example 1. The context $\mathcal{C}[] = (\lambda z.[])III$ erases $\lambda xy.zxy$ because, filling $[]$ by $\lambda xy.zxy$, we obtain a closed linear λ-term that reduces to I. □

In [15], Mackie proves that all closed linear λ-terms can be erased by means of very simple contexts.

Lemma 2 ([15])**.** *Let M be any closed linear λ-term. Then there exists $n \geq 0$ such that $MI.\overset{n}{\because}.I \rightarrow^*_\beta I$.*

The above result is closely related to solvability (see [3]): "*A λ-term M in the standard λ-calculus is said solvable if, for some n, there exist λ-terms $N_1, \dots, N_n$ such that $MN_1 \dots N_n =_\beta I$.*" Lemma 2 states that every closed linear λ-term is solvable by linear contexts.

In fact, the notion of erasability can be addressed in a more general setting.

Definition 5 (Erasable sets)**.** Let X be a set of linear λ-terms. We say that X is an *erasable set* if a linear λ-term $\mathtt{E}_X$ exists such that $\mathtt{E}_X\, M \rightarrow^*_{\beta\eta} I$, for all $M \in X$. We call $\mathtt{E}_X$ *eraser* of X.

Proposition 3. *A finite set X of linear λ-terms is erasable if and only if all terms in X are closed.*

Proof. Let X be a finite set of linear λ-terms. To prove the left-to-right direction, suppose X is erasable. By definition, there exists a linear λ-term $\mathtt{E}_X$ such that $\mathtt{E}_X\, M \rightarrow^*_{\beta\eta} I$, for all $M \in X$. Since I is closed, by Fact 1 each $M \in X$ must be closed too. Let us now suppose that all terms in X are closed, and let $M_1, \dots, M_n$ be such terms. By Lemma 2, for every $i \leq n$ there exists a $k_i \geq 0$ such that $M_i I.\overset{k_i}{\because}.I \rightarrow^*_\beta I$. It suffices to set $\mathtt{E}_X \triangleq \lambda x.xI.\overset{k}{\because}.I$, where $k = \max_{i=1}^n k_i$. □

Recall from Definition 3 that $\langle M, N\rangle \triangleq \lambda z.zMN$. In the same spirit of Definition 5, we now investigate duplicability in the linear λ-calculus.

Definition 6 (Duplicable sets)**.** Let X be a set of linear λ-terms. We say that X is a *duplicable set* if a linear λ-term $\mathtt{D}_X$ exists such that $\mathtt{D}_X\, M \rightarrow^*_{\beta\eta} \langle M, M\rangle$ and $FV(\mathtt{D}_X) \cap FV(M) = \varnothing$, for all $M \in X$. We call $\mathtt{D}_X$ *duplicator* of X.

Proposition 4. *If a finite set X of linear λ-terms is duplicable then all terms in X are closed.*

Proof. Let X be a finite set of linear λ-terms, and suppose X is duplicable. By definition, there exists a linear λ-term $\mathtt{D}_X$ such that $\mathtt{D}_X\, M \to^*_{\beta\eta} \langle M, M\rangle$, for all $M \in X$. Since both M and $\mathtt{D}_X$ are linear λ-terms, and $FV(\mathtt{D}_X) \cap FV(M) = \varnothing$, we have that $\mathtt{D}_X\, M$ is linear, for all $M \in X$. If there were a variable occurring free in a term $M \in X$, then it would occur twice in $\langle M, M\rangle$, contradicting Fact 1. ☐

We conjecture that the converse holds as well, as long as we restrict to sets of distincts $\beta\eta$-normal forms. Indeed, duplication in a linear setting ultimately relies on the following linear version of the general separation theorem for the standard λ-calculus proved by Coppo et al. [4]:

Conjecture 5 (General separation)**.** *Let $X = \{M_1, \dots, M_n\}$ be a set of distinct closed linear λ-terms in $\beta\eta$-normal form. Then, for all $N_1, \dots, N_n$ closed linear λ-terms, there exists a closed linear λ-term F such that $F\, M_i =_{\beta\eta} N_i$, $\forall i \le n$.*

Now, let $X = \{M_1, \dots, M_n\}$ be a finite set of distinct closed linear λ-terms in $\beta\eta$-normal form. If Conjecture 5 were true, by fixing $N_i \triangleq \langle M_i, M_i\rangle$ for all $i \le n$, there would exists a closed linear λ-term $\mathtt{D}_X$ such that $\mathtt{D}_X\, M_i \to^*_{\beta\eta} \langle M_i, M_i\rangle$. So, we could connect linear erasure and duplication to standard λ-calculus notions:

$$\begin{gathered}\textit{solvability} \text{ implies } \textit{linear erasability}\\ \textit{separation} \text{ implies } \textit{linear duplication}\ .\end{gathered}$$

This topic is left to future work (see Section 8).

3.2. The typed setting

Erasure and duplication are less direct and liberal in IMLL_2 which assigns types to linear λ-terms. Specifically, it is possible to erase or duplicate all values (Definition 2) of what we call "ground type", i.e. Mairson&Terui's notion of closed Π_1-type [16], whose formal definition will be recalled shortly. A typical example of ground type in IMLL_2 is the one representing booleans. The standard second-order intuitionistic formulation of booleans (i.e. $\forall\alpha.\alpha \multimap \alpha \multimap \alpha$) is meaningless for IMLL_2 due to the lack of free weakening. Mairson&Terui [16] define them as:

$$\mathbf{B} \triangleq \forall\alpha.\alpha \multimap \alpha \multimap \alpha \otimes \alpha \qquad \mathtt{tt} \triangleq \lambda x.\lambda y.\langle x, y\rangle \qquad \mathtt{ff} \triangleq \lambda x.\lambda y.\langle y, x\rangle \tag{1}$$

where the values "truth" $\mathtt{tt}$ and the "falsity" $\mathtt{ff}$ implement the "erasure by garbage collection": the first element of the pair is the "real" output, while the second one is garbage. Starting from (1), Mairson shows in [15] that IMLL is expressive enough to encode boolean functions. Mairson and Terui reformulate that encoding in IMLL_2 in order to prove results about the complexity of cut-elimination [16]. The advantage of IMLL_2 is to assign uniform types to the λ-terms representing boolean functions. An *eraser* $\mathtt{E}_\mathbf{B}$ and a *duplicator* $\mathtt{D}_\mathbf{B}$ are the keys to obtain the encoding:

$$\mathtt{E}_\mathbf{B} \triangleq \lambda z.\mathtt{let}\ zII\ \mathtt{be}\ x, y\ \mathtt{in}\ (\mathtt{let}\ y\ \mathtt{be}\ I\ \mathtt{in}\ x) : \mathbf{B} \multimap \mathbf{1} \tag{2}$$

$$\mathtt{D}_\mathbf{B} \triangleq \lambda z.\pi_1(z\langle \mathtt{tt}, \mathtt{tt}\rangle\langle \mathtt{ff}, \mathtt{ff}\rangle) : \mathbf{B} \multimap \mathbf{B} \otimes \mathbf{B} \tag{3}$$

$$\pi_1 \triangleq \lambda z.\mathtt{let}\ z\ \mathtt{be}\ x, y\ \mathtt{in}\ (\mathtt{let}\ \mathtt{E}_\mathbf{B}\, y\ \mathtt{be}\ I\ \mathtt{in}\ x) : (\mathbf{B} \otimes \mathbf{B}) \multimap \mathbf{B} \tag{4}$$

with $\mathbf{B}$ as in (1) and π_1 the linear λ-term projecting the first element of a pair. Switching to type-assignment setting we get uniform copying and erasing mechanisms of the whole *class* of values of a given ground type. Note that the type-theoretical constraints let the erasure of a typed linear λ-term make use of something more than mere stacks of identities as in Lemma 2. Also, note that *both* the possible results $\langle \mathtt{tt}, \mathtt{tt} \rangle$ and $\langle \mathtt{ff}, \mathtt{ff} \rangle$ of duplications are built-in components of $\mathtt{D}_{\mathbf{B}}$. In accordance with the given input, $\mathtt{D}_{\mathbf{B}}$ selects the right pair representing the result by *erasing* the unwanted one. Such a "linear" form of *duplication by selection and erasure* is a step-by-step elimination of useless data until the desired result shows up.

The analysis of (2) and (3) leads to the following formal notions:

Definition 7 (Duplicable and erasable types in IMLL_2)**.** Let A be a type in IMLL_2. It is a *duplicable type* if a linear λ-term $\mathtt{D}_A : A \multimap A \otimes A$ exists such that $\mathtt{D}_A\, V \rightarrow^*_{\beta\eta} \langle V, V \rangle$, for every value V of A.

Moreover, A is an *erasable type* if a linear λ-term $\mathtt{E}_A : A \multimap \mathbf{1}$ exists such that $\mathtt{E}_A\, V \rightarrow^*_{\beta\eta} I$, for every value V of A.

We call $\mathtt{D}_A$ *duplicator* of A and $\mathtt{E}_A$ its *eraser.* □

Duplicators and erasers in Definition 7 apply to values of a given type, i.e. *closed* and normal inhabitants. This is not a loss of generality because Proposition 3 and Proposition 4 say that only closed terms can be duplicated or erased linearly.

Definition 8 (Π_1, Σ_1-types [16] and Ground types)**.** The following mutually defined grammars generate Π_1 and Σ_1-types:

$$\Pi_1 \coloneqq \alpha \mid \Sigma_1 \multimap \Pi_1 \mid \forall \alpha.\Pi_1$$
$$\Sigma_1 \coloneqq \alpha \mid \Pi_1 \multimap \Sigma_1$$

We call *ground types* the *closed* Π_1-types. □

We note that the universal quantifier $\forall$ occurs only positively in a Π_1-type, hence in ground types.

The booleans $\mathbf{B}$ in (1), the unit $\mathbf{1}$ and the tensor $A \otimes B$ as in Definition 3 are ground types, if A and B are. In fact, following [16], tensors and units can occur *also* to the left-hand side of a linear implication "$\multimap$", even in negative positions. The reason is that we can ignore them in practice, thanks to the isomorphisms:

$$((A \otimes B) \multimap C) \multimapboth (A \multimap B \multimap C) \qquad\qquad (\mathbf{1} \multimap C) \multimapboth C \ .$$

Ground types represent *finite* data types, while the values with a ground type represent their data.

Fact 6. *Every closed linear λ-term M has a ground type.* □

Proof. Every closed linear λ-term M is typable in IMLL (see [11]). Types in IMLL are quantifier-free instances of Π_1-types. Hence, M has also a Π_1-type A in IMLL_2. Let $FV(A) = \{\alpha_1, \ldots, \alpha_n\}$. Since M inhabits A, it also inhabits $\overline{A} = \forall\alpha_1.\cdots.\forall\alpha_n.A$, which is a closed Π_1-type, i.e. a ground type in IMLL_2. □

The class of ground types is a subset of both the classes of duplicable and erasable types.

Theorem 7 ([16])**.** *Every ground type is erasable.*

Proof. The proof follows from proving two statements by simultaneous induction: (i) For every Π_1-type A with free type variables $\alpha_1, \dots, \alpha_n$ there exists a linear λ-term $\mathtt{E}_A$ such that $\vdash \mathtt{E}_A : A[\mathbf{1}/\alpha_1, \dots, \mathbf{1}/\alpha_n] \multimap \mathbf{1}$, and (ii) for every Σ_1-type A with free type variables $\alpha_1, \dots, \alpha_n$ there exists a linear λ-term $\mathtt{H}_A$ such that $\vdash \mathtt{H}_A : A[\mathbf{1}/\alpha_1, \dots, \mathbf{1}/\alpha_n]$. ☐

Theorem 8. *Every inhabited ground type is duplicable.*

Mairson and Terui sketch the proof of Theorem 8 in [16]. Appendix A, which we see as an integral and relevant part of this work, develops it in every detail.

4. The system LEM

Theorems 7 and 8 say that the ground types can be weakened and contracted in IMLL_2. We here logically internalize those kinds of weakening a contraction in the deductive system LEM (Linearly Exponential and Multiplicative). It extends IMLL_2 with inference rules for the modality "$\downarrow$" that closely recall the exponential rules in Linear Logic.

Definition 9 (Types of LEM)**.** Let $\mathcal{X}$ be a denumerable set of type variables. The following grammar (5) generates the *exponential types*, while the grammar (6) generates the *linear types*:

$$\sigma \coloneqq A \mid \downarrow\sigma \tag{5}$$

$$A \coloneqq \alpha \mid \sigma \multimap A \mid \forall\alpha.A \tag{6}$$

where $\alpha \in \mathcal{X}$ and, in the last clause of the grammar (5), i.e. the one introducing $\downarrow\sigma$, *the type σ must be closed and without negative occurrences of* $\forall$. The set of all types generated by the grammar (5) will be denoted $\Theta_\downarrow$. A type is *strictly exponential* if it is of the form $\downarrow\sigma$. A *strictly exponential context* is a context containing only strictly exponential types and, similarly, a *linear context* contains only linear types. Finally, $A\langle B/\alpha\rangle$ is the standard meta-level substitution of B, for every occurrence of α in A. ☐

Remark 2. The modality "$\downarrow$" identifies where the ground types (Definition 8) occur in the grammars (5) and (6) because it applies to closed types that are free from negative occurrences of $\forall$. So, the occurrences of $\downarrow\sigma$ identify where contraction and weakening rules can apply in the derivations of LEM. ☐

Also, we observe that syntactically replacing the Linear Logic modality "!" for "$\downarrow$" in (5) and (6) yields a subset of Gaboardi&Ronchi's *essential types* [8], introduced to prove Subject reduction in a variant of Soft Linear Logic. Essential types forbid the occurrences of modalities in the right-hand side of an implication, such as in $A \multimap\, !B$.

$$
\begin{aligned}
(\lambda x.M)N &\to M[N/x] \\
\mathtt{discard}_\sigma\, V \;\mathtt{in}\; M &\to M \\
\mathtt{copy}^U_\sigma\, V \;\mathtt{as}\; y,z \;\mathtt{in}\; M &\to M[V/y, V/z]
\end{aligned}
$$

Figure 2: Reduction on terms.

LEM will be defined as a type-assignment for the term calculus $\Lambda_\downarrow$, that is essentially the standard linear λ-calculus with explicit and type-dependent constructs for erasure and duplication, i.e. $\mathtt{discard}_\sigma$ and $\mathtt{copy}^V_\sigma$, the latter being also decorated with a value V. These new constructs are able to copy and discard values only, i.e. closed and normal linear λ-terms.

Definition 10 (Terms and reduction of LEM)**.** Let $\mathcal{V}$ be a denumerable set of variables. The *terms* of LEM are given by the grammar:

$$M, N \coloneqq x \mid \lambda x.M \mid MN \mid \mathtt{discard}_\sigma\, M \;\mathtt{in}\; N \mid \mathtt{copy}^V_\sigma\, M \;\mathtt{as}\; x,y \;\mathtt{in}\; N \tag{7}$$

where $x, y \in \mathcal{V}$, V is a value (Definition 2, Section 2), and $\sigma \in \Theta_\downarrow$. The set of all terms of LEM will be denoted $\Lambda_\downarrow$. The set of the free variables of a term, and the notion of size are standard for variables, abstractions, and applications. The extension to the new constructors are:

$$
\begin{aligned}
FV(\mathtt{discard}_\sigma\, M \;\mathtt{in}\; N) &= FV(M) \cup FV(N) \\
FV(\mathtt{copy}^V_\sigma\, M \;\mathtt{as}\; x,y \;\mathtt{in}\; N) &= FV(M) \cup (FV(N) \setminus \{x,y\}) \\
|\mathtt{discard}_\sigma\, M \;\mathtt{in}\; N| &= |M| + |N| + 1 \\
|\mathtt{copy}^V_\sigma\, M \;\mathtt{as}\; x,y \;\mathtt{in}\; N| &= |M| + |N| + |V| + 1\ .
\end{aligned} \tag{8}
$$

A term M in (7) is *linear* if all of its free variables occur once in it and every proper sub-term of M with form $\lambda x.N$ (resp. $\mathtt{copy}^V_\sigma\, M' \;\mathtt{as}\; y,z \;\mathtt{in}\; N$) is such that x (resp. y, z) must occur in N and N is linear. *Henceforth, we use linear terms only.*

The notions of meta-level substitution and context are as usual.

The *one-step reduction relation* $\to$ is a binary relation on terms. It is defined by the reduction rules in Figure 2 and by the commuting conversions in Figure 3. It applies in any context. Its reflexive and transitive closure is denoted $\to^*$. A term is said a (or is in) *normal form* if no reduction step applies to it. $\square$

Both the type and the term annotations in the constructs $\mathtt{discard}_\sigma$ and $\mathtt{copy}^V_\sigma$ will become meaningful once we introduce the type-assignment system. The value V will be an inhabitant of σ, a necessary condition in order to faithfully express the mechanism of linear duplication.

The structure of the types in Definition 9 drives the definition of LEM.

$$
\begin{aligned}
(\texttt{discard}_\sigma\, M \texttt{ in } N)P &\to \texttt{discard}_\sigma\, M \texttt{ in } (NP)\\
\texttt{discard}_\sigma\, (\texttt{discard}_\tau\, M \texttt{ in } N) \texttt{ in } P &\to \texttt{discard}_\tau\, M \texttt{ in } (\texttt{discard}_\sigma\, N \texttt{ in } P)\\
\texttt{copy}^V_\sigma\, (\texttt{discard}_\tau\, M \texttt{ in } N) \texttt{ as } y,z \texttt{ in } P &\to \texttt{discard}_\tau\, M \texttt{ in } (\texttt{copy}^V_\sigma\, N \texttt{ as } y,z \texttt{ in } P)\\
(\texttt{copy}^V_\sigma\, M \texttt{ as } y,z \texttt{ in } N)P &\to \texttt{copy}^V_\sigma\, M \texttt{ as } y,z \texttt{ in } (NP)\\
\texttt{discard}_\sigma\, (\texttt{copy}^V_\tau\, M \texttt{ as } y,z \texttt{ in } N) \texttt{ in } P &\to \texttt{copy}^V_\tau\, M \texttt{ as } y,z \texttt{ in } (\texttt{discard}_\sigma\, N \texttt{ in } P)\\
\texttt{copy}^U_\sigma\, (\texttt{copy}^V_\tau\, M \texttt{ as } y,z \texttt{ in } N) \texttt{ as } y',z' \texttt{ in } P &\to \texttt{copy}^V_\tau\, M \texttt{ as } y,z \texttt{ in } (\texttt{copy}^U_\sigma\, N \texttt{ as } y',z' \texttt{ in } P)
\end{aligned}
$$

Figure 3: Commuting conversions on terms.

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

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

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

$$
\frac{x_1 : {\downarrow}\sigma_1, \ldots, x_n : {\downarrow}\sigma_n \vdash M : \sigma}{x_1 : {\downarrow}\sigma_1, \ldots, x_n : {\downarrow}\sigma_n \vdash M : {\downarrow}\sigma}\,p \qquad \frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma, y : {\downarrow}\sigma \vdash M[y/x] : \tau}\,d
$$

$$
\frac{\Gamma \vdash M : \tau}{\Gamma, x : {\downarrow}\sigma \vdash \texttt{discard}_\sigma\, x \texttt{ in } M : \tau}\,w \qquad \frac{\Gamma, y : {\downarrow}\sigma, z : {\downarrow}\sigma \vdash M : \tau \qquad \vdash V : \sigma}{\Gamma, x : {\downarrow}\sigma \vdash \texttt{copy}^V_\sigma\, x \texttt{ as } y,z \texttt{ in } M : \tau}\,c
$$

Figure 4: The system LEM.

