

Light Affine Logic as a Programming Language: a First Contribution

LUCA ROVERSI

*Dipartimento di Informatica – Università di Torino
Corso Svizzera n. 185
10149 Torino – ITALY*

Received (received date)

Revised (revised date)

Communicated by Editor's name

ABSTRACT

This work is about an experimental paradigmatic functional language for programming with **P-TIME** functions. The language is designed from Intuitionistic Light Affine Logic. It can be typed automatically by a type inference algorithm that deduces polymorphic types *à la ML*.

Keywords: Functional programming languages, P-TIME computations, Light Affine Logic, Automatic type inference.

1. Introduction

This work is about a functional language Λ_{LA} , with a typable sub-set Λ_{LA}^T . The types for Λ_{LA}^T are polymorphic formulas of Intuitionistic Light Affine Logic (**ILAL** in the following.) Polymorphism is *à la ML* [12]: all the universal quantifications in any formula must occur as top-most operators.

As the formulas of **ILAL** are the types of Λ_{LA}^T , the functional language inherits the main computational property of **ILAL**: every term M of Λ_{LA}^T can be reduced to its normal form in a number of steps bound by a polynomial in the dimension of M . So, Λ_{LA}^T is an *experimental* language to program feasible functions.

A type inference algorithm for Λ_{LA} is defined. If a given term M of Λ_{LA} can have a formula of **ILAL** as type, then the algorithm deduces the most general type ν for M : all the other legal types for M can be obtained as instances of ν .

This paper contributes to solve the open problem Girard [4] left when he introduced Light Linear Logic. He wondered if some language could be designed from Light Linear Logic to program feasible functions. Unluckily, Intuitionistic Light Linear Logic was too cumbersome as a source for designing a language, using Curry-Howard principles [9]. Asperti [1] introduces **ILAL** to simplify the sequent calculus of Light Linear Logic. **ILAL** preserves both the language of the logical formulas, and the polynomial bound, significantly simplifying the set of rules. As-

perti's simplification consists of allowing unconstrained weakening in his logic. The sequent calculus of **ILAL** becomes simple enough to reformulate it as a natural deduction, and extract a manageable programming language.

Before concluding this introduction with the contents of the paper, we recall the main mechanism of **ILAL** to bound its cut elimination complexity. A thorough understanding of this mechanism will aid in understanding how Λ_{LA}^T works.

Let ς range over a countable set of names. The sequent calculus of **ILAL** derives formulas belonging to the language of the grammar in Figure 1, and has the rules

$$\nu ::= \varsigma \mid \nu \multimap \nu \mid \S\nu \mid !\nu \mid \forall\varsigma.\nu ,$$

Fig. 1. The Formulas of **ILAL**.

in Figure 2, where Γ , and Δ are multi-sets of formulas.

$$\begin{array}{c}
(Ax) \frac{}{\nu \vdash \nu} \quad (Cut) \frac{\Gamma \vdash \nu' \quad \Delta, \nu' \vdash \nu}{\Gamma, \Delta \vdash \nu} \\
\\
(W) \frac{\Gamma \vdash \nu}{\Gamma, \nu' \vdash \nu} \quad (C) \frac{\Gamma, !\nu', !\nu' \vdash \nu}{\Gamma, !\nu' \vdash \nu} \\
\\
(-\circ_l) \frac{\Gamma \vdash \nu' \quad \Delta, \nu'' \vdash \nu}{\Gamma, \Delta, \nu' \multimap \nu'' \vdash \nu} \quad (-\circ_r) \frac{\Gamma, \nu' \vdash \nu}{\Gamma \vdash \nu' \multimap \nu} \\
\\
(!) \frac{\nu_1 \dots \nu_n \vdash \nu \quad 0 \leq i \leq n \leq 1}{!\nu_1 \dots !\nu_n \vdash !\nu} \\
\\
(\S) \frac{\nu_1 \dots \nu_m, \nu'_1 \dots \nu'_n \vdash \nu \quad 0 \leq i \leq m \quad 0 \leq j \leq n \quad m+n \geq 0}{!\nu_1 \dots !\nu_m, \S\nu'_1 \dots \S\nu'_n \vdash \S\nu} \\
\\
(\forall_l) \frac{\Gamma, \{\nu'' \downarrow \varsigma\} \nu' \vdash \nu}{\Gamma, \forall\varsigma.\nu' \vdash \nu} \quad (\forall_r) \frac{\Gamma \vdash \nu \quad \varsigma \text{ not free in the types of } \Gamma}{\Gamma \vdash \forall\varsigma.\nu}
\end{array}$$

Fig. 2. The sequent calculus of **ILAL**.

The key feature of **ILAL** is to avoid any exponential proliferation of the contraction rule (C), during the cut elimination. This is achieved by restricting the form of the derivations that can be duplicated when eliminating the cuts:

- a derivation Π of **ILAL** can be duplicated only if it ends with (!)-rule;

- (!)-rule can be applied only to derivations that use *at most a single* assumption to derive their conclusion (Figure 2).

A situation originating the duplication of a derivation is:

$$(Cut) \frac{(!) \frac{\nu \vdash \nu'}{!\nu \vdash !\nu'} \quad (C) \frac{\Gamma, !\nu', !\nu' \vdash \nu''}{\Gamma, !\nu' \vdash \nu''}}{\Gamma, !\nu \vdash \nu''} .$$

The cut elimination step is in Figure 3, which uses graphs to represent sequent

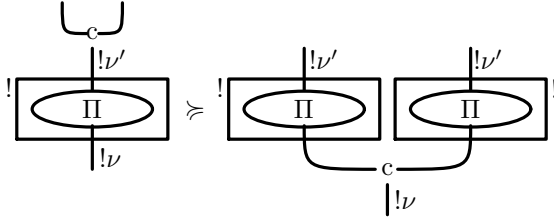


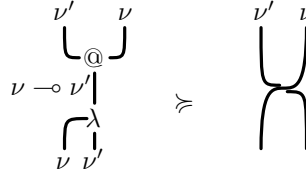
Fig. 3. A cut elimination step duplicating a derivation.

calculus derivations. The box, called !-box, stands for the application of (!)-rule to the derivation Π with conclusion $\nu \vdash \nu'$. In the leftmost picture, c -node represents the application of (C)-rule to the two occurrences of $!\nu'$ in $\Gamma, !\nu', !\nu' \vdash \nu''$. The upward link of the !-box is plugged into the downward link of c : so the application of (Cut)-rule is hidden. Finally, \succcurlyeq represents the cut elimination relation.

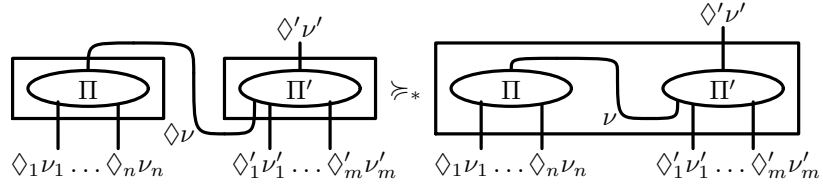
We use drawings to represent the derivations of **ILAL**, because the notion of *level* is more evident: the level l of a derivation Π contains all the links enclosed in exactly l boxes. Of course, the levels of any Π range between 0, and some finite integer L .

Assume the c -node in Figure 3 be at level l in some Π' . After \succcurlyeq , the single assumption of the !-box replicates the c -node without increasing the number of its occurrences: the c -node simply moves downward, through the !-box. After the cut elimination step, that c -node can keep duplicating other boxes. However, the duplication process is finite: the graphs for **ILAL** are acyclic because they are a sub-set of Proof nets [5]. As a result, the duplication steps at l are, at most, as many as the number n_l of links at l . Hence, when the duplication process at l stops, the number of links at level $l + 1$ has grown from n_{l+1} to, at most, $n_l n_{l+1}$. All other kinds of cut elimination steps strictly decrease the links at a given level. This means that, for every level, the cut elimination is finite. Figure 4 shows all the cut elimination steps not recalled yet. In particular, \succcurlyeq_* comes with the proviso that, if $\diamond' \equiv \S$, then, for any $m, n \geq 0$, and for any $1 \leq i \leq n$, and $1 \leq j \leq m$, we have $\diamond, \diamond_i, \diamond'_j \in \{!, \S\}$. Otherwise, if $\diamond' \equiv !$, then, for any $m = 0, n \leq 1$, and for any $1 \leq i \leq n$, we have $\diamond, \diamond_i \in \{!\}$.

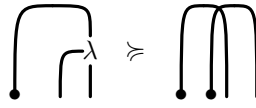
Now, the key observation is that the cut elimination can proceed by levels: from 0^{th} up to L^{th} . It is a finite process because it is finite at every level, and there is no way to create an eliminable cut at l when eliminating cuts at $l + 1$.



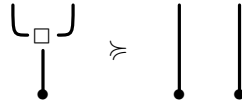
Cut elimination between a $(-o_l)$ -rule (upper node) and a $(-o_r)$ -rule (lower node.)



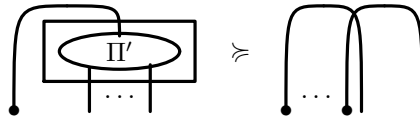
Cut elimination merging the borders of two boxes.



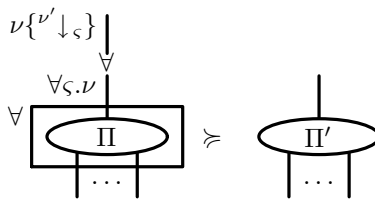
Cut elimination erasing a $(-o_r)$ -rule.



Cut elimination erasing either a $(-o_l)$ -rule or a (C) -rule.



Cut elimination erasing a box.



Cut elimination instantiating a polymorphic type variable:

Π' is Π , where ν' replaces ς everywhere in ν .

Fig. 4. The Cut Elimination of **ILAL**.

Assuming that L cannot grow, it is simple to extract a bound on the dimension of the normalized derivation, which is at most polynomial in the initial dimension of Π , with a degree exponential in L . Finally, **ILAL** is designed to avoid any growth of L . Figure 4 shows that the only way to change L is by decreasing it, through the erasure of a box. The access to the body of any box is realized by merging its border with the border of some other box. Accessing the bodies of the boxes serves to apply duplicable functions, with type $!(\nu \multimap \nu')$, for example, to an argument. Such an argument must be enclosed in a box as well, and its type will be either $!\nu$ or $\S\nu$. So, the application can take place only when both the argument, and the function are inside the same box. The level they live at is preserved. Of course, a duplicable function can require multiple arguments. Take $!(\nu_1 \multimap \nu_2 \multimap \nu)$ as an example. \S -boxes allow to feed such a function with as many arguments as needed, because \S -boxes can have more than one assumption. However, since \S -boxes are not duplicable, the polynomial bound is not broken.

This mechanism can be studied with the help of the graphs here above, or using Λ_{LA}^T that we are going to introduce with a lot of programming examples.

“Index”. Section 2 introduces the untyped syntax Λ of the typed functional language Λ_{LA}^T we are interested to. Λ has not any explicit constructs for encoding (C)-rule. This choice influences the overall design of Λ_{LA}^T . It must be defined on two disjoint sets of variables names: one contains the names for the terms that can be duplicated during the computations. The other set contains the names for linearly used terms. Hence, Λ_{LA}^T is a kind of *call-by-value* language. These aspects are discussed in Section 5, where the dynamics is defined, after the introduction of the types in Section 3, and of the type assignment in Section 4. In particular, Section 4 defines the natural deduction of **ILAL**, decorated with the functional terms, under Curry-Howard principles. Section 6 is about the expressiveness of Λ_{LA}^T . All the polynomials can be encoded in it. Some other interesting programming examples are developed, like a very efficient and compact version of the predecessor on a representation of Natural numbers. When reading Section 2, and 4, we suggest to refer to Section 6 for some programming examples. Section 7 addresses some of the syntactical problems that Λ_{LA}^T shares with other languages, used for encoding **ILAL** or Intuitionistic Light Linear Logic. Section 8 shows that there exists a poly-step reduction strategy on Λ_{LA}^T . Section 9 defines the type inference algorithm. Correctness and Completeness of it are given. This section also has a simulation of a type inference for better justifying some a syntactical choice made when for designing Λ_{LA}^T . Subsection 9.2 is about the main details for proving Correctness and Completeness of the type inference. Section 10 concludes the paper with some reference to related, and future work.

2. The Functional Syntax.

Let x, y, w, z range over the set of *linear* identifiers **Term-Variables**, and let X, Y, W, Z range over the set of *exponential* identifiers **!-Term-Variables**. Let also χ be ranging over **Term-Variables** \cup **!-Term-Variables**. The set Λ of the functional

terms is in Figure 5. The term constructors λ , $!$, \S , and **let** bind free variables. As

$$\begin{aligned}
M, N ::= & \text{Term-Variables} \mid \text{!-Term-Variables} \mid \\
& \lambda\chi.M \mid (MN) \mid \\
& !M \mid !(M)^{[N/\chi]} \mid \\
& \S M \mid \S(M)^{[M_1/\chi_1 \dots M_m/\chi_m ; N_1/\chi'_1 \dots N_n/\chi'_n]} \mid \\
& \text{let } X = M \text{ in } N
\end{aligned}$$

Fig. 5. The Functional Syntax.

usual, λ is such that, if χ is in the free variable set $\mathbf{FV}(M)$ of M , then it is not in $\mathbf{FV}(\lambda\chi.M)$. The term constructor $!$ can be applied either to a closed term M , yielding the closed $!$ -box $!M$, or to an open term M with a *single* free variable χ . In this case, the free variables of the obtained $!$ -box are in $\mathbf{FV}(N)$. Observe that the single free variable χ of M can occur more than once in M itself. The structure $[^N/\chi]$ is the *interface* of the $!$ -box, with the *single component* $^N/\chi$, and M is the *body* of the box. The operator \S builds \S -boxes. If it is applied to a closed term M , it yields the closed \S -box $\S M$. Otherwise, \S can be applied to a term M such that $\chi_1, \dots, \chi_m, \chi'_1, \dots, \chi'_n$, with $m + n \geq 1$, are *all* its free variables $\mathbf{FV}(M)$. Then, all the elements of $\mathbf{FV}(M)$ get bounded, and the free variables of the obtained \S -box are $(\cup_{i=1}^m \mathbf{FV}(M_i)) \cup (\cup_{i=1}^n \mathbf{FV}(N_i))$. The term M is the *body* of the \S -box. The structure $[^{M_1/\chi_1 \dots M_m/\chi_m ; N_1/\chi'_1 \dots N_n/\chi'_n}]$ is its interface. It has two parts to let the type inference working. For example, an instance of \S -box is:

$$\S(xy)^{[M/x ; N/y]} , \quad (2.1)$$

for some M and N .

Finally, if $X \in \mathbf{FV}(N)$, then **let** binds X .

The elements of Λ are considered up to α -equivalence: *consistently* renaming the bound variables of a term M yields another term M' which is equivalent to M . For example, the following pairs of terms are α -equivalent:

- $!(\lambda x.Yx)^{[N/Y]}$, and $!(\lambda x.Xx)^{[N/X]}$
- $!(\lambda x.Yx)^{[N/Y]}$, and $!(\lambda y.Yy)^{[N/Y]}$

The renaming is *consistent* if it preserves the kind of the variables: capital names replace only capital names. For example, $!(\lambda x.Yx)^{[N/Y]}$, and $!(\lambda X.YX)^{[N/Y]}$ are not α -equivalent. Moreover, consistent renaming requires to avoid usual variable clashes. For example, $!(\lambda x.yx)^{[N/y]}$, and $!(\lambda x.xx)^{[N/x]}$ are not α -equivalent.

