

A Polymorphic Language which is Typable and Poly-step

Luca Roversi

Institute de Mathématiques de Luminy
UPR 9016 – 163, avenue de Luminy – Case 907
13288 Marseille Cedex 9 – France
roversi@iml.univ-mrs.fr
<http://www.iml.univ-mrs.fr>

Abstract. A functional language Λ_{LA} is given. A sub-set Λ_{LA}^T of Λ_{LA} is automatically typable. The types are formulas of Intuitionistic Light Affine Logic with polymorphism *à la ML*. Every term of Λ_{LA}^T can reduce to its normal form in, at most, poly-steps. Λ_{LA}^T can be used as a prototype of programming language for **P-TIME** algorithms.

1 Introduction

In [3], Girard introduced Light Linear Logic which *captures P-TIME*. This means that the cut-elimination process of Light Linear Logic terminates in polynomial time with respect to the dimension of any given derivation, and, vice versa, all **P-TIME** Turing machines can be encoded as data-types in Light Linear Logic.

Girard left as an open problem to find a concrete syntax for **ILLL**, namely for Intuitionistic Light Linear Logic. This paper introduces an untyped functional language Λ_{LA} which has a typable sub-set Λ_{LA}^T . The types for Λ_{LA}^T are formulas of **ILLL** with a polymorphism *à la ML*. The types can be inferred automatically.

Before introducing Λ_{LA}^T , it is worth recalling the main mechanism of **ILLL** to bound the cut-elimination complexity. The key point is avoiding the proliferation of the contraction rule

$$(Contraction) \frac{\Gamma, A, A \vdash B}{\Gamma, A \vdash B} ,$$

that can take place when eliminating the cuts in a derivation of Intuitionistic Logic. For example, let C_n be the *typable* λ -term $\overline{22} \dots$, which has $n \geq 2$ Church Numerals $\overline{2} \stackrel{\text{def}}{=} \lambda xy.x(xy)$ in it. The length of the left-most reduction of C_n to its normal form grows exponentially in n . This happens essentially because there are redexes that, once reduced, yield two residual redexes each, thus developing a reduction space with the form of a tree. Logically, this means that there are contraction rules that duplicate other contraction rules as effect of the cut-elimination. Figure 1 gives an intuitive pictorial representation of this.

The non-exponential proliferation of the c -nodes in **ILLL** is a consequence of two main aspects. First, in **ILLL**, a derivation can be duplicated only if enclosed

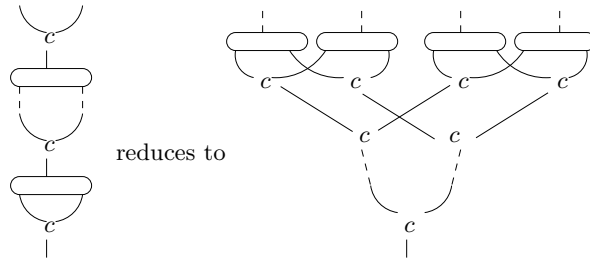
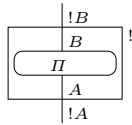
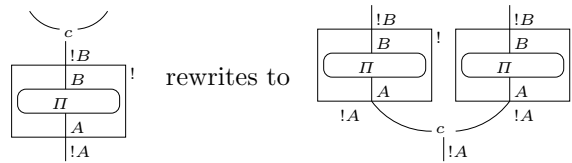


Fig. 1. From lists to trees of contractions

into a *region*, called $!$ -box. Second, any $!$ -box can derive a formula, from *at most a single assumption*. For example, let Π be a derivation of **ILLL**, proving B from *the single* assumption A . Then, the $!$ -box $!\Pi$, built on Π is:



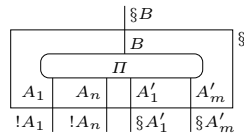
A cut-elimination step duplicating $!\Pi$ is such that:



The $!$ -box divides the space in two parts: one inside, and the one outside it. This means that, unlike Intuitionistic Logic, the c -node here above, which contracts $!A$, does not trivially extend any tree of c -nodes inside the $!$ -box: there is the $!$ -box border in between. In first approximation, this means that any exponentially growing tree of c -nodes, in a derivation Π of **ILLL**, is an (exponential) function of the maximal number of nested $!$ -boxes of Π . But, this is a detail...

The second feature of **ILLL** is that any $!$ -box has at most one assumption. This means that, if at some stage of the cut-elimination process, a $!$ -box has a tree of c -nodes in it, then every c -node was present somewhere, in the same $!$ -box, since the beginning of the cut-elimination. In this way, the non-exponential proliferation of c -nodes holds when composing the deductions of **ILLL**.

We complete the description of the novel part of **ILLL** by describing the behavior of a second region in it. If Π is a deduction proving B from two, possibly empty, sets of assumptions $A_1 \dots A_n$ and $A'_1 \dots A'_m$, then, the \S -box $\S\Pi$ with Π in it, is:



The \S -boxes increase the expressiveness of **ILLL**. Without \S -boxes, Church Numerals could not be encoded in **ILLL**. The presence of an arbitrary number of assumptions of a \S -box that can be contracted should not worry. The non-exponential proliferation of c -nodes is preserved because \S -boxes themselves can not be duplicated. Hence, the creation of trees with too many c -nodes below any \S -box is forbidden.

The computational behavior of the language Λ_{LA}^T incorporates the features here above of the cut-elimination of **ILLL**. The language Λ_{LA}^T , however, is the functional counterpart of a natural deduction for the sequent calculus that Asperti introduced in [1], where the original sequent calculus in [3] has greatly simplified. The point of [1] is to move from **ILLL** to its *affine* version **ILAL**, where unrestricted weakening is allowed.

Contributions. This work introduces an *untyped* functional language Λ_{LA} . It has a sub-set Λ_{LA}^T of *typable* elements. The types for Λ_{LA}^T are the formulas of a natural deduction that we introduce. The natural deduction proves a sub-set of the formulas derived by Asperti's sequent calculus for Intuitionistic Light Affine Logic **ILAL** [1]. The types have a polymorphism *à la* ML [8]: only external quantifiers are allowed. The types for the elements of Λ_{LA}^T can be automatically inferred by a type inference algorithm. In particular, the type τ inferred for any $M \in \Lambda_{LA}^T$ is *principal*.

In spite Asperti's sequent calculus for **ILAL** greatly simplifies the original system for Light Linear Logic in [3], the natural deduction it induces still has enough rules to make a concrete functional syntax quite heavy. Following a methodological hint already in [1], the following simplification is introduced: contraction is left as an implicit structural rule of the natural deduction. This choice influences the design of Λ_{LA}^T as follows: Λ_{LA}^T 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. This makes Λ_{LA}^T a sort of *call by value* language: the two kinds of variables "decide" what can be duplicated, and when.

The language Λ_{LA}^T is *strongly normalizable* and *Church-Rosser*. It is also *correct* with respect to **P-TIME**. Namely, if $M \in \Lambda_{LA}^T$, then M represents an algorithm in **P-TIME**. Moreover, Λ_{LA}^T admits a poly-step reduction strategy. This means that any term M of Λ_{LA}^T can be reduced to its normal form in, at most, a number of steps polynomial in the dimension of M .