Definition 11 (The system LEM)**.** It is the type-assignment system for the term calculus $\Lambda_\downarrow$ (Definition 10) in Figure 4. It extends IMLL_2 with the rules *promotion* p, *dereliction* d, *weakening* w and *contraction* c. As usual, $\multimap$R, $\forall$R, and p are right rules while $\multimap$L, $\forall$L, d, w, and c are left ones. □

First, we observe that ax cannot introduce exponential types, like in the *essential types* of the type systems in [8]. This is the base for proving:

Proposition 9 (Exponential context from exponential conclusion)**.** *If* $\mathcal{D} \lhd \Gamma \vdash M : {\downarrow}\sigma$ *is a derivation in* LEM*, then* Γ *is a strictly exponential context.*

Proof. By structural induction on the derivation $\mathcal{D}$. □

Also, we observe that p, d, w, c of LEM are reminiscent of the namesake Linear Logic exponential rules, but they only apply to types ${\downarrow}\sigma$ that (5) (Definition 9) generates, i.e. closed types with no negative occurrences of universal quantifiers. The rule c has one premise more than the contraction rule in Linear Logic. This premise "witnesses" that the type σ we want to contract is inhabited by at least one value V. This is because c expresses the mechanism of linear contraction discussed in the previous section for IMLL_2. As we shall see in Theorem 20, the term $\mathtt{copy}^V_\sigma$ that c introduces is a (very) compact notation for duplicators of ground types, whose detailed description is in Appendix A. Roughly, the duplicator of a ground type A is a linear λ-term that, taking a value U of type A as argument, implements the following three main operations:

1. expand U to its η-long normal form U_A whose dimension is bounded by the dimension of the type A. This is why using a modality to identify types for which this can be done is so important;

2. compile U_A to a linear λ-term $\lceil U_A \rceil$ which encodes U_A as a boolean tuple;

3. copy and decode $\lceil U_A \rceil$, obtaining the duplication $\langle U_A, U_A \rangle$ of U_A as final result. This duplication is by means of the term $\mathtt{dec}^s_A$ in Appendix A.3. It nests a series of `if-then-else` constructs which is a look-up table, possibly quite big, that stores all the pairs of normal inhabitants of A. Each of them represents a possible outcome of the duplication. Given a boolean tuple $\lceil U_A \rceil$ in input, the nested `if-then-else` select the corresponding pair $\langle U_A, U_A \rangle$, erasing all the remaining "candidates". The inhabitation condition for A stated in Theorem 8 ensures that the default pair $\langle V, V \rangle$ exists as a sort of "exception". We "throw" it each time the boolean tuple that $\mathtt{dec}^s_A$ receives as input does not encode any term.

Point 3 is the one implementing Mairson&Terui's "*duplication by selection and erasure*" discussed in Section 3. It involves the component of the duplicator which is exponential in the size of A. Therefore, as we shall see in Theorem 22, the construct $\mathtt{copy}^V_\sigma$ exponentially compresses the linear duplication mechanism encoded in a duplicator.

We conclude this section by commenting about how "$\downarrow$" and LL's "!" differ. Intuitively, the latter allows to duplicate or erase logical structure, or terms, *at once*, which is the standard way to computationally interpret contraction and weakening of a logical system. The modality "$\downarrow$" identifies duplication and erasure processes with a more *constructive* nature. The duplication proceeds step-by-step among a whole set of possible choices in order to identify those ones that cannot contain the copies of the term we are interested to duplicate, until it eventually reaches what it searches. Then, it exploits erasure. Erasing means eroding step-by-step a derivation or a term, according to the type that drives its construction.

5. Basic computational properties of LEM

The observations in the previous sections lead us to set the reduction rules on terms, which allow duplication and erasure for values only, as in Figure 2. Those reductions are more restrictive than the cut-elimination steps that we could perform on LEM if we look at it as it was a pure logical system, i.e. not a type-assignment. Since the cut-elimination of LEM works as in LL, once replaced "!" for "$\downarrow$", we can observe the effect of moving the *cut* in:

$$\dfrac{\dfrac{\dfrac{\vdots}{y:\downarrow A, z:\downarrow\mathbf{1} \vdash yz:\mathbf{1}}}{y:\downarrow A, z:\downarrow\mathbf{1} \vdash yz:\downarrow\mathbf{1}}\,p \qquad \dfrac{\dfrac{\vdots}{\Gamma, x_1:\downarrow\mathbf{1}, x_2:\downarrow\mathbf{1} \vdash M:\ \tau} \qquad \dfrac{\vdots}{\vdash I:\mathbf{1}}}{\Gamma, x:\downarrow\mathbf{1} \vdash \mathtt{copy}^{I}_{\mathbf{1}}\, x \ \mathtt{as}\ y,z\ \mathtt{in}\ M:\tau}\,c}{\Gamma, y:\downarrow A, z:\downarrow\mathbf{1} \vdash \mathtt{copy}^{I}_{\mathbf{1}}\, yz \ \mathtt{as}\ y,z\ \mathtt{in}\ M:\tau}\,cut \tag{9}$$

upward, in order to eventually eliminate it. The move would require to duplicate the *open* term yz, erroneously yielding a non linear term. So, at the proof-theoretical level, moves of the cut rule exist that cannot correspond to any reduction on terms. In order to circumvent the here above misalignment, we proceed as follows:

- We define the *lazy cut-elimination steps*. Their introduction rules out any attempt to eliminate instances of cuts like (9). The apparent drawback is to transform cuts like (9) into *deadlocks*, i.e. into instances of *cut* that we cannot eliminate.
- Deadlocks are not a problem. Once defined *lazy types*, we can show that a *lazy cut-elimination strategy* exists such that it eliminates all the cut rules that may occur in a derivation of a lazy type. The cost of the elimination is cubical (Theorem 14).

Last, we show that the reduction on terms in Figure 2 and Figure 3 enjoy Theorem 17, i.e. Subject reduction.

5.1. *Cut-elimination and its cubical complexity*

Definition 12 (The cuts of LEM)**.** Let (X, Y) identify an instance:

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash N:\sigma}\,X \qquad \dfrac{\vdots}{\Delta, x:\sigma \vdash M:\tau}\,Y}{\Gamma, \Delta \vdash M[N/x]:\tau}\,cut \tag{10}$$

of the rule *cut* that occurs in a given derivation $\mathcal{D}$ of LEM, where X and Y are two of the rules in Figure 4. *Axiom cuts* involve *ax*, and are of the form (X, ax) or (ax, Y), for some X and Y. *Exponential cuts* are (p,d), (p,w), and (p,c). *Principal cuts* are ($\multimap$R, $\multimap$L), ($\forall$R, $\forall$L) and every exponential cut. *Symmetric cuts* contain axiom and principal cuts. Every symmetric cut that is not exponential is *multiplicative*. *Commuting cuts* are all the remaining instances of *cut*, not mentioned here above, (p, p) included, for example.

A *lazy cut* is every instance of the cut (10) which is both exponential and such that N is a value.

A *deadlock* is every instance of the cut (10) which is both exponential and such that $\Gamma \neq \varnothing$. Otherwise, it is *safe*. $\square$

The lazy cut-elimination rules that we introduce here below are the standard ones, but restricted to avoid the elimination of non lazy instances of the exponential cuts (p, d), (p, w) and (p, c).

$$\dfrac{\dfrac{\overset{\mathcal{D}}{\Gamma, x:\sigma \vdash M:A}}{\Gamma \vdash \lambda x.M : \sigma \multimap A}\multimap R \qquad \dfrac{\overset{\mathcal{D}'}{\Delta \vdash N:\sigma} \qquad \overset{\mathcal{D}''}{\Theta, z:A \vdash P:\tau}}{\Delta,\Theta,y:\sigma\multimap A \vdash P[yN/z]:\tau}\multimap L}{\Gamma,\Delta,\Theta \vdash P[(\lambda x.M)N/z]:\tau}\mathit{cut} \rightsquigarrow \dfrac{\dfrac{\overset{\mathcal{D}'}{\Delta \vdash N:\sigma} \qquad \overset{\mathcal{D}}{\Gamma, x:\sigma \vdash M:A}}{\Gamma,\Delta \vdash M[N/x]:A}\mathit{cut} \qquad \overset{\mathcal{D}''}{\Theta, z:A \vdash P:\tau}}{\Gamma,\Delta,\Theta \vdash P[M[N/x]/z]:\tau}\mathit{cut}$$

$$\dfrac{\dfrac{\overset{\mathcal{D}}{\Gamma \vdash M : A\langle\gamma/\alpha\rangle}}{\Gamma \vdash M:\forall\alpha.A}\forall R \qquad \dfrac{\overset{\mathcal{D}'}{\Delta, x:A\langle B/\alpha\rangle \vdash N:\tau}}{\Delta, x:\forall\alpha.A \vdash N:\tau}\forall L}{\Gamma,\Delta \vdash N[M/x]:\tau}\mathit{cut} \rightsquigarrow \dfrac{\overset{\mathcal{D}\langle B/\alpha\rangle}{\Gamma \vdash M : A\langle B/\alpha\rangle} \qquad \overset{\mathcal{D}'}{\Delta, x:A\langle B/\alpha\rangle \vdash N:\tau}}{\Gamma,\Delta \vdash N[M/x]:\tau}\mathit{cut}$$

$$\dfrac{\dfrac{\overset{\mathcal{D}}{\vdash V:\sigma}}{\vdash V:\downarrow\sigma}p \qquad \dfrac{\overset{\mathcal{D}'}{\Gamma, x:\sigma \vdash M:\tau}}{\Gamma, y:\downarrow\sigma \vdash M[y/x]:\tau}d}{\Gamma \vdash M[V/x]:\tau}\mathit{cut} \rightsquigarrow \dfrac{\overset{\mathcal{D}}{\vdash V:\sigma} \qquad \overset{\mathcal{D}'}{\Gamma, x:\sigma \vdash M:\tau}}{\Gamma \vdash M[V/x]:\tau}\mathit{cut}$$

$$\dfrac{\dfrac{\overset{\mathcal{D}}{\vdash V:\sigma}}{\vdash V:\downarrow\sigma}p \qquad \dfrac{\overset{\mathcal{D}'}{\Delta \vdash M:\tau}}{\Delta, x:\downarrow\sigma \vdash \mathtt{discard}_\sigma\, x \;\mathtt{in}\; M:\tau}w}{\Delta \vdash \mathtt{discard}_\sigma\, V \;\mathtt{in}\; M:\tau}\mathit{cut} \rightsquigarrow \overset{\mathcal{D}'}{\Delta \vdash M:\tau}$$

$$\dfrac{\dfrac{\overset{\mathcal{D}}{\vdash V:\sigma}}{\vdash V:\downarrow\sigma}p \qquad \dfrac{\overset{\mathcal{D}_1}{\Delta, y:\downarrow\sigma, z:\downarrow\sigma \vdash M:\tau} \qquad \overset{\mathcal{D}_2}{\vdash U:\sigma}}{\Delta, x:\downarrow\sigma \vdash \mathtt{copy}^U_\sigma\, x \;\mathtt{as}\; y,z \;\mathtt{in}\; M:\tau}c}{\Delta \vdash \mathtt{copy}^U_\sigma\, V \;\mathtt{as}\; y,z \;\mathtt{in}\; M:\tau}\mathit{cut} \rightsquigarrow \dfrac{\dfrac{\overset{\mathcal{D}}{\vdash V:\sigma}}{\vdash V:\downarrow\sigma}p \qquad \dfrac{\dfrac{\overset{\mathcal{D}}{\vdash V:\sigma}}{\vdash V:\downarrow\sigma}p \qquad \overset{\mathcal{D}_1}{\Delta, y:\downarrow\sigma, z:\downarrow\sigma \vdash M:\tau}}{\Delta, z:\downarrow\sigma \vdash M[V/y]:\tau}\mathit{cut}}{\Delta \vdash M[V/y, V/z]:\tau}\mathit{cut}$$

Figure 5: *Lazy* cut-elimination rules for the principal cuts $(\multimap\mathrm{R}, \multimap\mathrm{L})$, $(\forall\mathrm{L}, \forall\mathrm{R})$, (p, d), (p, w), and (p, c).

Definition 13 (Lazy cut-elimination rules)**.** Figure 5 introduces the *lazy cut-elimination rules* for the principal cuts. The elimination rules for commuting and axiom cuts are standard, so we omit them all from Figure 5; the (possibly) less obvious commuting ones can be recovered from the reductions on terms in Figure 3. We remark that the elimination of the principal cuts $(\forall\mathrm{R}, \forall\mathrm{L})$ and (p, d) does not modify the subject of their concluding judgment. So, we call them *insignificant* as every other cut-elimination rule non influencing their concluding subject. Given a derivation $\mathcal{D}$, we write $\mathcal{D} \rightsquigarrow \mathcal{D}'$ if $\mathcal{D}$ rewrites to some $\mathcal{D}$ by one of the above rules. □

Lazy cut-elimination is a way of preventing the erasure and the duplication of terms that are not values, and hence to restore a correspondence between cut-elimination and term reduction. However, one can run into derivations containing exponential cuts that will never turn into lazy cuts, like the deadlock in (9). The solution we adopt is to identify a set of judgments whose derivations can be rewritten into cut-free ones by a sequence of lazy cut-elimination steps.

Definition 14 (Lazy types, lazy judgments and lazy derivations)**.** We say that σ is a *lazy type* if it contains no *negative* occurrences of $\forall$. Also, we say that $x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash M : \tau$ is a *lazy judgment* if τ is a lazy type and $\sigma_1, \ldots,$ σ_n contain no *positive* occurrences of $\forall$. Last, $\mathcal{D} \triangleleft \Gamma \vdash M : \tau$ is called a *lazy derivation* if $\Gamma \vdash M : \tau$ is a lazy judgment. □

Lemma 10 and 11 here below, as well as Definition 15 and 16, are the last preliminaries to show the relevance of lazy cuts that occur in lazy derivations.

Lemma 10.

(1) Every type of the form $\downarrow\sigma$ is closed and lazy.

(2) Every closed type has at least a positive quantification.

(3) Let ρ be any instance of $\forall$L, d, w, c, and p, the latter with a non empty context. The conclusion of ρ is not lazy.

(4) Let ρ be any instance of ax, $\multimap$ R, $\multimap$ L, $\forall$R, and p, the latter with an empty context. If the conclusion of ρ is lazy, then, every premise of ρ is lazy.

(5) If $\mathcal{D}$ is a cut-free and lazy derivation of LEM*, then all its judgments are lazy and no occurrences of $\forall$L, d, w, c, and p, the latter with a non empty context, can exist in $\mathcal{D}$.*

Proof. Point (1) holds by Definition 9. Point (2) is by a structural induction on types. Concerning Point (3), the conclusions of d, w, c, and p contain $\downarrow\sigma$. This is a closed type, hence, by Point (2), such conclusions are not lazy judgments. Moreover, $\forall$L introduces a positive occurrence of $\forall$ in the context of its conclusion, so that this latter cannot be a lazy judgment. Point (4) is a case analysis on every listed inference rule. As for Point (5), we can proceed by structural induction on $\mathcal{D}$. By definition, the conclusion of $\mathcal{D}$ is a lazy judgment. Point (3) excludes that one among $\forall$L, d, w, c, and p (with a non empty context) may be the last rule of $\mathcal{D}$. So, only one among ax, $\multimap$ R, $\multimap$ L, $\forall$R, and p (with an empty context) can be the concluding rule, say r, of $\mathcal{D}$. Point (4) implies that all the premises of r are lazy. Hence, we can apply the inductive hypothesis to the derivations of the premises of r and conclude. □

Definition 15 (Size of a derivation)**.** The *size* $|\mathcal{D}|$ of a derivation $\mathcal{D}$ in LEM is defined by induction:

1. If $\mathcal{D}$ is ax then $|\mathcal{D}| = 1$.
2. If $\mathcal{D}$ is a derivation $\mathcal{D}'$ that concludes by a rule with a single premise, then $|\mathcal{D}| = |\mathcal{D}'| + 1$.
3. If $\mathcal{D}$ composes two derivations $\mathcal{D}'$ and $\mathcal{D}''$ by a rule with two premises, but different from c, then $|\mathcal{D}| = |\mathcal{D}'| + |\mathcal{D}''| + 1$.
4. If $\mathcal{D}$ composes two derivations $\mathcal{D}'$ and $\mathcal{D}''$ by the rule c, then $|\mathcal{D}| = |\mathcal{D}'| + |\mathcal{D}''| + 3$. □

Remark 3. Adding "3" instead of the possibly expected "1" in the clause (4) of Definition 15 highlights the non linearity that the instances of c introduce in the course of the lazy cut-elimination on LEM of Definition 17 below. □

Lemma 11. *Let $\mathcal{D} \lhd x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash M : \sigma$ be a cut-free and lazy derivation.*

(1) M is a linear λ-term in normal form.

(2) $|M| \leq \sum_{i=1}^{n} |\sigma_i| + |\sigma|$.

(3) $|\mathcal{D}| = |M| + k$, where k is the number of $\forall$ and $\downarrow$ occurring in $\sigma_1, \ldots, \sigma_n, \sigma$.

(4) If $\mathcal{D}' \lhd x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash N : \sigma$ is a lazy and cut-free derivation, then $|N| \leq |M|$ implies $|\mathcal{D}'| \leq |\mathcal{D}|$.

(5) The set of values with lazy type σ is finite.

Proof. The assumptions on $\mathcal{D}$ allow to apply Lemma 10.(5) which implies that every judgment in $\mathcal{D}$ is lazy and free of $\forall$L, d, w, c and p (with a non empty context). Hence, we can prove Points (1)-(3) by induction on the structure of $\mathcal{D}$. Point (4) is a corollary of Point (3). Point (5) is a corollary of Point (2). ☐

Definition 16 (Height of an inference rule)**.** Let $\mathcal{D} \lhd \Gamma \vdash M : \sigma$ be a derivation and r a rule instance in it. The *height of* r, written $h(r)$, is the number of rule instances from the conclusion $\Gamma \vdash M : \sigma$ of $\mathcal{D}$ upward to the conclusion of r. The *height of* $\mathcal{D}$, written $h(\mathcal{D})$, is the largest $h(r)$ among its rule instances. ☐

Lemma 12 and 13 assure that we can eliminate exponential lazy cuts from a lazy derivation.

Lemma 12 (Existence of a lazy *cut*)**.** *Let $\mathcal{D}$ be a lazy derivation with only exponential cuts in it. At least one of those cuts is* safe.