The substitution of M for χ in N is denoted by $N\{^M\downarrow_\chi\}$. It extends, obviously, the variable-clash free substitution of terms for variables of the λ -Calculus, while preserving the two classes of variables. For example:

- $X\{^Y\downarrow_X\} = Y$,

- $x\{y\downarrow_x\} = y$

but $x\{Y\downarrow_x\}$ is undefined. So, the substitution $N\{M\downarrow_\chi\}$ on Λ is a partial function. It behaves like the substitution for the λ -calculus, *only* in the following cases:

- M is either a $!$ -box, or in $!$ -Term-Variables, and χ is in $!$ -Term-Variables,
- M is any term, and χ belongs to Term-Variables.

Otherwise, the substitution is undefined.

3. The Types

Let α, β, γ range over the set of *linear* identifiers **Type-Variables**, and let δ, ϵ range over the set of *exponential* identifiers $!$ -**Type-Variables**. Let also ς, ν range over **Type-Variables** \cup $!$ -**Type-Variables**. The *types* are defined by the grammar in Figure 6.

$$\begin{aligned} \tau, \rho &::= L \mid E \\ L &::= \text{Type-Variables} \mid \tau \multimap \rho \mid \S\tau \\ E &::= !\text{-Type-Variables} \mid !\tau . \end{aligned}$$

Fig. 6. The Types.

The *type schemes* originate from the grammar in Figure 7.

$$\sigma ::= \forall \varsigma_1 \dots \varsigma_n . \tau \quad n \geq 0 .$$

Fig. 7. The Type Schemes.

As usual, \forall is a binder: the free variables of $\forall \varsigma_1 \dots \varsigma_n . \tau$ are $\text{FV}(\tau) \setminus \{\varsigma_1 \dots \varsigma_n\}$ with $\text{FV}(\tau)$ having the obvious inductive definition. The type schemes are taken up to α -equivalence.

We distinguish two disjoint sets of type schemes. The first set contains the *linear type schemes*:

$$\forall \varsigma_1 \dots \varsigma_n . \tau \quad (n \geq 0) ,$$

where τ belongs to the sub-language with the start symbol L . The second has the *exponential type schemes*:

$$\forall \varsigma_1 \dots \varsigma_n . \tau \quad (n \geq 0) ,$$

where τ belongs to the sub-language with the start symbol E . For example, $\forall \alpha . \delta \multimap (!\alpha)$ is linear, while $\forall \beta . !\beta$ and $\forall \delta . \delta$ are exponential.

A *set of assumptions* is a set of pairs $\{\chi_1 : \sigma_1, \dots, \chi_n : \sigma_n\}$, where every σ_i is a *type scheme*, every χ_i belongs to $\text{Term-Variables} \cup \text{!-Term-Variables}$, and such that:

1. χ_i belongs to !-Term-Variables (Term-Variables) if, and only if, σ_i is an exponential (linear) type scheme, and
2. $\{\chi_1 : \sigma_1, \dots, \chi_n : \sigma_n\}$ is a function with finite domain $\{\chi_1, \dots, \chi_n\}$. Namely, if $i \neq j$ then $\chi_i \neq \chi_j$.

According to the nature of their co-domain, hence of their domain, we have *linear*, and *exponential* (sets of) assumptions. The co-domain of linear assumptions contains *only* linear type schemes. The co-domain of the exponential ones have *only* exponential type schemes.

We take Θ for ranging over exponential (sets of) assumptions, and Δ, Φ for ranging over the linear (sets of) assumptions. Γ is used as a meta-variable for generic (sets of) assumptions that cannot be classified neither linear nor exponential, or which we do not mind to be one of the two.

For any set Γ of assumptions, and any type τ , the notation $\forall \Gamma. \tau$ stands for the type $\forall \varsigma_1 \dots \varsigma_n. \tau$, where $\{\varsigma_1 \dots \varsigma_n\}$ is $\text{FV}(\tau) \setminus \text{FV}(\Gamma)$.

The *type substitutions* map $\text{Type-Variables} \cup \text{!-Type-Variables}$ to *types*. They replace linear types for linear variables, and exponential types for exponential variables. The simultaneous substitution of $\tau_1 \dots \tau_n$ for $\varsigma_1 \dots \varsigma_n$, which is the identity on all the type variables different from $\varsigma_1, \dots, \varsigma_n$ is denoted by $\{\tau_1 \downarrow_{\varsigma_1} \dots \tau_n \downarrow_{\varsigma_n}\}$. The *support* $\text{supp}(S)$ of S is the set $\{\varsigma \mid S(\varsigma) \text{ different from } \varsigma\}$. The type substitutions are ranged over by S, R , and U . Moreover, for any type scheme σ , and any set of assumptions Γ , we write $S\sigma$, and $S\Gamma$ for the application of the obvious extensions of S to the type schemes and to the assumptions.

Finally, the type schemes can be *ordered* as follows:

$$\forall \varsigma_1 \dots \varsigma_m. \tau \geq \forall v_1 \dots v_n. S\tau.$$

with $n, m \geq 0$, and:

$$\begin{aligned} \text{supp}(S) &\subseteq \{\varsigma_1, \dots, \varsigma_m\} \\ \text{FV}(\forall \varsigma_1 \dots \varsigma_m. \tau) &\subseteq \text{FV}(\forall v_1 \dots v_n. S\tau) . \end{aligned} \tag{3.1}$$

4. The Typing Rules

For any set of assumptions Γ , any functional term M , and any type τ :

$$\Gamma \vdash_{\text{T}} M : \tau$$

says that M has type τ from Γ . The rules for deriving the judgments are in Figure 8. The linear assumptions are used multiplicatively, while those exponentials have additive occurrences. Also observe that $(Ax_l), (Ax_e), (!_\emptyset)$, and (\S_\emptyset) have implicit weakening, while $(\multimap_E), (\S)$, and (let) have implicit contraction.

$$\begin{array}{c}
(Ax_l) \frac{}{\Gamma, x : \tau \vdash_{\mathbf{T}} x : \tau} \qquad (Ax_e) \frac{\sigma \geq \tau}{\Gamma, X : \sigma \vdash_{\mathbf{T}} X : \tau} \\
\\
(\multimap_E) \frac{\Theta, \Delta_1 \vdash_{\mathbf{T}} M : \tau' \multimap \tau \quad \Theta, \Delta_2 \vdash_{\mathbf{T}} N : \tau'}{\Theta, \Delta_1, \Delta_2 \vdash_{\mathbf{T}} MN : \tau} \qquad (\multimap_I) \frac{\Gamma, \chi : \tau' \vdash_{\mathbf{T}} M : \tau}{\Gamma \vdash_{\mathbf{T}} \lambda \chi. M : \tau' \multimap \tau} \\
\\
(!) \frac{\Gamma \vdash_{\mathbf{T}} N : !\tau' \quad \chi : \tau' \vdash_{\mathbf{T}} M : \tau}{\Gamma \vdash_{\mathbf{T}} !(M)[^N/\chi] : !\tau} \qquad (!\emptyset) \frac{\vdash_{\mathbf{T}} M : \tau}{\Gamma \vdash_{\mathbf{T}} !M : !\tau} \\
\\
\begin{array}{l}
m + n \geq 1 \\
\Theta, \Delta_i \vdash_{\mathbf{T}} M_i : !\tau_i \qquad (0 \leq i \leq m) \\
\Theta, \Phi_j \vdash_{\mathbf{T}} N_j : \S\rho_j \qquad (0 \leq j \leq n)
\end{array} \\
(\S) \frac{\chi_1 : \tau_1 \dots \chi_m : \tau_m, \chi'_1 : \rho_1 \dots \chi'_n : \rho_n \vdash_{\mathbf{T}} M : \tau}{\Theta \dots \Delta_i \dots \Phi_j \dots \vdash_{\mathbf{T}} \S(M)[\dots^{M_i}/\chi_i \dots; \dots^{N_j}/\chi'_j \dots] : \S\tau} \\
\\
(\S\emptyset) \frac{\vdash_{\mathbf{T}} M : \tau}{\Gamma \vdash_{\mathbf{T}} \S M : \S\tau} \\
\\
(\text{let}) \frac{\Theta, \Delta_1 \vdash_{\mathbf{T}} M : \tau' \quad X : \forall \Theta, \Delta_1. \tau', \Theta, \Delta_2 \vdash_{\mathbf{T}} N : \tau}{\Theta, \Delta_1, \Delta_2 \vdash_{\mathbf{T}} \text{let } X = M \text{ in } N : \tau}
\end{array}$$

Fig. 8. The Typing Rules.

As a first, simple example of typing in \vdash_T , we encode the Church numeral $\overline{2} \stackrel{\text{def}}{=} \lambda f x. f(f x)$ of the λ -Calculus in Λ :

$$(-\circ_I) \frac{(\S) \frac{\mathcal{D} \quad \mathcal{D} \quad \mathcal{D}'}{X :!(\alpha \multimap \alpha) \vdash_T \S(\lambda y. z_1(z_2 y)) [^X/z_1 \ ^X/z_2;] : \S(\alpha \multimap \alpha)}}{\lambda X. \S(\lambda y. z_1(z_2 y)) [^X/z_1 \ ^X/z_2;] :!(\alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha)}}{,}$$

where \mathcal{D} is:

$$(Ax_e) \frac{}{X :!(\alpha \multimap \alpha) \vdash_T X :!(\alpha \multimap \alpha)},$$

\mathcal{D}' is:

$$(-\circ_I) \frac{(-\circ_E) \frac{(Ax_l) \frac{}{z_1 : \alpha \multimap \alpha \vdash_T z_1 : \alpha \multimap \alpha} \quad \mathcal{D}''}{z_1 : \alpha \multimap \alpha, z_2 : \alpha \multimap \alpha, y : \alpha \vdash_T z_1(z_2 y) : \alpha}}{z_1 : \alpha \multimap \alpha, z_2 : \alpha \multimap \alpha \vdash_T \lambda y. z_1(z_2 y) : \alpha \multimap \alpha}}{,}$$

and \mathcal{D}'' is:

$$(-\circ_E) \frac{(Ax_l) \frac{}{z_2 : \alpha \multimap \alpha \vdash_T z_2 : \alpha \multimap \alpha} \quad (Ax_l) \frac{}{y : \alpha \vdash_T y : \alpha}}{z_2 : \alpha \multimap \alpha, y : \alpha \vdash_T z_2 y : \alpha}.$$

Later we shall need the following terminology. The rule $(-\circ_I)$ is called *introduction* rule. All the others, but (Ax_l) , (Ax_e) , and (let) , are called *elimination* rules.

Λ^T is the set of all the *typable* terms of Λ , being $M \in \Lambda$ *typable* if, and only if, there exist Γ , and τ such that $\Gamma \vdash_T M : \tau$.

The terms of Λ^T , and the substitution of a term for a free variable, that we already defined, well behave with respect to the type assignment. Indeed, the Substitution property holds. Firstly, we have:

Lemma 1 (Substitution distributivity) *If $\Gamma \vdash_T M : \tau$, then $S\Gamma \vdash_T M : S\tau$, for any S .*

Proof. By induction on M , the “worst” case being $M \in \text{!-Term-Variables}$. In this case $\Gamma \vdash_T M : \tau$ coincides with $\Gamma', X : \forall \varsigma_1 \dots \varsigma_n. \tau \vdash_T X : R\tau$, for some R . The proof exploits the α -rule on the types. We can always choose $\varsigma_1 \dots \varsigma_n$ not belonging to $\text{supp}(S) \cup (\cup_{\varsigma \in \text{supp}(S)} \text{FV}(S\varsigma))$. \square

Lemma 2 (Substitution) *1. Let $\Theta, x : \tau, \Delta \vdash_T M : \tau'$. Then, $\Theta, \Phi, \Delta \vdash_T M\{^N \downarrow_x\} : \tau'$, for any $\Theta, \Phi \vdash_T N : \tau$.*

2. Let $\Theta, X : \forall \varsigma_1 \dots \varsigma_n. \tau, \Delta \vdash_T M : \tau'$. Then, $\Theta, \Phi, \Delta \vdash_T M\{^N \downarrow_X\} : \tau'$, for any $\Theta, \Phi \vdash_T N : \tau$ such that N is either a !-box or an element of !-Term-Variables.

Proof. The first statement can be simply proved by induction on M .

The second statement requires some simple remarks. The bound variables $\varsigma_1 \dots \varsigma_n$ can be chosen so that they do not belong to the type variables free in Θ, Φ . Then, call *relevant* all the instances $\Gamma_i, X : \forall \varsigma_1 \dots \varsigma_n. \tau \vdash_T X : R_i\tau$ of (Ax_e) , with $0 \leq i \leq m$ and some $m \geq 0$, in the derivation of $\Theta, X : \forall \varsigma_1 \dots \varsigma_n. \tau, \Delta \vdash_T M : \tau'$. Thanks to (3.1), and Lemma 1, $\Theta, \Phi \vdash_T N : R_i\tau$, for every R_i . Then, derive $\Theta, \Phi, \Delta \vdash_T M\{^N \downarrow_X\} : \tau'$ as follows: proceed like for deducing $\Theta, X :$

$\forall \varsigma_1 \dots \varsigma_n. \tau, \Delta \vdash_{\mathbb{T}} M : \tau'$, using $\Gamma, \Theta \vdash_{\mathbb{T}} N : R_i \tau$ in place of the relevant axiom corresponding to it. \square

5. The Dynamics.

We describe the normalization steps of the typing rules on a simplification of the terms in $\Lambda^{\mathbb{T}}$. The *simplification* consists of *erasing* every occurrence of “;” in the interfaces of the \S -boxes of the terms in $\Lambda^{\mathbb{T}}$. Call $\Lambda_{\perp}^{\mathbb{T}}$ the new obtained language. The rewriting system \rightsquigarrow on $\Lambda_{\perp}^{\mathbb{T}} \times \Lambda_{\perp}^{\mathbb{T}}$ is the contextual closure of the rewriting relations in Figure 9 through 13.

$$\begin{array}{lcl}
(\lambda x.M)N & \triangleright_1 & M\{^N \downarrow_x\} \\
(\lambda X.M)Y & \triangleright_2 & M\{^Y \downarrow_X\} \\
(\lambda X.M)!N & \triangleright_3 & M\{!^N \downarrow_X\} \\
(\lambda X.M)!(N)^{[Y/X]} & \triangleright_4 & M\{!(N)^{[Y/X]} \downarrow_X\} \\
(\lambda X.M)!(N)^{[P/X]} & \triangleright_5 & (\lambda Y.M\{!(N)^{[Y/X]} \downarrow_X\})P \quad \text{if } P \notin \text{!-Term-Variables}
\end{array}$$

Fig. 9. β -group

$$\begin{array}{lcl}
!(M)^{[!N/X]} & \triangleright_1 & !M\{^N \downarrow_x\} \\
!(M)^{[!(N)^{[P/X]}/X]} & \triangleright_2 & !(M\{^N \downarrow_x\})^{[P/X]} \\
!(M)^{[!(Y)^{[P/Y]}/X]} & \triangleright_3 & !(M\{^Y \downarrow_x\})^{[P/Y]} \\
!(M)^{[!!N/X]} & \triangleright_4 & !(M\{!^N \downarrow_X\}) \\
!(M)^{[!!(N)^{[P/X]}/X]} & \triangleright_5 & !(M\{!(N)^{[Y/X]} \downarrow_X\})^{[!P/Y]} \\
!(M)^{[!(N)^{[P/X]}]^{[Q/X']}/X]} & \triangleright_6 & !(M\{!(N)^{[Y/X]} \downarrow_X\})^{[!(P)^{[Q/X']}/Y]}
\end{array}$$