The completeness of Λ_{LA}^T with respect to **P-TIME** is left open.

"Index". Section 2 introduces the untyped language Λ_{LA} . Section 3 defines the natural deduction for **ILAL**, decorated with the terms of Λ_{LA}^T . Section 4 is about the correctness of Λ_{LA} with respect to **P-TIME**. Section 5 recalls the expressiveness of Λ_{LA}^T and shows some encodings of usual λ -Calculus terms in it. When reading Section 2, and 3, we suggest to refer to Section 5 for having programming examples on Λ_{LA}^T . Section 6 introduces the poly-step strategy. Section 7 defines the type inference algorithm in natural deduction style. Section 8 concludes the

paper with some reference to related, and future work. Appendix A introduces some of the details about \mathbf{G}_{LA} that Asperti skipped in [1].

Acknowledgments. This work has been developed also with some useful discussions with Andrea Asperti, Stefano Guerrini, Tom Kranz, Yves Lafont. The work has been supported by a TMR-Marie Curie Grant, contract n. ERBFM-BICT961411.

2 The Functional Language

The Syntax. Let **Term-Variables** be a set of identifiers ranged over by x, y, w, z , and **!-Term-Variables** be another set of identifiers ranged over by X, Y, W, Z . Moreover, let χ be ranging over **Term-Variables** \cup **!-Term-Variables**. The set Λ of the functional terms is given by:

$$M, N, P, Q ::= \text{Term-Variables} \mid \text{!-Term-Variables} \mid \lambda\chi.M \mid (MN) \mid !M \mid \\ !(M)[^N/\chi] \mid \S M \mid \S(M)[^{M_1/\chi_1} \cdots ^{M_n/\chi_n}] \mid \text{let } X = M \text{ in } N$$

The term constructors $\lambda, !, \S$, and **let** bind free variables. As usual, λ is such that, if χ is in the free variable set $\text{FV}(M)$ of M , then it can not be in $\text{FV}(\lambda\chi.M)$. 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* variable χ . In this case, the free variables of the $!$ -box obtained are in $\text{FV}(N)$. In particular, N is called the *interface* of the $!$ -box just built, and M is its *body*. The operator \S builds \S -boxes. It can be applied to a term M with free variables χ_1, \dots, χ_n , being $n \geq 1$. All χ_1, \dots, χ_n get bounded, and the free variables of the obtained \S -boxes are in $\cup_{i=1}^n \text{FV}(M_i)$. Again, all M_i s are the interface of the \S -box just built, and M is its *body*. Finally, if $X \in \text{FV}(N)$, then **let** binds X . The square brackets in $!$ and \S -boxes belong to the syntax, and delimit the interface of the boxes themselves.

The substitution of M for χ in N is denoted by $N\{^M\downarrow_\chi\}$. It is defined as a partial function. Namely, it behaves like the usual variable-clash free substitution only in one of the two 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.

The elements of Λ are considered up to α -equivalence. For example, $!(M)[^N/\chi]$ is α -equivalent to $!(M\{^{\chi'}\downarrow_\chi\})[^N/\chi]$. Parenthesis will be omitted when writing terms, if no ambiguity exists.

The Dynamics. The rewriting system \rightsquigarrow on Λ is the contextual closure of the relation \triangleright on $\Lambda_{\text{LA}} \times \Lambda_{\text{LA}}$ here below:

- β -group

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

- !!-group

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

- §!-group

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

- §§-group

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

- let -group

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

Of course, the α -equivalence must be used to avoid variable clashes when rewriting terms. As usual, \sim^* is the reflexive, and transitive closure of \sim on Λ . The functional language Λ_{LA} , subject of this work, is $\langle \Lambda, \sim \rangle$.

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

Λ_{LA} is a kind of restriction of the untyped call-by-value λ -Calculus[9], which rewriting rule is:

$$(\lambda x.M)N \rightarrow_{\beta_v} M\{\downarrow_x\} \text{ if } N \text{ is a value ,}$$

where the variables, and the λ -abstractions are values. Namely, only the terms with a specific form can be substituted for the variables. The rewriting system \sim behaves analogously to \rightarrow_{β_v} . Following the definition of the *partial* substitution of terms for variables, given above, no constraints exist when replacing $x \in \mathbf{Term-Variables}$ by *any* term. The idea is that, in the typable sub-set Λ_{LA}^T , x stands for any non duplicable, or *linear*, entity. Consequently, replacing

M for x can not result in the duplication of M . On the contrary, only the $!$ -boxes and the elements of **!-Term-Variables** can be substituted for a variable $X \in \text{!-Term-Variables}$. This because, in the typable sub-set $\Lambda_{\text{LA}}^{\text{T}}$, X represents duplicable, or *non linear*, resources. In the usual call-by-value terminology, any term is a value, with respect to the linear variables. On the other side, only the $!$ -boxes, and the **!-Term-Variables**, which represent the duplicable regions of **ILLL**, are values for the non linear variables.

Take the β -group, for example. The first four axioms follow what just said here above. 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 also not be a $!$ -box.

As a second example, consider the $!!$ -group. The relation defined by the $!!$ -rules makes two terms communicating, when such two terms are contained in two distinct $!$ -boxes. The communication takes place by substituting the term contained in one $!$ -box for the free variables of the term contained in the other $!$ -box. The rule \triangleright_1 deals with the case where one term N is in a $!$ -box which constitutes the interface of another $!$ -box, whose content is M . The communication between N and M can take place independently from the form of N , accordingly to what said above. 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.

All the other groups preserve the definition of the substitution, and are defined to cover all *disjoint* cases, according to the form of the term being substituted.

3 The Type Assignment

The Types. Let assume to have a set **Type-Variables**, ranged over by α, β, γ , and δ . The *types* are defined by the grammar: $\tau, \rho, \mu, \nu ::= \text{Type-Variables} \mid \tau \multimap \rho \mid !\tau \mid \S\tau$. The *type schemes* originate from the grammar: $\sigma ::= \forall \alpha_1 \dots \alpha_n. \tau$, with $n \geq 0$. As usual, \forall is a binder: the free variables of $\forall \alpha_1 \dots \alpha_n. \tau$ are $\text{FV}(\tau) \setminus \{\alpha_1 \dots \alpha_n\}$. We say that τ is *!-exponential*, and we write $!\text{-exp}(\tau)$, if there is a type τ' such that $\tau = \forall \alpha_1 \dots \alpha_n. !\tau'$, with $n \geq 0$.

A *set of assumptions* takes the form $\{\chi_1 : \sigma_1, \dots, \chi_n : \sigma_n\}$, where every σ_i is a *type scheme*, and every χ_i belongs to **Term-Variables** \cup **!-Term-Variables**. A set of assumptions $\Gamma = \{\chi_1 : \sigma_1, \dots, \chi_n : \sigma_n\}$ is *well formed* if it satisfies two constraints: (i) χ_i belongs to **!-Term-Variables** if, and only if, $!\text{-exp}(\sigma_i)$ holds with $n \geq i \geq 0$; (ii) Γ can be thought of as a function with finite domain $\{\chi_1, \dots, \chi_n\}$. Namely, $\chi : \sigma_1$, and $\chi : \sigma_2$ can not belong to Γ , if $\sigma_1 \neq \sigma_2$, up to the obvious α -equivalence on types.