Proof. Let $\Gamma \vdash M : \tau$ be the conclusion of $\mathcal{D}$. By contradiction, let us suppose that every occurrence of (exponential) *cut* in $\mathcal{D}$ is a deadlock. At least one of them, say c_m, has minimal height $h(c_m)$, i.e. no other *cut* occurs in the sequence of rule instances, say $r_1, \ldots r_n$, from the conclusion of c_m down to the one of $\mathcal{D}$. Since c_m is a deadlock, its leftmost premise has form $\Delta \vdash N : \downarrow\sigma$, where $\Delta \neq \emptyset$. By Proposition 9, Δ is strictly exponential and the whole $\Delta \vdash N : \downarrow\sigma$ is a non lazy judgment by Lemma 10.(1) and Lemma 10.(2). The contraposition of Lemma 10.(4) implies that the non lazy judgment in c_m can only be transformed to a non lazy judgment by every r_i, with $1 \leq i \leq n$, letting the conclusion of $\mathcal{D}$ non lazy, so contradicting the assumption. Hence, c_m must be safe. ☐

Lemma 13 (Eliminating a lazy *cut*)**.** *Let $\mathcal{D}$ be a lazy derivation with only exponential cuts in it. A lazy derivation $\mathcal{D}^*$ exists such that both $\mathcal{D} \rightsquigarrow \mathcal{D}^*$, by reducing a lazy cut, and $|\mathcal{D}^*| < |\mathcal{D}|$.*

Proof. Lemma 12 implies that $\mathcal{D}$ contains at least an exponential cut which is safe. Let us take (p, X) with maximal height $h((p, X))$ among those safe instances of *cut*. So, if (p, X) has form:

$$\dfrac{\dfrac{\dfrac{\mathcal{D}'}{\vdash N : \sigma}}{\vdash N : \downarrow\sigma}\, p \qquad \dfrac{\vdots}{\Delta, x : \downarrow\sigma \vdash M : \tau}\, X}{\Delta \vdash M[N/x] : \tau}\, cut \quad , \tag{11}$$

then $\mathcal{D}'$ is a lazy derivation because $\downarrow\sigma$ is a lazy type by Lemma 10.(1). Since $\mathcal{D}'$ is lazy and can only contain exponential cuts, by Lemma 12 and by maximality of $h((p, X))$, it is forcefully cut-free. So, by Lemma 11.(1), we have that N is a value, i.e. (p, X) is lazy and we can reduce it to obtain $\mathcal{D}^*$. If X in (11) is d or w, then it is simple to show that $|\mathcal{D}^*| < |\mathcal{D}|$. Let X be c. Then, (11) is:

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}'\\ \vdash V:\sigma\end{array}}{\vdash V:\downarrow\sigma}\,p \qquad \dfrac{\begin{array}{c}\mathcal{D}''\\ \Delta, y:\downarrow\sigma, z:\downarrow\sigma \vdash M':\tau\end{array} \qquad \begin{array}{c}\mathcal{D}'''\\ \vdash U:\sigma\end{array}}{\Delta, x:\downarrow\sigma\vdash \texttt{copy}^U_\sigma\ x\ \texttt{as}\ y,z\ \texttt{in}\ M':\tau}\,c}{\Delta\vdash \texttt{copy}^U_\sigma\ V\ \texttt{as}\ y,z\ \texttt{in}\ M':\tau}\,cut \quad , \tag{12}$$

with $\mathcal{D}'''$ lazy and cut-free for the same reasons as $\mathcal{D}'$ is. So, (12) can reduce to:

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}'\\ \vdash V:\sigma\end{array}}{\vdash V:\downarrow\sigma}\,p \qquad \dfrac{\dfrac{\begin{array}{c}\mathcal{D}'\\ \vdash V:\sigma\end{array}}{\vdash V:\downarrow\sigma}\,p \qquad \begin{array}{c}\mathcal{D}''\\ \Delta, y:\downarrow\sigma, z:\downarrow\sigma \vdash M':\tau\end{array}}{\Delta, z:\downarrow\sigma\vdash M'[V/y]:\tau}\,cut}{\Delta\vdash M'[V/y, V/z]:\tau}\,cut \quad . \tag{13}$$

By Lemma 11.(5), we can safely assume that U is a value with largest size among values of type σ. So, Lemma 11.(2) implies $|V| \leq |U|$, from which $|\mathcal{D}'| \leq |\mathcal{D}'''|$, by Lemma 11.(4). By applying Definition 15.4 to (12) and (13), we have $|\mathcal{D}^*| < |\mathcal{D}|$. □

Definition 17 (Lazy cut-elimination strategy)**.** Let $\mathcal{D}$ be a lazy derivation of LEM. Let a *round* on $\mathcal{D}$ be defined as follows:

{1} Let eliminate all the commuting instances of *cut*.

{2} If any instance of *cut* remains, it is necessarily symmetric. Let reduce a multiplicative cut, if any. Otherwise, let reduce a lazy exponential cut, if any.

The *lazy cut-elimination strategy* iterates rounds, starting from $\mathcal{D}$, until instances of *cut* exist in the obtained derivation. □

Theorem 14 (Lazy cut-elimination has a cubic bound)**.** *Let $\mathcal{D}$ be a lazy derivation. The lazy cut-elimination can reduce $\mathcal{D}$ to a cut-free $\mathcal{D}^*$ in $\mathcal{O}(|\mathcal{D}|^3)$ steps.*

Proof. Let $H(\mathcal{D})$ be the sum of the heights $h(\mathcal{D}')$ of all sub-derivations $\mathcal{D}'$ of $\mathcal{D}$ whose conclusion is an instance of *cut*. We proceed by induction on the lexicographically order of the pairs $\langle|\mathcal{D}|, H(\mathcal{D})\rangle$. To show that the lazy cut-elimination strategy in Definition 17 terminates, we start by applying a round to $\mathcal{D}$, using step {1}. Every commuting cut-elimination step just moves an instance of *cut* upward, strictly decreasing $H(\mathcal{D})$ and leaving $|\mathcal{D}|$ unaltered. Let us continue by applying step {2} of the round. As usual, $|\mathcal{D}|$ shrinks when eliminating a multiplicative cut. If only exponential instances of *cut* remain, by Lemma 13 we can rewrite $\mathcal{D}$ to $\mathcal{D}'$ by reducing a lazy exponential cut in such a way that $|\mathcal{D}'| < |\mathcal{D}|$. Therefore, the lazy cut-elimination strategy terminates with a cut-free derivation $\mathcal{D}^*$.

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

$$\mathcal{D} = \mathcal{D}_0 \underbrace{\longrightarrow}_{cc_0} \mathcal{D}'_0 \rightsquigarrow \mathcal{D}_1 \cdots \underbrace{\longrightarrow}_{cc_i} \mathcal{D}'_i \rightsquigarrow \mathcal{D}_{i+1} \underbrace{\longrightarrow}_{cc_{i+1}} \cdots \mathcal{D}'_{n-1} \rightsquigarrow \mathcal{D}_n \underbrace{\longrightarrow}_{cc_n} \mathcal{D}'_n = \mathcal{D}^* \quad , \qquad (14)$$

where every cc_j denotes the number of commuting cuts applied from derivation $\mathcal{D}_j$ to derivation $\mathcal{D}'_j$, for every $0 \le j \le n$. A bound on every cc_j is $|\mathcal{D}_j|^2$ because every instance of rule in $\mathcal{D}_j$ can, in principle, be commuted with every other. The first part of the proof implies $|\mathcal{D}_j| = |\mathcal{D}'_j|$, for every $0 \le j \le n$. Lemma 13 implies $|\mathcal{D}'_j| > |\mathcal{D}_{j+1}|$, for every $0 \le j \le n-1$. So, $n \le |\mathcal{D}|$ and the total number of cut-elimination steps in (14) is $O(|\mathcal{D}| \cdot |\mathcal{D}|^2)$. □

Remark 4. The cubic bound on the lazy strategy keeps holding also in the case we apply the *lazy* cut-elimination to *non lazy* derivations. Of course, in that case, deadlocks may remain in the final derivation where no instance of *cut* can be further eliminated. □

5.2. *Subject reduction theorem*

The proof of the Subject reduction requires some typical preliminaries.

Lemma 15 (Substitution)**.** *If $\Gamma \vdash M : \tau$ then $\Gamma[A/\alpha] \vdash M : \tau[A/\alpha]$, for every* linear *type A.*

Lemma 16 (Generation)**.**

(1) If $\mathcal{D} \lhd \Gamma \vdash \lambda x.M : \tau$, then $\tau = \downarrow^n \forall \vec{\alpha}.(\sigma \multimap A)$, where $\downarrow^n \triangleq \downarrow \overset{n}{\ldots} \downarrow$ and $\vec{\alpha} = \alpha_1, \ldots, \alpha_m$, for some $n \ge 0$ and $m \ge 0$.

(2) If $\mathcal{D} \lhd \Delta, x : \forall \alpha.A \vdash P : \tau$, then an instance r of $\forall$L exists in $\mathcal{D}$ with conclusion $\Delta', x : \forall \alpha.A \vdash P' : \tau'$, for some Δ', P' and τ'. I.e., r introduces $x : \forall \alpha.A$.

(3) If $\mathcal{D} \lhd \Delta, x : \sigma \multimap B \vdash P[xN/y] : \tau$, then an instance r of $\multimap$L exists in $\mathcal{D}$ with conclusion $\Delta', x : \sigma \multimap B \vdash P'[xN'/y] : \tau'$, for some Δ', P', N' and τ'. I.e., r introduces $x : \sigma \multimap B$.

(4) If $\mathcal{D} \lhd \Gamma \vdash \lambda x.M : \forall \alpha.A$, then a derivation $\mathcal{D}'$ exists which is $\mathcal{D}$ with some rule permuted in order to obtain an instance of $\forall$R as last rule of $\mathcal{D}'$.

(5) If $\mathcal{D} \lhd \Gamma \vdash \lambda x.P : \sigma \multimap B$, then a derivation $\mathcal{D}'$ exists which is $\mathcal{D}$ with some rule permuted in order to obtain an instance of $\multimap$R as last rule of $\mathcal{D}'$.

(6) If $\mathcal{D} \lhd \Delta, x : \downarrow\sigma \vdash P[xN/y] : \tau$, then an instance r of d exists in $\mathcal{D}$ with conclusion $\Delta', x : \downarrow\sigma \vdash P'[xN'/y] : \tau'$, for some Δ', P', N' and τ'. I.e., d introduces $x : \downarrow\sigma$.

(7) *If $\mathcal{D} \lhd \Gamma \vdash M : \downarrow\sigma$, then a derivation $\mathcal{D}'$ exists which is $\mathcal{D}$ with some rule permuted in order to get an instance of p as last rule of $\mathcal{D}'$.*

(8) *If $\mathcal{D} \lhd \Delta, x : \downarrow\sigma \vdash \texttt{discard}_\sigma\, x \texttt{ in } P : \tau$, then an instance r of w exists in $\mathcal{D}$ with conclusion $\Delta', x : \downarrow\sigma \vdash \texttt{discard}_\sigma\, x \texttt{ in } P' : \tau'$, for some Δ', P' and τ'. I.e., r introduces $x : \downarrow\sigma$.*

(9) *If $\mathcal{D} \lhd \Delta, x : \downarrow\sigma \vdash \texttt{copy}^U_\sigma\, x \texttt{ as } x_1, x_2 \texttt{ in } P : \tau$, then an instance r of c exists in $\mathcal{D}$ with conclusion $\Delta', x : \downarrow\sigma \vdash \texttt{copy}^U_\sigma\, x \texttt{ as } x_1, x_2 \texttt{ in } P' : \tau'$, for some Δ', P' and τ'. I.e., r introduces $x : \downarrow\sigma$.*

Proof. We can adapt the proof by Gaboardi&Ronchi in [8] to LEM because the types in Definition 9 are a sub-set of Gaboardi&Ronchi's *essential types*. In particular, Point *(7)* relies on Proposition 9. □

Theorem 17 (Subject reduction)**.** *If $\Gamma \vdash M : \tau$ and $M \to M'$, then $\Gamma \vdash M' : \tau$.*

Proof. We proceed by structural induction on $\mathcal{D}$. The crucial case of $M \to M'$ (Figure 2) is when $(\lambda x.P)Q$ exists in M and $\mathcal{D}$ contains:

$$\dfrac{\mathcal{D}' \lhd \Delta \vdash \lambda x.P : \sigma \qquad \mathcal{D}'' \lhd \Sigma, y : \sigma \vdash N[yQ/z] : \tau}{\mathcal{D} \lhd \Delta, \Sigma \vdash N[(\lambda x.P)Q/z] : \tau}\,cut \quad .$$

Lemma 16.*(1)* implies that $\sigma = \downarrow^n \forall\vec{\alpha}.(\sigma_1 \multimap C)$, where $\downarrow^n \triangleq \downarrow .\overset{n}{\cdots}. \downarrow$ and $\vec{\alpha} = \alpha_1, \dots, \alpha_m$, for some $n \geq 0$ and $m \geq 0$. Lemma 16.*(2)*, 16.*(3)*, and 16.*(6)*, imply that $\mathcal{D}''$ has form:

$$\begin{array}{c}
\dfrac{\dfrac{\vdots}{\Sigma''_1 \vdash Q'' : \sigma'_1} \qquad \dfrac{\vdots}{\Sigma''_2, z : C' \vdash N'' : \tau''}}{\Sigma''_1, \Sigma''_2, y''' : \sigma'_1 \multimap C' \vdash N''[y'''Q''/z] : \tau''}\multimap\mathrm{L} \\
\vdots \\
\dfrac{\Sigma', y'' : \forall\vec{\alpha}.(\sigma_1 \multimap C) \vdash N'[y''Q'/z] : \tau'}{\Sigma', y' : \downarrow\forall\vec{\alpha}.(\sigma_1 \multimap C) \vdash N'[y'Q'/z] : \tau'}\,d \\
\vdots \\
\Sigma, y : \downarrow^n \forall\vec{\alpha}.(\sigma_1 \multimap C) \vdash N[yQ/z] : \tau
\end{array} \tag{15}$$

where $\sigma'_1 = \sigma_1[A_1/\alpha_1, \dots, A_m/\alpha_m]$ and $C' = C[A_1/\alpha_1, \dots, A_m/\alpha_m]$, for some $\Sigma', \Sigma''_1, \Sigma''_2, y', y'', y''', N', N'', Q', Q'', \tau', \tau'', A_1, \dots, A_m$. Lemma 16.*(4)*, 16.*(5)* and 16.*(7)* imply that, permuting some of its rules, $\mathcal{D}'$ can be reorganized as:

$$\begin{array}{cl}
\vdots & \\
\Delta, x : \sigma_1 \vdash P : C & \\
\hline
\Delta \vdash \lambda x.P : \sigma_1 \multimap C & \multimap\mathrm{R} \\
\hline\hline
\Delta \vdash \lambda x.P : \forall\vec{\alpha}.(\sigma_1 \multimap C) & \forall\mathrm{R} \\
\hline\hline
\Delta \vdash \lambda x.P : \downarrow^n \forall\vec{\alpha}.(\sigma_1 \multimap C) & p
\end{array} \quad ,$$

where the concluding instances of p are necessary if $n > 0$ and are legally introduced because Δ is strictly exponential as consequence of Proposition 9 that we can apply to the judgment beacause $\downarrow\sigma$ is strictly exponential as well. Moreover, Lemma 15 assures that a derivation of $\Delta, x : \sigma_1' \vdash P : C'$ exists because $\alpha_1, \ldots, \alpha_m$ are not free in Δ. Therefore:

$$\dfrac{\dfrac{\dfrac{\vdots}{\Sigma_1'' \vdash Q'' : \sigma_1'} \qquad \dfrac{\vdots}{\Delta, x : \sigma_1' \vdash P : C'}}{\Delta, \Sigma_1'' \vdash P[Q''/x] : C'}\,cut \qquad \dfrac{\vdots}{\Sigma_2'', z : C' \vdash N'' : \tau''}}{\dfrac{\Delta, \Sigma_1'', \Sigma_2'' \vdash N''[P[Q''/x]/z] : \tau''}{\dfrac{\vdots}{\Delta, \Sigma \vdash N[P[Q/x]/z] : \tau}}}\,cut \quad .$$

which concludes with the same rules as in (15). A similar proof exists, which relies on Lemma 16.*(8)*, or Lemma 16.*(9)*, when reducing $\texttt{discard}_\sigma\ V\ \texttt{in}\ M$, or $\texttt{copy}^U_\sigma\ V\ \texttt{as}\ y, z\ \texttt{in}\ M$. All the remaining cases are straightforward. □

6. Translation of LEM into IMLL$_2$ and exponential compression

The system LEM provides a logical status to copying and erasing operations that exist in IMLL$_2$. In what follows, we show that a translation $(\_)^\bullet$ from LEM into IMLL$_2$ exists that "unpacks" both the constructs $\texttt{discard}_\sigma$ and $\texttt{copy}^V_\sigma$ by turning them into, respectively, an eraser and a duplicator of ground types. Then, we show that the reduction steps in Figure 2 and the commuting conversions in Figure 3 can be simulated by the $\beta\eta$-reduction of the linear λ-calculus, as long as we restrict to terms of $\Lambda_\downarrow$ typable in LEM. Last, we discuss the complexity of the translation, and we prove that every term typable in LEM is mapped to a linear λ-term whose size can be is exponential in the one of the original term.

We start with defining the translation from LEM to IMLL$_2$.

