

## LIGHT LOGICS AND THE CALL-BY-VALUE LAMBDA CALCULUS

PAOLO COPPOLA <sup>a</sup>, UGO DAL LAGO <sup>b</sup>, AND SIMONA RONCHI DELLA ROCCA <sup>c</sup>

<sup>a</sup> Dipartimento di Matematica e Informatica, Università di Udine. via delle Scienze 206, 33100 Udine, Italy.

*e-mail address:* coppola@dimi.uniud.it

<sup>b</sup> Dipartimento di Scienze dell'Informazione, Università di Bologna. via Mura Anteo Zamboni 7, 40127 Bologna, Italy

*e-mail address:* dallago@cs.unibo.it

<sup>c</sup> Dipartimento di Informatica, Università di Torino corso Svizzera 185, 10129 Torino, Italy

*e-mail address:* ronchi@di.unito.it

---

**ABSTRACT.** The so-called light logics [13, 1, 2] have been introduced as logical systems enjoying quite remarkable normalization properties. Designing a type assignment system for pure lambda calculus from these logics, however, is problematic, as discussed in [6]. In this paper we show that shifting from usual call-by-name to call-by-value lambda calculus allows regaining strong connections with the underlying logic. This will be done in the context of Elementary Affine Logic (EAL), designing a type system in natural deduction style assigning EAL formulae to lambda terms.

### 1. INTRODUCTION

The so-called light logics [13, 1, 2] have been introduced as logical counterparts of complexity classes, namely polynomial and elementary time functions. After their introduction, they have been shown to be relevant for optimal reduction [10, 11], programming language design [2, 16] and set theory [15]. However, proof languages for these logics, designed through the Curry-Howard correspondence, are syntactically quite complex and can hardly be proposed as programming languages. An interesting research challenge is the design of type systems assigning light logics formulae to pure lambda-terms, forcing the class of typable terms to enjoy the same remarkable properties which can be proved for the logical systems. The mismatch between  $\beta$ -reduction in the lambda-calculus and cut-elimination in logical systems, however, makes it difficult to both getting the subject reduction property and inheriting the complexity properties from the logic, as discussed in [6]. Indeed,

---

*1998 ACM Subject Classification:* F.4.1, F.1.1, F.1.3.

*Key words and phrases:* Linear logic, lambda calculus, implicit computational complexity.

<sup>a,b,c</sup> All authors have been partially supported by PRIN project “FOLLIA” and french ANR “NO-CoST” project (JC05\_43380).

$\beta$ -reduction is more permissive than the restrictive copying discipline governing calculi directly derived from light logics. Consider, for example, the following expression in  $\Lambda_{\text{LA}}$  (see [16]):

$$\text{let } M \text{ be } !x \text{ in } N$$

This rewrites to  $N\{x/P\}$  if  $M$  is  $!P$ , but is not a redex if  $M$  is, say, an application. It is not possible to map this mechanism into pure lambda calculus. The solution proposed by Baillot and Terui [6] in the context of Light Affine Logic (**LAL**, see [1, 2]) consists in defining a type-system which is strictly more restrictive than the one induced by the logic. In this way, they both achieve subject reduction and a strong notion of polynomial time soundness.

Now, notice that mapping the above let expression to the application

$$(\lambda x.N)M$$

is not meaningless if we shift from the usual call-by-name lambda calculus to the call-by-value lambda calculus, where  $(\lambda x.N)M$  is not necessarily a redex. In this paper, we make the best of this idea, introducing a type assignment system, that we call **ETAS**, assigning formulae of Elementary Affine Logic (**EAL**) to lambda-terms. **ETAS** enjoys the following remarkable properties:

- The language of types coincides with the language of **EAL** formulae.
- Every proof of **EAL** can be mapped into a type derivation in **ETAS**.
- (Call-by-value) subject reduction holds.
- Elementary bounds can be given on the length of any reduction sequence involving a typable term. A similar bound holds on the size of terms involved in the reduction.
- Type inference is decidable and the principal typings can be inferred in polynomial time.