The statement $!\text{-exp}(\Gamma)$ holds on a well formed set of assumptions Γ if all types in the co-domain of Γ are $!$ -exponential or, equivalently, if the domain of Γ is a sub-set of **!-Term-Variables**. Consequently, a well formed set of assumptions, whose domain is contained in **Term-Variables**, is *linear*. We take Γ as a meta-variable for ranging over generic well formed sets of assumptions, Θ for

denoting !-exponential sets of well formed assumptions, and, Δ, Φ, Ψ , and Υ for dealing with the linear sets of well formed assumptions.

The *type substitutions* are functions from **Type-Variables** to *types*. The notation: $\{\tau_1 \downarrow_{\alpha_1} \cdots \tau_n \downarrow_{\alpha_n}\}$, stands for a type substitution that simultaneously replaces every τ_i for α_i , and which is the identity on all the type variables different from $\alpha_1, \dots, \alpha_n$. The type substitutions are ranged over by S, R, T , and U . Moreover, for any type σ and any set Γ of assumptions, $S\sigma$, and $S\Gamma$ denote the application of the obvious extensions of S to type schemes and sets of assumptions. In general, the substitutions do not preserve well formed sets of assumptions. For example, $\Delta = \{x : \alpha\}$ is well formed, but $S\Delta$ is not, if $S = \{\downarrow_{\alpha}\}$. So, for any *well formed* set of assumptions Γ , and substitution S , the *compatibility* predicate S -compatible with- Γ holds exactly when $S\Gamma$ is well formed. Moreover, it holds:

Lemma 1. *If S is $S_1 S_2$, and S -compatible with- Γ , then so it is S_2 .*

The type schemes can be ordered: $\forall \beta_1 \dots \beta_n. \tau \geq \forall \alpha_1 \dots \alpha_m. S\tau$ if both $\text{FV}(\tau) \supseteq \{\beta_1, \dots, \beta_n\}$, and $\text{FV}(S\tau) \supseteq \{\alpha_1, \dots, \alpha_m\}$, for a given S .

The Typing Rules. For any well formed set of assumptions Γ , any functional term M , and any type scheme σ , we write the judgment: $\Gamma \vdash_{\text{T}} M : \sigma$ if it is a conclusion of a deduction in the following system:

$$\begin{array}{c}
(Ax) \frac{}{\Gamma, \chi : \sigma \vdash_{\text{T}} \chi : \sigma} \\
(\forall_E) \frac{\Gamma \vdash_{\text{T}} M : \forall \alpha. \sigma}{\Gamma \vdash_{\text{T}} M : \{\tau \downarrow_{\alpha}\} \sigma} \qquad (\forall_I) \frac{\Gamma \vdash_{\text{T}} M : \sigma \quad \alpha \notin \Gamma}{\Gamma \vdash_{\text{T}} M : \forall \alpha. \sigma} \\
(\multimap_E) \frac{\Theta, \Delta_1 \vdash_{\text{T}} M : \tau' \multimap \tau \quad \Theta, \Delta_2 \vdash_{\text{T}} N : \tau'}{\Theta, \Delta_1, \Delta_2 \vdash_{\text{T}} MN : \tau} \qquad (\multimap_I) \frac{\Gamma, \chi : \tau \vdash_{\text{T}} M : \tau'}{\Gamma \vdash_{\text{T}} \lambda \chi. M : \tau \multimap \tau'} \\
(!) \frac{\Gamma \vdash_{\text{T}} N : \tau' \quad \chi : \tau' \vdash_{\text{T}} M : \tau}{\Gamma \vdash_{\text{T}} !(M)[\tau'/\chi] : \tau} \qquad (!_{\emptyset}) \frac{\vdash_{\text{T}} M : \tau}{\Gamma \vdash_{\text{T}} !M : \tau} \\
\begin{array}{l}
m + n + p + q \geq 1 \\
\Theta, \Delta_i \vdash_{\text{T}} M_i : \tau_i \qquad (1 \leq i \leq m) \\
\Theta, \Phi_j \vdash_{\text{T}} N_j : \rho_j \qquad (1 \leq j \leq n) \\
\Theta, \Psi_k \vdash_{\text{T}} P_k : \mu_k \qquad (1 \leq k \leq p) \\
\Theta, \Upsilon_l \vdash_{\text{T}} Q_l : \nu_l \qquad (1 \leq l \leq q) \\
x_1 : \tau_1 \dots x_m : \tau_m, X_1 : \rho_1 \dots X_n : \rho_n, \\
y_1 : \mu_1 \dots y_p : \mu_p, Y_1 : \nu_1 \dots Y_q : \nu_q \vdash_{\text{T}} M : \tau
\end{array} \\
(\S) \frac{}{\Theta \dots \Delta_i \dots \Phi_j \dots \Psi_k \dots \Upsilon_l \dots \vdash_{\text{T}} \S(M) \left[\begin{array}{l}
M_1/x_1 \dots M_m/x_m, \\
N_1/X_1 \dots N_n/X_n, \\
P_1/y_1 \dots P_p/y_p, \\
Q_1/Y_1 \dots Q_q/Y_q
\end{array} \right] : \S\tau} \\
(\S_{\emptyset}) \frac{\vdash_{\text{T}} M : \tau}{\Gamma \vdash_{\text{T}} \S M : \S\tau} \qquad (\text{let}) \frac{\Theta, \Delta_1 \vdash_{\text{T}} M : \sigma \quad X : \sigma, \Theta, \Delta_2 \vdash_{\text{T}} N : \tau}{\Theta, \Delta_1, \Delta_2 \vdash_{\text{T}} \text{let } X = M \text{ in } N : \tau}
\end{array}$$

Observe that (Ax) , $(!_{\emptyset})$, and (\S_{\emptyset}) have implicit weakening, while (\multimap_E) , (\S) , and (let) have implicit contraction.

Definition 1. $\Lambda_{\text{LA}}^{\text{T}}$ is the sub-set of Λ_{LA} typable by \vdash_{T} , namely, $M \in \Lambda_{\text{LA}}^{\text{T}}$, if, and only if, there are Γ , and σ such that $\Gamma \vdash_{\text{T}} M : \sigma$.

Lemma 2. The following rules are admissible in \vdash_{T} :

$$(\geq) \frac{\chi : \sigma', \Gamma \vdash_{\text{T}} M : \sigma'' \quad \sigma \geq \sigma'}{\chi : \sigma, \Gamma \vdash_{\text{T}} M : \sigma''}$$

$$(WR) \frac{\Gamma \vdash_{\text{T}} M : \sigma \quad \text{domain}(\Gamma') \cap \text{domain}(\Gamma) = \emptyset \quad \mathbf{FV}(M) \subseteq \mathbf{V} \subseteq \text{domain}(\Gamma)}{\{\chi : \Gamma(\chi) \mid \chi \in \mathbf{V}\}, \Gamma' \vdash_{\text{T}} M : \tau}$$

Proof. By structural induction on M .