Definition 18 (From LEM to IMLL$_2$)**.** The map $(\_)^\bullet : \mathsf{LEM} \longrightarrow \mathsf{IMLL}_2$ translates a derivation $\mathcal{D} \lhd \Gamma \vdash_{\mathsf{LEM}} M : \tau$ into a derivation $\mathcal{D}^\bullet \lhd \Gamma^\bullet \vdash_{\mathsf{IMLL}_2} M^\bullet : \tau^\bullet$:

1. For all types $\sigma \in \Theta_\downarrow$:

$$\begin{aligned} \alpha^\bullet &\triangleq \alpha \\ (\tau \multimap A)^\bullet &\triangleq \tau^\bullet \multimap A^\bullet \\ (\forall\alpha.A)^\bullet &\triangleq \forall\alpha.A^\bullet \\ (\downarrow\tau)^\bullet &\triangleq \tau^\bullet \quad . \end{aligned}$$

2. For all contexts $\Gamma = x_1 : \sigma_1, \ldots, x_n : \sigma_n$, we set $\Gamma^\bullet \triangleq x_1 : \sigma_1^\bullet, \ldots, x_n : \sigma_n^\bullet$;

$$\left(\dfrac{\begin{array}{c}\mathcal{D}\\ x_1:\downarrow\sigma_1,\ldots,x_n:\downarrow\sigma_n\vdash M:\sigma\end{array}}{x_1:\downarrow\sigma_1,\ldots,x_n:\downarrow\sigma_n\vdash M:\downarrow\sigma}\,p\right)^{\bullet}\triangleq\left(\begin{array}{c}\mathcal{D}\\ x_1:\downarrow\sigma_1,\ldots,x_n:\downarrow\sigma_n\vdash M:\sigma\end{array}\right)^{\bullet}$$

$$\left(\dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma,x:\sigma\vdash M:\tau\end{array}}{\Gamma,y:\downarrow\sigma\vdash M[y/x]:\tau}\,d\right)^{\bullet}\triangleq\dfrac{\dfrac{}{y:\sigma^{\bullet}\vdash y:\sigma^{\bullet}}\,\mathrm{ax}\qquad\left(\begin{array}{c}\mathcal{D}\\ \Gamma,x:\sigma\vdash M:\tau\end{array}\right)^{\bullet}}{\Gamma^{\bullet},y:\sigma^{\bullet}\vdash M^{\bullet}[y/x]:\tau^{\bullet}}\,cut$$

$$\left(\dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma\vdash M:\tau\end{array}}{\Gamma,x:\downarrow\sigma\vdash\mathtt{discard}_{\sigma}\,x\ \mathtt{in}\ M:\tau}\,w\right)^{\bullet}\triangleq\dfrac{\begin{array}{c}\vdots\\ x:\sigma^{\bullet}\vdash\mathtt{E}_{\sigma^{\bullet}}\,x:\mathbf{1}\end{array}\qquad\begin{array}{c}\left(\begin{array}{c}\mathcal{D}\\ \Gamma\vdash M:\tau\end{array}\right)^{\bullet}\\ \vdots\\ \Gamma^{\bullet},y:\mathbf{1}\vdash\mathtt{let}\ y\ \mathtt{be}\ I\ \mathtt{in}\ M^{\bullet}:\tau^{\bullet}\end{array}}{\Gamma^{\bullet},x:\sigma^{\bullet}\vdash\mathtt{let}\ \mathtt{E}_{\sigma^{\bullet}}\,x\ \mathtt{be}\ I\ \mathtt{in}\ M^{\bullet}:\tau^{\bullet}}\,cut$$

$$\left(\dfrac{\begin{array}{c}\mathcal{D}_1\\ \Gamma,x_1:\downarrow\sigma,x_2:\downarrow\sigma\vdash M:\tau\end{array}\qquad\begin{array}{c}\mathcal{D}_2\\ \vdash V:\sigma\end{array}}{\Gamma,x:\downarrow\sigma\vdash\mathtt{copy}^{V}_{\sigma}\,x\ \mathtt{as}\ x_1,x_2\ \mathtt{in}\ M:\tau}\,c\right)^{\bullet}\triangleq\dfrac{\begin{array}{c}\mathcal{D}_2^{\bullet}\\ \vdash V^{\bullet}:\sigma^{\bullet}\\ \vdots\\ x:\sigma^{\bullet}\vdash\mathtt{D}^{V^{\bullet}}_{\sigma^{\bullet}}\,x:\sigma^{\bullet}\otimes\sigma^{\bullet}\end{array}\qquad\begin{array}{c}\left(\begin{array}{c}\mathcal{D}_1\\ \Gamma,x_1:\downarrow\sigma,x_2:\downarrow\sigma\vdash M:\tau\end{array}\right)^{\bullet}\\ \vdots\\ \Gamma^{\bullet},y:\sigma^{\bullet}\otimes\sigma^{\bullet}\vdash\mathtt{let}\ y\ \mathtt{be}\ x_1,x_2\ \mathtt{in}\ M^{\bullet}:\tau^{\bullet}\end{array}}{\Gamma^{\bullet},x:\sigma^{\bullet}\vdash\mathtt{let}\ \mathtt{D}^{V^{\bullet}}_{\sigma^{\bullet}}\,x\ \mathtt{be}\ x_1,x_2\ \mathtt{in}\ M^{\bullet}:\tau^{\bullet}}\,cut$$

Figure 6: The translation of the rules p, d, w and c.

3. For all typable terms $M \in \Lambda_{\downarrow}$:

$$\begin{aligned} x^{\bullet} &\triangleq x \\ (\lambda x.P)^{\bullet} &\triangleq \lambda x.P^{\bullet} \\ (PQ)^{\bullet} &\triangleq P^{\bullet}Q^{\bullet} \\ (\mathtt{discard}_{\sigma}\ P\ \mathtt{in}\ Q)^{\bullet} &\triangleq \mathtt{let}\ \mathtt{E}_{\sigma^{\bullet}}\ P^{\bullet}\ \mathtt{be}\ I\ \mathtt{in}\ Q^{\bullet} \\ (\mathtt{copy}^{V}_{\sigma}\ P\ \mathtt{as}\ x_1,x_2\ \mathtt{in}\ Q)^{\bullet} &\triangleq \mathtt{let}\ \mathtt{D}^{V^{\bullet}}_{\sigma^{\bullet}}\ P^{\bullet}\ \mathtt{be}\ x_1,x_2\ \mathtt{in}\ Q^{\bullet}\quad, \end{aligned}$$

where $\mathtt{E}_{\sigma^{\bullet}}$ and $\mathtt{D}^{V^{\bullet}}_{\sigma^{\bullet}}$ (see Remark 6 in Appendix A) are the eraser and the duplicator of $\sigma^{\bullet}$ which is both ground, because σ is closed and with no negative occurrences of $\forall$, and inhabited by $V^{\bullet}$.

4. The definition of $(\_)^{\bullet}$ extends to any derivation $\mathcal{D} \lhd \Gamma \vdash M : \sigma$ of LEM in the obvious way, following the structure of $M^{\bullet}$. Figure 6 collects the most interesting cases. □

Remark 5. Both $\mathtt{E}_{\sigma^{\bullet}}$ and $\mathtt{D}^{V^{\bullet}}_{\sigma^{\bullet}}$ in point 3 of Definition 18 here above exist by Theorem 7 and Theorem 8. □

The simulation theorem requires some preliminaries.

Lemma 18. *For every typable value V:*

(1) $V^{\bullet} = V$.

(2) V has type σ if and only if $V^{\bullet}$ has type $\sigma^{\bullet}$.

Proof. Straightforward consequence of Definition 18. □

Lemma 19. *For all terms $M, N \in \Lambda_\downarrow$ typable in* LEM, $M^\bullet[N^\bullet/x] = (M[N/x])^\bullet$.

Proof. We can proceed by a standard structural induction on M. □

We now show that every reduction on terms typable in LEM can be simulated in the linear λ-calculus by means of the $\beta\eta$-reduction relation. We recall that Subject reduction holds on every typable $M \in \Lambda_\downarrow$ (Theorem 17). Moreover, every linear λ-term that has a type in IMLL, has one in IMLL_2 (see [11]). So, we state the simulation theorem for terms, rather than the related derivations.

Theorem 20 (Simulation). *Let $\mathcal{D} \lhd \Gamma \vdash M : \sigma$ be a derivation in* LEM. *If $M_1 \to M_2$ then $M_1^\bullet \to^*_{\beta\eta} M_2^\bullet$:*

$$\begin{tikzcd} M_1 \arrow[r] \arrow[d, dashed] & M_2 \arrow[d, dashed] \\ M_1^\bullet \arrow[r, "*", "\beta\eta"'] & M_2^\bullet \end{tikzcd} .$$

Proof. We can proceed by structural induction on M_1. One of the most interesting cases is when M_1 is $(\lambda x.P)Q$ and $M_2 = P[Q/x]$. Lemma 19 implies $((\lambda y.P)Q)^\bullet = (\lambda y.P^\bullet)Q^\bullet \to_\beta P^\bullet[Q^\bullet/x] = (P[Q/x])^\bullet$. If M_1 is $\texttt{discard}_\sigma\, V \texttt{ in } N$ and M_2 is N, then V is a value of type σ. By Lemma 18.(2), $V^\bullet$ is a value of $\sigma^\bullet$. Hence:

$$(\texttt{discard}_\sigma\, V \texttt{ in } N)^\bullet \triangleq \texttt{let } \texttt{E}_{\sigma^\bullet}\, V^\bullet \texttt{ be } I \texttt{ in } N^\bullet \to^*_\beta N^\bullet$$

by Theorem 7. If M_1 is $\texttt{copy}^U_\sigma\, V \texttt{ as } x_1, x_2 \texttt{ in } N$ and M_2 is $N[V/x_1, V/x_2]$, then U and V are both values of type σ. By Lemma 18.(2), $U^\bullet$ and $V^\bullet$ are both values of type $\sigma^\bullet$. Hence:

$$\begin{aligned}
(\texttt{copy}^U_\sigma\, V \texttt{ as } x_1, x_2 \texttt{ in } N)^\bullet &\triangleq \texttt{let } \texttt{D}^{U^\bullet}_{\sigma^\bullet}\, V^\bullet \texttt{ be } x_1, x_2 \texttt{ in } N^\bullet \\
&\to^*_{\beta\eta} \texttt{let } \langle V^\bullet, V^\bullet\rangle \texttt{ be } x_1, x_2 \texttt{ in } N^\bullet && \text{Theorem 8} \\
&\to^*_\beta N^\bullet[V^\bullet/x_1, V^\bullet/x_2] \\
&\triangleq (N^\bullet[V^\bullet/x_1])[V^\bullet/x_2] \\
&= ((N[V/x_1])[V/x_2])^\bullet && \text{Lemma 19} \\
&\triangleq (N[V/x_1, V/x_2])^\bullet \ .
\end{aligned}$$

□

We conclude by estimating the impact of the translation on the size of terms produced by “unpacking” the constructs $\texttt{discard}_\sigma$ and $\texttt{copy}^V_\sigma$. We need to bound the dimension of erasers and duplicators with ground type A. We rely on the map $(\_)^-$ (Definition 24 in Appendix A) that, intuitively, strips every occurrence of $\forall$ away from a given type (Remark 6 in Appendix A.)

Lemma 21 (Size of duplicators and erasers). *For every ground type A:*

(1) $|\mathtt{E}_A| \in \mathcal{O}(|A^-|)$.

(2) If V is a value of A, then $|\mathtt{D}_A^V| \in \mathcal{O}(2^{|A^-|^2})$.

Proof. Point (1) is straightforward by looking at the proof of Theorem 7. Concerning Point (2), from Appendix A we know that $\mathtt{D}_A^V$ is $\mathtt{dec}_A^s \circ \mathtt{enc}_A^s \circ \mathtt{sub}_A^s$, where $s = \mathcal{O}(|A^-| \cdot \log |A^-|)$. The components of $\mathtt{D}_A^V$ with a size not linear in $|A^-|$ are $\mathtt{dec}_A^s$ and $\mathtt{enc}_A^s$. The λ-term $\mathtt{dec}_A^s$ (see point 3 in Section 4) nests occurrences of `if-then-else` each containing 2^s pairs of normal inhabitants of A, every of which, by Lemma 29, has size bounded by $|A^-|$. Similarly, $\mathtt{enc}_A^s$ alternates instances of λ-terms $\mathtt{abs}^s$ and $\mathtt{app}^s$ which, again, nest occurrences of `if-then-else` every one with 2^s instances of boolean strings of size s. The overall complexity of $\mathtt{D}_A^V$ is $\mathcal{O}(s \cdot 2^s) = \mathcal{O}(2^{|A^-|^2})$. □

Theorem 22 (Exponential Compression)**.** *Let $\mathcal{D} \lhd \Gamma \vdash M : \sigma$ be a derivation in* LEM*. Then, $|M^\bullet| = \mathcal{O}(2^{|M|^k})$, for some $k \geq 1$.*

Proof. The proof is by structural induction on M. The interesting case is when M is $\mathtt{copy}_\sigma^V\, P$ `as` x_1, x_2 `in` Q. Since M is typable, V has type σ, that is closed and free from negative occurrences of $\forall$, hence lazy. By Lemma 11.(5), it is safe to assume that V is a value with largest size among values of type σ. By Lemma 18, $V = V^\bullet$ is also the largest value of type $\sigma^\bullet$ in IMLL_2. Finally, by Lemma 29, this implies that V is a η-long normal form of type $\sigma^\bullet$. Now, by using Definition 3, we have $|M^\bullet| = |\mathtt{let}\ \mathtt{D}_{\sigma^\bullet}^V\, P^\bullet\ \mathtt{be}\ x_1, x_2\ \mathtt{in}\ Q^\bullet| = |\mathtt{D}_{\sigma^\bullet}^V| + |P^\bullet| + |Q^\bullet| + 4$. On the one hand, by induction hypothesis, we obtain $|P^\bullet| = \mathcal{O}(2^{|P|^{k'}})$ and $|Q^\bullet| = \mathcal{O}(2^{|Q|^{k''}})$, for some $k', k'' \geq 1$. On the other hand, by applying both Lemma 21 and Lemma 29, we have $|\mathtt{D}_{\sigma^\bullet}^V| = \mathcal{O}(2^{|A^-|^2}) = \mathcal{O}(2^{2 \cdot |V|^2})$. Therefore, there exists $k \geq 1$ such that $|M^\bullet| = \mathcal{O}(2^{(|V|+|P|+|Q|+1)^k}) = \mathcal{O}(2^{|M|^k})$. □

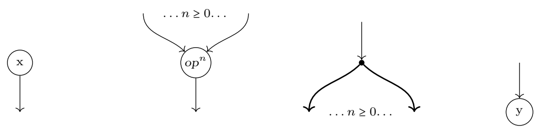
7. The expressiveness of LEM and applications

Theorem 20 says that LEM is not "algorithmically" more expressive than IMLL_2. Nonetheless, terms with type in LEM, and their evaluation mechanisms, exponentially compress the corresponding linear λ-terms and evaluations in IMLL_2 (Theorem 22). The goal of this section is to explore the benefits of this compression.

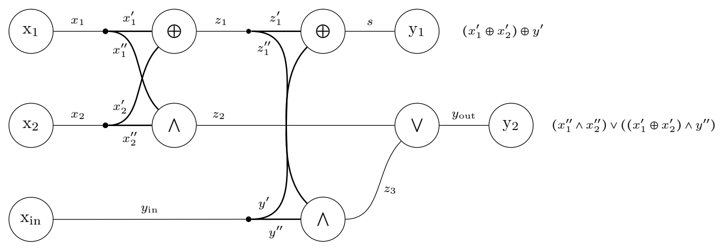
7.1. Boolean circuits in LEM

We encode boolean circuits as terms of LEM (Definition 21) and we prove a simulation result (Proposition 23).

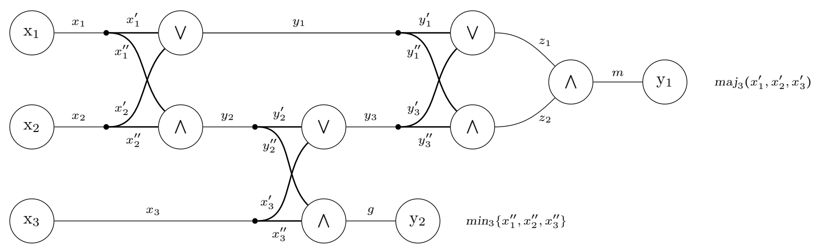
The encoding is inspired by Mairson&Terui [15]. Other encodings of the boolean circuits have been shown in Terui [26], Mogbil&Rahli [18] and Aubert [2] by considering the *unbounded* proof-nets for the multiplicative fragment MLL of Linear Logic. Unbounded proof-nets are an efficient language able to express n-ary tensor products by single nodes and to characterize parallel computational



(a) From left, input, internal, fan-out, and output nodes.



(b) The 2-bits full-adder boolean circuit



(c) The 3-bits majority boolean circuit

Figure 7: Nodes of boolean circuits and some examples. Writing, for example, $maj_3\{x_1', x_2', x_3'\}$ in place of $maj_3\{x_1, x_2, x_3\}$ would be equivalent. The current notation just highlights which is the component of the fan-out nodes that an output depends on.

complexity classes such as NC, AC, and $\mathsf{P}/_{\text{poly}}$. The contribution of this work to these encodings is in the use of `copy` and `discard` to directly express the fan-out nodes that allow a more compact and modular representation of circuits. In particular, as compared to [26, 18, 2], our encoding is able to get rid of the garbage that accumulates in the course of the simulation.

We start by briefly recalling the basics of boolean circuits from Vollmer [28].

Definition 19 (Boolean circuits)**.** A *boolean circuit* C is a finite, directed and acyclic graph with n *input nodes*, m *output nodes*, *internal nodes* and *fan-out nodes* as in Figure 7(a). The incoming (resp. outgoing) edges of a node are *premises* (resp. *conclusions*). The *fan-in* of an internal node is the number of its premises. Labels for the n input nodes of C are $\mathrm{x}_1, \dots, \mathrm{x}_n$ and those ones for the m outputs are $\mathrm{y}_1, \dots, \mathrm{y}_m$. Each internal node with fan-in $n \geq 0$ has an n-ary

$\neg$	$\texttt{not} \triangleq \lambda b.\lambda x.\lambda y.byx$	$: \mathbf{B} \multimap \mathbf{B}$
$\wedge^0$	$\texttt{and}^0 \triangleq \lambda x.\lambda y.\langle x, y\rangle$	$: \mathbf{B}$
$\wedge^1$	$\texttt{and}^1 \triangleq I$	$: \mathbf{B} \multimap \mathbf{B}$
$\wedge^2$	$\texttt{and}^2 \triangleq \lambda x_1.\lambda x_2.\pi_1(x_1 x_2\, \texttt{ff})$	$: \mathbf{B} \multimap \mathbf{B} \multimap \mathbf{B}$
$\wedge^{n+2}$	$\texttt{and}^{n+2} \triangleq \lambda x_1 \ldots x_{n+1} x_{n+2}.\texttt{and}^2\,(\texttt{and}^{n+1}\, x_1\, \ldots\, x_{n+1})\, x_{n+2}$	$: \mathbf{B} \multimap \overset{n+2}{\ldots} \multimap \mathbf{B} \multimap \mathbf{B}$
$\vee^0$	$\texttt{or}^0 \triangleq \lambda x.\lambda y.\langle y, x\rangle$	$: \mathbf{B}$
$\vee^1$	$\texttt{or}^1 \triangleq I$	$: \mathbf{B} \multimap \mathbf{B}$
$\vee^2$	$\texttt{or}^2 \triangleq \lambda x_1.\lambda x_2.\pi_1(x_1 \texttt{tt}\, x_2)$	$: \mathbf{B} \multimap \mathbf{B} \multimap \mathbf{B}$
$\vee^{n+2}$	$\texttt{or}^{n+2} \triangleq \lambda x_1 \ldots x_{n+1} x_{n+2}.\texttt{or}^2\,(\texttt{or}^{n+1}\, x_1\, \ldots\, x_{n+1})\, x_{n+2}$	$: \mathbf{B} \multimap \overset{n+2}{\ldots} \multimap \mathbf{B} \multimap \mathbf{B}$
fo^0	$\texttt{out}^0 \triangleq \lambda x.\texttt{discard}_{\mathbf{B}}\, x \texttt{ in } I$	$: \downarrow\mathbf{B} \multimap \mathbf{1}$
fo^1	$\texttt{out}^1 \triangleq I$	$: \downarrow\mathbf{B} \multimap \mathbf{B}$
fo^2	$\texttt{out}^2 \triangleq \lambda x.\texttt{copy}^{\texttt{tt}}_{\mathbf{B}}\, x \texttt{ as } x_1, x_2 \texttt{ in } \langle x_1, x_2\rangle$	$: \downarrow\mathbf{B} \multimap \downarrow\mathbf{B} \otimes \downarrow\mathbf{B}$
fo^{n+2}	$\texttt{out}^{n+2} \triangleq \lambda x.\texttt{copy}^{\texttt{tt}}_{\mathbf{B}}\, x \texttt{ as } x_1, x_2 \texttt{ in } \langle \texttt{out}^{n+1}\, x_1, x_2\rangle$	$: \downarrow\mathbf{B} \multimap \downarrow\mathbf{B} \otimes \overset{n+2}{\ldots} \otimes \downarrow\mathbf{B}$

Figure 8: Encoding of boolean functions and fan-out.

boolean function op^n as its label, provided that if $n = 0$, then op^n is a boolean constant in $\{0, 1\}$. The fan-out nodes have no label. Input and internal nodes are *logical nodes* and their conclusions are *logical edges*. If ν and ν' are logical nodes, then ν' is a *successor* of ν if a directed path from ν to ν' exists which crosses no logical node. The *size* $|C|$ of C is the number of nodes. Its *depth* $\delta(C)$ is the length of the longest path from an input node to a output node. A *basis* $\mathcal{B}$ is a set of boolean functions. A boolean circuit C is *over a basis* $\mathcal{B}$ if the label of every of its internal nodes belong to $\mathcal{B}$ only. The standard unbounded fan-in basis is $\mathcal{B}_1 = \{\neg, (\wedge^n)_{n\in\mathbb{N}}, (\vee^n)_{n\in\mathbb{N}}\}$. □

When representing boolean circuits as terms we label edges by λ-variables, we omit their orientation, we assume that every fan-out always has a logical edge as its premise and we draw non-logical edges, i.e. conclusions of fan-out nodes, as thick lines. Figures 7(b) and 7(c) are examples. A 2-bits full-adder is in the first one. It takes two bits x_1, x_2 and a carrier y_{in} as inputs. Its outputs are the sum $s = (x_1' \oplus x_2') \oplus y'$ and the carrier $y_{\text{out}} = (x_1'' \wedge x_2'') \vee ((x_1' \oplus x_2') \wedge y'')$, where $\oplus$ is the exclusive or that we can obtain by the functionally complete functions in $\mathcal{B}_1$. Figure 7(c) is the 3-bits majority function $maj_3(x_1, x_2, x_3)$. It serially composes three occurrences of the boolean circuit that switches two inputs x_1 and x_2 in order to put the greatest on the topmost output and the smallest on the bottommost one, under the convention that 0 is smaller than 1. So, the 3-bits majority circuit first sorts its input bits and then checks if the topmost two, i.e. the majority, are both set to 1. The lowermost output is garbage.