Fig. 10. !!-group

$$\begin{array}{lcl}
\S(M)[\dots !^N/x_i \dots] & \triangleright_1 & \S(M\{^N \downarrow_{x_i}\})[\dots \dots] \\
\S(M)[\dots !(N)^{[P/X]}/x_i \dots] & \triangleright_2 & \S(M\{^N \downarrow_{x_i}\})[\dots ^P/X \dots] \\
\S(M)[\dots !(Y)^{[P/Y]}/x_i \dots] & \triangleright_3 & \S(M\{^Y \downarrow_{x_i}\})[\dots ^P/Y \dots] \\
\S(M)[\dots !!N/x_i \dots] & \triangleright_4 & \S(M\{!^N \downarrow_{x_i}\})[\dots \dots] \\
\S(M)[\dots !!(N)^{[P/X]}/x_i \dots] & \triangleright_5 & \S(M\{!(N)^{[Y/X]} \downarrow_{x_i}\})[\dots !P/Y \dots] \\
\S(M)[\dots !(N)^{[P/X]}]^{[Q/X']}/x_i \dots] & \triangleright_6 & \S(M\{!(N)^{[Y/X]} \downarrow_{x_i}\})[\dots !(P)^{[Q/X']}/Y \dots]
\end{array}$$

Fig. 11. $\S!$ -group

The substitution of terms for variables on $\Lambda^{\mathbb{T}}$ works on $\Lambda_{\perp}^{\mathbb{T}}$ as well. The α -equivalence must be used to avoid variable clashes when rewriting terms.

$$\begin{aligned}
& \S(M)[\dots \S^N/x_i \dots] \triangleright_1 \S(M\{^N \downarrow x_i\})[\dots \dots] \\
& \S(M)[\dots \S^{(N)[\dots^P/x_i \dots]}/x_i \dots] \triangleright_2 \S(M\{^N \downarrow x_i\})[\dots \dots^P/x_i \dots \dots] \\
& \S(M)[\dots \S^{(Y)[^P/Y]}/x_i \dots] \triangleright_3 \S(M\{^Y \downarrow x_i\})[\dots^P/Y \dots] \\
& \S(M)[\dots \S^{!N}/x_i \dots] \triangleright_4 \S(M\{^{!N} \downarrow x_i\})[\dots \dots] \\
& \S(M)[\dots \S^{!(N)[^P/x_i]}/x_i \dots] \triangleright_5 \S(M\{^{!(N)[^Y/x_i] \downarrow x_i\})[\dots \S^P/Y \dots] \\
& \S(M)[\dots \S^{!(N)[^P/x_i][\dots^Q/x_i' \dots]}/x_i \dots] \triangleright_6 \S(M\{^{!(N)[^Y/x_i] \downarrow x_i\})[\dots \dots \S^{(P)[\dots^Q/x_i' \dots]}/Y \dots \dots]
\end{aligned}$$

Fig. 12. $\S\S$ -group

$$\begin{aligned}
\mathbf{let} X = Y \mathbf{in} P & \triangleright_1 P\{^Y \downarrow X\} \\
\mathbf{let} X = !M \mathbf{in} P & \triangleright_2 P\{^{!M} \downarrow X\} \\
\mathbf{let} X = !(M)[^N/x_i] \mathbf{in} P & \triangleright_3 \mathbf{let} Y = N \mathbf{in} P\{^{!(M)[^Y/x_i] \downarrow X\}
\end{aligned}$$

Fig. 13. \mathbf{let} -group

Every term to the left of \triangleright is a *redex*, while every right-hand term is the *reduct*.

The subject reduction theorem here below assures the correctness of the rewriting system \rightsquigarrow :

Theorem 1 (Subject Reduction) *Let M, N be terms of Λ^T . Assume M_- , and N_- be the terms of Λ^T , obtained from M , and N , respectively. If $\Gamma \vdash_T M : \tau$, and $M_- \rightsquigarrow N_-$, then $\Gamma \vdash_T N : \tau$.*

Proof. By induction on the definition of \rightsquigarrow , rearranging the structure of the derivation, according to \rightsquigarrow . \square

The reflexive, and transitive closure of \rightsquigarrow on Λ is \rightsquigarrow^* .

The presence of the \mathbf{let} -construct requires to consider the set of terms up to *commuting equivalences*. For example, assume:

$$(\mathbf{let} X = P \mathbf{in} (\lambda\chi.M))N \tag{5.1}$$

have type τ from Γ in \vdash_T . (5.1) can be rewritten only if some redexes exist in one among P, M or N . However, it is not difficult to verify that:

$$\mathbf{let} X = P \mathbf{in} ((\lambda\chi.M)N) \tag{5.2}$$

has type τ from Γ as well, and, moreover, (5.2) potentially has a redex more than (5.1). This is a usual problem of the functional languages with \mathbf{let} -like construct, which is solved by taking the terms up to some commuting equivalences. Our equivalences are in Figure 14. They are consistent with respect to the types: both sides of every equivalence have the same type from the same set of assumptions. The result is immediately verifiable for the four uppermost equivalences. The last one requires:

$$\begin{aligned}
(\mathbf{let} X = P \mathbf{in} (\lambda\chi.M))N &\equiv \mathbf{let} X = P \mathbf{in} ((\lambda\chi.M)N) \\
!(M)[^{\mathbf{let} X=P \mathbf{in} N/\chi}] &\equiv \mathbf{let} X = P \mathbf{in} !(M)[^N/\chi] \\
\mathfrak{S}(M)[\dots^{\mathbf{let} X=P \mathbf{in} N/\chi}\dots] &\equiv \mathbf{let} X = P \mathbf{in} \mathfrak{S}(M)[\dots^N/\chi\dots] \\
\mathbf{let} X = (\mathbf{let} Y = P \mathbf{in} M) \mathbf{in} N &\equiv \mathbf{let} Y = P \mathbf{in} \mathbf{let} X = M \mathbf{in} N
\end{aligned}$$

Fig. 14. The Commuting Equivalences

Lemma 3 *The rule:*

$$(\leq) \frac{\Gamma, X : \sigma \vdash_{\mathsf{T}} M : \tau \quad \sigma \leq \sigma'}{\Gamma, X : \sigma' \vdash_{\mathsf{T}} M : \tau}$$

is admissible in \vdash_{T} .

Proof. By induction on the depth of the derivation of $\Gamma, X : \sigma \vdash_{\mathsf{T}} M : \tau$, using Lemma 2. \square

Lemma here above will also be used later for proving Completeness of the type inference algorithm with respect to \vdash_{T} .

A term M is \rightsquigarrow -normal or, simply *normal*, if \rightsquigarrow cannot rewrite M any more. The normal terms are the typable terms of the language generated by the grammar in Figure 15. For example, $\mathfrak{S}(z)[^{xy/z}]$ can be well typed, and it is normal: xy cannot

$$\begin{aligned}
M &::= \lambda\chi.M \mid B \mid \\
&\quad !M \mid !(M)[^B/\chi] \mid \\
&\quad \mathfrak{S}M \mid \mathfrak{S}(M)[^B/\chi_1 \dots^B/\chi_m] \mid \\
&\quad \mathbf{let} X = A \mathbf{in} M \\
B &::= A \mid \mathbf{Term-Variables} \mid \mathbf{!-Term-Variables} \\
A &::= (\lambda X.M)(x \underbrace{M \dots M}_{n \geq 1}) \mid x \underbrace{M \dots M}_{p \geq 1}
\end{aligned}$$

Fig. 15. The Values

be rewritten into any kind of box. In first approximation, the reduction stops in presence of an application which is both unable to yield some box, and which is argument of one among an application, a \mathbf{let} , or an interface of some box. The commuting equivalences shift the \mathbf{let} -prefixes when they can prevent the formation of some redexes.

From now on, we will tend to use the (“erased”) terms of $\Lambda_{\perp}^{\mathsf{T}}$. In particular, $\Lambda_{\text{LA}}^{\mathsf{T}}$ will refer to the language $\langle \Lambda_{\perp}^{\mathsf{T}}, \rightsquigarrow \rangle$. We will talk about Λ_{LA} as well, which stands for $\langle \Lambda_{\perp}, \rightsquigarrow \rangle$, where Λ_{\perp} is the obvious erasure of the untyped language Λ . However, when dealing with the type inference algorithm, we will be forced to refer to the terms of Λ^{T} , defined in Figure 5, where the the interfaces of the \mathfrak{S} -boxes are split.

5.1. Discussion.

It is worth giving some intuition about the meaning and the behavior of the dynamics.

Λ_{LA}^T is a kind of restriction of a typed call-by-value λ -Calculus [14], which rewriting rule is:

$$(\lambda x.M)N \rightarrow_{\beta_v} M\{^N \downarrow_x\} \text{ if } N \text{ belongs to } \text{Values} = \text{Variables} \cup \lambda\text{-Abstractions} .$$

Namely, only the terms with a specific form can be substituted for the variables of the call-by-value λ -Calculus. The rewriting system \rightsquigarrow behaves analogously to \rightarrow_{β_v} . No constraints exist when replacing $x \in \text{Term-Variables}$ by *any* term. The idea is that, in Λ_{LA}^T , x stands for any non duplicable entity, also called *linear*. Consequently, replacing M for x never duplicates M . On the other hand, only the !-boxes, and the elements of !-Term-Variables can be substituted for a variable $X \in \text{!-Term-Variables}$. In Λ_{LA}^T , X represents duplicable resources, also called *exponential*. So, borrowing a call-by-value terminology, any term of Λ_{LA}^T is a value with respect to the linear variables, while, only the !-boxes, and the !-Term-Variables are values for the exponential variables. We can stress more the need of two kinds of variables, using a simple example. Consider the term:

$$(\lambda X.\S(M)[^X /_{\chi_1} \ X /_{\chi_2}])(xy) \tag{5.3}$$

According to \rightsquigarrow , the left-most application cannot reduce. Assume, now, to have only a single kind of variables. Then, (5.3) would be:

$$(\lambda x.\S(M)[^x /_{\chi_1} \ x /_{\chi_2}])(xy) \tag{5.4}$$

Without any type information, we could reduce (5.4) to $(\lambda x.\S(M)[^{xy} /_{\chi_1} \ xy /_{\chi_2}])$. This could not correspond to any normalization step of the logic, because something which is not a !-box was duplicated. So, the two kinds of variables carry with them a least type information. It is required to perform the right rewriting steps, in absence of the whole type information in the functional syntax. Similar ideas are used in [15] to encode Intuitionistic Linear Logic in a very compact functional language.

Let us observe now the behavior of the β -group in Figure 9. The first four axioms follow what just said about the linear and the exponential values. The axiom \triangleright_5 needs a side condition to take it apart from \triangleright_4 . In particular, \triangleright_5 serves to avoid the substitution for X of the interface P , as it could not be a !-box.

Consider also the !!-group in Figure 10. The relation defined by the !!-rules makes two terms communicating, when they are the bodies of two distinct !-boxes. The communication takes place by substituting the body of one !-box for the occurrences of the free variable of the body of the other !-box. The rule \triangleright_1 deals with the case where N is the body of a !-box in the interface of another !-box, whose body is M . The communication between N and M takes place independently from the form of N , accordingly to the substitution. Otherwise, N must reduce to a further, deeper !-box, before the substitution takes place: see the rule \triangleright_4 . The remaining !!-rules cover all the possible *disjoint* cases, according to the form of the !-box in the interface. Remark that saying: “the bodies of two different boxes communicate”, means that the body of two boxes can be accessed. However, this access does not eliminate the border of the boxes.

All the rewriting groups behave accordingly to the intuitive idea, used for describing how the (partial) substitution works. The rewriting groups are defined to cover all the *disjoint* cases, depending on the form of the term being substituted.

The next part of this discussion, shows that the functional terms are truly untyped: there are terms of Λ_{LA} that cannot be typed, but which evaluate to their normal form by means of \rightsquigarrow . A first example is M' , defined as:

$$(\lambda Y.\underbrace{\S(\S(z_1 z_2)[y_1 \underbrace{!(\lambda y.y)}_I / z_1 \quad y_2 \underbrace{!(\lambda w z.z)}_K / z_2)]}_{\Delta'}[Y / y_1 \quad Y / y_2])! \underbrace{(\lambda X.\S(x_1 x_2)[X / x_1 \quad X / x_2])}_{\Delta'},$$

which is a possible translation in Λ_{LA} , of the λ -Calculus term:

$$M \stackrel{\text{def}}{=} (\lambda x.x \ I(x \ K))! \underbrace{(\lambda y.y \ y)}_{\Delta} .$$

The term M' cannot be typed in \vdash_{T} , for M has not types in System F [6], of which \vdash_{T} is a sub-system.

An evaluation sequence of M' is:

$$\begin{aligned} M' &\rightsquigarrow_{\beta_3} \S(\S(z_1 z_2)[y_1 \ !I / z_1 \quad y_2 \ !K / z_2])[! \Delta' / y_1 \quad ! \Delta' / y_2] \\ &\rightsquigarrow_{\S_{1,1}^*}^* \S(\S(z_1 z_2)[\Delta' \ !I / z_1 \quad \Delta' \ !K / z_2]) \\ &\rightsquigarrow_{\beta_{3,3}}^* \S(\S(z_1 z_2)[\S(x_1 x_2)[!I / x_1 \quad !I / x_2] / z_1 \quad \S(x_1 x_2)[!K / x_1 \quad !K / x_2] / z_2]) \\ &\rightsquigarrow_{\S_{1,1,1,1}^*}^* \S(\S(z_1 z_2)[\S(II) / z_1 \quad \S(KK) / z_2]) \\ &\rightsquigarrow_{\S_{1,1}^*}^* \S(\S(II(KK))) \\ &\rightsquigarrow_{\beta_{1,1,1}^*}^* \S\S I . \end{aligned}$$

As a second example, we show a class \mathcal{C} of terms without types in \vdash_{T} . \mathcal{C} contains the approximations of the translation of the λ -Calculus fix-point operator $\Theta\Theta$, where:

$$\Theta \stackrel{\text{def}}{=} \lambda xy.y(x \ x \ y) .$$

Let:

$$\Theta' \stackrel{\text{def}}{=} \lambda x.\S(y \underbrace{(\lambda w y z.w(y \ z))}_J)[x / y] ,$$

and, for any closed term M of Λ_{LA} , define:

$$\begin{aligned} T^0 &= \lambda x.x \ M \ I \ \S I \\ T^{n+1} &= \lambda x.x \ M \ \Theta' \ \S T^n . \end{aligned}$$

Then $\Theta' \S T^n$ reduces to:

$$\S(M \ \S(M \ \dots \ \S(M \ \S I) \dots))$$

with $n + 1$ nested \S -boxes, i.e., with $n + 2$ \S -boxes in total. For example:

$$\begin{aligned}
\Theta' \S T^1 &\rightsquigarrow \S(y J)[\S T^1 / y] \\
&\rightsquigarrow \S(T^1 J) \\
&\rightsquigarrow \S(J M \Theta' \S T^0) \\
&\rightsquigarrow \S(M(\Theta' \S T^0)) \\
&\rightsquigarrow^* \S(M \S(M \S I)) .
\end{aligned}$$

It is interesting to understand why we cannot simulate the possibly infinite reduction of $\Theta \Theta N$ in Λ_{LA} . The point is the lack of an operator that allows to increase the nesting level of a term by enclosing it into more \S and (or) $!$ -boxes. So, we can only approximate the computation of $\Theta \Theta N$ with a term that contains a finite number of nested boxes in the right position. Recall that forbidding the growth of the number of nested boxes is the key point to control the complexity of the cut elimination for Light Affine Logic.

Two remarks to conclude.

The first is about the non existence of terms in Λ_{LA} that do not type-check because they break the poly-time reduction bound. This is a consequence of its design principles. Λ_{LA} essentially realizes a Curry-Howard Isomorphism [9] for **ILAL** from which it inherits all the structural properties. The poly-time bound on the cut elimination of **ILAL** is purely structural: once written a derivation of **ILAL**, the types can be erased, and the cut elimination still has the wanted bound. The bound is taken under control by limiting the possibility of applying functions to functions. Typically, the representation in **ILAL** of the Church numeral $\bar{2}$ can be applied to itself only by including it into a box, otherwise the occurrence of $\bar{2}$, used as argument, cannot be duplicated. The possibility of duplicating only $!$ -boxes holds also on the untyped structures that are at the base of **ILAL**. So, we can get to the same conclusions on Λ_{LA} .