Clearly, rule (WR) simultaneously weakens, and extends Γ .

Lemma 3 (Substitution).

1. Let $\Theta, x : \sigma, \Delta \vdash_{\text{T}} M : \sigma'$. Then, $\Theta, \Phi, \Delta \vdash_{\text{T}} M\{^N \downarrow_x\} : \sigma'$, for any $\Theta, \Phi \vdash_{\text{T}} N : \sigma$.
2. Let $\Theta, X : !\tau, \Delta \vdash_{\text{T}} M : \sigma$. Then, $\Theta, \Phi, \Delta \vdash_{\text{T}} M\{^N \downarrow_X\} : \sigma$, for any $\Theta, \Phi \vdash_{\text{T}} N : !\tau$ such that N is a $!$ -box or it belongs to $!$ -Term-Variables.

Proof. By induction on M .

Theorem 1 (Subject Reduction). If $\Gamma \vdash_{\text{T}} M : \sigma$, and $M \rightsquigarrow N$, then $\Gamma \vdash_{\text{T}} N : \sigma$.

Proof. By induction on M , using Substitution Lemma here above.

4 Correctness

A possible statement of correctness for $\Lambda_{\text{LA}}^{\text{T}}$ states: any $M \in \Lambda_{\text{LA}}^{\text{T}}$ represents an algorithm in **P-TIME**.

The proof of such a statement is very simple. First, we consider a language \mathbf{G}_{LA} of graphs. A strategy that reduces any graph G in a time at most polynomial in the dimension of G exists. Second, \mathbf{G}_{LA} is proved to be a model of $\Lambda_{\text{LA}}^{\text{T}}$. So, $\Lambda_{\text{LA}}^{\text{T}}$ becomes a functional notation for dealing with **P-TIME**.

The language \mathbf{G}_{LA} is explicitly introduced in Appendix A. It is the graph version of the sequent calculus of Intuitionistic Light Affine Logic **ILAL** in [1].

Let use \mathbf{G}_{LA} as a model for $\Lambda_{\text{LA}}^{\text{T}}$. First, let \succ be the reduction relation on \mathbf{G}_{LA} , which we recall in Appendix A. The intuition about \succ says that, up to some irrelevant details, it mimics the cut-elimination steps of **ILAL** on \mathbf{G}_{LA} . Then, take \diamond as the equational theory of \succ on \mathbf{G}_{LA} , namely, its reflexive, transitive, and symmetric closure.

Theorem 2 (Correctness). There exists an embedding $(.)^\bullet$ of $\Lambda_{\text{LA}}^{\text{T}}$ into \mathbf{G}_{LA} such that, if $M \rightsquigarrow^* N$, then $M^\bullet \diamond N^\bullet$.

$(.)^\bullet$ is the obvious adaptation to Λ_{LA}^T of the embedding of the natural deduction of Intuitionistic Logic into its sequent calculus, where $(\mathbf{let} X = N \mathbf{in} M)^\bullet \stackrel{\text{def}}{=} ((\lambda X.M)N)^\bullet$. Then, the correctness of Λ_{LA}^T is implied by the *confluence* of \succ .

Theorem 3 (Confluence). *The rewriting system \rightsquigarrow is confluent.*

This theorem is implied by the *strong normalizability*, and the *local confluence* of Λ_{LA}^T . In particular, Λ_{LA}^T is *strongly normalizing* because, following Girard in [3], we can embed it into System F [4]. The local confluence comes from verifying that \rightsquigarrow has not critical pairs.

5 Expressiveness

Any polynomial $f(x_1, \dots, x_n)$ of arity $n \geq 0$, either linear, or not, can be represented as a term $\langle f(x_1, \dots, x_n) \rangle$ of Λ_{LA}^T . In particular, the \mathbf{let} constructor is required when $f(x_1, \dots, x_n)$ is not linear.

Following also a referee's suggestion, we do not introduce the whole encoding of the polynomials. Only the Church Numerals and some operation on them are given, together with some further intuition about how the reduction complexity is controlled. Those interested to more details about the encoding of polynomials are referred to [10], where also the predecessor is introduced.

The Church Numerals. Let \mathbf{int}_τ be an abbreviation for $!(\tau \multimap \tau) \multimap \S(\tau \multimap \tau)$. For any $n \geq 1$, define:

$$\begin{aligned} \bar{0} &\stackrel{\text{def}}{=} \lambda X. \S \lambda y. y : \mathbf{int}_\tau \\ \overline{n+1} &\stackrel{\text{def}}{=} \lambda X. \S (\lambda y. y_1 (\dots (y_n y) \dots)) [^X /_{y_1} \dots ^X /_{y_n}] : \mathbf{int}_\tau . \end{aligned}$$

Now, define an *erasure* function on Λ_{LA}^T as follows: from a given M , 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 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. Applying this erasure to any term of Λ_{LA} would yield a term of the usual λ -Calculus. In particular, the terms here above, would be mapped into the usual Church Numerals. Some combinators on them:

$$\begin{aligned} \mathit{succ} &\stackrel{\text{def}}{=} \lambda z X. \S (\lambda y. y_1 (y_2 y)) [^X /_{y_1} (^{zX}) /_{y_2}] : \mathbf{int}_\tau \multimap \mathbf{int}_\tau \\ \mathit{sum} &\stackrel{\text{def}}{=} \lambda w z X. \S (\lambda y. y_1 (y_2 y)) [^{(wX)} /_{y_1} (^{zX}) /_{y_2}] : \mathbf{int}_\tau \multimap \mathbf{int}_\tau \multimap \mathbf{int}_\tau \\ \mathit{iter} &\stackrel{\text{def}}{=} \lambda x_n X_s x_b. \S (y w) [^{(x_n X_s)} /_y ^{x_b} /_w] : \mathbf{int}_\tau \multimap !(\tau \multimap \tau) \multimap \S \tau \multimap \S \tau \\ \mathit{mult} &\stackrel{\text{def}}{=} \lambda x Y. \mathit{iter} x !(\lambda w. \mathit{sum} z w) [^Y /_z] \S \bar{0} : \mathbf{int}_{\mathbf{int}_\tau} \multimap !\mathbf{int}_\tau \multimap \S \mathbf{int}_\tau . \end{aligned}$$

The successor succ , the sum , and the multiplication mult do the obvious thing. The iteration iter takes as arguments a numeral, a *step function*, and a *base* where to start the iteration from.

Observe that *iter* $\overline{\lambda}$ ($!\overline{\lambda}$) ($\S\overline{\lambda}$) can not have type, for $\overline{\lambda}$, and any numeral here above, can not be used as a step function. This because the step function is required to have identical domain and co-domain. This should not surprise. In the introduction we pointed out that the application of $\overline{\lambda}$ to $\overline{\lambda}$ is the starting point to yield exponentially growing computations.