Translating boolean circuits as terms of LEM requires to encode the boolean functions in $\mathcal{B}_1$ and the fan-out nodes. Figure 8 reports them, where $\texttt{tt}$ and $\texttt{ff}$ encode the boolean values in (1), and π_1 is the projection in (4). As a

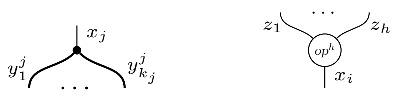


Figure 9: From left, a fan-out node and an internal node.

typographical convention, $\mathtt{i} \in \{\mathtt{tt}, \mathtt{ff}\}$ will code the boolean constant $i \in \{0,1\}$, and $\mathtt{op}^n$ the n-ary boolean function op^n, according to Figure 8. We shorten $\mathtt{and}^0$, $\mathtt{or}^0$, $\mathtt{and}^2$, $\mathtt{or}^2$, and $\mathtt{out}^2$ as $\mathtt{tt}$, $\mathtt{ff}$, $\mathtt{and}$, $\mathtt{or}$, and $\mathtt{out}$, respectively. The encoding of the binary exclusive or $\oplus$ is $\mathtt{xor}$.

We recall that boolean circuits are a model of parallel computation, while the λ-calculus models sequential computations. Mapping the former into the latter requires some technicalities. The notion of *level* allows to topologically sort the structure of the boolean circuits in order to preserve the node dependencies:

Definition 20 (Level)**.** The *level* l of a logical node ν in a boolean circuit C is:

1. 0 if ν has no successors, and
2. $\max\{l_1, \dots, l_k\} + 1$ if ν has successors $\nu_1, \dots, \nu_k$ with levels $l_1, \dots, l_k$.

The *level* of a logical edge is the level of the logical node it is the conclusion of. The *level* of a boolean circuit is the greatest level of its logical nodes. □

We define a level-by-level translation of unbounded fan-in boolean circuits over $\mathcal{B}_1$ into terms typable in LEM taking inspiration from Schubert [25]:

Definition 21 (From boolean circuits to terms)**.** Let C be a boolean circuit with n inputs and m outputs. We define the term $\mathtt{level}^l_C$ by induction on $l-1$:

1. $\mathtt{level}^{-1}_C \triangleq \langle x_1, \dots, x_n \rangle$, where $x_1, \dots, x_n$ are the variables labelling the logical edges of level 0.
2. $\mathtt{level}^l_C \triangleq (\lambda x_1 \dots x_n x_{n+1} \dots x_m.\mathtt{let}\,(\mathtt{out}^{k_1}\, x_1)\ \mathtt{be}\ y^1_1, \dots, y^1_{k_1}\ \mathtt{in}\ \dots$ $\mathtt{let}\,(\mathtt{out}^{k_n}\, x_n)\ \mathtt{be}\ y^n_1, \dots, y^n_{k_n}\ \mathtt{in}\ \mathtt{level}^{l-1}_C)\, B_1 \dots B_m$, where:
   (a) $x_1, \dots, x_n, x_{n+1}, \dots, x_m$ are the variables labelling the logical edges of level l,
   (b) for all $1 \le j \le n$, x_j is the premise of a fan-out node with conclusions labelled with $y^j_1, \dots, y^j_{k_j}$ (see Figure 9).
   (c) for all $1 \le i \le m$, if x_i is the variable labeling the conclusion of an internal node op^h with premises labeled by $z_1, \dots, z_h$, respectively (see Figure 9), then $B_i \triangleq \mathtt{op}^h\, z_1\, \dots\, z_h$. If x_i is the variable labelling the conclusion of an input node then $B_i \triangleq x_i$.

Last, if the input nodes have conclusions labeled by $x_1, \dots, x_n$, respectively, and if C has level l, then we define $\lambda(C) \triangleq \lambda x.\mathtt{let}\ x\ \mathtt{be}\ x_1, \dots, x_n\ \mathtt{in}\ \mathtt{level}^l_C$. □

Example 2 (2-bits full adder)**.** The level-by-level translation of the boolean circuit C in Figure 7(b) is the following:

$$\begin{aligned}
\texttt{level}^{-1}_C &\triangleq \langle s, y_{\text{out}}\rangle\\
\texttt{level}^{0}_C &\triangleq (\lambda s.\lambda y_{\text{out}}.\texttt{level}^{-1}_C)(\texttt{xor}\, z_1'\, y')(\texttt{or}\, z_2\, z_3)\\
\texttt{level}^{1}_C &\triangleq (\lambda z_2.\lambda z_3.\texttt{level}^{0}_C)(\texttt{and}\, x_1''\, x_2'')(\texttt{and}\, z_1''\, y'')\\
\texttt{level}^{2}_C &\triangleq (\lambda z_1.\lambda y_{\text{in}}.\texttt{let}\,(\texttt{out}\, z_1)\ \texttt{be}\ z_1', z_1''\ \texttt{in}\\
&\qquad\qquad \texttt{let}\,(\texttt{out}\, y_{\text{in}})\ \texttt{be}\ y', y''\ \texttt{in}\ \texttt{level}^{1}_C)(\texttt{xor}\, x_1'\, x_2')\, y_{\text{in}}\\
\texttt{level}^{3}_C &\triangleq (\lambda x_1.\lambda x_2.\texttt{let}\,(\texttt{out}\, x_1)\ \texttt{be}\ x_1', x_1''\ \texttt{in}\\
&\qquad\qquad \texttt{let}\,(\texttt{out}\, x_2)\ \texttt{be}\ x_2', x_2''\ \texttt{in}\ \texttt{level}^{2}_C)\, x_1\, x_2
\end{aligned}$$

where we set $\lambda(C) \triangleq \lambda x.\texttt{let}\, x\ \texttt{be}\ x_1, x_2, y_{\text{in}}\ \texttt{in}\ \texttt{level}^3_C$ which reduces to:

$$\begin{aligned}
&\lambda x.\texttt{let}\, x\ \texttt{be}\ x_1, x_2, y_{\text{in}}\ \texttt{in}\ (\texttt{let}\,(\texttt{out}\, x_1)\ \texttt{be}\ x_1', x_1''\ \texttt{in}\\
&\texttt{let}\,(\texttt{out}\, x_2)\ \texttt{be}\ x_2', x_2''\ \texttt{in}\ (\texttt{let}\,(\texttt{out}\, y_{\text{in}})\ \texttt{be}\ y', y''\ \texttt{in}\\
&\texttt{let}\,(\texttt{out}\,(\texttt{xor}\, x_1'\, x_2'))\ \texttt{be}\ z_1', z_2''\ \texttt{in}\ \langle \texttt{xor}\, z_1'\, y', \texttt{or}\,(\texttt{and}\, x_1''\, x_2'')(\texttt{and}\, z_1''\, y'')\rangle\ ))\ . \qquad \square
\end{aligned}$$

Example 3 (3-bits majority)**.** The level-by-level translation of the boolean circuit C in Figure 7(c) is the following:

$$\begin{aligned}
\texttt{level}^{-1}_C &\triangleq \langle m, g\rangle\\
\texttt{level}^{0}_C &\triangleq (\lambda m.\lambda g.\texttt{level}^{-1}_C)(\texttt{and}\, z_1\, z_2)(\texttt{and}\, y_2''\, x_3'')\\
\texttt{level}^{1}_C &\triangleq (\lambda z_1.\lambda z_2.\texttt{level}^{0}_C)(\texttt{or}\, y_1'\, y_3')(\texttt{and}\, y_1''\, y_3'')\\
\texttt{level}^{2}_C &\triangleq (\lambda y_1.\lambda y_3.\texttt{let}\,(\texttt{out}\, y_1)\ \texttt{be}\ y_1', y_1''\ \texttt{in}\\
&\qquad \texttt{let}\,(\texttt{out}\, y_3)\ \texttt{be}\ y_3', y_3''\ \texttt{in}\ \texttt{level}^{1}_C)(\texttt{or}\, x_1'\, x_2')(\texttt{or}\, y_2'\, x_3')\\
\texttt{level}^{3}_C &\triangleq (\lambda y_2.\lambda x_3.\texttt{let}\,(\texttt{out}\, y_2)\ \texttt{be}\ y_2', y_2''\ \texttt{in}\\
&\qquad \texttt{let}(\texttt{out}\, x_3)\ \texttt{be}\ x_3', x_3''\ \texttt{in}\ \texttt{level}^{2}_C)(\texttt{and}\, x_1''\, x_2'')\, x_3\\
\texttt{level}^{4}_C &\triangleq (\lambda x_1.\lambda x_2.\texttt{let}\,(\texttt{out}\, x_1)\ \texttt{be}\ x_1', x_1''\ \texttt{in}\\
&\qquad \texttt{let}\,(\texttt{out}\, x_2)\ \texttt{be}\ x_2', x_2''\ \texttt{in}\ \texttt{level}^{3}_C)\, x_1\, x_2
\end{aligned}$$

where we set $\lambda(C) \triangleq \lambda x.\texttt{let}\ x\ \texttt{be}\ x_1, x_2, x_3\ \texttt{in}\ \texttt{level}^4_C$ which reduces to:

$$\begin{aligned}
&\lambda x.\texttt{let}\ x\ \texttt{be}\ x_1, x_2, x_3\ \texttt{in}\ \texttt{let}\,(\texttt{out}\, x_1)\ \texttt{be}\ x_1', x_1''\ \texttt{in}\\
&\texttt{let}\,(\texttt{out}\, x_2)\ \texttt{be}\ x_2', x_2''\ \texttt{in}\ (\texttt{let}\,(\texttt{out}\, x_3)\ \texttt{be}\ x_3', x_3''\ \texttt{in}\\
&\texttt{let}\,(\texttt{out}\,(\texttt{or}\, x_1'\, x_2'))\ \texttt{be}\ y_1', y_1''\ \texttt{in}\ (\texttt{let}\,(\texttt{out}\,(\texttt{and}\, x_1''\, x_2''))\ \texttt{be}\ y_2', y_2''\ \texttt{in}\\
&\texttt{let}\,(\texttt{out}\,(\texttt{or}\, y_2'\, x_3'))\ \texttt{be}\ y_3', y_3''\ \texttt{in}\ \langle \texttt{and}\,(\texttt{or}\, y_1'\, y_3')(\texttt{and}\ y_1''\, y_3''), \texttt{and}\, y_2''\, x_3''\rangle\ ))\ . \qquad \square
\end{aligned}$$

The size of the term coding an internal node depends on its fan-in. Likewise, the size of the term coding a fan-out node depends on the number of conclusions. The size of the circuit bounds both values. Moreover, by Theorem 17, reducing a typable term yields a typable term. These observations imply:

Proposition 23 (Simulation of circuit evaluation)**.** *If C is an unbounded fan-in boolean circuit over $\mathcal{B}_1$ with n inputs and m outputs then $\lambda(C)$ is such that:*

1. *its size is $O(|C|)$,*
2. *it has type $(\downarrow\mathbf{B} \otimes \overset{n}{\ldots} \otimes \downarrow\mathbf{B}) \multimap (\mathbf{B} \otimes \overset{m}{\ldots} \otimes \mathbf{B})$ in LEM, and*
3. *for all $(i_1, \ldots, i_n) \in \{0,1\}^n$, the evaluation of C on input $(i_1, \ldots, i_n)$ outputs the tuple $(i'_1, \ldots, i'_m) \in \{0,1\}^m$ iff $\lambda(C)\,\langle \mathtt{i}_1, \ldots, \mathtt{i}_n\rangle \to^* \langle \mathtt{i}'_1, \ldots, \mathtt{i}'_m\rangle$.*

It should not be surprising that the translation cannot preserve the depth of a given circuit, since LEM has only *binary* logical operators. That is why we use nested instances "`let`" (Definition (3)) to access single elements of $A_1 \otimes \ldots \otimes A_n$. We could preserve the depth by extending LEM with unbounded tensor products as done, for example, in [26] for the multiplicative fragment of linear logic MLL.

7.2. Numerals in LEM

We introduce a class $\mathcal{N}$ of terms in LEM, called *numerals*, that represent natural numbers. We give a successor `S` and an addition `A` on numerals, both typable in LEM, and we show that they behave as expected using Subject reduction. Moreover, the numerals can operate as iterators on a class of terms in LEM that form a group with respect to the application.

Definition 22 (Terms and types for $\mathcal{N}$)**.** Let us recall that $\mathbf{1}$ is $\forall\alpha.\alpha \multimap \alpha$ and I is $\lambda x.x$ (Section 2). The numerals of $\mathcal{N}$ have form:

$$\begin{aligned} \overline{0} &\triangleq \lambda f x.\mathtt{discard}_{\mathbf{1}}\, f \,\mathtt{in}\, x : \mathbb{N} \\ \overline{1} &\triangleq \lambda f x. f x : \mathbb{N} \\ \overline{n+2} &\triangleq \lambda f x.\mathtt{copy}^{I}_{\mathbf{1}}\, f \,\mathtt{as}\, f_1 \ldots f_n \,\mathtt{in}\, f_1(\ldots(f_n\, x)\ldots) : \mathbb{N} \ , \end{aligned} \tag{16}$$

where, for any M, $\mathtt{copy}^{I}_{\mathbf{1}}\, f_0 \,\mathtt{as}\, f_1 \ldots f_n \,\mathtt{in}\, M$ in (16) stands for:

$$\mathtt{copy}^{I}_{\mathbf{1}}\, f_0 \,\mathtt{as}\, f_1, f'_2 \,\mathtt{in}\,(\mathtt{copy}^{I}_{\mathbf{1}}\, f'_2 \,\mathtt{as}\, f_2, f'_3 \,\mathtt{in}\, \ldots (\mathtt{copy}^{I}_{\mathbf{1}}\, f'_{n-1} \,\mathtt{as}\, f_{n-1}, f_n \,\mathtt{in}\, M)\ldots) \ ,$$

and $\mathbb{N} \triangleq \mathbf{N}[\mathbf{1}/\alpha]$ with $\mathbf{N} \triangleq (\downarrow\alpha) \multimap \alpha$. In order to identify terms that represent the same natural number, we take numerals up to the following equivalences:

$$\begin{aligned} &\mathtt{copy}^{I}_{1}\, f \,\mathtt{as}\, f_1, f_2 \,\mathtt{in}\,(\mathtt{copy}^{I}_{1}\, f_2 \,\mathtt{as}\, f_3, f_4 \,\mathtt{in}\, M) \\ &\qquad\qquad\qquad = \mathtt{copy}^{I}_{1}\, f \,\mathtt{as}\, f_2, f_4 \,\mathtt{in}\,(\mathtt{copy}^{I}_{1}\, f_2 \,\mathtt{as}\, f_1, f_3 \,\mathtt{in}\, M) \\ &f(\mathtt{copy}^{I}_{1}\, f' \,\mathtt{as}\, g, h \,\mathtt{in}\, M) = \mathtt{copy}^{I}_{1}\, f' \,\mathtt{as}\, g, h \,\mathtt{in}\, f\, M \ . \end{aligned}$$ □

The elements of $\mathcal{N}$ are the analogue of the Church numerals. Let us compare $\mathbb{N}$ to the type **int** of the Church numerals in Linear Logic:

$$\begin{aligned} \mathbf{int} &\triangleq \forall\alpha.(!(\alpha \multimap \alpha) \multimap (\alpha \multimap \alpha)) \\ \mathbb{N} &\triangleq (\downarrow\forall\alpha.(\alpha \multimap \alpha)) \multimap \forall\alpha.(\alpha \multimap \alpha) \ . \end{aligned}$$

In the former the universal quantification is in positive position, while in the latter it occurs on both sides of the main implication. This is because we can apply the modality $\downarrow$ only to ground types, which are closed. Also, observe that the lack of an external quantifier in $\mathbb{N}$ limits the use of numerals as iterators.

The analogy with the Church numerals can be pushed further by defining a successor $\mathtt{S}$ and an addition $\mathtt{A}$:

$$\begin{aligned}\mathtt{S} &\triangleq \lambda nfx.\mathtt{copy}^{I}_{1}\, f\, \mathtt{as}\, f_1, f_2\, \mathtt{in}\, f_1(nf_2x) : \mathbb{N} \multimap \mathbb{N}\\ \mathtt{A} &\triangleq \lambda mnfx.\mathtt{copy}^{I}_{1}\, f\, \mathtt{as}\, f_1, f_2\, \mathtt{in}\, mf_1(nf_2x) : \mathbb{N} \multimap \mathbb{N} \multimap \mathbb{N}\ .\end{aligned}$$

Sticking to the computational behaviour of the terms (Figures 2 and 3), not of the underlying derivations, the Subject reduction (Theorem 17) implies:

Proposition 24. *For all $n, m \geq 0$, $\mathtt{S}\,\overline{n} \to^* \overline{n+1}$ and $\mathtt{A}\,\overline{m}\,\overline{n} \to^* \overline{m+n}$.*

For example, the following reduction is legal:

$$\begin{aligned}\mathtt{S}\,\overline{2} &\triangleq (\lambda nfx.\mathtt{copy}^{I}_{1}\, f\, \mathtt{as}\, f_1, f_2\, \mathtt{in}\, f_1(nf_2x))(\lambda gy.\mathtt{copy}^{I}_{\mathbf{1}}\, g\, \mathtt{as}\, g_1, g_2\, \mathtt{in}\, g_1(g_2\, y))\\ &\to \lambda fx.\mathtt{copy}^{I}_{1}\, f\, \mathtt{as}\, f_1, f_2\, \mathtt{in}\, f_1((\lambda gy.\mathtt{copy}^{I}_{\mathbf{1}}\, g\, \mathtt{as}\, g_1, g_2\, \mathtt{in}\, g_1(g_2\, y))f_2x)\\ &\to \lambda fx.\mathtt{copy}^{I}_{1}\, f\, \mathtt{as}\, f_1, f_2\, \mathtt{in}\, f_1((\lambda y.\mathtt{copy}^{I}_{\mathbf{1}}\, f_2\, \mathtt{as}\, g_1, g_2\, \mathtt{in}\, g_1(g_2\, y))x)\\ &\to \lambda fx.\mathtt{copy}^{I}_{1}\, f\, \mathtt{as}\, f_1, f_2\, \mathtt{in}\, f_1(\mathtt{copy}^{I}_{\mathbf{1}}\, f_2\, \mathtt{as}\, g_1, g_2\, \mathtt{in}\, g_1(g_2\, x))\\ &= \lambda fx.\mathtt{copy}^{I}_{1}\, f\, \mathtt{as}\, f_1, f_2\, \mathtt{in}\, (\mathtt{copy}^{I}_{\mathbf{1}}\, f_2\, \mathtt{as}\, g_1, g_2\, \mathtt{in}\, f_1(g_1(g_2\, x))) \triangleq \overline{3}\end{aligned}$$

Observe that Proposition 24 considers typable terms and term reductions by exploiting Theorem 17. A similar result cannot be restated for the related derivations and the lazy-cut elimination (Definition 17). For example, the here above term $\mathtt{S}\,\overline{2}$ has type $\mathbb{N}$, that is not lazy (Definition 14), due to the presence of a universal quantification in negative position. Indeed, the lazy cut-elimination strategy of a derivation of $\mathtt{S}\,\overline{2}$ runs into deadlocks before producing a cut-free derivation of $\overline{3}$.