The basic idea underlying **ETAS** consists in partitioning premises into three classes, depending on whether they are used once, or more than once, or they are in an intermediate status. We believe this approach can work for other light logics too, and some hints will be given.

The proposed system is the first one satisfying the above properties for light logics. A notion of typability for lambda calculus has been defined in [10, 11, 7] for **EAL**, and in [4] for **LAL**. Type inference has been proved to be decidable. In both cases, however, the notion of typability is not preserved by  $\beta$ -reduction.

Noticeably, the proposed approach can be extended to Light Affine Logic and Soft Affine Logic (**SAL**, see [5, 14]).

A preliminary version of the present paper is [9]: here some results have been improved. In particular a new type inference algorithm is presented, and its complexity is analyzed: it turns out that our type inference algorithm for **EAL** has a complexity of the same order than the type inference for simple types. Moreover some discussions about possible extensions of this method have been added.

The paper is organized as follows: in Section 2 a comparison with existing work is made, in Section 3 some preliminary notions about **EAL** and lambda calculus are recalled, in Section 4 the **ETAS** system is introduced, and in Section 5 and 6 its main properties, namely complexity bounds and a type inference algorithm, are explained. Section 7 presents two possible extensions, allowing to reach completeness for elementary functions, and in Section 8 some hints on how to apply our idea to other light logics are given. Section 9 contains a short summary of the obtained results.

## 2. COMPARISON WITH EXISTING WORK

This work is not the first contribution on type systems derived from light logics. We should mention works on (principal) type inference for Elementary Affine Logic and Light Affine Logic by Baillot, Coppola, Martini and Ronchi Della Rocca [10, 4, 11]. There, the goal was basically proving decidability of type inference. The proposed type systems were the ones directly induced from logical systems. Typable lambda terms can be efficiently reduced using Lamping’s abstract algorithm, although basic properties like subject reduction and complexity bounds were not necessarily verified.

Baillot and Terui [6] proposed a type system inspired by light logics and enjoying subject reduction and polynomial time normalization, called Dual Light Affine Logic (**DLAL**). The underlying term system is ordinary lambda-calculus with usual, call-by-name reduction. They’ve recently proved [3] that system **F** terms can be decorated with light types in polynomial time, following similar work for Elementary Affine Logic [7].

Our approach should be understood as complementary to the one proposed by Baillot and Terui [6]: we exploit call-by-value evaluation and this allows us to stay closer to logical systems. On the other hand, the way our type system is formulated prevents us from getting the full power of second-order quantification. Nevertheless, second-order quantification is not as crucial with call-by-value as with usual call-by-name, where data can be encoded in Church-style, following Berarducci and Böhm [8].

## 3. PRELIMINARIES

In this section we recall the proof calculus for Elementary Affine Logic,  $\Lambda^{\text{EA}}$ . Then relations with the lambda calculus will be discussed.

**Definition 3.1** (Terms, Types, Contexts).

- i) The set  $\Lambda$  of terms of the lambda calculus is defined by the grammar  $M ::= x \mid MM \mid \lambda x.M$ , where  $x \in \text{Var}$ , a countable set of variables.
- ii) The grammar generating the set  $\Lambda^{\text{EA}}$  of terms of the Elementary Lambda Calculus (**EA-terms** for short) is obtained from the previous one by adding rules

$$M ::= !(M) \left[ \frac{M}{x}, \dots, \frac{M}{x} \right] \mid [M]_{M=x,y}$$

and by constraining all variables to occur at most once. These two constructs interpret promotion and contraction, respectively.

- iii) **EA-types** are formulae of (Propositional) Elementary Affine Logic (hereby **EAL**), and are generated by the grammar  $A ::= a \mid A \multimap A \mid !A$  where  $a$  belongs to a countable set of basic type constants. **EA-types** will be ranged over by  $A, B, C$ .
- iv) **EA-contexts** are finite subsets of **EA-type** assignments to variables, where all variables are different. Contexts are ranged over by  $\Phi, \Psi$ . If  $\Phi = \{x_1 : A_1, \dots, x_n : A_n\}$ , then  $\text{dom}(\Phi) = \{x_1, \dots, x_n\}$ . Two contexts are *disjoint* if their domains have empty intersection.
- v) The type assignment system in natural-deduction style for **EA-terms** ( $\vdash_{\text{NEAL}}$  for short) assigns **EA-types** to **EA-terms**. The system is given in Table 1. With a slight abuse of notation, we will denote by **NEAL** the set of typable terms in  $\Lambda^{\text{EA}}$ .