Notice, however, that there are variations of $\overline{\lambda}$, and $\overline{\lambda}$ that we can use as step function for *iter* :

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

We conclude with an example about reduction:

$$\begin{aligned}\text{sum } \overline{\lambda} & \\ \rightsquigarrow_{1,1}^{\beta} \lambda X.\S(\lambda y.y_1(y_2 y))^{[\overline{\lambda} X/y_1 \ \overline{\lambda} X/y_2]} & \\ \rightsquigarrow_{2,2}^{\beta} \lambda X.\S(\lambda y.y_1(y_2 y))^{[\S(\lambda z.z_1 z)^{[X/z_1]}/y_1 \ \S(\lambda w.w_1 w)^{[X/w_1]}/y_2]} & \\ \rightsquigarrow_{2,2}^{\S\S} \lambda X.\S(\lambda y.(\lambda z.z_1 z)((\lambda w.w_1 w)y))^{[X/z_1 \ X/w_1]} & \\ \rightsquigarrow_{1,1}^{\beta} \lambda X.\S(\lambda y.z_1(w_1 y))^{[X/z_1 \ X/w_1]} & \\ \stackrel{\text{def}}{=} \overline{\lambda} . &\end{aligned}$$

For example, the notation $\rightsquigarrow_{2,2}^{\S\S}$ stands for a sequence of two contextual applications of \triangleright_2 , belonging to the $\S\S$ -group.

6 Poly-step Reduction Strategy

$\Lambda_{\text{LA}}^{\text{T}}$ can be proved to have a *poly-step* normalization process. This means that, given a term M , there is a *normal strategy* such that M rewrites to a normal term N with no more \rightsquigarrow -steps than the dimension $|M|$ of M . Notice that, for a term M , being poly-step in the above sense, is not exactly as having a **P-TIME** normalization process in $|M|$. Counting the time would mean to consider at least the cost of the renaming operations, and of the substitution of the terms for the variables. But this is not an issue, as Λ_{LA} is only an abstract language to program with, and not an implementation language.

The proof that $\Lambda_{\text{LA}}^{\text{T}}$ is poly-step split in two parts.

First, we show that the rewriting relation \triangleright of the language of graphs \mathbf{G}_{LA} , is computationally adequate with respect to the rewriting relation \rightsquigarrow of Λ_{LA} . This means showing:

Theorem 4 (Adequacy). *There exists an embedding $(.)^p$ from Λ_{LA} to \mathbf{G}_{LA} such that, if $M \rightsquigarrow^* N$, then $M^p \triangleright^* N^p$.*

Second, we define a *canonical* reduction strategy \rightsquigarrow_p^* on \rightsquigarrow . Then, the existence of a poly-time reduction strategy \succ_p^* for \mathbf{G}_{LA} , allows to prove:

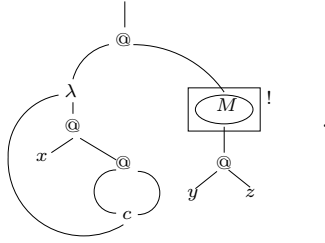
Theorem 5 (Poly-step Adequacy). *For any term M of $\Lambda_{\text{LA}}^{\text{T}}$, if $M \rightsquigarrow_p^* N$, then $M^P \succ_p^* N^P$.*

Proving Adequacy. The embedding $(.)^P$ can not be as simple as $(.)^\bullet$, otherwise we could only prove Correctness, as we did in Section 4.

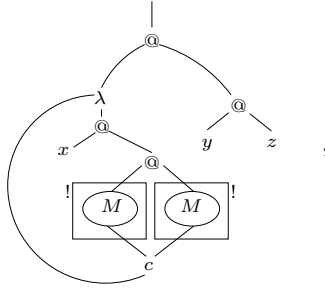
Here, we want more: any reduction step of $\Lambda_{\text{LA}}^{\text{T}}$, in fact, has to stand for a reduction sequence of \mathbf{G}_{LA} (Appendix A.) The problem for showing this is that Λ_{LA} does not have explicit contraction, unlike \mathbf{G}_{LA} . Recall that any contraction node of \mathbf{G}_{LA} determines whether a graph can be duplicated, or not; in Λ_{LA} , the same effect is obtained with two distinguished sets of variables. The result is that the situations where a contraction node of \mathbf{G}_{LA} gets stuck, correspond to pairs of term *constructors* of $\Lambda_{\text{LA}}^{\text{T}}$ that can not annihilate each other. For example, consider the following one-step reduction of a term to its normal form:

$$P \stackrel{\text{def}}{=} (\lambda X.xXX)!(M)^{[yz/x]} \rightsquigarrow (\lambda Y.x!(M)^{[Y/x]}!(M)^{[Y/x]})(yz) \stackrel{\text{def}}{=} Q .$$

Taking $(.)^P$ as $(.)^\bullet$, the graph P^P would be the one expected:



On the contrary, Q^P would be:



which is not quite what we want: Q^P would not be the graph that P^P reduces to. The difference between $(.)^\bullet$ and $(.)^P$ must be the ability to detect when a configuration in a term of $\Lambda_{\text{LA}}^{\text{T}}$ must be translated in the obvious way, as for P , or if it must be simplified, as for Q . Situations like the one here above involve

pair of boxes as well. For example, let $P \stackrel{\text{def}}{=} \lambda xyz. xyz$. Then, take the terms:

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

and reduce them: $(.)^\bullet$ would not allow to prove adequacy. Summing up, $(.)^p$ must be sensitive to the “history” of some pair of term constructors in a given term.

This need is accomplished by taking a labeling function which maps the term constructors λ , $!$ -box, \S -box, and the application of Λ_{LA} to natural numbers. Then, for any M , $(M)^p$ consists of two main steps. The first yields the same result, say G_M , as $(M)^\bullet$. The second step operates on G_M by reducing it with \triangleright . These reductions eliminate *only* those pairs of nodes of G_M which are images of two term constructors with the same label.

Of course something has to correctly set the labeling. This can be done by \rightsquigarrow , when yielding the right-hand side of: \triangleright_5^β , $\triangleright_{5,6}^{!!}$, $\triangleright_{5,6}^{\S!}$, and $\triangleright_{5,6}^{\S\S}$.

Proving $\Lambda_{\text{LA}}^\text{T}$ being Poly-step. Let M be given. We say that N is at depth $i \geq 0$, 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, which are the left-hand side terms of the rewriting relation \triangleright in Section 2. 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 1^{st} and stops (at most) at the d^{th} .

$\Lambda_{\text{LA}}^\text{T}$ is poly-step because the strategy \rightsquigarrow_p^* is the obvious adaptation of the poly-time strategy \triangleright_p^* on \mathbf{G}_{LA} . To get \triangleright_p^* as in [1], just replace \triangleright for \rightsquigarrow , and let the β -redexes be the left-hand graphs of $\triangleright_{1,3,4,5,6,7,8,9,10}$ in Appendix A, while the box-redexes be all the other ones.

7 The Type Inference

This section is essentially technical. It defines the adaptation to Λ_{LA} of the Damas-Milner type inference for ML [2]. For a less verbose presentation it is worth introducing some notations.

Never before used type variables are called *fresh*.