As far as we could see, the "zero-test", the predecessor and the subtraction on numerals cannot have type in LEM. The problem is the position of the universal quantifiers of $\mathbb{N}$. Consider, for example, the following predecessor:

$$\begin{aligned}\mathtt{P} &\triangleq \lambda nsz.n\, S[s]\, B[z] && \text{(Predecessor)}\\ S[M] &\triangleq \lambda p.\mathtt{let}\ p\ \mathtt{be}\ l, r\ \mathtt{in}\ \langle M, lr\rangle && \text{(Step function)}\\ B[N] &\triangleq \langle I, N\rangle && \text{(Base function)}\end{aligned}$$

introduced by Roversi [23]. Giving a type to $\mathtt{P}$ would require to substitute $(\alpha \multimap \alpha) \otimes \alpha$ for α in $\mathbb{N}$, as suggested by the application of $n : \mathbb{N}$ to $S[s]$. The position of the universal quantifiers in $\mathbb{N}$ forbids it. Were such an instance legal, we could iterate functions, contradicting the cubic bound on the cut-elimination (Theorem 14.)

Further, we can generalize $\mathbb{N}$ to $\mathbf{N}[\overline{(A \multimap A)}/\gamma]$, where $\overline{(A \multimap A)}$ is the closure of a quantifier-free type $A \multimap A$, and find that *Hereditarily finite permutations*

(HFP) by Dezani [7] inhabit $\overline{A \multimap A}$. An HFP is a λ-term of the form $P \triangleq \lambda z x_1 \ldots x_n . z(P_1 x_{\rho(1)}) \ldots (P_n x_{\rho(n)})$, for some $n \geq 0$, where $\rho \in S_n$ (the symmetric group of $\{1, \ldots, n\}$) and $P_1, \ldots, P_n$ are HFP. The class $\mathcal{H}_{\text{lin}}$ of linear λ-terms which are HFP (considered modulo $\beta\eta$-equivalence) forms a group:

1. The binary operation is $\lambda f g x . f(g x)$;
2. The identity is I;
3. If $P = \lambda z x_1 \ldots x_n . z(P_1 x_{\rho(1)}) \ldots (P_n x_{\rho(n)})$ is in $\mathcal{H}_{\text{lin}}$, the inductively defined inverse is:

$$P^{-1} \triangleq \lambda z' x_1 \ldots x_n . z'(P^{-1}_{\rho^{-1}(1)} x_{\rho^{-1}(1)}) \ldots (P^{-1}_{\rho^{-1}(n)} x_{\rho^{-1}(n)})$$

   where, for all $1 \leq i \leq n$, ρ_i^{-1} is the inverse permutation of ρ_i and P_i^{-1} is the inverse of P_i.

For example, let $P = \lambda wabc.w(\lambda xy.ayx)(\lambda xy.bxy)c$ which belongs to HFP since $\lambda xy.ayx =_\beta (\lambda zxy.zyx)a$, $\lambda xy.bxy =_\beta (\lambda zxy.zxy)b$, $c =_\beta Ic$, where $(\lambda zxy.zyx)$, $(\lambda zxy.zxy)$ and I are in HFP. Then, P has type $\forall \alpha_x \alpha_y \alpha_a \alpha_b \alpha_c \alpha . A \multimap A$, which is a ground type where A is $(\alpha_x \multimap \alpha_y \multimap \alpha_a) \multimap (\alpha_x \multimap \alpha_y \multimap \alpha_b) \multimap \alpha_c \multimap \alpha$. These observations show an unexpected link between LEM and reversible computation (see Perumalla [22] for a thorough introduction) that can well be expressed in terms of monoidal structures where permutations play a central role [19, 20, 21].

8. Conclusions

We introduce LEM. It is a type-assignment for the linear λ-calculus extended with new constructs that can duplicate or erase values, i.e. closed and normal linear λ-terms. LEM enjoys a mildly weakened cut-elimination, and the Subject reduction. The internalization of the mechanism of linear weakening and contraction by means of modal rules allows to exponentially compress derivations of IMLL_2. On one side, this enables to represent boolean circuits more compactly, as compared to previous ones, based on the multiplicative fragment of Linear Logic. On the other, LEM can represent Church-like encoding of the natural numbers, with successor and the addition for them.

We conclude by briefly discussing possible future works.

In Section 3, we conjecture that a version of the general separation property holds in the linear λ-calculus (Conjecture 5.) Were it true, we could show that a duplicator exists for all the finite sets of closed terms in $\beta\eta$-normal form, and connect linear duplication with the standard notion of separation.

In Section 4, we design LEM to express linear weakening and contraction in the same spirit as of the exponential rules of LL. We are working to push the analogy further, by formulating a type-assignment that extends IMLL_2 with "linear additives". A candidate rule is:

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

where V is a value and A, A_1, A_2 are closed and without negative occurrences of $\forall$. The intuition behind this rule is the one we discuss in Section 4 for the contraction rule of LEM. We also claim that the new system would keep the normalization cost linear, unlike standard additives (see [16]).

Section 7.1 presents an encoding of the boolean circuits not preserving their depth. Moving to unbounded fan-in proof nets for LEM would improve the correspondence, where the rules p, w, c and d would be expressed by nodes and boxes. Operations on them would compactly perform duplication and get rid of garbage, possibly improving [26, 18, 2]. A reasonable question would then be whether the use of alternative and weaker exponential rules in LEM could be the right approach to capture circuit complexity classes like NC, AC, and $\mathsf{P}/_{\text{poly}}$ in analogy with the implicit characterizations of the Polynomial and Elementary-time computational complexity classes by means of Light Logics [13, 6].

Section 7.2 contributes to the problem of defining numeral systems in linear settings. In [14], Mackie has recently introduced linear variants of numeral systems. He shows that successor, addition, predecessor, and subtraction have representatives in the linear λ-calculus. We could not find how giving type in LEM to some of the terms of Mackie's numeral systems. However, by merging Mackie's encoding and Scott numerals [5], numeral systems seem to exist which LEM can give a type to. The cost would be to extend LEM with recursive types, following Roversi&Vercelli [24].

References

References

[1] Sandra Alves, Maribel Fernández, Mário Florido, and Ian Mackie. The Power of Linear Functions. In Zoltán Ésik, editor, *Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL, Szeged, Hungary, September 25-29, 2006, Proceedings*, volume 4207 of *Lecture Notes in Computer Science*, pages 119–134. Springer, 2006.

[2] Clément Aubert. Sublogarithmic uniform boolean proof nets. In Jean-Yves Marion, editor, *Proceedings Second Workshop on Developments in Implicit Computational Complexity, Saarbrücken, DICE 2011, Germany, April 2nd and 3rd, 2011.*, volume 75 of *EPTCS*, pages 15–27, 2011.

[3] Hendrik Pieter Barendregt. The lambda calculus: Its syntax and semantics. 1984. *Studies in Logic and the Foundations of Mathematics*, 1984.

[4] Mario Coppo, Mariangiola Dezani-Ciancaglini, and S Ronchi Della Rocca. (semi)-separability of finite sets of terms in scott's d∞-models of the λ-calculus. In *International Colloquium on Automata, Languages, and Programming*, pages 142–164. Springer, 1978.

[5] H. B. Curry and R. Feys. *Combinatory Logic, Volume I.* North-Holland, 1958. Second printing 1968.

[6] Vincent Danos and Jean-Baptiste Joinet. Linear logic and elementary time. *Information and Computation*, 183(1):123–137, 2003.

[7] Mariangiola Dezani-Ciancaglini. Characterization of normal forms possessing inverse in the *lambda-beta-eta*-calculus. *Theor. Comput. Sci.*, 2(3):323–337, 1976.

[8] Marco Gaboardi and Simona Ronchi Della Rocca. From light logics to type assignments: a case study. *Logic Journal of the IGPL*, 17(5):499–530, 2009.

[9] Jean-Yves Girard. Linear Logic. *Theor. Comput. Sci.*, 50:1–102, 1987.

[10] Jean-Yves Girard. Linear logic. *Theoretical computer science*, 50(1):1–101, 1987.

[11] J Roger Hindley. Bck-combinators and linear λ-terms have types. *Theoretical Computer Science*, 64(1):97–105, 1989.

[12] Jan Willem Klop. Combinatory reduction systems. 1980.

[13] Yves Lafont. Soft linear logic and polynomial time. *Theoretical Computer Science*, 318(1):163–180, 2004.

[14] Ian Mackie. Linear Numeral Systems. *Journal of Automated Reasoning*, Feb 2018.

[15] Harry G. Mairson. Linear Lambda Calculus and PTIME-completeness. *J. Funct. Program.*, 14(6):623–633, November 2004.

[16] Harry G. Mairson and Kazushige Terui. On the Computational Complexity of Cut-Elimination in Linear Logic. In Carlo Blundo and Cosimo Laneve, editors, *Theoretical Computer Science*, pages 23–36, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[17] Satoshi Matsuoka. Weak typed Bohm Theorem on IMLL. *Annals of Pure and Applied Logic*, 145(1):37–90, 2007.

[18] Virgile Mogbil and Vincent Rahli. Uniform circuits, & boolean proof nets. In Sergei N. Artëmov and Anil Nerode, editors, *Logical Foundations of Computer Science, International Symposium, LFCS 2007, New York, NY, USA, June 4-7, 2007, Proceedings*, volume 4514 of *Lecture Notes in Computer Science*, pages 401–421. Springer, 2007.

[19] Luca Paolini, Mauro Piccolo, and Luca Roversi. A certified study of a reversible programming language. In *TYPES*, volume 69 of *LIPIcs*, pages 7:1–7:21. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[20] Luca Paolini, Mauro Piccolo, and Luca Roversi. A class of reversible primitive recursive functions. *Electronic Notes in Theoretical Computer Science*, 322:227–242, 2016. Elsevier, Netherlands.

[21] Luca Paolini, Mauro Piccolo, and Luca Roversi. On a class of reversible primitive recursive functions and its turing-complete extensions. *New Generation Computing*, 36(3):233–256, Jul 2018.

[22] Kalyan S. Perumalla. *Introduction to Reversible Computing*. Chapman & Hall/CRC Computational Science. Taylor & Francis, 2013.

[23] Luca Roversi. A P-Time Completeness Proof for Light Logics. In *Ninth Annual Conference of the EACSL (CSL'99)*, volume 1683 of *Lecture Notes in Computer Science*, pages 469 – 483, Madrid (Spain), September 1999. Springer-Verlag.

[24] Luca Roversi and Luca Vercelli. Safe Recursion on Notation into a Light Logic by Levels. In *Proceedings of the Workshop on Developments in Implicit Computational complexity (DICE 2010)*, volume 23 of *Electronic Proceedings in Theoretical Computer Science*, pages 63 – 77. On-line, March 2010.

[25] Aleksy Schubert. The complexity of β-reduction in low orders. In *International Conference on Typed Lambda Calculi and Applications*, pages 400–414. Springer, 2001.

[26] Kazushige Terui. Proof nets and boolean circuits. In *19th IEEE Symposium on Logic in Computer Science (LICS 2004), 14-17 July 2004, Turku, Finland, Proceedings*, pages 182–191. IEEE Computer Society, 2004.

[27] Anne Sjerp Troelstra and Helmut Schwichtenberg. *Basic proof theory*. Number 43. Cambridge University Press, 2000.

[28] Heribert Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Springer-Verlag, Berlin, Heidelberg, 1999.

Appendix A. The proof of Theorem 8

In this section we give a detailed proof of Theorem 8 for IMLL_2, which states that if A is an inhabited ground type, i.e an inhabited closed Π_1-type, then A is also a duplicable type. By Definition 7, this amounts to show that a linear λ-term $\mathtt{D}_A : A \multimap A \otimes A$ exists such that $\mathtt{D}_A\, V \rightarrow^*_{\beta\eta} \langle V, V \rangle$ holds for every value V of A. We shall construct $\mathtt{D}_A$ as the composition of three linear λ-terms, as diagrammatically displayed in Figure A.10. For each such component we dedicate a specific subsection. For the sake of presentation, in this section we focus on terms of IMLL_2 rather than on derivations, so that when we say that a term M has type A with context Γ, we clearly mean that a derivation $\mathcal{D}$ exists such that $\mathcal{D} \lhd \Gamma \vdash M : A$. Moreover, as assumed in Section 2, terms are considered modulo α-equivalence.

$$\begin{tikzcd}
A \arrow[r, "\mathtt{sub}^s_A"] & A^{-}[\mathbf{B}^s] \arrow[r, "\mathtt{enc}^s_A"] & \mathbf{B}^s \arrow[r, "\mathtt{dec}^s_A"] & A \otimes A \\
V \arrow[r, mapsto] & V_A \arrow[r, mapsto] & \lceil V_A \rceil \arrow[r, mapsto] & \langle V_A, V_A \rangle
\end{tikzcd}$$

Figure A.10: The diagrammatic representation of $\mathtt{D}_A$.

Appendix A.1. The linear λ-term $\mathtt{sub}^s_A$

Roughly, the λ-term $\mathtt{sub}^s_A$, when applied to a value V of ground type A, produces its η-long normal form V_A whose type is obtained from A as follows: we strip away every occurrence of $\forall$ and we substitute each type variable with the s-ary tensor of boolean datatypes $\mathbf{B}^s = \mathbf{B} \otimes \overset{s}{\ldots} \otimes \mathbf{B}$, for some $s > 0$.
Before introducing the λ-term $\mathtt{sub}^s_A$, we need the definition of η-long normal form:

Definition 23 (η-long normal forms)**.** Let $\mathcal{D} \lhd \Gamma \vdash M : B$ be cut-free. We define the *η-expansion* of $\mathcal{D}$, denoted $\mathcal{D}^{\Gamma}_{B}$, as the derivation obtained from $\mathcal{D}$ by substituting every occurrence of:

$$\frac{}{x : A \vdash x : A}\ ax$$

with a derivation of $x : A \vdash M' : A$, for some M', whose axioms have form $y : \alpha \vdash y : \alpha$. The η-expansion is unique and transforms the λ-term M to its *η-long normal form*, denoted by M^{Γ}_{B} and such that $M^{\Gamma}_{B} \rightarrow^{*}_{\eta} M$. If the context Γ of an η-expanded $\mathcal{D}$ is $x_1 : A_1, \ldots, x_n : A$ we may write $\mathcal{D}^{A_1,\ldots,A_n}_{B}$ and $M^{A_1,\ldots,A_n}_{B}$. If Γ is empty, we feel free to write $\mathcal{D}_B$ and M_B.

Lemma 25. *Let $\mathcal{D} \lhd \Gamma \vdash M : A$ be a cut-free derivation in IMLL_2, and let M^{Γ}_{A} denote the η-long normal form obtained by η-expanding $\mathcal{D}$. Then:*

1 If $M = x$, $A = \alpha$, and $\Gamma = x : \alpha$ then $x^{\alpha}_{\alpha} = x$.

2 If $M = x$, $A = \forall\alpha.B$, and $\Gamma = x : \forall\alpha.B$ then $x^{\forall\alpha.B}_{\forall\alpha.B} = x^{B}_{B}$.

3 If $M = x$, $A = B \multimap C$, and $\Gamma = x : B \multimap C$ then $x^{B\multimap C}_{B\multimap C} = \lambda y.(x y^{B}_{B})^{C}_{C}$.

4 If $A = \forall\alpha.B$ then $M^{\Gamma}_{\forall\alpha.B} = M^{\Gamma}_{B\langle\gamma/\alpha\rangle}$, for some γ.

5 If $M = \lambda x.N$ and $A = B \multimap C$ then $(\lambda x.N)^{\Gamma}_{B\multimap C} = \lambda x.N^{\Gamma,x:B}_{C}$.

6 If $M = P[yN/x]$ and $\Gamma = \Delta, \Sigma, y : B \multimap C$, where P has type A with context $\Delta, x : C$ and N has type B with context Σ, then $(P[yN/x])^{\Gamma}_{A} = P^{\Delta,x:C}_{A}[yN^{\Sigma}_{B}/x]$.

7 If $M = P[yN/x]$ and $\Gamma = \Gamma', y : \forall\alpha.B$ then we have $(P[yN/x])^{\Gamma',y:\forall\alpha.B}_{A} = (P[yN/x])^{\Gamma',y:B\langle D/\alpha\rangle}_{A}$, for some type D.

Proof. Just follow the definition of η-long normal form. $\square$

Definition 24. Let A be a type in IMLL_2. We define A^- by induction on the complexity of the type:

$$\begin{aligned} \alpha^- &\triangleq \alpha \\ (A \multimap B)^- &\triangleq A^- \multimap B^- \\ (\forall\alpha.A)^- &\triangleq A^-\langle\gamma/\alpha\rangle \ , \end{aligned}$$

where γ is taken from the head of an infinite list of fresh type variables. The notation $A[B]$ denotes the type obtained by replacing B for every free type variable of A. Moreover, if $\Gamma = x_1 : A_1, \dots, x_n : A_n$, then Γ^- stands for $x_1 : A_1^-, \dots, x_n : A_n^-$, and $\Gamma[B]$ stands for $x_1 : A_1[B], \dots, x_n : A_n[B]$. □

Definition 25 (The linear λ-term $\mathtt{sub}^s_A$)**.** Let $s > 0$. We define the linear λ-terms $\mathtt{sub}^s_A : A[\mathbf{B}^s] \multimap A^-[\mathbf{B}^s]$, where A is a Π_1-type, and $\overline{\mathtt{sub}}^s_A : A^-[\mathbf{B}^s] \multimap A[\mathbf{B}^s]$, where A is a Σ_1-type, by simultaneous induction on the size of A:

$$\begin{aligned} \mathtt{sub}^s_\alpha &\triangleq \lambda x.x & \overline{\mathtt{sub}}^s_\alpha &\triangleq \lambda x.x \\ \mathtt{sub}^s_{\forall\alpha.B} &\triangleq \mathtt{sub}^s_B & & \\ \mathtt{sub}^s_{B\multimap C} &\triangleq \lambda x.\lambda y.\mathtt{sub}^s_C(x\,(\overline{\mathtt{sub}}^s_B\,y)) & \overline{\mathtt{sub}}^s_{B\multimap C} &\triangleq \lambda x.\lambda y.\overline{\mathtt{sub}}^s_C(x\,(\mathtt{sub}^s_B\,y)). \end{aligned}$$ □

The following will be used to compact the proof of some of the coming lemmas.