Both  $\Lambda^{\text{EA}}$  and  $\Lambda$  are ranged over by  $M, N, P, Q$ . The context should help avoiding ambiguities. Symbol  $\equiv$  denotes syntactic identity on both types and terms. The identity on terms is

$\frac{\Phi, x : A \vdash_{\text{NEAL}} x : A \quad A}{\Phi, x : A \vdash_{\text{NEAL}} M : B} I_{\multimap}$	$\frac{\Phi \vdash_{\text{NEAL}} M : !A \quad \Psi, x : !A, y : !A \vdash_{\text{NEAL}} N : B}{\Phi, \Psi \vdash_{\text{NEAL}} [N]_{M=x, y} : B} C$
$\frac{\Psi_1 \vdash_{\text{NEAL}} M_1 : !A_1 \quad \dots \quad \Psi_n \vdash_{\text{NEAL}} M_n : !A_n \quad x_1 : A_1, \dots, x_n : A_n \vdash_{\text{NEAL}} N : B}{\Phi, \Psi_1, \dots, \Psi_n \vdash_{\text{NEAL}} !(N) [^{M_1/x_1, \dots, M_n/x_n}] : !B} !$	$\frac{\Phi \vdash_{\text{NEAL}} M : A \multimap B \quad \Psi \vdash_{\text{NEAL}} N : A}{\Phi, \Psi \vdash_{\text{NEAL}} M N : B} E_{\multimap}$

Table 1: Type assignment system for **EA**-terms. Contexts with different names are intended to be disjoint.

$(\lambda x. M N) \quad \rightarrow_{\beta} \quad M\{N/x\}$
$[N]_{!(M)[^{M_1/x_1, \dots, M_n/x_n}] = x, y} \quad \rightarrow_{\text{dup}} \quad \dots [N\{^{!(M)[^{x'_1/x_1, \dots, x'_n/x_n}] / x}\{^{!(M')[^{y'_1/y_1, \dots, y'_n/y_n}] / y}\}_{M_1=x'_1, y'_1} \dots\}_{M_n=x'_n, y'_n}}$
$!(M)[^{M_1/x_1, \dots, M_n/x_n}] \quad \rightarrow_{!-c} \quad !(M\{N/x_i\})[^{M_1/x_1, \dots, P_1/y_1, \dots, P_m/y_m, \dots, M_n/x_n}]$
$([M]_{M_1=x_1, x_2} N) \quad \rightarrow_{@-c} \quad [(M\{x'_1/x_1, x'_2/x_2\} N)]_{M_1=x'_1, x'_2}$
$(M [N]_{N_1=x_1, x_2}) \quad \rightarrow_{@-c} \quad [(M N\{x'_1/x_1, x'_2/x_2\})]_{N_1=x'_1, x'_2}$
$!(M)[^{M_1/x_1, \dots, M_i/N=y, z/x_i, \dots, M_n/x_n}] \quad \rightarrow_{!-c} \quad [!(M)[^{M_1/x_1, \dots, M_i\{y'/y, z'/z\}/x_i, \dots, M_n/x_n}]]_{N=y', z'}$
$[M]_{[N]_{P=y_1, y_2=x_1, x_2}} \quad \rightarrow_{c-c} \quad [[M]_{N\{y'_1/y_1, y'_2/y_2\}=x_1, x_2}]_{P=y'_1, y'_2}$
$\lambda x. [M]_{N=y, z} \quad \rightarrow_{\lambda-c} \quad [\lambda x. M]_{N=y, z} \quad \text{where } x \notin \text{FV}(N)$

where  $M'$  in the  $\rightarrow_{\text{dup}}$ -rule is obtained from  $M$  replacing all its free variables with fresh ones ( $x_i$  is replaced with  $y_i$ );  $x'_1$  and  $x'_2$  in the  $\rightarrow_{@-c}$ -rule,  $y'$  and  $z'$  in the  $\rightarrow_{!-c}$ -rule and  $y'_1, y'_2$  in the  $\rightarrow_{c-c}$ -rule are fresh variables.