The second remark deals with the terms that do not break the poly-step reduction bound, and which do not type-check anyway. This, actually, addresses an open problem. It is not known if $\Lambda_{\text{LA}}^{\text{T}}$ can encode all **P-TIME** Turing machines. To have an idea about the meaning of encoding **P-TIME** Turing machines into functional language refer to Roversi [18]. There, a functional language, more compact than $\Lambda_{\text{LA}}^{\text{T}}$, which represents the derivations of **ILAL** under Curry-Howard Isomorphism principles, encodes all **P-TIME** Turing machines. The basic point of such an encoding is that it exploits the fully polymorphic types of **ILAL**, which are not available in $\Lambda_{\text{LA}}^{\text{T}}$.

6. The Expressiveness

This section shows that $\Lambda_{\text{LA}}^{\text{T}}$ is expressive enough to represent the polynomials on integers. Moreover, a remarkable programming example will be the encoding of the predecessor on integers. It will be also useful to address some aspects of the syntax of Λ_{LA} that requires improvements.

6.1. The Polynomials

To encode the polynomials in $\Lambda_{\text{LA}}^{\text{T}}$ we follow Girard [4]. Some changes will be necessary, however, because the type assignment for $\Lambda_{\text{LA}}^{\text{T}}$ has not the full expres-

siveness of the second order quantifiers, for, we recall, we want the decidability of the type inference.

The result of this section is that any polynomial $f(x_1, \dots, x_n)$ of arity $n \geq 0$ can be represented as a term $\langle f(x_1, \dots, x_n) \rangle$ of Λ_{LA}^T .

We are not going to develop the proof in its full generality, for sake of readability. We choose, instead, to work on an example, and to encode the non linear polynomial x^3+1 . This is done in three steps. Firstly, we introduce in Λ_{LA}^T the Church numerals and the operations on them that we need. Secondly, the encoding of the *linear* polynomial $x_1 \times x_2 \times x_3 + 1$ will be defined. Finally, $x^3 + 1$ is represented.

6.1.1. The Church numerals.

The Church numerals on Λ_{LA}^T are:

$$\begin{aligned} \bar{0} &\stackrel{\text{def}}{=} \lambda X. \S \lambda y. y : \mathbf{int}_\tau \\ \bar{n} &\stackrel{\text{def}}{=} \lambda X. \S (\lambda w. w_1 (\dots (w_n w) \dots)) [^X /_{w_1} \dots ^X /_{w_n}] : \mathbf{int}_\tau , \end{aligned}$$

where: $\mathbf{int}_\tau \stackrel{\text{def}}{=} !(\tau \multimap \tau) \multimap \S(\tau \multimap \tau)$.

The combinators here above are called Church numerals because there exist a translation from Λ_{LA} to the λ -Calculus that, applied to $\bar{0}$ and \bar{n} , yields the λ -Calculus Church numerals:

$$\lambda f x. \underbrace{f(\dots (f x) \dots)}_{n \geq 0} .$$

The translation is as simple as follows: from a given $M \in \Lambda_{LA}$, delete all the occurrences of $!$, and \S . Then, for any $[\dots^N /_\chi \dots]$ of any box, substitute N for χ in the body, no matter what the forms of N and χ are. Finally, erase all the interfaces, and collapse **Term-Variables** and **!-Term-Variables** into a single set. Of course, the substitutions must avoid variable clash.

A first combinator on our Church numerals is:

$$\mathit{succ} \stackrel{\text{def}}{=} \lambda z X. \S (\lambda y. y_1 (y_2 y)) [^X /_{y_1} (z^X) /_{y_2}] : \mathbf{int}_\tau \multimap \mathbf{int}_\tau .$$

Then, the numeral next to \bar{n} is:

$$\begin{aligned} \mathit{succ} \bar{n} &\rightsquigarrow \lambda X. \S (\lambda y. y_1 (y_2 y)) [^X /_{y_1} (\bar{n}^X) /_{y_2}] \\ &\rightsquigarrow \lambda X. \S (\lambda y. y_1 (y_2 y)) [^X /_{y_1} \S (\lambda w. w_1 (\dots (w_n w) \dots)) [^X /_{w_1} \dots ^X /_{w_n}] /_{y_2}] \\ &\rightsquigarrow \lambda X. \S (\lambda y. y_1 ((\lambda w. w_1 (\dots (w_n w) \dots)) y)) [^X /_{y_1} ^X /_{w_1} \dots ^X /_{w_n}] \\ &\rightsquigarrow \lambda X. \S (\lambda y. y_1 (w_1 (\dots (w_n y) \dots))) [^X /_{y_1} ^X /_{w_1} \dots ^X /_{w_n}] \\ &=_\alpha \lambda X. \S (\lambda y. y_1 (y_1 (\dots (y_n y) \dots))) [^X /_{y_1} ^X /_{y_2} \dots ^X /_{y_{n+1}}] \\ &\stackrel{\text{def}}{=} \overline{n+1} . \end{aligned}$$

Other combinators are:

$$\begin{aligned}
sum &\stackrel{\text{def}}{=} \lambda w z X. \S(\lambda y. y_1(y_2 y)) [{}^w X /_{y_1} z X /_{y_2}] : \mathbf{int}_\tau \multimap \mathbf{int}_\tau \multimap \mathbf{int}_\tau \\
iter &\stackrel{\text{def}}{=} \lambda x_n X_s x_b. \S(y w) [{}^{x_n} X_s /_y x_b /_w] : \mathbf{int}_\tau \multimap !(\tau \multimap \tau) \multimap \S \tau \multimap \S \tau \\
mult &\stackrel{\text{def}}{=} \lambda x Y. iter x !(\lambda w. sum z w) [{}^Y /_z] \S \bar{\theta} : \mathbf{int}_{\mathbf{int}_\tau} \multimap !\mathbf{int}_\tau \multimap \S \mathbf{int}_\tau \\
coerc &\stackrel{\text{def}}{=} \lambda x. \S(y \bar{\theta}) [{}^x !succ /_y] : \mathbf{int}_{\mathbf{int}_\tau} \multimap \S \mathbf{int}_\tau .
\end{aligned}$$

sum adds two numerals. *iter* takes as arguments a numeral, a *step* function, and a *base* where to start the iteration from. Observe that *iter* $\bar{\mathcal{D}} !\bar{n} \S \bar{\theta}$ cannot have type, for any numeral \bar{n} . This because the step function is required to have identical domain and co-domain. This should not surprise. Taking the λ -Calculus Church numeral $\bar{\mathcal{D}}$, and applying it to itself we get an exponentially costing computation. Notice, however, that there are variations of $\bar{\theta}$, and \bar{I} that we can use as step function for *iter*:

$$\begin{aligned}
\bar{\theta}' &\stackrel{\text{def}}{=} \lambda x. \S \lambda y. y : \S(\alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha) \\
\bar{\theta}'' &\stackrel{\text{def}}{=} \lambda X. !\lambda y. y : !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \\
\bar{I}' &\stackrel{\text{def}}{=} \lambda x. \S(\lambda y. y_1 y) [{}^x /_{y_1}] : \S(\alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha) \\
\bar{I}'' &\stackrel{\text{def}}{=} \lambda X. !(\lambda y. y_1 y) [{}^X /_{y_1}] : !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) .
\end{aligned}$$

Clearly the terms here above can be step functions because they are *fake* iterations. The *true* ones start from $\bar{\mathcal{D}}$ upward.

mult is defined as an iterated sum, for multiplying two numerals.

Finally, *coerc*(ion) embeds a numeral into a \S -box, preserving its value:

$$\begin{aligned}
coerc \bar{n} & \\
&\rightsquigarrow \S(y \bar{\theta}) [{}^{\bar{n}} !succ /_y] \\
&\rightsquigarrow \S(y \bar{\theta}) [{}^{\S(\lambda w. w_1(\dots(w_n w)\dots))} [{}^{!succ /_{w_1} \dots !succ /_{w_n}}] /_y] \\
&\rightsquigarrow \S(y \bar{\theta}) [{}^{\S(\lambda w. succ(\dots(succ w)\dots))} /_y] \\
&\rightsquigarrow \S((\lambda w. succ(\dots(succ w)\dots)) \bar{\theta}) \\
&\rightsquigarrow \S(succ(\dots(succ \bar{\theta})\dots)) \\
&\rightsquigarrow^* \S \bar{n} .
\end{aligned}$$

6.1.2. The Polynomials.

we develop a leading example about how encoding the polynomials. In particular, we develop the explicit construction of an encoding of the *non linear* polynomial $P_n \stackrel{\text{def}}{=} x^3 + 1$, passing through an encoding $\langle x_1 \times x_2 \times x_3 + 1 \rangle$ of its *linear* version $P_l \stackrel{\text{def}}{=} x_1 \times x_2 \times x_3 + 1$.

The construction of $\langle P_l \rangle$ proceeds inductively. So, $\langle P_l \rangle$ must be:

$$\langle x_1 \rangle \langle \times \rangle \langle x_2 \rangle \langle \times \rangle \langle x_3 \rangle \langle + \rangle \langle 1 \rangle .$$

At this point, the structure of the types of Λ_{LA}^T imposes a “bottom-up” translation strategy. We start encoding the leftmost multiplication:

$$\langle x_1 \rangle \langle \times \rangle \langle x_2 \rangle \stackrel{\text{def}}{\equiv} \text{mult } y_1 \ Y_2 : \S \mathbf{int}_{\mathbf{int}_\tau} , \quad (6.1)$$

being:

$$\begin{aligned} \langle x_1 \rangle &\stackrel{\text{def}}{\equiv} y_1 : \mathbf{int}_{\mathbf{int}_{\mathbf{int}_\tau}} \\ \langle x_2 \rangle &\stackrel{\text{def}}{\equiv} Y_2 : !\mathbf{int}_{\mathbf{int}_\tau} . \end{aligned}$$

The type we have assumed for y_1 and Y_2 are not the most general, at this point. However, they serve to conclude the whole encoding with its most general type.

The type of (6.1) forces to use a \S -box to open it, namely for accessing its innermost type, before it can be multiplied by $\langle x_3 \rangle \stackrel{\text{def}}{\equiv} y_3$. This suggests to assume $y_3 : \S !\mathbf{int}_\tau$ such that:

$$(\langle x_1 \rangle \langle \times \rangle \langle x_2 \rangle) \langle \times \rangle \langle x_3 \rangle \stackrel{\text{def}}{\equiv} \S(\text{mult } z \ W)[^{\text{mult } y_1 \ Y_2 / z \ y_3 / W}] : \S^2 \mathbf{int}_\tau \quad (6.2)$$

Before the encoding of the sum between the result of (6.2), and the constant 1, we introduce an abbreviation. It contracts the consecutive nesting of boxes. We give it only by using an example. The term:

$$\S(\S(!(\text{succ } y)[^W / y])[^X / W])[^w / X][^z / w]$$

becomes:

$$\S^2(!^2(\text{succ } y)[^X / y])[^z / X] .$$

Now, we are ready to encode the sum between the result of (6.2) and 1. The result of the encoding of the multiplication is two \S -boxes deep. So, both the sum, and the translation of 1 must be into two nested \S -boxes as well:

$$\langle P_1 \rangle \stackrel{\text{def}}{\equiv} \S^2(\text{sum } x \ y)[^{\S(\text{mult } z \ W)[^{\text{mult } y_1 \ Y_2 / z \ y_3 / W}] / x \ \S^2 T / y}] : \S^2 \mathbf{int}_\tau , \quad (6.3)$$

from the assumptions $y_1 : \mathbf{int}_{\mathbf{int}_{\mathbf{int}_\tau}}$, $Y_2 : !\mathbf{int}_{\mathbf{int}_\tau}$, and $y_3 : \S !\mathbf{int}_\tau$.

For a simpler readability of $\langle P_n \rangle$, that we are going to introduce, let see some generalizations of $\bar{0}$, succ , and coerc :

$$\begin{aligned} \bar{0}_0 &\stackrel{\text{def}}{\equiv} \bar{0} : \mathbf{int}_\tau \\ \bar{0}_q &\stackrel{\text{def}}{\equiv} \S^{p!q} \bar{0} : \S^{p!q} \mathbf{int}_\tau \\ \text{succ}_0^0 &\stackrel{\text{def}}{\equiv} \text{succ} : \mathbf{int}_\tau \multimap \mathbf{int}_\tau \\ \text{succ}_q^p &\stackrel{\text{def}}{\equiv} \lambda x. \S^p (!^q (\text{succ } y)[^X / y])[^z / X] : \S^{p!q} \mathbf{int}_\tau \multimap \S^{p!q} \mathbf{int}_\tau \\ \text{coerc}_0^0 &\stackrel{\text{def}}{\equiv} \text{coerc} : \mathbf{int}_{\mathbf{int}_\tau} \multimap \S \mathbf{int}_\tau \\ \text{coerc}_q^p &\stackrel{\text{def}}{\equiv} \lambda x. \S(y \ \bar{0}_q^p)[^{x \ !\text{succ}_q^p / y}] : \mathbf{int}_{\S^{p!q} \mathbf{int}_\tau} \multimap \S^{p+1!q} \mathbf{int}_\tau , \end{aligned}$$

with $p, q \geq 0$.

Finally, the encoding of $x^3 + 1$. Firstly, we transform (6.3) so that the types of its free variables are instances of the same type. (6.3) becomes:

$$\begin{aligned} & \S(\S^2(\text{sum } x \ y)[\S(\text{mult } z \ W)[^{\text{mult } y_1 \ Y_2 / z \ y_3 / w} / x \ \S^2 \bar{T} / y] \\ & \quad)[\text{coerc}_0^0 \ x_1 / y_1 \ \text{coerc}_1^0 \ x_2 / Y_2 \ \text{coerc}_1^1 \ x_3 / y_3] : \S^3 \mathbf{int}_\tau \ , \end{aligned} \quad (6.4)$$

where $x_1 : \mathbf{int}_{\mathbf{int}_{\mathbf{int}_{\mathbf{int}_\tau}}$, $x_2 : \mathbf{int}_{\mathbf{int}_{\mathbf{int}_\tau}}$, and $x_3 : \mathbf{int}_{\mathbf{int}_{\mathbf{int}_\tau}}$.

For any numeral \bar{n} , (6.4) allows to write:

$$\begin{aligned} \langle P_n \rangle & \stackrel{\text{def}}{=} \text{let } X = \bar{n} \\ & \text{in } \S(\S(\S^2(\text{sum } x \ y)[\S(\text{mult } z \ W)[^{\text{mult } y_1 \ Y_2 / z \ y_3 / w} / x \ \S^2 \bar{T} / y] \\ & \quad)[\text{coerc}_0^0 \ x_1 / y_1 \ \text{coerc}_1^0 \ x_2 / Y_2 \ \text{coerc}_1^1 \ x_3 / y_3] \\ & \quad) [X / x_1 \ X / x_2 \ X / x_3] : \S^4 \mathbf{int}_\tau \ . \end{aligned}$$

Notice that $\langle P_n \rangle$ is not the unique encoding of P_n . However, it evaluates to a numeral four \S -boxes deep.

In general, two kinds of type assumptions are used to represent a *linear* polynomial $f(x_1, \dots, x_n)$. One of them is \mathbf{int}_ρ , where ρ is a suitable ‘‘tower’’ of integers $\mathbf{int}_{\dots_{\mathbf{int}_\tau}}$. The other is $\mathbf{int}_{\S^p \mathbf{int}_\rho}$, with $p, q \geq 0$. The type for the representation becomes $\S^k \mathbf{int}_\tau$, with k depending on the structure of the polynomial. The *non linear* polynomials are encoded as closed terms, using the `let` constructor to identify sub-sets of free variables.