For any sub-set \mathbf{V} of **Term-Variables** \cup **!-Term-Variables**, and for any well formed set of assumptions Γ , $\Gamma^\mathbf{V}$ stands for Γ restricted to $\chi \in \mathbf{V}$, while $\Gamma_\mathbf{V}$ is Γ restricted to $\chi \notin \mathbf{V}$.

Let $\tau_1, \dots, \tau_n, \tau'_1, \dots, \tau'_n$ be any tuple of types. Then, $\vdash_U \{\tau_1 = \tau'_1, \dots, \tau_n = \tau'_n\} \Rightarrow U$, denotes the obvious algorithm that yields the *most general unifier* U , of the pairs $\tau_1 = \tau'_1, \dots, \tau_n = \tau'_n$, if any. Otherwise, $\vdash_U \{\tau_1 = \tau'_1, \dots, \tau_n = \tau'_n\} \Rightarrow \mathbf{failure}$.

The identity substitution on types is \mathcal{I} .

For any set of assumptions Γ , and any type τ , the notation $\forall \Gamma. \tau$ stands for the type $\forall \alpha_1 \dots \alpha_n. \tau$, where $\{\alpha_1 \dots \alpha_n\}$ is $\mathbf{FV}(\tau) \setminus \mathbf{FV}(\Gamma)$.

The Algorithm For any well formed set of assumptions Γ , any functional term M , any substitution S , and any type τ , the algorithm for the type inference derives two kinds of judgments: either $\vdash_{\text{TI}} \Gamma; M \Rightarrow S; \tau$ or $\vdash_{\text{TI}} \Gamma; M \Rightarrow \mathbf{failure}$. The first corresponds to the success of the algorithm, while, the second to its failure.

The following rules in Natural Semantics[5] define \vdash_{TI} to derive judgments of the first kind. The rules for the second kind of judgment are omitted, because obvious. The rules of the algorithm are:

$$\begin{array}{c}
n \geq 0 \\
\gamma_1, \dots, \gamma_n \text{ fresh} \\
S = \{\gamma_1 \downarrow_{\alpha_1} \dots \gamma_n \downarrow_{\alpha_n}\} \\
(Ax) \frac{}{\vdash_{\text{TI}} \Gamma, \chi : \forall \alpha_1 \dots \alpha_n. \tau; \chi \Rightarrow \mathcal{I}; S\tau}
\end{array}$$

$$\begin{array}{c}
\vdash_{\text{TI}} \Gamma; M \Rightarrow S_M; \tau_M \\
\gamma \text{ fresh} \\
\vdash_U \{\tau_M = !\gamma\} \Rightarrow U \\
\vdash_{\text{TI}} X : (\forall S_M \Gamma. !U\gamma), S_M \Gamma_{\{X\}}; N \Rightarrow S_N; \tau_N \\
S_N S_M \text{-compatible with- } \Gamma \\
(\text{let}) \frac{}{\vdash_{\text{TI}} \Gamma; \text{let } X = M \text{ in } N \Rightarrow S_N S_M; \tau_N}
\end{array}
\qquad
\begin{array}{c}
\vdash_{\text{TI}} \Gamma; M \Rightarrow S_M; \tau_M \\
\vdash_{\text{TI}} S_M \Gamma; N \Rightarrow S_N; \tau_N \\
\gamma \text{ fresh} \\
\vdash_U \{S_N \tau_M = \tau_N \multimap \gamma\} \Rightarrow U \\
US_N S_M \text{-compatible with- } \Gamma \\
(\multimap E) \frac{}{\vdash_{\text{TI}} \Gamma; MN \Rightarrow US_N S_M; U\gamma}
\end{array}$$

$$\begin{array}{c}
\gamma \text{ fresh} \\
\vdash_{\text{TI}} \Gamma_{\{x\}}, x : \gamma; M \Rightarrow S; \tau \\
S \text{-compatible with- } (\Gamma_{\{x\}}, x : \gamma) \\
S \text{-compatible with- } \Gamma^{\{x\}} \\
(\multimap_{Ix}) \frac{}{\vdash_{\text{TI}} \Gamma; \lambda x. M \Rightarrow S; S\gamma \multimap \tau}
\end{array}
\qquad
\begin{array}{c}
\gamma \text{ fresh} \\
\vdash_{\text{TI}} \Gamma_{\{X\}}, X : !\gamma; M \Rightarrow S; \tau \\
S \text{-compatible with- } (\Gamma_{\{X\}}, X : !\gamma) \\
S \text{-compatible with- } \Gamma^{\{X\}} \\
(\multimap_{IX}) \frac{}{\vdash_{\text{TI}} \Gamma; \lambda X. M \Rightarrow S; (!S\gamma) \multimap \tau}
\end{array}$$

$$\begin{array}{c}
\vdash_{\text{TI}} \Gamma; N \Rightarrow S_N; \tau_N \\
\gamma \text{ fresh} \\
\vdash_U \{\tau_N = !\gamma\} \Rightarrow U \\
\text{not}(!\text{-exp}(U\gamma)) \\
\vdash_{\text{TI}} x : U\gamma, US_N \Gamma_{\{x\}}; M \Rightarrow S_M; \tau_M \\
S_M US_N \text{-compatible with- } \Gamma \\
(!_x) \frac{}{\vdash_{\text{TI}} \Gamma; !(M)^N / x \Rightarrow S_M US_N; !\tau_M}
\end{array}
\qquad
\begin{array}{c}
\vdash_{\text{TI}} \Gamma; N \Rightarrow S_N; \tau_N \\
\gamma \text{ fresh} \\
\vdash_U \{\tau_N = !!\gamma\} \Rightarrow U \\
\vdash_{\text{TI}} X : !U\gamma, US_N \Gamma_{\{X\}}; M \Rightarrow S_M; \tau_M \\
S_M US_N \text{-compatible with- } \Gamma \\
(!_X) \frac{}{\vdash_{\text{TI}} \Gamma; !(M)^N / X \Rightarrow S_M US_N; !\tau_M}
\end{array}$$

$$\begin{array}{c}
\vdash_{\text{TI}} \emptyset; M \Rightarrow S; \tau \\
S \text{-compatible with- } \Gamma \\
(!_\emptyset) \frac{}{\vdash_{\text{TI}} \Gamma; !M \Rightarrow S; !\tau}
\end{array}
\qquad
\begin{array}{c}
\vdash_{\text{TI}} \emptyset; M \Rightarrow S; \tau \\
S \text{-compatible with- } \Gamma \\
(\S_\emptyset) \frac{}{\vdash_{\text{TI}} \Gamma; \S M \Rightarrow S; \S \tau}
\end{array}$$