Table 2: Normalization rules in  $\Lambda^{\text{EA}}$ .

taken modulo names of bound variables and modulo permutation in the list  $M/x, \dots, M/x$  inside  $!(M)[^{M/x, \dots, M/x}]$ .

On  $\Lambda$ , both the call-by-name and the call-by-value  $\beta$ -reduction will be used, according to the following definition.

**Definition 3.2** (Reduction).

- i) We refer to the contextual closure of the rule  $(\lambda x.M)N \rightarrow_n M\{N/x\}$ , where  $M\{N/x\}$  denotes the capture free substitution of  $N$  to the free occurrences of  $x$  in  $M$ , as the *call-by-name  $\beta$ -reduction*;
- ii) *Values* are generated by the grammar  $V ::= x \mid \lambda x.M$  where  $x$  ranges over  $Var$  and  $M$  ranges over  $\Lambda$ .  $\mathcal{V}$  is the set of all values. Values are denoted by  $V, U, W$ . The *call-by-value  $\beta$ -reduction* is the contextual closure of the rule  $(\lambda x.M)V \rightarrow_v M\{V/x\}$  where  $V$  ranges over values.
- iii) Let  $t \in \{n, v\}$ ; symbols  $\rightarrow_t^+$  and  $\rightarrow_t^*$  denote the transitive closure and the symmetric and transitive closure of  $\rightarrow_t$ , respectively.

A term in  $\Lambda^{\mathbf{EA}}$  can be transformed naturally to a term in  $\Lambda$  by performing the substitutions which are explicit in it, and forgetting the modality  $!$ . Formally, the translation function  $(\cdot)^* : \Lambda^{\mathbf{EA}} \rightarrow \Lambda$  is defined by induction on the structure of  $\mathbf{EA}$ -terms as follows:

$$\begin{aligned}
(x)^* &= x \\
(\lambda x.M)^* &= \lambda x.(M)^* \\
(MN)^* &= (M)^*(N)^* \\
([M]_{N=x_1, x_2})^* &= (M)^*\{(N)^*/x_1, (N)^*/x_2\} \\
(! (N) [M_1/x_1, \dots, M_n/x_n])^* &= (N)^*\{(M_1)^*/x_1, \dots, (M_n)^*/x_n\}
\end{aligned}$$

where  $M\{M_1/x_1, \dots, M_n/x_n\}$  denotes the simultaneous substitution of all free occurrences of  $x_i$  by  $M_i$  ( $1 \leq i \leq n$ ).

The map  $(\cdot)^*$  easily induces a type-assignment system  $\mathbf{NEAL}^*$  for pure lambda-calculus: take  $\mathbf{NEAL}$  and replace every occurrence of a term  $M$  by  $M^*$  in every rule. Normalization in  $\mathbf{NEAL}$  (see Table 2), however, is different from normalization in lambda-calculus —  $\mathbf{NEAL}^*$  does not even satisfy subject-reduction. Moreover, lambda calculus does not provide any mechanism for sharing: the argument is duplicated as soon as  $\beta$ -reduction fires. This, in turn, prevents from analyzing normalization in the lambda calculus using the same techniques used in logical systems. This phenomenon has catastrophic consequences in the context of Light Affine Logic, where polynomial time bounds cannot be transferred from the logic to pure lambda-calculus [6].