6.2. The Predecessor

For a compact representation of the predecessor, encode the type *tensor product* \otimes in Λ_{LA}^T as follows:

$$\tau_1 \otimes_\rho \dots \otimes_\rho \tau_n \stackrel{\text{def}}{=} (\tau_1 \multimap \dots \multimap \tau_n \multimap \rho) \multimap \rho \ ,$$

for any type ρ . This tensor type has two canonical terms associated: one to produce, and the other for accessing its components. The first is:

$$M_1 \otimes_\rho \dots \otimes_\rho M_n \stackrel{\text{def}}{=} \lambda x. x M_1 \dots M_n : \tau_1 \otimes_\rho \dots \otimes_\rho \tau_n \ ,$$

for any ρ , assuming $\Theta, \Delta_i \vdash_{\text{T}} M : \tau_i$, for every $1 \leq i \leq n$. The term accessing the components of a tensor is:

$$\lambda \chi_1 \otimes_\rho \dots \otimes_\rho \chi_n. P \stackrel{\text{def}}{=} \lambda x. x (\lambda \chi_1 \dots \chi_n. P) : \tau_1 \otimes_\rho \dots \otimes_\rho \tau_n \multimap \rho \ ,$$

where $\Gamma, \chi_1 : \tau_1 \dots \chi_n : \tau_n \vdash_{\text{T}} P : \rho$, for any ρ . Thanks to the α -equivalence, the choice of all χ_i s can be such that:

$$\begin{aligned} (\lambda \chi_1 \otimes_\rho \dots \otimes_\rho \chi_n. P)(M_1 \otimes_\rho \dots \otimes_\rho M_n) & \rightsquigarrow^* \\ & (P\{M_1 \downarrow_{\chi_1}\}) \dots \{M_n \downarrow_{\chi_n}\} \equiv P\{M_1 \downarrow_{\chi_1} \dots M_n \downarrow_{\chi_n}\} \ , \end{aligned}$$

where $P\{M_1 \downarrow_{\chi_1} \dots M_n \downarrow_{\chi_n}\}$ is the simultaneous substitution of every M_i for the corresponding χ_i in P .

Observe also that the tensor has the projections, as consequence of the unrestricted weakening.

Now, we have what we need to define the predecessor:

$$pred \stackrel{\text{def}}{=} \lambda w X. \xi(\lambda y. \pi_2(z(\text{base } y))) [w \text{ !}(\text{step } x) [x/z] / z] : \mathbf{int}_{(\alpha \multimap \alpha) \otimes_\alpha \alpha} \multimap \mathbf{int}_\alpha ,$$

where:

$$\begin{aligned} step &\stackrel{\text{def}}{=} \lambda xy. x \otimes_\alpha ((\lambda w \otimes_\alpha z. w z) y) : \\ &\quad (\alpha \multimap \alpha) \multimap ((\alpha \multimap \alpha) \otimes_\alpha \alpha) \multimap ((\alpha \multimap \alpha) \otimes_\alpha \alpha) \\ base &\stackrel{\text{def}}{=} \lambda y. (\lambda w. w) \otimes_\alpha y : \alpha \multimap (\alpha \multimap \alpha) \otimes_\alpha \alpha \\ \pi_2 &\stackrel{\text{def}}{=} \lambda x \otimes_\alpha y. y : (\alpha \multimap \alpha) \otimes_\alpha \alpha \multimap \alpha , \end{aligned}$$

for any α . The behavior of $pred$ is more evident if we rephrase it in the λ -Calculus with pair constructors $[M, N]$. Let:

$$\begin{aligned} xy^0 &\stackrel{\text{def}}{=} [I, y] \\ xy^{n+1} &\stackrel{\text{def}}{=} [x, \underbrace{x(\dots(xy)\dots)}_n] , \end{aligned}$$

with $n \geq 0$. Assume xy^n be an *alternative* representation of the Church numeral $\overline{n} x y$ in the λ -Calculus, with $n \geq 0$. Observe that $\underbrace{(\lambda[w, z]. [x, w z])}_{step}(xy^n)$ yields the

alternative representation xy^{n+1} of $\overline{n+1} x y$. So, iterating n times $step$ from xy^0 we end up with the *alternative* representation $[x, xy^{n-1}]$ of $\overline{n} x y$. We can apply to it the second projection, getting the representation of $\overline{n-1} x y$. So, we have calculated the predecessor of \overline{n} . To move from the λ -Calculus to Λ_{LA}^T , it is enough to replace pairs with tensors, and to use the boxes when required.

The predecessor just defined on Λ_{LA}^T simplifies Asperti's [1] definition. This because, our predecessor does not make any use of the encoding of the additive types by means of the second order quantification.

7. The Garbage Collection

The introduction of the predecessor in Subsection 6.2 allows to focus about a syntactical problem of Λ_{LA} .

For any α , consider the following reduction:

$$\begin{aligned}
& \text{pred } \overline{2} \\
& \rightsquigarrow \lambda X. \S(\lambda y. \pi_2(z(\text{base } y)))[\overline{2} \ !(\text{step } x)[^X/x_1]/z] \\
& \rightsquigarrow \lambda X. \S(\lambda y. \pi_2(z(\text{base } y)))[\S(\lambda y. y_1(y_2 \ y))[\!(\text{step } x_1)[^X/x_1]/y_1 \ \!(\text{step } x_2)[^X/x_2]/y_2]/z] \\
& \rightsquigarrow \lambda X. \S(\lambda y. \pi_2(z(\text{base } y)))[\S(\lambda y. (\text{step } x_1)((\text{step } x_2) \ y))[^X/x_1 \ ^X/x_2]/z] \\
& \rightsquigarrow \lambda X. \S(\lambda y. \pi_2((\lambda w. (\text{step } x_1)((\text{step } x_2) \ w))(\text{base } y)))[^X/x_1 \ ^X/x_2] \\
& \rightsquigarrow \lambda X. \S(\lambda y. \pi_2((\text{step } x_1)((\text{step } x_2)(\text{base } y))))[^X/x_1 \ ^X/x_2] \\
& \rightsquigarrow \lambda X. \S(\lambda y. \pi_2((\text{step } x_1)((\text{step } x_2)(I \otimes_\alpha y))))[^X/x_1 \ ^X/x_2] \\
& \rightsquigarrow^* \lambda X. \S(\lambda y. \pi_2((\text{step } x_1)(x_2 \otimes_\alpha y)))[^X/x_1 \ ^X/x_2] \\
& \rightsquigarrow^* \lambda X. \S(\lambda y. \pi_2(x_1 \otimes_\alpha (x_2 \ y)))[^X/x_1 \ ^X/x_2] \\
& \rightsquigarrow \lambda X. \S(\lambda y. x_2 \ y)[^X/x_1 \ ^X/x_2] . \tag{7.1}
\end{aligned}$$

Morally, the result just obtained is $\overline{1}$. However, syntactically, it contains a redundant occurrence of X . We can overcome the problem introducing some garbage collection rules. They are the contextual closure of the rewriting relation:

$$\begin{aligned}
& \!(M)[^N/\chi] \quad \blacktriangleright \quad \!M \\
& \S(M)[\dots^N/\chi \dots] \quad \blacktriangleright \quad \!M[\dots \dots] ,
\end{aligned}$$

which must be applied only to \rightsquigarrow -normal terms, if $\chi \notin \mathbf{FV}(M)$. For example, garbage collecting (7.1) yields:

$$\lambda X. \S(\lambda y. x_2 \ y)[^X/x_1 \ ^X/x_2] \quad \blacktriangleright \quad \S(\lambda y. x_2 \ y)[^X/x_2] \stackrel{\text{def}}{=} \overline{1} .$$

A remark is worth here. The just outlined syntactical issue that requires the garbage collection is not only a problem of $\Lambda_{\text{LA}}^{\text{T}}$. It is already present in the Proof nets for Intuitionistic Light Logic [4], and in Asperti's graphs language for Intuitionistic Affine Logic [1]. The problem originates from the interaction between the weakening internal to the boxes, and the boxes themselves. A better syntax, without the explicit use of the borders to delimit the boxes would solve the problem. This is the subject of further developments.

8. Poly-step Reduction Strategy

The language $\Lambda_{\text{LA}}^{\text{T}}$ has a *poly-step* evaluation process: given a term M , there is a *normal strategy* such that M rewrites to a normal term N in a number of \rightsquigarrow -steps bound by the dimension $|M|$ of M . Being poly-step in the above sense is not exactly as having a **P-TIME** normalization process in $|M|$. Counting the time means to consider, at least, the cost of the renaming operations, and of the substitution of terms for variables. But this is not an issue: $\Lambda_{\text{LA}}^{\text{T}}$ is only a paradigmatic language to program with, and not an implementation language.

This section is organized as follows.

We recall the notions about a typed language of graphs $\mathbf{G}_{\text{LA}}^{\text{T}}$ [1], already given in Section 1.

Then, we introduce the *erased* version \mathbf{G}_{LA} of $\mathbf{G}_{\text{LA}}^{\text{T}}$. The language \mathbf{G}_{LA} is the set of graphs obtained from $\mathbf{G}_{\text{LA}}^{\text{T}}$ by erasing all type information. \mathbf{G}_{LA} inherits the

P-TIME bound on the computational complexity: if the normalization length of the typed graph G^T is at most a polynomial in its dimension $|G^T|$, then the same holds for the erasure G of G^T , with respect to $|G|$.

Finally, we prove that Λ_{LA}^T is poly-step. This proof splits into two parts. Firstly, we show that a rewriting relation \succ of \mathbf{G}_{LA} , is computationally adequate with respect to \rightsquigarrow on Λ_{LA}^T . Namely:

Theorem 2 (Adequacy) *There exists an embedding $(.)^p$ from Λ_{LA}^T to \mathbf{G}_{LA} such that, if $M \rightsquigarrow N$, then $M^p \succ^* N^p$.*

Secondly, we define the steps \rightsquigarrow_p of a *canonical* reduction strategy \rightsquigarrow_p^* on \rightsquigarrow . The poly-time reduction strategy \succ_p^* for \mathbf{G}_{LA} yields:

Theorem 3 (Single-Poly-step Adequacy) *For any term M of Λ_{LA}^T , if $M \rightsquigarrow_p N$, then $M^p \succ_p^* N^p$,*

which can be generalized by replacing $M \rightsquigarrow_p N$ with $M \rightsquigarrow_p^* N$.

8.1. The Typed Graphs.

Let \mathbf{G}_{LA}^T be the language of types of the grammar in Figure 16. **Type-Variables**

$$\nu ::= \text{Type-Variables} \mid \nu \multimap \nu \mid \S\nu \mid !\nu \mid \forall \varepsilon. \nu$$

Fig. 16. The types of \mathbf{G}_{LA}^T .

is ranged over by ε . It contains type variables unrelated to both the linear, and the exponential type variables of \vdash_T .

\mathbf{G}_{LA}^T is the least language of graphs which contains the axioms:

$$\begin{array}{l} \longleftarrow \text{ conclusion} \\ \nu \\ \longleftarrow \text{ assumption} \end{array}$$

and such that, if $G, G' \in \mathbf{G}_{\text{LA}}^T$, then the graphs in Figure 17. belong to \mathbf{G}_{LA}^T as well. The graphs of \mathbf{G}_{LA}^T stand for the derivations of intuitionistic sequent calculus. This means that they have a single conclusion, obtained by a, possibly empty, set of assumptions. The conclusion are upward links, while the assumptions are those downward.

\mathbf{G}_{LA}^T is the tool Asperti used to prove the complexity of **ILAL** [1].

The rewriting system of \mathbf{G}_{LA}^T is \succ . It was introduced in Section 1. We recall it here: \succ is the least relation on $\mathbf{G}_{\text{LA}}^T \times \mathbf{G}_{\text{LA}}^T$, which contains the *contextual closure* of the relation \succ_* between *parts* of graphs, defined in Figure 18, and Figure 19. In particular, \succ_* comes with the proviso that, if $\diamond' \equiv \S$, then, for any $m, n \geq 0$, and for any $1 \leq i \leq n$, and $1 \leq j \leq m$, we have $\diamond, \diamond_i, \diamond'_j \in \{!, \S\}$. Otherwise, if $\diamond' \equiv !$, then, for any $m = 0, n \leq 1$, and for any $1 \leq i \leq n$, we have $\diamond, \diamond_i \in \{!\}$.

The reflexive, and transitive closure of \succ on \mathbf{G}_{LA}^T is \succ^* .

The left-hand sides of the rules defining \succ are the *redexes*. Their right-hand sides are the *reducts*.

A graph G is \succ -*normal* or, simply *normal*, if \succ can no more rewrite G .

The strategy that reduces any graph G in poly-time, with respect to its dimension is called \succ_p^* . We need some notions to recall it. Take $G \in \mathbf{G}_{\text{LA}}^T$. The graph

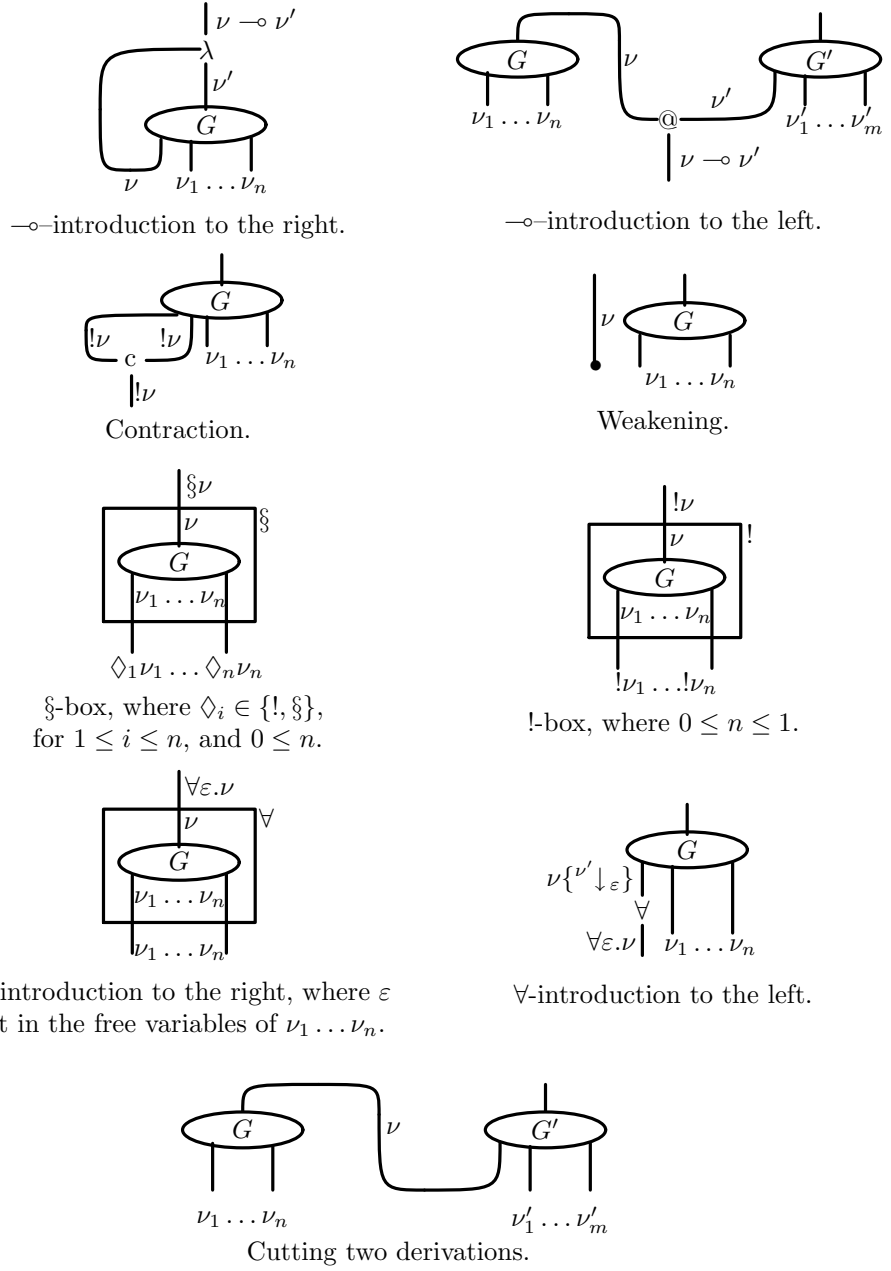
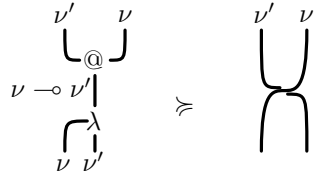
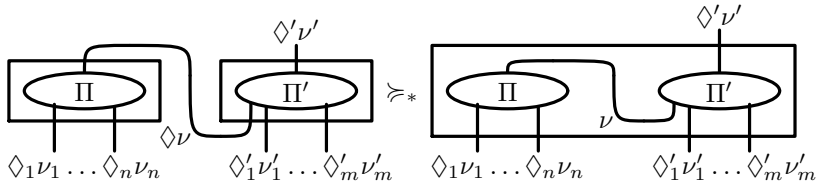


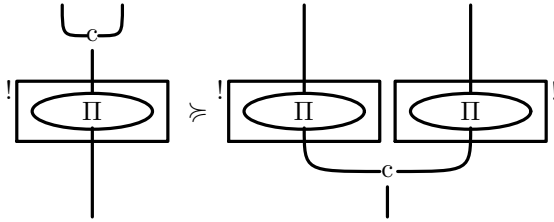
Fig. 17. The typed graphs.