Definition 26. Let $s > 0$. Let A be a Π_1-type. Let $\Gamma = x_1 : A_1, \dots, x_n : A_n$ be a context of Σ_1-types. Let M be an inhabitant of $A[\mathbf{B}^s]$ with context $\Gamma[\mathbf{B}^s]$. Then $M[\Gamma]$ denotes the substitution:

$$M[\overline{\mathtt{sub}}^s_{A_1}\,x'_1/x_1, \dots, \overline{\mathtt{sub}}^s_{A_n}\,x'_n/x_n]$$

for some $x'_1, \dots, x'_n$. □

Lemma 26. *Let $s > 0$ and z be of type $A[\mathbf{B}^s]$.*

(1) If A is a Π_1-type, then $\mathtt{sub}^s_A\,z \rightarrow^*_\beta z^A_A$.

(2) If A is a Σ_1-type, then $\overline{\mathtt{sub}}^s_A\,z \rightarrow^*_\beta z^A_A$.

Proof. We prove both points by simultaneous induction on $|A|$:

1. Case $A = \alpha$. Both the statements are straightforward since we have $z^\alpha_\alpha = z$ by Lemma 25.1.

2. Case $A = \forall\alpha.B$. This case applies to point (1) only. By induction hypothesis, for every variable x of type $B[\mathbf{B}^s]$, $\mathtt{sub}^s_B\,x \rightarrow^*_\beta x^B_B$. The λ-term $\mathtt{sub}^s_B$ has type $B[\mathbf{B}^s] \multimap B^-[\mathbf{B}^s]$, which is equal to $(B\langle\mathbf{B}^s/\alpha\rangle)[\mathbf{B}^s] \multimap (\forall\alpha.B)^-[\mathbf{B}^s]$. Hence, $\mathtt{sub}^s_B$ has also type $(\forall\alpha.B)[\mathbf{B}^s] \multimap (\forall\alpha.B)^-[\mathbf{B}^s]$. Moreover, by Definition 25 we have $\mathtt{sub}^s_B = \mathtt{sub}^s_{\forall\alpha.B}$. Therefore, for every variable z of type $(\forall\alpha.B)[\mathbf{B}^s]$ we have $\mathtt{sub}^s_{\forall\alpha.B}\,z = \mathtt{sub}^s_B\,z \rightarrow^*_\beta z^B_B$. But $z^B_B = z^{\forall\alpha.B}_{\forall\alpha.B}$ by Lemma 25.2.

3. Case $A = B \multimap C$. We prove point (1) only (point (2) is similar). Let z be of type $(B \multimap C)[\mathbf{B}^s] = B[\mathbf{B}^s] \multimap C[\mathbf{B}^s]$. Then we have

$$\begin{aligned}
\mathtt{sub}^s_{B\multimap C}\, z &= (\lambda x.\lambda y.\mathtt{sub}^s_C(x(\overline{\mathtt{sub}}^s_B\, y)))z && \text{Definition 25}\\
&\to_\beta \lambda y.\mathtt{sub}^s_C(z(\overline{\mathtt{sub}}^s_B\, y))\\
&= \lambda y.(\mathtt{sub}^s_C\, w)[z(\overline{\mathtt{sub}}^s_B\, y)/w]\\
&\to^*_\beta \lambda y.w^C_C[z(\overline{\mathtt{sub}}^s_B\, y)/w] && \text{induction hyp., point (1)}\\
&\to^*_\beta \lambda y.w^C_C[zy^B_B/w] && \text{induction hyp., point (2)}\\
&= \lambda y.(zy^B_B)^C_C\\
&= z^{B\multimap C}_{B\multimap C} && \text{Lemma 25.3.}
\end{aligned}$$

□

Lemma 27. *Let $s > 0$. If $z : A[\mathbf{B}^s]$, where A is a Π_1-type, then* $\mathtt{sub}^s_A\, z^A_A \to^*_\beta z^A_A$.

Proof. We prove it by induction on $|A|$:

1. Case $A = \alpha$. The statement is straightforward since we have $z^\alpha_\alpha = z$ by Lemma 25.1
2. Case $A = \forall\alpha.B$. By Definition 25, $\mathtt{sub}^s_{\forall\alpha.B} = \mathtt{sub}^s_B$ and we use the induction hypothesis.
3. Case $A = B \multimap C$. Then we have

$$\begin{aligned}
\mathtt{sub}^s_{B\multimap C}\, z^{B\multimap C}_{B\multimap C} &= (\lambda x.\lambda y.\mathtt{sub}^s_C(x(\overline{\mathtt{sub}}^s_B\, y)))z^{B\multimap C}_{B\multimap C} && \text{Definition 25}\\
&= (\lambda x.\lambda y.\mathtt{sub}^s_C(x(\overline{\mathtt{sub}}^s_B\, y)))(\lambda w.(zw^B_B)^C_C) && \text{Lemma 25.3}\\
&\to_\beta \lambda y.\mathtt{sub}^s_C((\lambda w.(zw^B_B)^C_C)(\overline{\mathtt{sub}}^s_B\, y))\\
&\to_\beta \lambda y.\mathtt{sub}^s_C(z(\overline{\mathtt{sub}}^s_B\, y)^B_B)^C_C\\
&\to^*_\beta \lambda y.\mathtt{sub}^s_C(zy^B_B)^C_C && \text{Lemma 26.(2)}\\
&= \lambda y.(\mathtt{sub}^s_C\, w^C_C)[zy^B_B/w]\\
&\to^*_\beta \lambda y.w^C_C[zy^B_B/w] && \text{induction hyp.}\\
&= \lambda y.(zy^B_B)^C_C\\
&=_\alpha z^{B\multimap C}_{B\multimap C} && \text{Lemma 25.3}
\end{aligned}$$

□

Lemma 28. *Let $s > 0$. Let A be a Π_1-type, and let $\Gamma = x_1 : A_1, \ldots, x_n : A_n$ be a context of Σ_1-types. If $\Gamma[\mathbf{B}^s] \vdash M : A[\mathbf{B}^s]$, with M normal, then:*

$$\mathtt{sub}^s_A\, M[\Gamma] \to^*_\beta M^\Gamma_A\ .$$

Proof. Let $Q_{\Gamma,A}$ be the number of universal quantifications in $A_1, \dots, A_n, A$. We prove the result by induction on $|M| + Q_{\Gamma,A}$. If $M = z$ then $\Gamma = z : A$ and $\mathtt{sub}^s_A\, M[\Gamma] = \mathtt{sub}^s_A(\overline{\mathtt{sub}}^s_A\, z)$. By point (2) of Lemma 26 and by Lemma 27 we have $\mathtt{sub}^s_A(\overline{\mathtt{sub}}^s_A\, z) \to^*_\beta \mathtt{sub}^s_A\, z^A_A \to^*_\beta z^A_A$. If $M = \lambda z.N$ then we have two cases depending on the type of M:

1. Case $A = \forall\alpha.B$. The λ-term $\mathtt{sub}^s_B$ has type $B[\mathbf{B}^s] \multimap B^-[\mathbf{B}^s]$, which is equal to $(B\langle\mathbf{B}^s/\alpha\rangle)[\mathbf{B}^s] \multimap (\forall\alpha.B)^-[\mathbf{B}^s]$, so that $\mathtt{sub}^s_B$ has also type $(\forall\alpha.B)[\mathbf{B}^s] \multimap (\forall\alpha.B)^-[\mathbf{B}^s]$. By Definition 25 we have $\mathtt{sub}^s_B = \mathtt{sub}^s_{\forall\alpha.B}$. By using the induction hypothesis, for every M of type $(\forall\alpha.B)[\mathbf{B}^s]$ with context $\Gamma[\mathbf{B}^s]$, we have $\mathtt{sub}^s_{\forall\alpha.B}\, M[\Gamma] = \mathtt{sub}^s_B\, M[\Gamma] \to^*_\beta M^\Gamma_B$. Moreover, by Lemma 25.4, $M^\Gamma_B = M^\Gamma_{\forall\alpha.A}$.

2. Case $A = B \multimap C$. Then we have:

$$\begin{aligned}
\mathtt{sub}^s_{B\multimap C}\, M[\Gamma] &= (\lambda x.\lambda y.\mathtt{sub}^s_C(x\,(\overline{\mathtt{sub}}^s_B\, y)))(\lambda z.N)[\Gamma] && \text{Definition 25}\\
&\to_\beta \lambda y.\mathtt{sub}^s_C((\lambda z.N)[\Gamma]\,(\overline{\mathtt{sub}}^s_B\, y))\\
&\to_\beta \lambda y.\mathtt{sub}^s_C((N[\Gamma])[\overline{\mathtt{sub}}^s_B\, y/z])\\
&= \lambda y.\mathtt{sub}^s_C(N[\Gamma, y : B]) && \text{Definition 26}\\
&\to^*_\beta \lambda y.N^{\Gamma,y:B}_C && \text{induction hyp.}\\
&= (\lambda y.N)^\Gamma_{B\multimap C} && \text{Lemma 25.5}\\
&=_\alpha M^\Gamma_{B\multimap C}.
\end{aligned}$$

If $M = P[zN/w]$ then the type of z cannot have an outermost universal quantification, because Γ is a context of Σ_1-types. So z has type of the form $B \multimap C$ in Γ. Let Γ' and Γ'' be contexts such that $\Gamma = \Gamma', \Gamma'', z : B \multimap C$, $\mathrm{dom}(\Gamma') = FV(P)$, and $\mathrm{dom}(\Gamma'') = FV(N)$. Then we have:

$$\begin{aligned}
\mathtt{sub}^s_A\, M[\Gamma] &= \mathtt{sub}^s_A(P[zN/w])[\Gamma]\\
&= \mathtt{sub}^s_A(P[\Gamma'][(zN)[\Gamma'', z : B \multimap C]/w])\\
&= \mathtt{sub}^s_A(P[\Gamma'][(\overline{\mathtt{sub}}^s_{B\multimap C}\, z)(N[\Gamma''])/w])\\
&\to^*_\beta \mathtt{sub}^s_A(P[\Gamma'][\overline{\mathtt{sub}}^s_C(z\,(\mathtt{sub}^s_B\,(N[\Gamma''])))/w]) && \text{Definition 25}\\
&\to^*_\beta \mathtt{sub}^s_A(P[\Gamma'][\overline{\mathtt{sub}}^s_C(zN^{\Gamma''}_B)/w]) && \text{induction hyp.}\\
&= (\mathtt{sub}^s_A\, P[\Gamma'][\overline{\mathtt{sub}}^s_C\, w/w])[zN^{\Gamma''}_B/w]\\
&= (\mathtt{sub}^s_A\, P[\Gamma', w : C])[zN^{\Gamma''}_B/w] && \text{Definition 26}\\
&\to^*_\beta P^{\Gamma',w:C}_A[zN^{\Gamma''}_B/w] && \text{induction hyp.}\\
&= (P[zN/w])^\Gamma_A && \text{Lemma 25.6}
\end{aligned}$$

□

Appendix A.2. The linear λ-term $\mathtt{enc}^s_A$

One missing ingredient in the previous subsection is the value of s, which is fixed to some strictly positive integer. To determine s we need the following property:

Lemma 29. *For every cut-free derivation $\mathcal{D} \lhd \Gamma \vdash M : B$ in IMLL_2 which does not contain applications of $\forall$L, the following inequations hold:*

$$|M| \leq |M^\Gamma_B| \leq |\Gamma^-| + |B^-| \leq 2 \cdot |M^\Gamma_B| \quad , \tag{A.1}$$

where $(\_)^-$ is as in Definition 24, and M^Γ_A is as in Definition 23.

Proof. The inequation $|M| \leq |M^\Gamma_B|$ is by definition of η-long normal form. Now, let $\mathcal{D}^\Gamma_B$ be the η-expansion of $\mathcal{D}$, so that $\mathcal{D}^\Gamma_B \lhd \Gamma \vdash M^\Gamma_B : B$. We prove the remaining two inequations by induction on $\mathcal{D}^\Gamma_B$. If it is an axiom then, by definition of η-expansion, it must be of the form $x : \alpha \vdash x : \alpha$, where $M^\Gamma_B = x$. Hence, $|x| \leq 2 \cdot |\alpha| \leq 2 \cdot |x|$. Both the rules $\multimap$R and $\multimap$L, increase by one the overall size of the types in a judgment and of the corresponding term, so the inequalities still hold. Last, the rules for $\forall$ do not affect the size of both Γ^-, B^- and M^Γ_B. $\square$

Notice that Lemma 29 does not hold in general whenever $\mathcal{D}$ contains instances of the inference rule $\forall$L, since one can exploit the inference rule $\forall$L to "compress" the size of a type.
Now, consider a cut-free derivation $\mathcal{D} \lhd \ \vdash M : A$, where A is a ground type. Since negative occurrences of $\forall$ are not allowed in A, $\mathcal{D}$ contains no application of $\forall$L and, by Lemma 29, this implies that $|M| \leq |A^-|$. This limits the number of variables a generic inhabitant of A has, so that we can safely say that the variables of M must certainly belong to a fixed set $\{\mathrm{x}_1, \ldots, \mathrm{x}_{|A^-|}\}$. The next step is to show that we can encode every normal form as a tuple of booleans, i.e. as elements in $\mathbf{B}^s$ with a sufficiently large s. Actually, we are interested in η-long normal forms only, due to the way the linear λ-term $\mathtt{sub}^s_A$ acts on inhabitants of A as shown in the previous subsection. So, given a ground type A, we can represent the η-long normal forms of type A with tuples of type $\mathbf{B}^{\mathcal{O}(|A^-| \cdot \log |A^-|)}$, since each such linear λ-term has at most $|A^-|$ symbols, each one encoded using around $\log |A^-|$ bits. By setting $s = c \cdot (|A^-| \cdot \log |A^-|)$ for some $c > 0$ large enough, there must exist a coding function $\lceil \_ \rceil : \Lambda_s \longrightarrow \mathbf{B}^s$, where Λ_s is the set of all normal linear λ-terms having size bounded by s. The role of the λ-term $\mathtt{enc}^s_A$ is to internalize the coding function $\lceil \_ \rceil$ in IMLL_2 as far as the η-long normal forms of a fixed type A are concerned.

The coming Lemma 31 relies on an iterated selection mechanism, i.e. a nested `if-then-else` construction. In order to define selection, we first we need to extend the projection in (4) (Section 3).

Definition 27 (Generalized projection)**.** Let A be a ground type. For all $k \geq 0$ and $\vec{m} = m_1, \ldots, m_k \geq 0$, the linear λ-term $\pi^{\vec{m}}_1$ is defined below:

$$\pi^{\vec{m}}_1 \triangleq \begin{cases} \lambda z.\mathtt{let}\ z\ \mathtt{be}\ x, y\ \mathtt{in}\ (\mathtt{let}\ \mathrm{E}_A\, y\ \mathtt{be}\ I\ \mathtt{in}\ x) & \text{if } k = 0 \\ \lambda z.\mathtt{let}\ z\ \mathtt{be}\ x, y\ \mathtt{in}\ (\mathtt{let}\ \mathrm{E}_A\, (y\, \mathtt{tt}^{m_1} \ldots \mathtt{tt}^{m_k})\ \mathtt{be}\ I\ \mathtt{in}\ x) & \text{if } k > 0 \end{cases}$$

with type $B \otimes B \multimap B$, where $B \triangleq \mathbf{B}^{m_1} \multimap \ldots \multimap \mathbf{B}^{m_k} \multimap A$. When $k = 0$ we simply write π_1 in place of $\pi_1^{\vec{m}}$, whose type is $A \otimes A \multimap A$.

Definition 28 (Generalized selection)**.** Let A be a ground type and let $M_{\mathtt{tt}^n}$, $M_{\langle \mathtt{tt}^{n-1}, \mathtt{ff}\rangle}$, $\ldots$, $M_{\langle \mathtt{tt}, \mathtt{ff}^{n-1}\rangle}$, $M_{\mathtt{ff}^n}$ be (not necessarily distinct) normal inhabitants of $\mathbf{B}^{m_1} \multimap \ldots \multimap \mathbf{B}^{m_k} \multimap A$, for some $n \geq 1$, $k \geq 0$, and $\vec{m} = m_1, \ldots, m_k \geq 0$. We define the linear λ-term:

$$\mathtt{if}\ x\ \mathtt{then}\ [M_{\mathtt{tt}^n}, M_{\langle \mathtt{tt}^{n-1}, \mathtt{ff}\rangle}, \ldots, M_{\langle \mathtt{tt}, \mathtt{ff}^{n-1}\rangle}, M_{\mathtt{ff}^n}]^{\vec{m}} \tag{A.2}$$

with type $\mathbf{B}^{m_1} \multimap \ldots \multimap \mathbf{B}^{m_k} \multimap A$ and context $x : \mathbf{B}^n$ by induction on n:

- $n = 1$: $\mathtt{if}\ x\ \mathtt{then}\ [M_{\mathtt{tt}}, M_{\mathtt{ff}}]^{\vec{m}} \triangleq \pi_1^{\vec{m}}(x\, M_{\mathtt{tt}}\, M_{\mathtt{ff}})$.
- $n > 1$: $\mathtt{if}\ x\ \mathtt{then}\ [M_{\mathtt{tt}^n}, M_{\langle \mathtt{tt}^{n-1}, \mathtt{ff}\rangle}, \ldots, M_{\langle \mathtt{tt}, \mathtt{ff}^{n-1}\rangle}, M_{\mathtt{ff}^n}]^{\vec{m}} \triangleq$

  $\mathtt{let}\ x\ \mathtt{be}\ x_1, x_2\ \mathtt{in}\ (\mathtt{if}\ x_2\ \mathtt{then}$
  $\big[(\lambda y_1.\mathtt{if}\ y_1\ \mathtt{then}\ [P_{\mathtt{tt}^{n-1}}, P_{\langle \mathtt{tt}^{n-2}, \mathtt{ff}\rangle}, \ldots, P_{\langle \mathtt{tt}, \mathtt{ff}^{n-2}\rangle}, P_{\mathtt{ff}^{n-1}}]^{\vec{m}}),$
  $(\lambda y_2.\mathtt{if}\ y_2\ \mathtt{then}\ [Q_{\mathtt{tt}^{n-1}}, Q_{\langle \mathtt{tt}^{n-2}, \mathtt{ff}\rangle}, \ldots, Q_{\langle \mathtt{tt}, \mathtt{ff}^{n-2}\rangle}, Q_{\mathtt{ff}^{n-1}}]^{\vec{m}})\big]^{n-1, \vec{m}})\, x_1$

  where, $\pi_1^{\vec{m}}$ is as in Definition 27 and, for every n-tuple $\langle \mathtt{b}_1, \ldots, \mathtt{b}_n\rangle$ of booleans, $P_{\langle \mathtt{b}_1, \ldots, \mathtt{b}_n\rangle} \triangleq M_{\langle\langle \mathtt{b}_1, \ldots, \mathtt{b}_n\rangle, \mathtt{tt}\rangle}$, $Q_{\langle \mathtt{b}_1, \ldots, \mathtt{b}_n\rangle} \triangleq M_{\langle\langle \mathtt{b}_1, \ldots, \mathtt{b}_n\rangle, \mathtt{ff}\rangle}$.

when $k = 0$ we feel free of ruling out the apex $\vec{m}$ in (A.2).