$$\begin{array}{l}
\vdash_{\text{TI}} S_{i-1} \cdots S_1 \Gamma; M_i \Rightarrow S_i; \tau_i \quad (1 \leq i \leq m) \\
\vdash_{\text{TI}} R_{j-1} \cdots R_1 S_m \cdots S_1 \Gamma; N_j \Rightarrow R_j; \rho_j \quad (1 \leq j \leq n) \\
\vdash_{\text{TI}} T_{k-1} \cdots T_1 R_n \cdots R_1 S_m \cdots S_1 \Gamma; P_k \Rightarrow T_k; \mu_k \quad (1 \leq k \leq p) \\
\vdash_{\text{TI}} U_{l-1} \cdots U_1 T_p \cdots T_1 R_n \cdots R_1 S_m \cdots S_1 \Gamma; Q_l \Rightarrow U_l; \nu_l \quad (1 \leq l \leq q) \\
\gamma_1 \dots \gamma_m, \alpha_1 \dots \alpha_n, \beta_1 \dots \beta_p, \delta_1 \dots \delta_q \text{ fresh} \\
\vdash_{\text{U}} \cup_{1 \leq i \leq m} \{U_q \cdots U_1 T_p \cdots T_1 R_n \cdots R_1 S_m \cdots S_{i+1} \tau_i = !\gamma_i\} \cup \\
\cup_{1 \leq j \leq n} \{U_q \cdots U_1 T_p \cdots T_1 R_n \cdots R_{j+1} \rho_j = !\alpha_j\} \cup \\
\cup_{1 \leq k \leq p} \{U_q \cdots U_1 T_p \cdots T_{k+1} \mu_k = \S \beta_k\} \cup \\
\cup_{1 \leq l \leq q} \{U_q \cdots U_{l+1} \nu_l = \S \delta_l\} \Rightarrow U \\
\text{not}(!\text{-exp}(U\gamma_i)) \quad (1 \leq i \leq n) \\
\text{not}(!\text{-exp}(U\beta_k)) \quad (1 \leq k \leq p) \\
\Gamma_1 = \{x_1 : \gamma_1, \dots, x_m : \gamma_m, X_1 : !\alpha_1, \dots, X_n : !\alpha_n, \\
y_1 : \beta_1, \dots, y_p : \beta_p, Y_1 : !\delta_1, \dots, Y_q : !\delta_q\} \\
\Gamma_2 = U_q \cdots U_1 T_p \cdots T_1 R_n \cdots R_1 S_m \cdots S_1 \Gamma_{\{x_1 \dots x_m, X_1 \dots X_n, y_1 \dots y_p, Y_1 \dots Y_q\}} \\
\vdash_{\text{TI}} U\Gamma_1, U\Gamma_2; M \Rightarrow S_M; \tau_M \\
\text{§} \frac{SUU_q \cdots U_1 T_p \cdots T_1 R_n \cdots R_1 S_m \cdots S_1 \text{-compatible with-} \Gamma}{\vdash_{\text{TI}} \Gamma; \S(M) \left[\begin{array}{l} M_1/x_1 \cdots M_m/x_m, \\ N_1/X_1 \cdots N_n/X_n, \\ P_1/y_1 \cdots P_p/y_p, \\ Q_1/Y_1 \cdots Q_q/Y_q \end{array} \right] \Rightarrow S_M U U_q \cdots U_1 T_p \cdots T_1 R_n \cdots R_1 S_m \cdots S_1; \S \tau_M}
\end{array}$$

The last rule is a “nightmare” because of the complexity of the type assignment rule for the §-box. For example, the line:

$$\vdash_{\text{TI}} S_{i-1} \cdots S_1 \Gamma; M_i \Rightarrow S_i; \tau_i \quad (1 \leq i \leq m)$$

stands for the statements:

$$\begin{array}{l}
\vdash_{\text{TI}} \Gamma; M_1 \Rightarrow S_1; \tau_1 \\
\vdash_{\text{TI}} S_1 \Gamma; M_2 \Rightarrow S_2; \tau_2 \\
\vdots \\
\vdash_{\text{TI}} S_{m-1} \cdots S_1 \Gamma; M_m \Rightarrow S_m; \tau_m
\end{array}$$

Theorem 6 (Correctness). *Let Γ be well formed. If $\vdash_{\text{TI}} \Gamma; M \Rightarrow S; \tau$, then $S\Gamma \vdash_{\text{T}} M : \tau$. In particular, S -compatible with- Γ .*

Proof. Induction on the calls to \vdash_{TI} , using Lemma 2.

Theorem 7 (Completeness). *Let Γ , and $S\Gamma$ be well formed. If $S\Gamma \vdash_{\text{T}} M : \sigma$, then $\vdash_{\text{TI}} \Gamma; M \Rightarrow S_M; \tau_M$, and there is \bar{S} such that $S = \bar{S}S_M$, and $\bar{S}(\forall S_M \Gamma. \tau_M) \geq \sigma$.*

Proof. Induction on the length of the derivation of $S\Gamma \vdash_{\text{T}} M : \tau$, using Lemma 1, and 2.

Completeness states that any type assigned by \vdash_{T} to M can be obtained as an instance of the type τ that \vdash_{TI} infers for M . So, τ is *principal* for M .

8 Conclusions

This work has presented an untyped functional language Λ_{LA} which 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 Intuitionistic Light Affine Logic.

The main properties of $\Lambda_{\text{LA}}^{\text{T}}$ are related to the functions it can represent, and to the complexity of its rewriting system. Every term of $\Lambda_{\text{LA}}^{\text{T}}$ represents a **P-TIME** algorithm. Moreover, there is a poly-step reduction strategy: it gets to the normal form of any term M in, at most, a polynomial number of steps, with respect to the dimension of M . Finally, $\Lambda_{\text{LA}}^{\text{T}}$ is Church-Rosser. So, we can think of it as a programming language to deal with algorithms with a computational complexity which is predictable and, at least in principle, reasonably low.

However, Λ_{LA} still needs improvements. Its syntax is still quite heavy. The main goal, like also a referee suggested, is to eliminate the interfaces from boxes.

The completeness of $\Lambda_{\text{LA}}^{\text{T}}$ is still open.

There are other languages to program algorithms in **P-TIME** [6, 7]. The weakness of $\Lambda_{\text{LA}}^{\text{T}}$, with respect to them, is that completeness is still open. Its strength come from its clean logical base.

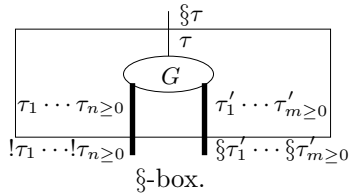
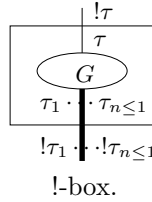
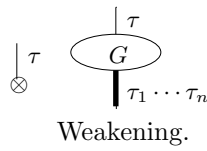
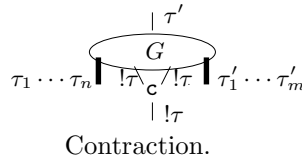
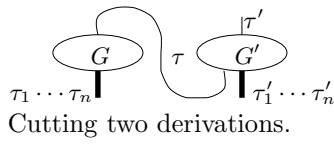
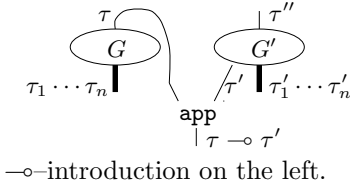
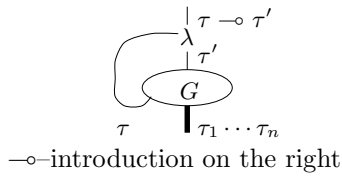
References