Cut elimination between two higher-order nodes

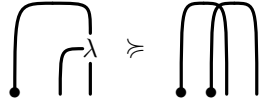


Cut elimination merging the borders of two boxes.

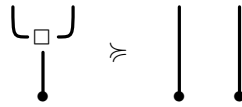


Cut elimination duplicating a !-box.

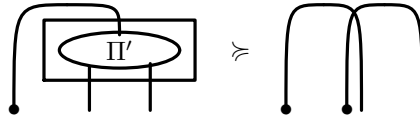
Fig. 18. The rewriting system for the typed graphs: first half.



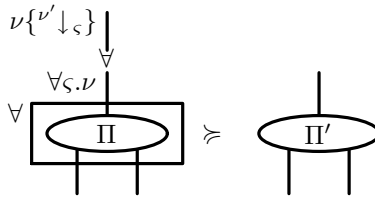
Cut elimination erasing a $-o_r$ -rule.



Cut elimination erasing either a $-o_l$ -rule or a C -rule.



Cut elimination erasing a box.



Cut elimination instantiating a polymorphic type variable:

Π' is Π , where ν' replaces ς everywhere in ν .

Fig. 19. The rewriting system for the typed graphs: second half.

G' is at depth $i \geq 0$ in G , written G'^i , if it is in the body of i nested boxes of G . The notion of depth can obviously be used also for the redexes, classified in two sets. The β -redexes are all the axioms in Figure 18, and Figure 19, but \succ_* . The remaining one is the *box-redex*. Assume G having at most d nested boxes. For any $0 \leq i \leq d$, the i^{th} reduction round reduces all the β -redexes G^i , and all the box-redexes G^{i-1} , in any order. The *poly-time reduction strategy* \succ_p^* is the sequence of reduction rounds which starts from the 1st and stops (at most) at the d^{th} . We have already given in Section 1 some hints at how getting the polynomial bound, thanks to \succ_p^* . For more details about it look at Asperti's paper [1].

A useful terminology is: \rightarrow -introduction to the right is the *right-rule* of $\mathbf{G}_{\text{LA}}^{\text{T}}$. All the others, but the cut rule, are *left-rules*.

8.2. The Untyped Graphs.

The language \mathbf{G}_{LA} of the untyped graphs is defined in three steps from $\mathbf{G}_{\text{LA}}^{\text{T}}$. Erase the introductions to the left, and to the right of the universal quantification in Figure 17. Erase all the type labels on the edges of the elements of $\mathbf{G}_{\text{LA}}^{\text{T}}$. Erase the last rule in Figure 19.

The proof of Theorem 2 requires the extension of \succ with the garbage collecting rules in Figure 20. We keep calling \succ the rewriting system on \mathbf{G}_{LA} just obtained.

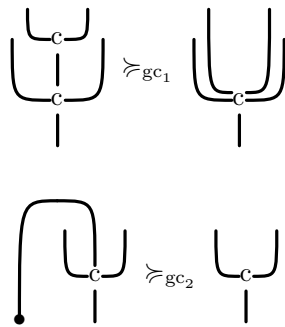


Fig. 20. The garbage collecting rules.

The poly-time strategy \succ_p^* is extended with garbage collecting rules, which become part of the β -redexes. This extension does not break the complexity bound. Firstly, the untyped graphs of \mathbf{G}_{LA} are essentially typed, by definition. In particular, $\mathbf{G}_{\text{LA}}^{\text{T}}$ is to \mathbf{G}_{LA} what Girard's System \mathcal{F} [6], is to the set of functional, and polymorphic, λ -terms, which are typable by what Mitchell calls *Pure Typing Theory* [13]. This means that every reduction step of \mathbf{G}_{LA} is a reduction step of $\mathbf{G}_{\text{LA}}^{\text{T}}$. Secondly, the garbage collecting rules simplify the structure of the graphs, by erasing nodes. Recently, Di Cosmo and Kesner [3] proved that our two garbage collection rules can be added to the cut elimination of the Proof Nets for Linear Logic [5] without breaking Strong Normalizability, and without increasing the length of the maximal reduction path. The same result can be rephrased here.

The terminology about right and left rules of $\mathbf{G}_{\text{LA}}^{\text{T}}$ adapts to \mathbf{G}_{LA} obviously.

8.3. Proving Adequacy

We need a translation $(.)^P$ from Λ_{LA}^T to G_{LA} . A better understanding of $(.)^P$ comes from a simpler version $(.)^\bullet$ of it. We shall use $(.)^P$ because $(.)^\bullet$ cannot work.

The function $(.)^\bullet$ behaves like the embedding of the *natural deduction* of Intuitionistic Logic into its *sequent calculus*. We mean that $(.)^\bullet$ translates:

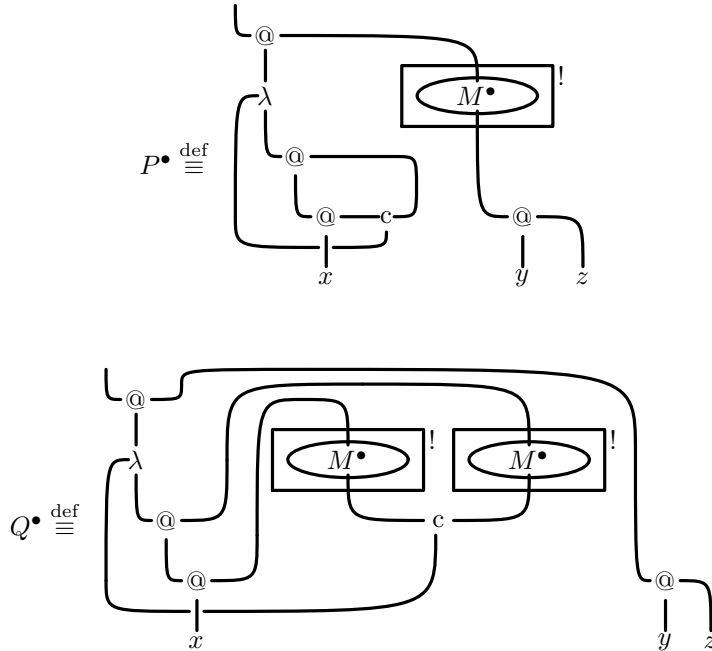
- every variable of Λ_{LA}^T into an axiom of G_{LA} ;
- every λ -abstraction $\lambda\chi.M$ of Λ_{LA}^T into the right-rule of G_{LA} . If, however, χ does not occur as a free variable of M , a weakening rule of G_{LA} must be used first;
- every term of Λ_{LA}^T , which encodes an elimination rule of \vdash_T , into (a suitable) composition of both a left-rule of G_{LA} , and some cut-rules. The way this works should be almost obvious for those acquainted with the representation of the elimination rules of the natural deduction of Intuitionistic Logic into derivations of its sequent calculus [6];
- $(\text{let } X = N \text{ in } M)^\bullet \stackrel{\text{def}}{=} ((\lambda X.M)N)^\bullet$.

Let us see how $(.)^\bullet$ works on an example. If we apply $(.)^\bullet$ to:

$$P \stackrel{\text{def}}{=} (\lambda X.x \ X \ X)!(M)[y \ z/\chi]$$

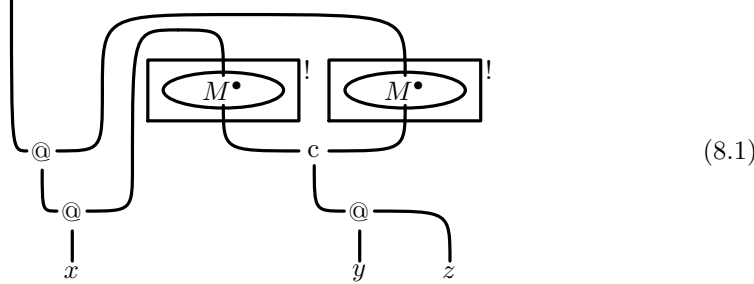
$$Q \stackrel{\text{def}}{=} (\lambda Y.x!(M)[^Y/\chi]!(M)[^Y/\chi])(y \ z) ,$$

we get the two graphs:



Remark that P^\bullet , and Q^\bullet are drawn as derivations of G_{LA}^T , where the types are erased. May be that, for a more comfortable read back of the terms from the graphs here above, one could draw the graphs with the more usual tree-like form.

The two translation examples here above allow us to see why $(\cdot)^\bullet$ is not enough to prove the Poly-step Adequacy for Λ_{LA}^T . We can observe that P reduces to Q in a single \rightsquigarrow -step, and that Q is normal. However, P^\bullet evaluates to:



which is not quite what we want: Q^\bullet is not the graph that P^\bullet reduces to because the leftmost uppermost nodes λ and $@$ of Q^\bullet have not reduced, and they *cannot* do it.

The problem is that Λ_{LA}^T does not have explicit contraction, while G_{LA} has. Any contraction node of G_{LA} determines whether a graph can be duplicated, or not. In Λ_{LA}^T , the same effect is obtained with two distinguished sets of variables. So, the situation where a contraction node of G_{LA} gets stuck because it cannot duplicate a sub-graph G , can correspond, like in the example above, to a term where some λ -abstractions cannot annihilate with the corresponding application. This because the argument of the application cannot be duplicated during the substitution. In fact, there is a whole class of terms not allowing to prove the adequacy of \succ^* to \rightsquigarrow through $(\cdot)^\bullet$. For some examples, let $P \stackrel{\text{def}}{=} \lambda x y z. x y z$, take the terms:

$$\begin{aligned}
 &!(P X X)[!^{(M)[^N/x]}/X] \\
 &!(P X X)[!^{(M)[^N/x]}[!^{(Q)/x'}/X] \\
 &\S(P X X)[!^{(M)[^N/x]}/X] \\
 &\S(P X X)[!^{(M)[^N/x]}[!^{(Q)/x'}/X] \\
 &\S(P X X)[\S^{(M)[^N/x]}/X] \\
 &\S(P X X)[\S^{(M)[^N/x]}[\dots^{(Q)/x'\dots}]/X]
 \end{aligned}$$

and, finally, reduce and translate them.

The translation $(\cdot)^p$ must be “smarter” than $(\cdot)^\bullet$. In particular, it must be sensitive to the “reduction history” of the term it is applied to.

This is achieved as follows.

Define a function such that, for any given $M \in \Lambda_{LA}^T$, it labels every λ , $!$ -box, \S -box, and application nodes of M . Each label must be *unique*. In our first example, P would become:

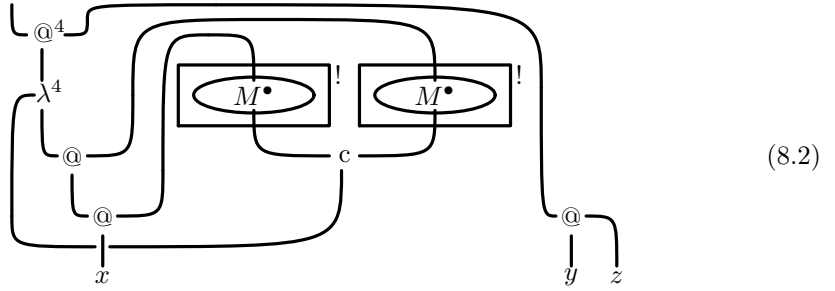
$$P \stackrel{\text{def}}{=} ((\lambda^1 X. x X X)!^3(M)[^y z/x])^2 ,$$

where 2 labels the application of the λ -abstraction to the !-box, and 3 the !-box itself. Then, modify \rightsquigarrow to identify the labels of those nodes that do not annihilate each other, to prevent the substitution of a term that cannot be duplicated for an exponential variable. For example, update \rightsquigarrow so that:

$$P \stackrel{\text{def}}{=} ((\lambda^1 X.x X X)!^3(M)[y z/x])^2 \rightsquigarrow ((\lambda^4 Y.x!^3(M)[Y/x]!^3(M)[Y/x])(y z))^4 \stackrel{\text{def}}{=} Q ,$$

where 1, and 2 are identified in 4 because xy cannot be duplicated. On the contrary, 3 occurs twice because it labels a !-box just duplicated.

Now, the behavior of $(.)^p$ can be described in two steps. Firstly, for any labeled term M , $(M)^p$ yields the same result, say G_M , as $(M)^\bullet$, with the proviso that the labels of M are preserved in G_M . For example, the first step of Q^p yields:



The second step of $(M)^p$ reduces the pairs $\langle \lambda^k, @^k \rangle$ of G_M . So, the second step of Q^p let (8.2) here above into (8.1).

The changes on \rightsquigarrow to correctly manipulate the labels of Λ_{LA}^T are:

$$\begin{aligned}
((\lambda^i X.M)!^j(N)[P/x])^j &\triangleright_{\beta_5} ((\lambda^k Y.M\{!(N)[Y/x]\downarrow_X\})P)^k \\
&\quad \text{if } P \notin \text{!-Term-Variables} \\
!^i(M)[!^j!(N)[P/x]/X] &\triangleright_{!!_5} !^k(M\{!(N)[Y/x]\downarrow_X\})[!^k P/Y] \\
!^i(M)[!^j!(N)[P/x][Q/x']]/X] &\triangleright_{!!_6} !^k(M\{!(N)[Y/x]\downarrow_X\})[!^k(P)[Q/x']/Y] \\
\S^i(M)[\dots !^j!(N)[P/x]/X_i \dots] &\triangleright_{\S!_5} \S^k(M\{!(N)[Y/x]\downarrow_{X_i}\})[\dots !^k P/Y \dots] \\
\S^i(M)[\dots !^j!(N)[P/x][Q/x']]/X_i \dots] &\triangleright_{\S!_6} \\
&\quad \S^k(M\{!(N)[Y/x]\downarrow_{X_i}\})[\dots !^k(P)[Q/x']/Y \dots] \\
\S^i(M)[\dots \S^j!(N)[P/x]/X_i \dots] &\triangleright_{\S\S_5} \S^k(M\{!(N)[Y/x]\downarrow_{X_i}\})[\dots \S^k P/Y \dots] \\
\S^i(M)[\dots \S^j!(N)[P/x][\dots Q/x' \dots]/X_i \dots] &\triangleright_{\S\S_6} \\
&\quad \S^k(M\{!(N)[Y/x]\downarrow_{X_i}\})[\dots \dots \S^k(P)[\dots Q/x' \dots]/Y \dots \dots]
\end{aligned}$$

with the proviso that fresh numbers must be used to identify labels. The rules of \rightsquigarrow not listed here above simply have the obvious effects on the indexes, by copying them from redexes to reducts. Of course, if a !-box with label i is duplicated, the two copies have index i . It should be clear, however, that, if the label of one of the two copies becomes k , because it is identified with some other label, then the label i of the untouched copy is unchanged.