Lemma 30. *Let A be a ground type and let $M_{\mathtt{tt}^n}$, $M_{\langle \mathtt{tt}^{n-1}, \mathtt{ff}\rangle}$, $\ldots$, $M_{\langle \mathtt{tt}, \mathtt{ff}^{n-1}\rangle}$, $M_{\mathtt{ff}^n}$ be (not necessarily distinct) normal inhabitants of $\mathbf{B}^{m_1} \multimap \ldots \multimap \mathbf{B}^{m_k} \multimap A$, for some $n \geq 1$, $k \geq 0$, and $\vec{m} = m_1, \ldots, m_k \geq 0$. For every n-tuple of booleans $\langle \mathtt{b}_1, \ldots, \mathtt{b}_n\rangle$ it holds that:*

$$\mathtt{if}\ \langle \mathtt{b}_1, \ldots, \mathtt{b}_n\rangle\ \mathtt{then}\ (M_{\mathtt{tt}^n}, M_{\langle \mathtt{tt}^{n-1}, \mathtt{ff}\rangle}, \ldots, M_{\langle \mathtt{tt}, \mathtt{ff}^{n-1}\rangle}, M_{\mathtt{ff}^n}) \to_\beta^* M_{\langle \mathtt{b}_1, \ldots, \mathtt{b}_n\rangle}\ .$$

Proof. Straightforward. □

Notice that, if $n = 1$ and $k = 0$ in Definition (28), we get the usual `if-then-else` construction defined in [8] as:

$$\mathtt{if}\ x\ \mathtt{then}\ M_1\ \mathtt{else}\ M_2 \triangleq \pi_1(x\, M_1\, M_2) \tag{A.3}$$

with type A and context $x : \mathbf{B}$, where $\pi_1 : A \otimes A \multimap A$ is as in Definition 27. Clearly, if $\mathtt{b}_1 \triangleq \mathtt{tt}$ and $\mathtt{b}_2 \triangleq \mathtt{ff}$, then $\mathtt{if}\ \mathtt{b}_i\ \mathtt{then}\ M_1\ \mathtt{else}\ M_2 \to_\beta^* M_i$ for $i = 1, 2$.

Before defining the linear λ-term $\mathtt{enc}_A^s$ we need to encode the λ-abstractions and the applications in IMLL_2.

Lemma 31. *Let $s > 0$. The following statements hold:*

(1) A linear λ-term $\mathtt{abs}^s : \mathbf{B}^s \multimap \mathbf{B}^s \multimap \mathbf{B}^s$ exists such that $\mathtt{abs}\lceil x\rceil\lceil M\rceil \to_\beta^* \lceil \lambda x.M\rceil$, if $|\lambda x.M| \leq s$ and $x \in \{\mathrm{x}_1, \ldots, \mathrm{x}_s\}$.

(2) A linear λ-term $\texttt{app}^s : \mathbf{B}^s \multimap \mathbf{B}^s \multimap \mathbf{B}^s$ *exists such that* $\texttt{app}\lceil M\rceil\lceil N\rceil \rightarrow^*_\beta \lceil MN\rceil$, *if* $|MN| \leq s$.

Proof. We sketch the proof of Point (1) only, since Point (2) is similar. Recall the notation in Definition 28. We let boolean values range over $b_1, b_2, \dots$ and with $\texttt{b}$ we denote the corresponding encoding of the boolean value b in IMLL_2. The linear λ-term $\texttt{abs}$ is of the form:

$$\lambda x.\lambda y.(\texttt{if } x \texttt{ then } [P_{\texttt{tt}^s}, P_{\langle \texttt{tt}^{s-1},\texttt{ff}\rangle}, \dots, P_{\langle \texttt{ff},\texttt{tt}^{s-1}\rangle}, P_{\texttt{ff}^s}]^s)\, y$$

where, for all s-tuple of booleans $T = \langle \texttt{b}_1, \dots, \texttt{b}_s\rangle$, the linear λ-term P_T with type $\mathbf{B}^s \multimap \mathbf{B}^s$ is as follows:

$$\lambda y.\texttt{if } y \texttt{ then } [Q^T_{\texttt{tt}^s}, Q^T_{\langle \texttt{tt}^{s-1},\texttt{ff}\rangle}, \dots, Q^T_{\langle \texttt{ff},\texttt{tt}^{s-1}\rangle}, Q^T_{\texttt{ff}^s}] \ .$$

For all $T = \langle \texttt{b}_1, \dots, \texttt{b}_s\rangle$ and for all $T' = \langle \texttt{b}'_1, \dots, \texttt{b}'_s\rangle$ we define:

$$Q^T_{T'} = \begin{cases} \lceil \lambda x.M\rceil & \text{if } \langle \texttt{b}_1, \dots, \texttt{b}_s\rangle = \lceil x\rceil,\ \langle \texttt{b}'_1, \dots, \texttt{b}'_s\rangle = \lceil M\rceil, \\ & \text{and } |\lambda x.M| \leq s \\ & \\ \langle \texttt{tt}, \overset{s}{\dots}, \texttt{tt}\rangle & \text{otherwise.} \end{cases}$$

□

The λ-term $\texttt{enc}^s_A$, given a value V_A in η-long normal form and of type A, combines the λ-terms $\texttt{abs}^s$ and $\texttt{app}^s$ to construct its encoding.

Definition 29 (The linear λ-term $\texttt{enc}^s_A$)**.** Let $s > 0$. We define the linear λ-terms $\texttt{enc}^s_A : A^-[\mathbf{B}^s] \multimap \mathbf{B}^s$, where A is a Π_1-type, and $\overline{\texttt{enc}}^s_A : \mathbf{B}^s \multimap A^-[\mathbf{B}^s]$, where A is a Σ_1-type, by simultaneous induction on the size of A:

$$\begin{array}{ll} \texttt{enc}^s_\alpha \triangleq \lambda z.z & \texttt{enc}^s_{B\multimap C} \triangleq \lambda z.\texttt{abs}^s\lceil x\rceil\,(\texttt{enc}^s_C\,(z\,(\overline{\texttt{enc}}^s_B\,\lceil x\rceil))) \\ \overline{\texttt{enc}}^s_\alpha \triangleq \lambda z.z & \overline{\texttt{enc}}^s_{B\multimap C} \triangleq \lambda z.\lambda x.\overline{\texttt{enc}}^s_C\,(\texttt{app}^s z\,(\texttt{enc}^s_B\, x)) \end{array}$$

with x chosen fresh in $\{\mathrm{x}_1, \dots, \mathrm{x}_s\}$. □

The following will be used to compact the proof of some of the coming lemmas.

Definition 30. Let $s > 0$, and let A be a Π_1-type and $\Gamma = x_1 : A_1, \dots, x_n : A_n$ be a context of Σ_1-types. If M is an inhabitant of type $A^-[\mathbf{B}^s]$ with context $\Gamma^-[\mathbf{B}^s]$ then $M[\Gamma]$ denotes the substitution:

$$M[\overline{\texttt{enc}}^s_{A_1}\, x'_1/x_1, \dots, \overline{\texttt{enc}}^s_{A_n}\, x'_n/x_n]$$

for some $x'_1, \dots, x'_n$. □

To prove that $\texttt{enc}^s_A$ is able to encode a value V_A of type A we need an intermediate step. We first prove that $\texttt{enc}^s_A$ substitutes every λ-abstraction in V_A with an instance of $\texttt{abs}^s$, and every application with an instance of $\texttt{app}^s$, thus producing a "precode". Then we prove that, when every free variable in it has been substituted with its respective encoding, the precode reduces to $\lceil V_A\rceil$.

Definition 31. Let $s > 0$. If M is a linear λ-term in normal form such that $|M| \leq s$, we define M^s by induction on $|M|$:

1. $M = x$ if and only if $M^s = x$,
2. $M = \lambda x.N$ if and only if $M^s = \texttt{abs}^s \lceil x' \rceil N^s[\lceil x' \rceil / x]$,
3. $M = PQ$ if and only if $M^s = \texttt{app}^s P^s Q^s$,

where x' is fresh, chosen in $\{\mathrm{x}_1, \dots, \mathrm{x}_s\}$. ◻

Lemma 32. *Let $s > 0$. If M and N are linear λ-terms, then $M^s[N^s/x] = (M[N/x])^s$.*

Proof. By induction on $|M|$. If $M = x$ then $x^s[N^s/x] = x[N^s/x] = N^s = (x[N/x])^s$. If $M = PQ$ then either x occurs in P or it occurs in Q, and let us consider the case $x \in FV(P)$, the other case being similar: by using the induction hypothesis we have $(PQ)^s[N^s/x] = \texttt{app}^s P^s[N^s/x] Q^s = \texttt{app}^s (P[N/x])^s Q^s = (P[N/x]Q)^s = ((PQ)[N/x])^s$. If $M = \lambda y.P$ then we have that $(\lambda y.P)^s[N^s/x] = \texttt{abs}^s \lceil y' \rceil P^s[N^s/x][\lceil y' \rceil / y] = \texttt{abs}^s \lceil y' \rceil (P[N/x])^s[\lceil y' \rceil / y] = (\lambda y.P[N/x])^s = ((\lambda y.P)[N/x])^s$. ◻

Lemma 33. *Let $s > 0$. If M is a linear λ-term in normal form such that $|M| \leq s$ with free variables $x_1, \dots, x_n$ then*

$$M^s[\lceil \vec{x'} \rceil] \to_\beta^* \lceil M[x'_1/x_1, \dots, x'_n/x_n] \rceil$$

where $\lceil \vec{x'} \rceil = [\lceil x'_1 \rceil / x_1, \dots, \lceil x'_n \rceil / x_n]$ and $x'_1, \dots, x'_n$ are distinct and fresh in $\{\mathrm{x}_1, \dots, \mathrm{x}_s\}$.

Proof. By induction on $|M|$. If $M = x$ then $\exists i \leq n \; x_i = x$, so that $x^s[\lceil x'_i \rceil / x] = x[\lceil x'_i \rceil / x] = \lceil x'_i \rceil = \lceil x[x'_i/x] \rceil$. If $M = \lambda y.N$ then, using the induction hypothesis, we have:

$$\begin{aligned}
(\lambda y.N)^s[\lceil \vec{x'} \rceil] &= (\texttt{abs}^s \lceil y' \rceil N^s[\lceil y' \rceil / y])[\lceil \vec{x'} \rceil] \\
&= \texttt{abs}^s \lceil y' \rceil N^s[\lceil \vec{x'} \rceil, \lceil y' \rceil / y] \\
&\to_\beta^* \texttt{abs}^s \lceil y' \rceil \lceil N[x'_1/x_1, \dots, x'_n/x_n, y'/y] \rceil \\
&\to_\beta^* \lceil \lambda y'.N[x'_1/x_1, \dots, x'_n/x_n, y'/y] \rceil && \text{Lemma 31} \\
&=_\alpha \lceil (\lambda y.N)[x'_1/x_1, \dots, x'_n/x_n] \rceil.
\end{aligned}$$

If $M = PQ$ then let $y_1, \dots, y_m$ (resp. $z_1, \dots, z_k$) be the free variables of P (resp. Q), and let $\vec{x'} = y'_1, \dots, y'_m, z'_1, \dots, z'_k$. Then we have:

$$\begin{aligned}
&(PQ)^s[\lceil \vec{x'} \rceil] = \\
&= \texttt{app}^s P^s[\lceil \vec{y'} \rceil] Q^s[\lceil \vec{z'} \rceil] \\
&\to_\beta^* \texttt{app}^s \lceil P[y'_1/y_1, \dots, y'_m/y_m] \rceil \lceil Q[z'_1/z_1, \dots, z'_k/z_k] \rceil \\
&\to_\beta^* \lceil P[y'_1/y_1, \dots, y'_m/y_m] Q[z'_1/z_1, \dots, z'_k/z_k] \rceil && \text{Lemma 31} \\
&= \lceil (PQ)[x'_1/x_1, \dots, x'_n/x_n] \rceil.
\end{aligned}$$

◻

It is easy to check that if M is an inhabitant of a Π_1-type A with context $\Gamma = x_1 : A_1, \ldots, x_n : A_n$ of Σ_1-types, then M has also type $A^-[\mathbf{B}^s]$ with context $\Gamma^-[\mathbf{B}^s]$.

Lemma 34. *Let M be a η-long normal form of type A with context $\Gamma = x_1 : A_1, \ldots, x_n : A_n$, where A is a Π_1-type and Γ is a context of Σ_1-types, and let $\sum_{i=1}^{m} |A_i^-| + |A^-| = k$ and $s = c \cdot (k \cdot \log k)$, for some c large enough. Then:*

$$(\texttt{enc}_A^s\, M[\Gamma])[\lceil \vec{x'} \rceil] \to_\beta^* \lceil M[x'_1/x_1, \ldots, x'_n/x_n] \rceil \ ,$$

where $\lceil \vec{x'} \rceil = [\lceil x'_1 \rceil / x_1, \ldots, \lceil x'_n \rceil / x_n]$, with $x'_1, \ldots, x'_n$ distinct and chosen fresh in $\{\mathrm{x}_1, \ldots, \mathrm{x}_s\}$.

Proof. By Lemma 33 it suffices to prove by induction on $|M|$ that the reduction $\texttt{enc}_A^s\, M[\Gamma] \to_\beta^* M^s$ holds. If $M = x$ then $A = \alpha$ and $\Gamma = x : \alpha$, because M is in η-long normal form, so that we have $\texttt{enc}_\alpha^s\, x[x : \alpha] = \texttt{enc}_\alpha^s (\overline{\texttt{enc}}_\alpha^s\, x) \to_\beta^* x = x^s$. If $M = \lambda y.N$ then $A = B \multimap C$, so that:

$$\begin{array}{ll}
\texttt{enc}_{B\multimap C}^s ((\lambda y.N)[\Gamma]) & \\
\to_\beta \texttt{abs}^s \lceil x' \rceil (\texttt{enc}_C^s ((\lambda y.N[\Gamma])(\overline{\texttt{enc}}_B^s \lceil y' \rceil))) & \text{Definition 29} \\
\to_\beta \texttt{abs}^s \lceil y' \rceil (\texttt{enc}_C^s (N[\Gamma][\overline{\texttt{enc}}_B^s \lceil y' \rceil / y])) & \\
= \texttt{abs}^s \lceil y' \rceil (\texttt{enc}_C^s (N[\Gamma][\overline{\texttt{enc}}_B^s\, x / x]))[\lceil y' \rceil / y] & \\
= \texttt{abs}^s \lceil y' \rceil (\texttt{enc}_C^s (N[\Gamma, y : B]))[\lceil y' \rceil / y] & \text{Definition 29} \\
\to_\beta^* \texttt{abs}^s \lceil y' \rceil (N^s[\lceil y' \rceil / y]) & \text{induction hyp.} \\
= (\lambda y.N)^s. &
\end{array}$$

Last, suppose $M = P[yN/x]$, and let Σ, Δ be contexts such that $\Gamma = \Sigma, \Delta, y : B \multimap C$, $\mathrm{dom}(\Sigma) = FV(P)$, and $\mathrm{dom}(\Delta) = FV(N)$. Then we have:

$$\begin{array}{ll}
\texttt{enc}_A^s (P[yN/x])[\Gamma] & \\
= \texttt{enc}_A^s (P[\Sigma][(yN)[\Delta, y : B \multimap C]/x]) & \\
= \texttt{enc}_A^s (P[\Sigma][(\overline{\texttt{enc}}_{B\multimap C}^s\, y) N[\Delta]/x]) & \\
\to_\beta^* \texttt{enc}_A^s (P[\Sigma][\overline{\texttt{enc}}_C^s (\texttt{app}^s\, y (\texttt{enc}_B^s\, N[\Delta]))/x]) & \text{Definition 29} \\
\to_\beta^* \texttt{enc}_A^s (P[\Sigma][\overline{\texttt{enc}}_C^s (\texttt{app}^s\, y\, N^s)/x]) & \text{induction hyp.} \\
= \texttt{enc}_A^s (P[\Sigma][\overline{\texttt{enc}}_C^s (yN)^s/x]) & \\
= \texttt{enc}_A^s (P[\Sigma, x : C])[(yN)^s/x] & \text{Definition 29} \\
\to_\beta^* P^s[(yN)^s/x] & \text{induction hyp.} \\
= (P[yN/x])^s & \text{Lemma 32.}
\end{array}$$

□

Appendix A.3. The linear λ-term $\texttt{dec}_A^s$

The linear λ-term $\texttt{dec}_A^s$ is the component of $\texttt{D}_A$ requiring the type inhabitation. Roughly, it takes in input a tuple of boolean values encoding the η-long normal form V_A of a ground type A, and it produces the pair $\langle V_A, V_A\rangle$. To ensure that $\mathtt{dec}^s_A$ is defined on all possible inputs, it is built in such a way that it returns a default inhabitant of A whenever the tuple of booleans in input does not encode any λ-term.

Definition 32 (The linear λ-term $\mathtt{dec}^s_A$)**.** Let A be a ground type and let U be a value of type A. If for some c large enough $s = c \cdot (|A^-| \cdot \log |A^-|)$, then we define the linear λ-term $\mathtt{dec}^s_A : \mathbf{B}^s \multimap A \otimes A$ as follows:

$$\lambda x.\mathtt{if}\ x\ \mathtt{then}\ [P_{\mathtt{tt}^s}, P_{\langle \mathtt{tt}^{s-1}, \mathtt{ff}\rangle}, \ldots, P_{\langle \mathtt{ff}, \mathtt{tt}^{s-1}\rangle}, P_{\mathtt{ff}^s}]$$

where, for all $T = \langle \mathtt{b}_1, \ldots, \mathtt{b}_s\rangle$ of type $\mathbf{B}^s$:

$$P_T = \begin{cases} \langle V_A, V_A\rangle & \text{if } \langle \mathtt{b}_1, \ldots, \mathtt{b}_s\rangle = \lceil V_A \rceil \\ \langle U, U\rangle & \text{otherwise.} \end{cases}$$

We are now able to prove the fundamental result of this section:

Theorem 35 (Duplication [16])**.** *Every inhabited ground type is duplicable.*

Proof. The duplicator $\mathtt{D}_A$ of a inhabited ground type is defined as follows: we fix $s = c \cdot (|A^-| \cdot \log |A^-|)$, we fix a default value U of A (see Definition 32), and we set:

$$\mathtt{D}_A \triangleq \mathtt{dec}^s_A \circ \mathtt{enc}^s_A \circ \mathtt{sub}^s_A$$

which has type $A \multimap A \otimes A$. By Lemma 28, Lemma 34, and Definition 32 the conclusion follows. Moreover, for all values V of type A, we have:

$$\mathtt{D}_A\, V \to^*_\beta \langle V_A, V_A\rangle \to^*_\eta \langle V, V\rangle\ .$$

□

Remark 6. If A is a ground type inhabited by the value U, we shall write $\mathtt{D}^U_A$ to stress that the default inhabitant of A used in constructing the duplicator $\mathtt{D}_A$ of A is U.