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 Symposium on Logic in Computer Science LICS'82*, January 1982.
3. J.-Y. Girard. Light Linear Logic. *Information and Computation*, 143:175 – 204, 1998.
4. J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Cambridge University Press, 1989.
5. C.A. Gunter. *Semantics of Programming Languages: Structures and Techniques*. Foundations of Computing. The MIT Press, 1992.
6. 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, 1997.
7. D. Leivant, and J.-Y. Marion. Lambda calculus characterizations of poly-time. *Fundamenta Informaticae*, 19:167 – 184, 1993.
8. R. Milner, M. Tofte, and R. Harper. *The Definition of Standard ML*. The MIT Press, 1990.
9. G. Plotkin. Call-by-name, call-by-value and the λ -calculus. *Theoretical Computer Science*, 1:125 – 159, 1975.
10. L. Roversi. Concrete syntax for intuitionistic light affine logic with polymorphic type assignment. In *Sixth Italian Conference on Theoretical Computer Science*. World Scientific, 9 – 11 November (Prato – Italy) 1998.

A The Language of Graphs

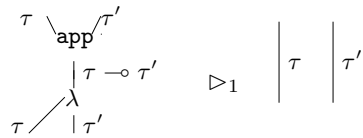
This section introduces \mathbf{G}_{LA} . The language \mathbf{G}_{LA} is the graph language Asperti refers to in [1] to prove that the sequent calculus for **ILAL** captures **P-TIME**.

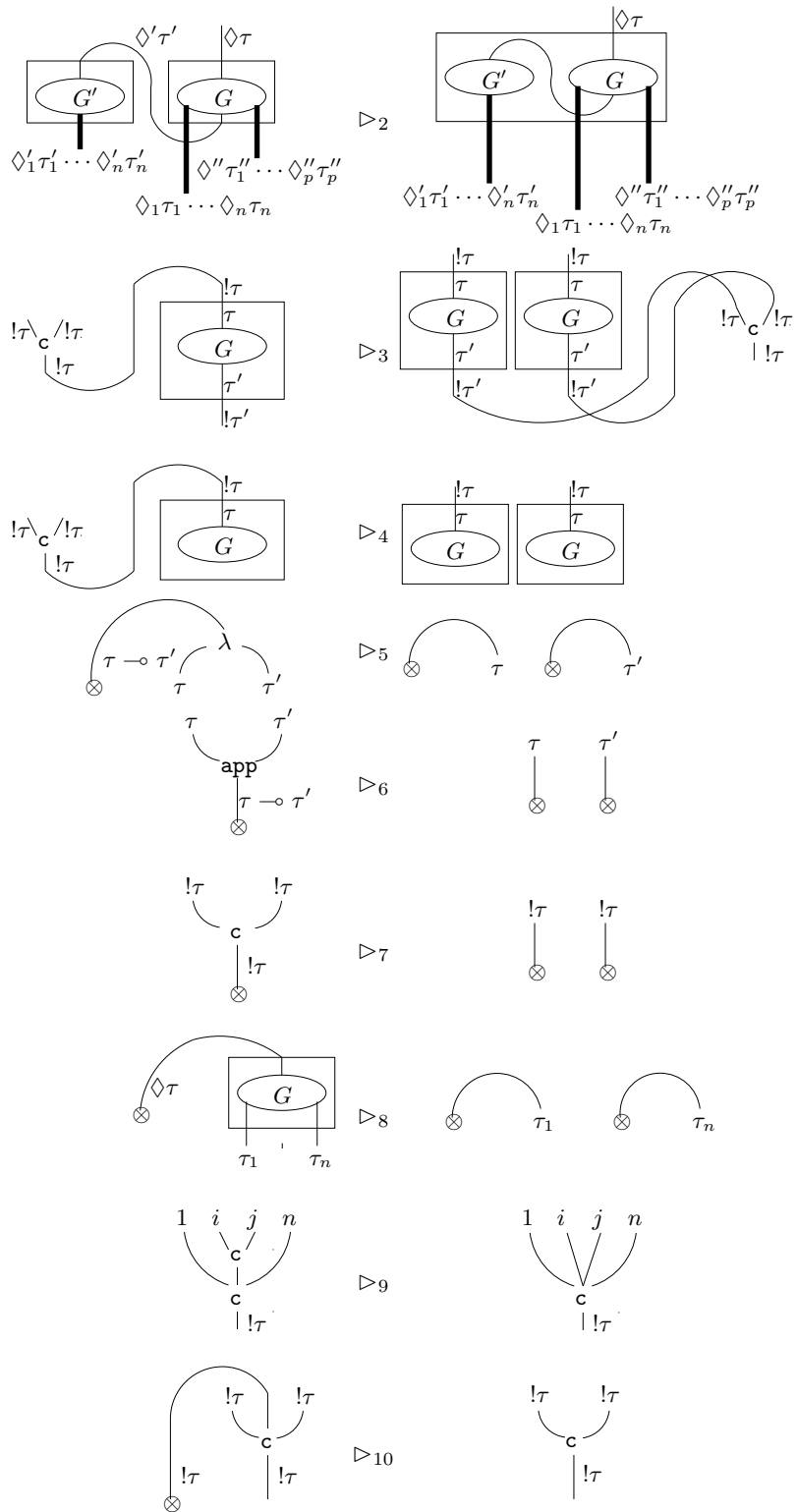
The language \mathbf{G}_{LA} is the least set of graphs containing the wires: $\begin{array}{c} | \\ \tau \end{array}$ which correspond to the axioms, and such that, if G , and G' are in \mathbf{G}_{LA} , then the following graphs belong to \mathbf{G}_{LA} as well:



The elements of \mathbf{G}_{LA} have a single upward link: the *root*. They have also a, possibly empty, set of sticking down links: the *inputs*. Multiple inputs are denoted by thick lines.

The rewriting system \triangleright is the least relation on $\mathbf{G}_{\text{LA}} \times \mathbf{G}_{\text{LA}}$ containing the *contextual closure* of a relation \triangleright between *parts* of graphs. The relation \triangleright is:





The rule \triangleright_2 is defined with the following proviso: if $\diamond \equiv \S$, then $\diamond' \in \{!, \S\}$. Otherwise, if $\diamond \equiv !$, then \diamond' can only be $!$. Of course, the bounds for n, p , and q are consistent with these two cases. Moreover, every $\diamond_i, \diamond'_j, \diamond''_k$ range over the set $\{!, \S\}$. Rule \triangleright_8 applies to both cases $\diamond \equiv !$, and $\diamond \equiv \S$. The rules \triangleright_9 , and \triangleright_{10} are a sort of garbage-collection.

We insist recalling that both \mathbf{G}_{LA} and its rewriting relation \triangleright are the language effectively used by Asperti to capture **P-TIME**[1].

The reflexive, and transitive closure \triangleright^* of the rewriting system here above is *locally confluent*, and *strongly normalizing*. Hence, it is also *Church-Rosser*. The strategy able to reduce to any graph G in poly-time, with respect to its dimension, is recalled in Section 6.