At this point, Adequacy can be proved by checking that diagrams like:

$$\begin{array}{ccc} M & \rightsquigarrow & N \\ \downarrow & & \downarrow \\ M^p & \succ^* & N^p \end{array}$$

commute, using the labeled terms and the updated \rightsquigarrow on them.

8.4. Proving $\Lambda_{\text{LA}}^{\text{T}}$ being Poly-step.

This subsection hints how proving Theorem 3.

The point is to show the existence of a *canonical strategy* \rightsquigarrow_p^* for \rightsquigarrow such that every single reduction step $M \rightsquigarrow_p N$ of \rightsquigarrow_p^* between M and N can be simulated only by steps of \succ_p^* . Recall once more, that \succ_p^* is the strategy that evaluates any $G \in \mathbb{G}_{\text{LA}}$ to its normal form in **P-TIME** with respect to the dimension of G .

Let us define \rightsquigarrow_p^* .

Take $M \in \Lambda_{\text{LA}}^{\text{T}}$. We say that N is at depth $i \geq 0$ in M , and we write N^i , if it is in the body of i nested boxes of M . The notion of depth can obviously be used also for the redexes. Let us classify the redexes in two sets. The β -redexes belong to the β -group or to the **let**-group. The box-redexes are all the others. Now, assume M having at most d nested boxes. For any $0 \leq i \leq d$, the i^{th} reduction round reduces all the β -redexes N^i and all the box-redexes N^{i-1} , in any order. Finally, the *poly-step reduction strategy* \rightsquigarrow_p^* is the sequence of reduction rounds which starts from the 1st and stops (at most) at the d^{th} .

At this point, Adequacy can be proved by checking that diagrams like:

$$\begin{array}{ccc} M & \rightsquigarrow_p & N \\ \downarrow & & \downarrow \\ M^p & \succ_p^* & N^p \end{array}$$

commute, using the labeled terms and the updated \rightsquigarrow on them.

9. The Type Inference

This section is about a type inference algorithm \vdash_{TI} for $\Lambda_{\text{LA}}^{\text{T}}$, built on a type inference for ML [2] The details on \vdash_{TI} that we shall skip can be found in [10].

The main component of \vdash_{TI} is a unification algorithm \vdash_{U} which must be applied to sets of pairs of *types*. Let $\tau_1 \dots \tau_n$, and $\tau'_1 \dots \tau'_n$ be two (equally long) tuples of *types*. Then, $\vdash_{\text{U}} \{\tau_1 = \tau'_1, \dots, \tau_n = \tau'_n\}$ denotes the application of the unification algorithm \vdash_{U} to the given set of pairs. The result can be twofold. If, for every $1 \leq i \leq n$, it does not exist any substitution U such that $U\tau_i$ and $U\tau'_i$ are the same type, then the result is **failure**. We write:

$$\vdash_{\text{U}} \{\tau_1 = \tau'_1, \dots, \tau_n = \tau'_n\} \Rightarrow \mathbf{failure} .$$

Otherwise, if such a U exists, it is the *most general unifier* of the given set of pairs, and we write:

$$\vdash_{\text{U}} \{\tau_1 = \tau'_1, \dots, \tau_n = \tau'_n\} \Rightarrow U .$$

The unification is defined inductively on the syntax of the types in the obvious way. However, it is worth remarking the behavior of \vdash_{U} on the two kinds of variable

types. The elements of **Type-Variables** can only be unified with linear types; those in **!-Type-Variables** can only be unified with exponential types. For example, \vdash_U yields **failure** if one of the following singletons is a sub-set of its argument:

$$\begin{aligned} &\{\alpha = \delta\} \\ &\{\alpha = !\tau\} \\ &\{\delta = \alpha\} \\ &\{\delta = \tau \multimap \tau'\} \\ &\{\delta = \S\tau\} . \end{aligned}$$

9.1. The Algorithm

The type inference algorithm \vdash_{TI} derives two kinds of judgments. One of them is $\vdash_{\text{TI}} \Gamma; M; \tau \Rightarrow S$, and the other is $\vdash_{\text{TI}} \Gamma; M; \tau \Rightarrow \mathbf{failure}$, where Γ is a set of assumptions, M is a functional term, τ is a type, and S is a substitution. The first judgment corresponds to the success of the algorithm. The second to its failure.

The meaning of the succeeding judgments is given by the following theorem:

Theorem 4 (Correctness) *If $\vdash_{\text{TI}} \Gamma; M; \tau \Rightarrow S$, then $S\Gamma \vdash_{\text{T}} M : S\tau$.*

Namely, if the type inference algorithm succeeds, its substitution S serves to derive a type from an assumption for the given term. Some comments about the proof of Correctness are in Subsection 9.2.

The set of rules in Natural Semantics [7] defining \vdash_{TI} is here below, and is divided into two parts. One contains the rules proving the judgments

$$\Gamma; M; \tau \Rightarrow S$$

where τ can be unified with a linear type. The other has rules such that τ can be unified with exponential types. The alternatives must be clearly mutually exclusive, so the set of rules defines a deterministic algorithm. Call *fresh* newly generated variables. The rules of the type inference are in Figure 21, and 22.

We simulate \vdash_{TI} on:

$$\S(x)[y \ z/x;] , \tag{9.1}$$

starting from the set of assumptions $\{y : \alpha, z : \beta\}$. We shall get a better understanding about why we split the interface of \S -boxes. The main steps of the simulation are:

$$\begin{aligned} &\vdash_{\text{TI}} y : \alpha, z : \beta; \S(x)[y \ z/x;]; \beta_1 \\ &\text{call } (\S_I) \quad \gamma \text{ fresh} \\ &\quad \vdash_U \{\beta_1 = \S\gamma\} \Rightarrow \{\S\gamma \downarrow_{\beta_1}\} \equiv U \\ &\quad \alpha_1 \text{ fresh} \\ &\quad \vdash_{\text{TI}} y : \alpha, z : \beta; y \ z; !\alpha_1 \\ &\quad \text{call } (\multimap_{E_l}) \quad \alpha_2 \text{ fresh} \\ &\quad \quad \vdash_{\text{TI}} y : \alpha, z : \beta; y; \alpha_2 \multimap !\alpha_1 \Rightarrow \{\alpha_2 \multimap !\alpha_1 \downarrow_{\alpha}\} \equiv S_y \\ &\quad \quad \vdash_{\text{TI}} y : \alpha_2 \multimap !\alpha_1, z : \beta; z; \alpha_2 \Rightarrow \{\alpha_2 \downarrow_{\beta}\} \equiv S_z \\ &\quad \text{return from } (\multimap_{E_l}) \Rightarrow \{\alpha_2 \multimap !\alpha_1 /_{\alpha} \alpha_2 /_{\beta}\} \equiv S_z S_y \end{aligned}$$

$$\begin{array}{c}
(Ax_l) \frac{\vdash_U \{\tau = \rho\} \Rightarrow U}{\vdash_{\text{TI}} \Gamma, x : \tau; x; \rho \Rightarrow U} \\
\\
(Ax_e) \frac{\begin{array}{c} n \geq 0 \\ v_i \text{ fresh and} \\ v_i \text{ linear iff } \varsigma_i \text{ linear} \\ \text{with } 1 \leq i \leq n \\ \vdash_U \{\{v_1 \downarrow_{\varsigma_1} \cdots v_n \downarrow_{\varsigma_n}\} \tau = \rho\} \Rightarrow U \end{array}}{\vdash_{\text{TI}} \Gamma, X : \forall \varsigma_1 \dots \varsigma_n. \tau; X; \rho \Rightarrow U} \\
\\
(-\circ_{E_l}) \frac{\begin{array}{c} \alpha \text{ fresh} \\ \vdash_{\text{TI}} \Gamma; M; \alpha \multimap \tau \Rightarrow S_M \\ \vdash_{\text{TI}} S_M \Gamma; N; S_M \alpha \Rightarrow S_N \end{array}}{\vdash_{\text{TI}} \Gamma; M N; \tau \Rightarrow S_N S_M} \\
(-\circ_{E_e}) \frac{\begin{array}{c} \delta \text{ fresh} \\ \vdash_{\text{TI}} \Gamma; M; \delta \multimap \tau \Rightarrow S_M \\ \vdash_{\text{TI}} S_M \Gamma; N; S_M \delta \Rightarrow S_N \end{array}}{\vdash_{\text{TI}} \Gamma; M N; \tau \Rightarrow S_N S_M} \\
\\
(-\circ_{I_l}) \frac{\begin{array}{c} \alpha, \varsigma \text{ fresh and} \\ \varsigma \text{ linear iff } \chi \text{ linear} \\ \vdash_U \{\tau = \varsigma \multimap \alpha\} \Rightarrow U \\ \vdash_{\text{TI}} U \Gamma_{\{\chi\}}, \chi : U \varsigma; M; U \alpha \Rightarrow S_M \end{array}}{\vdash_{\text{TI}} \Gamma; \lambda \chi. M; \tau \Rightarrow S_M U} \\
(-\circ_{I_e}) \frac{\begin{array}{c} \delta, \varsigma \text{ fresh and} \\ \varsigma \text{ linear iff } \chi \text{ linear} \\ \vdash_U \{\tau = \varsigma \multimap \delta\} \Rightarrow U \\ \vdash_{\text{TI}} U \Gamma_{\{\chi\}}, \chi : U \varsigma; M; U \delta \Rightarrow S_M \end{array}}{\vdash_{\text{TI}} \Gamma; \lambda \chi. M; \tau \Rightarrow S_M U} \\
\\
(!\theta_l) \frac{\begin{array}{c} \alpha \text{ fresh} \\ \vdash_U \{\tau = !\alpha\} \Rightarrow U \\ \vdash_{\text{TI}} \emptyset; M; U \alpha \Rightarrow S_M \end{array}}{\vdash_{\text{TI}} \Gamma; !M; \tau \Rightarrow S_M U} \\
(!\theta_e) \frac{\begin{array}{c} \delta \text{ fresh} \\ \vdash_U \{\tau = !\delta\} \Rightarrow U \\ \vdash_{\text{TI}} \emptyset; M; U \delta \Rightarrow S_M \end{array}}{\vdash_{\text{TI}} \Gamma; !M; \tau \Rightarrow S_M U} \\
\\
(\S\theta_l) \frac{\begin{array}{c} \alpha \text{ fresh} \\ \vdash_U \{\tau = \S\alpha\} \Rightarrow U \\ \vdash_{\text{TI}} \emptyset; M; U \alpha \Rightarrow S_M \end{array}}{\vdash_{\text{TI}} \Gamma; \S M; \tau \Rightarrow S_M U} \\
(\S\theta_e) \frac{\begin{array}{c} \delta \text{ fresh} \\ \vdash_U \{\tau = \S\delta\} \Rightarrow U \\ \vdash_{\text{TI}} \emptyset; M; U \delta \Rightarrow S_M \end{array}}{\vdash_{\text{TI}} \Gamma; \S M; \tau \Rightarrow S_M U} \\
\\
(!_l) \frac{\begin{array}{c} \alpha \text{ fresh} \\ \vdash_U \{\tau = !\alpha\} \Rightarrow U \\ \varsigma \text{ fresh and} \\ \varsigma \text{ linear iff } \chi \text{ linear} \\ \vdash_{\text{TI}} U \Gamma; N; !\varsigma \Rightarrow S_N \\ \vdash_{\text{TI}} \chi : S_N \varsigma; M; S_N U \alpha \Rightarrow S_M \end{array}}{\vdash_{\text{TI}} \Gamma; !(M)[^N/\chi]; \tau \Rightarrow S_M S_N U} \\
(!_e) \frac{\begin{array}{c} \delta \text{ fresh} \\ \vdash_U \{\tau = !\delta\} \Rightarrow U \\ \varsigma \text{ fresh and} \\ \varsigma \text{ linear iff } \chi \text{ linear} \\ \vdash_{\text{TI}} U \Gamma; N; !\varsigma \Rightarrow S_N \\ \vdash_{\text{TI}} \chi : S_N \varsigma; M; S_N U \delta \Rightarrow S_M \end{array}}{\vdash_{\text{TI}} \Gamma; !(M)[^N/\chi]; \tau \Rightarrow S_M S_N U} \\
\\
(\text{let}) \frac{\begin{array}{c} \delta \text{ fresh} \\ \vdash_{\text{TI}} \Gamma; M; \delta \Rightarrow S_M \\ \vdash_{\text{TI}} X : (\forall S_M \Gamma. S_M \delta), S_M \Gamma_{\{X\}}; N; S_M \tau \Rightarrow S_N \end{array}}{\vdash_{\text{TI}} \Gamma; \text{let } X = M \text{ in } N; \tau \Rightarrow S_N S_M}
\end{array}$$

Fig. 21. The Type Inference: first part.

$$\begin{array}{l}
\alpha \text{ fresh} \\
\vdash_U \{\tau = \S\alpha\} \Rightarrow U \\
\varsigma_i \text{ fresh and } \varsigma_i \text{ linear iff } \chi_i \text{ linear with } 1 \leq i \leq m \\
v_j \text{ fresh and } v_j \text{ linear iff } \chi'_j \text{ linear with } 1 \leq j \leq n \\
\vdash_{\text{TI}} \Gamma; M_1; !\varsigma_1 \Rightarrow S_1 \\
\vdots \\
\vdash_{\text{TI}} S_{i-1} \cdots S_1 \Gamma; M_i; !\varsigma_i \Rightarrow S_i \\
\vdots \\
\vdash_{\text{TI}} S_{m-1} \cdots S_1 \Gamma; M_m; !\varsigma_m \Rightarrow S_m \\
\vdash_{\text{TI}} S_m \cdots S_1 \Gamma; N_1; \S v_1 \Rightarrow R_1 \\
\vdots \\
\vdash_{\text{TI}} R_{j-1} \cdots R_1 S_m \cdots S_1 \Gamma; N_j; \S v_j \Rightarrow R_j \\
\vdots \\
\vdash_{\text{TI}} R_{n-1} \cdots R_1 S_m \cdots S_1 \Gamma; N_n; \S v_n \Rightarrow R_n \\
\Gamma' \stackrel{\text{def}}{=} \{\chi_i : R_n \cdots R_1 S_m \cdots S_i \varsigma_i \mid 1 \leq i \leq m\} \cup \\
\{\chi'_j : R_n \cdots R_j v_j \mid 1 \leq j \leq n\} \\
\vdash_{\text{TI}} \Gamma'; M; R_n \cdots R_1 S_m \cdots S_1 U \alpha \Rightarrow S_M \\
\hline
(\S i) \frac{}{\vdash_{\text{TI}} \Gamma; \S(M) [^{M_1/\chi_1} \cdots ^{M_m/\chi_m}; ^{N_1/\chi'_1} \cdots ^{N_n/\chi'_n}]; \tau \Rightarrow S_M R_n \cdots R_1 S_m \cdots S_1 U}
\end{array}$$

$$\begin{array}{l}
\delta \text{ fresh} \\
\vdash_U \{\tau = \S\delta\} \Rightarrow U \\
\varsigma_i \text{ fresh and } \varsigma_i \text{ linear iff } \chi_i \text{ linear with } 1 \leq i \leq m \\
v_j \text{ fresh and } v_j \text{ linear iff } \chi'_j \text{ linear with } 1 \leq j \leq n \\
\vdash_{\text{TI}} \Gamma; M_1; !\varsigma_1 \Rightarrow S_1 \\
\vdots \\
\vdash_{\text{TI}} S_{i-1} \cdots S_1 \Gamma; M_i; !\varsigma_i \Rightarrow S_i \\
\vdots \\
\vdash_{\text{TI}} S_{m-1} \cdots S_1 \Gamma; M_m; !\varsigma_m \Rightarrow S_m \\
\vdash_{\text{TI}} S_m \cdots S_1 \Gamma; N_1; \S v_1 \Rightarrow R_1 \\
\vdots \\
\vdash_{\text{TI}} R_{j-1} \cdots R_1 S_m \cdots S_1 \Gamma; N_j; \S v_j \Rightarrow R_j \\
\vdots \\
\vdash_{\text{TI}} R_{n-1} \cdots R_1 S_m \cdots S_1 \Gamma; N_n; \S v_n \Rightarrow R_n \\
\Gamma' \stackrel{\text{def}}{=} \{\chi_i : R_n \cdots R_1 S_m \cdots S_i \varsigma_i \mid 1 \leq i \leq m\} \cup \\
\{\chi'_j : R_n \cdots R_j v_j \mid 1 \leq j \leq n\} \\
\vdash_{\text{TI}} \Gamma'; M; R_n \cdots R_1 S_m \cdots S_1 U \delta \Rightarrow S_M \\
\hline
(\S e) \frac{}{\vdash_{\text{TI}} \Gamma; \S(M) [^{M_1/\chi_1} \cdots ^{M_m/\chi_m}; ^{N_1/\chi'_1} \cdots ^{N_n/\chi'_n}]; \tau \Rightarrow S_M R_n \cdots R_1 S_m \cdots S_1 U}
\end{array}$$

Fig. 22. The Type Inference: second part.

Call $\vdash_{\overline{\mathbb{T}}}$ the type assignment $\vdash_{\mathbb{T}}$ without the rules for the boxes, and $\vdash_{\overline{\mathbb{T}\mathbb{I}}}$ the corresponding sub-system of $\vdash_{\mathbb{T}\mathbb{I}}$. Observe that $\vdash_{\overline{\mathbb{T}}}$ is an adaptation of the type assignment $\vdash_{\overline{\mathbb{T}}}^{\lambda}$ for the simply typed λ -Calculus to the types with linear implication, built on two kinds of type variables. To recover $\vdash_{\overline{\mathbb{T}\mathbb{I}}}^{\lambda}$ from $\vdash_{\overline{\mathbb{T}\mathbb{I}}}$ just replace \multimap by the usual implication, and forget the differences between linear and exponential for both the term and the type variables. So, if we show that $\vdash_{\overline{\mathbb{T}}}$ enjoys the properties of $\vdash_{\overline{\mathbb{T}}}^{\lambda}$, used to prove Correctness and Completeness for $\vdash_{\overline{\mathbb{T}\mathbb{I}}}^{\lambda}$, then we know that Correctness and Completeness hold for $\vdash_{\overline{\mathbb{T}\mathbb{I}}}$ as well.

Part of the properties we refer to describe the behavior of the type substitutions. One of the main properties is Lemma 1, proved in Section 4 for the whole $\vdash_{\mathbb{T}}$, and that we recall here:

Lemma (Substitution distributivity) If $\Gamma \vdash_{\mathbb{T}} M : \tau$, then $S\Gamma \vdash_{\mathbb{T}} M : S\tau$, for any S .

Further fundamental properties are Lemma 3, and Lemma 4 here below, which can be proved by structural induction on M :

Lemma 4

$$(WE) \frac{\Gamma \vdash M : \tau \quad \text{domain}(\Gamma') \cap \text{domain}(\Gamma) = \emptyset \quad \text{FV}(M) \subseteq \mathbb{V} \subseteq \text{domain}(\Gamma)}{\{\chi : \Gamma(\chi) \mid \chi \in \mathbb{V}\}, \Gamma' \vdash M : \tau}$$

is admissible in $\vdash_{\mathbb{T}}$.

Rule (WE) simultaneously weakens, and extends Γ . Together with the Substitution distributivity, it serves for proving Correctness.

Correctness, and Completeness require some further properties about the composition of substitutions, and the distributivity of the substitutions over the type schemes. These properties certainly hold on our system as they relate to the syntax of both the types, and the substitution themselves, and do not involve the type assignment at all. So, we skip them.

The proof of:

Theorem (Correctness) If $\vdash_{\mathbb{T}\mathbb{I}} \Gamma; M; \tau \Rightarrow S$, then $S\Gamma \vdash_{\mathbb{T}} M : S\tau$, proceeds by induction on M and it is obvious.

We shall develop the full proof of one inductive case of Completeness whose statement we recall here:

Theorem (Completeness) If $R\Gamma \vdash_{\mathbb{T}} M : R\tau$, then $\vdash_{\mathbb{T}\mathbb{I}} \Gamma; M; \tau \Rightarrow S$, and there is \bar{S} such that $R = \bar{S}S$ on all the variables not in $\text{new}(\vdash_{\mathbb{T}\mathbb{I}} \Gamma; M; \tau)$, abbreviated as $R =_{\text{new}(\vdash_{\mathbb{T}\mathbb{I}} \Gamma; M; \tau)} \bar{S}S$.

In particular, we prove Completeness when M is a non closed !-box. The case where M is a closed !-box is simpler. When M is a \S -box the proof gets a bit more complicate, but analogous to the case we are going to develop in full detail. This because the type inference rules for the \S -boxes are generalizations of the ones for the !-boxes.

Let us see the details of the proof.

The instance of Completeness we want to prove is:

$$\text{If } R\Gamma \vdash_{\mathbb{T}} !(M)[^N/\chi] : R\rho \text{ ,} \tag{9.4}$$

$$\text{then } \vdash_{\mathbb{T}\mathbb{I}} \Gamma; !(M)[^N/\chi]; \rho \Rightarrow S \tag{9.5}$$

$$\text{and } \exists \bar{S}. R =_{\text{new}(\vdash_{\mathbb{T}\mathbb{I}} \Gamma; !(M)[^N/\chi]; \rho)} \bar{S}S \text{ .} \tag{9.6}$$

- From the definition of \vdash_T , (9.4) tells that:

$$R\rho = !\tau , \quad (9.7)$$

for some τ , that we assume to be linear. With an exponential τ , the proof would be identical, but it would involve the type inference rule $(!_e)$ in place of $(!_l)$, as we are going to do in what follows.

- Take:

$$R' \stackrel{\text{def}}{=} \{\tau \downarrow_{\beta}\} \cup R , \quad (9.8)$$

with β generated by the call of \vdash_{TI} on $(!_l)$.

- From (9.8) we get:

$$\begin{aligned} R'\rho & \stackrel{\beta \text{ fresh}}{=} R\rho \\ & \stackrel{(9.7)}{=} !\tau \\ & \stackrel{\text{def}}{=} R'!\beta . \end{aligned} \quad (9.9)$$

Namely, R' unifies ρ and $!\beta$. So, there exists the most general unifier U produced by the $(!_l)$ rule:

$$\vdash_{\cup} \{\rho = !\beta\} \Rightarrow U \text{ and} \quad (9.10)$$

$$\exists \bar{U}. R' = \bar{U}U . \quad (9.11)$$

- Thanks to β fresh, using the definition of R' , (9.10), and (9.11) we rewrite (9.4):

$$\bar{U}U\Gamma \vdash_T!(M)[^N/\chi] : \bar{U}U!\beta . \quad (9.12)$$

- By (9.12) and the definition of \vdash_T :

$$\bar{U}U\Gamma \vdash_T N : !\rho' \quad (9.13)$$

$$\chi : \rho' \vdash_T M : \bar{U}U!\beta , \quad (9.14)$$

for some ρ' .

- Take:

$$\bar{U}' \stackrel{\text{def}}{=} \bar{U} \cup \{\rho' \downarrow_{\varsigma}\} , \quad (9.15)$$

with ς generated by the call of \vdash_{TI} on $(!_l)$. The definition here above, implies:

$$\bar{U}'U\Gamma = \bar{U}U\Gamma \quad (9.16)$$

$$\bar{U}'\varsigma = \rho' . \quad (9.17)$$

- Using (9.16) and (9.17), rewrite (9.13):

$$\bar{U}'U\Gamma \vdash_T N : \bar{U}'!\varsigma . \quad (9.18)$$

- By induction on (9.18), we get:

$$\vdash_{\text{TI}} U\Gamma; N; !\varsigma \Rightarrow S_N \quad (9.19)$$

$$\exists \bar{S}_N. \bar{U}' =_{\text{new}(\vdash_{\text{TI}} U\Gamma; N; !\varsigma)} \bar{S}_N S_N . \quad (9.20)$$

- Using (9.17), rewrite (9.14):

$$\vdash_{\text{TI}} \chi : \bar{U}'\varsigma \vdash_{\text{T}} M : \bar{U}'U!\beta . \quad (9.21)$$

- Observe that (9.20) implies:

$$\bar{U}'\varsigma = \bar{S}_N S_N \varsigma \quad (9.22)$$

$$\bar{U}'U!\beta = \bar{S}_N S_N U!\beta . \quad (9.23)$$

Namely, the inductive hypothesis is vital to get the equality between \bar{U}' and $\bar{S}_N S_N$ on β and ς which are not among the variables generated by $\vdash_{\text{TI}} U\Gamma; N; !\varsigma$.

- With (9.22) and (9.23) rewrite (9.21):

$$\vdash_{\text{TI}} \chi : \bar{S}_N S_N \varsigma \vdash_{\text{T}} M : \bar{S}_N S_N U!\beta . \quad (9.24)$$

- By induction on (9.24):

$$\vdash_{\text{TI}} \chi : S_N \varsigma; M; S_N U!\beta \Rightarrow S_M \quad (9.25)$$

$$\exists \bar{S}_M. \bar{S}_N =_{\text{new}(\vdash_{\text{TI}} \chi : S_N \varsigma; M; S_N U!\beta)} \bar{S}_M S_M . \quad (9.26)$$

- Use (9.8), (9.11), ς fresh with respect to \bar{U} and (9.15), (9.20), and (9.26) for writing:

$$R \cup \{\tau \downarrow_{\beta}\} = \bar{S}_M S_M S_N U . \quad (9.27)$$

- The statements (9.5) and (9.6) we want to prove require that $R \cup \{\tau \downarrow_{\beta}\}$ and $\bar{S}_M S_M S_N U$ must be applied to ρ and Γ not containing neither β nor ς . We can conclude:

$$R = \bar{S}_M S_M S_N U , \quad (9.28)$$

by restricting ourselves to the type variables different from β and ς . By taking \bar{S}_M as \bar{S} , and $S_M S_N U$ as S in (9.6) we have done.

10. Conclusions

This work is an extended version of both [16, 17].

It is a first proposal for using **ILAL** as a programming language. This is accomplished by introducing an untyped functional language Λ_{LA} . Λ_{LA} has a sub-set $\Lambda_{\text{LA}}^{\text{T}}$ that can be typed automatically by a type inference algorithm. The types for the terms of $\Lambda_{\text{LA}}^{\text{T}}$ are polymorphic formulas of **ILAL**.

The main properties of Λ_{LA}^T are related to the functions it can represent, and to the complexity of its rewriting system. Every term of Λ_{LA}^T represents a **P-TIME** algorithm. Moreover, there is a poly-step reduction strategy for it which normalizes any term M in a number of steps bound by a polynomial in the dimension of M .

We can think of Λ_{LA}^T as an *experimental* paradigmatic programming language to deal with algorithms with both a predictable and, at least in principle, low computational complexity.

However, Λ_{LA} still needs improvements.

Its syntax is still quite heavy. The main goal is to eliminate the interfaces from boxes.

Moreover, the completeness of Λ_{LA}^T is still open. This means that we have not yet proved that all the **P-TIME** Turing machines can be encoded in Λ_{LA}^T . This seems very unlikely because of the limited polymorphism allowed by the types for Λ_{LA}^T . The challenge is to extend a bit the expressiveness of the type system \vdash_T , in order to get the completeness, but without losing the decidability of the type inference.

Once obtained the completeness, it will be certainly interesting comparing the “**P-TIME** programming discipline” of Λ_{LA}^T with respect to that of other languages, introduced in [8, 11] for programming **P-TIME** algorithms.

Acknowledgments. This work has been developed thanks to some useful discussions with Andrea Asperti, and Yves Lafont. Also Stefano Guerrini, and Tom Kranz gave some help. An anonymous referee suggested how to make the paper more readable. The work has been financially supported by two projects: a TMR-Marie Curie Grant, contract n. ERBFMBICT972805, and by MURST *Progetto di ricerca cofinanziato: “Tecniche formali per la specifica, l’analisi, la verifica, la sintesi e la trasformazione di sistemi software”*. The work was submitted when the author was at *Institut de Mathématiques de Luminy in Marseille*.

References

1. A. Asperti. Light Affine Logic. In *Proceedings of Symposium on Logic in Computer Science LICS’98*, 1998.
2. L. Damas and R. Milner. Principal type-schemes for functional programs. In *Proceedings of 9th ACM Symposium on Principles of Programming Languages POPL’82*, January 1982.
3. R. Di Cosmo and D. Kesner. Strong normalization of explicit substitutions via cut elimination in proof nets (extended abstract). In *Proceedings of Twelfth Annual IEEE Symposium on Logic in Computer Science LICS’98*, pages 35 – 46, June – July 1997.
4. J.-Y. Girard. Light Linear Logic. *Information and Computation*, 143:175 – 204, 1998.
5. J.-Y. Girard, Y. Lafont, and L. Regnier. *Advances in Linear Logic*, volume 222 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, 1995.
6. J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Cambridge University Press, 1989.

7. C.A. Gunter. *Semantics of Programming Languages: Structures and Techniques*. Foundations of Computing. The MIT Press, 1992.
8. M. Hoffmann. A mixed modal/linear lambda calculus with applications to Bellantoni-Cook safe recursion. In *Proceedings of Computer Science Logic 1997 (CSL'97)*, Aarhus, Denmark, volume To appear, 1997.
9. W.A. Howard. *To H.B. Curry: Essay on Combinatory Logic, Lambda Calculus and Formalisms*, chapter The formulae-as-type notion of construction, pages 479–490. J.R.Hindley and J.P. Seldin editors, Academic press, 1980.
10. Oukseh Lee and Kwangkeun Yi. Proofs about a folklore let-polymorphic type inference algorithm. *Transactions on Programming Languages and Systems*, 20(4):707 – 723, 1998.
11. D. Leivant and J.-Y. Marion. Lambda calculus characterizations of poly-time. *Fundamenta Informaticae*, 19:167 – 184, 1993.
12. R. Milner, M. Tofte, and R. Harper. *The Definition of Standard ML*. The MIT Press, 1990.
13. J.C. Mitchell. Polymorphic type inference and containment. *Information and Computation*, 76:211 – 249, 1988.
14. G. Plotkin. Call-by-name, call-by-value and the λ -calculus. *Theoretical Computer Science*, 1:125 – 159, 1975.
15. S. Ronchi della Rocca and L. Roversi. Lambda calculus and intuitionistic linear logic. *Studia Logica*, 57:417 – 448, 1997.
16. L. Roversi. Concrete syntax for intuitionistic light affine logic with polymorphic type assignment. In *Sixth Italian Conference on Theoretical Computer Science*, pages 24 – 36. World Scientific, 9 – 11 November (Prato – Italy) 1998.
17. L. Roversi. A polymorphic language which is typable and poly-step. In *Advances in Computing Science – ASIAN'98*, volume LNCS 1538, pages 43 – 60. Springer-Verlag, 8 – 10 December (Manila – The Philippines) 1998.
18. L. Roversi. A P-Time completeness proof for light logics. In *Proceedings of Computer Science Logic 1999 (CSL'99) (Madrid – Spain)*, volume LNCS. Springer-Verlag, 20 – 25 September 1999.