Consider now a different translation  $(\cdot)^\# : \Lambda^{\mathbf{EA}} \rightarrow \Lambda$ :

$$\begin{aligned}
(x)^\# &= x \\
(\lambda x.M)^\# &= \lambda x.(M)^\# \\
(MN)^\# &= (M)^\# (N)^\# \\
([N]_{M=x, y})^\# &= \begin{cases} (N)^\#\{M/x, M/y\} & \text{if } M \text{ is a variable} \\ (\lambda z.(N)^\#\{z/x, z/y\})(M)^\# & \text{otherwise} \end{cases} \\
(! (N) [M_1/x_1, \dots, M_n/x_n])^\# &= \begin{cases} (N)^\# & \text{if } n = 0 \\ (! (N) [M_2/x_2, \dots, M_n/x_n])^\#\{M_1/x_1\} & \text{if } n \geq 1 \text{ and } M_1 \text{ is a variable} \\ (\lambda x_1.(! (N) [M_2/x_2, \dots, M_n/x_n])^\#)(M_1)^\# & \text{if } n \geq 1 \text{ and } M_1 \text{ is not a variable} \end{cases}
\end{aligned}$$

Please observe that while  $(\cdot)^\#$  maps  $[N]_{M=x,y}$  and  $!(N) [^{M_1/x_1, \dots, M_n/x_n}]$  to applications (except when the arguments are variables),  $(\cdot)^*$  maps them to terms obtained by substitution. Indeed, if lambda calculus is endowed with ordinary  $\beta$ -reduction, the two translations are almost equivalent:

**Lemma 3.3.** *For every EA-term  $M$ ,  $(M)^\# \rightarrow_n^* (M)^*$ .*

*Proof.* By induction on  $M$ . □

However, it is certainly not true that  $(M)^\# \rightarrow_v^* (M)^*$ . The map  $(\cdot)^\#$ , differently from  $(\cdot)^*$ , does not cause an exponential blowup on the length of terms. The *length*  $L(M)$  of a term  $M$  is defined inductively as follows:

$$\begin{aligned} L(x) &= 1 \\ L(\lambda x.M) &= 1 + L(M) \\ L(M N) &= 1 + L(M) + L(N) \end{aligned}$$

The same definition can be extended to EA-terms by way of the following equations:

$$\begin{aligned} L(! (M)) &= L(M) + 1 \\ L(! (M) [^{M_1/x_1, \dots, M_n/x_n}]) &= L(! (M) [^{M_1/x_1, \dots, M_{n-1}/x_{n-1}}]) + L(M_n) + 1 \\ L([M]_{N=x,y}) &= L(M) + L(N) + 1 \end{aligned}$$

**Proposition 3.4.** *For every  $N \in \Lambda^{\text{EA}}$ ,  $L(N^\#) \leq 2L(N)$ .*

*Proof.* By induction on  $N$ . The cases for variables, abstractions and applications are trivial. Let us now consider the other two inductive cases. Suppose  $N = [P]_{Q=x,y}$ . If  $Q$  is a variable, then  $L(N^\#) = L(P^\#) \leq 2L(P) \leq 2L(N)$ . If  $Q$  is not a variable, then  $L(N^\#) = L(P^\#) + L(Q^\#) + 2 \leq 2L(P) + 2L(Q) + 2 = 2(L(P) + L(Q) + 1) = 2L(N)$ . If, on the other hand,  $N = !(M) [^{M_1/x_1, \dots, M_n/x_n}]$ , then we can proceed by induction on  $n$ . If  $n = 0$ , then the inequality is trivially verified. If, on the other hand,  $n > 0$ , then we must distinguish two different cases: if  $M_n$  is a variable, then the inequality is trivially satisfied; if  $M_n$  is not a variable, then  $N^\#$  is  $(\lambda x_n. (! (M) [^{M_1/x_1, \dots, M_{n-1}/x_{n-1}}])^\#) M_n^\#$  and, by the induction hypothesis on  $n$  and  $M_n$ , we get

$$\begin{aligned} L(N^\#) &= 2 + L((! (M) [^{M_1/x_1, \dots, M_{n-1}/x_{n-1}}])^\#) + L(M_n^\#) \\ &\leq 2 + 2L(! (M) [^{M_1/x_1, \dots, M_{n-1}/x_{n-1}}]) + 2L(M_n^\#) \\ &= 2L(N) \end{aligned}$$

This concludes the proof. □

#### 4. THE ELEMENTARY TYPE ASSIGNMENT SYSTEM

In this section we will define a type assignment system typing lambda-terms with **EAL** formulae. We want the system to be *almost* syntax directed, the difficulty being the handling of  $C$  and  $!$  rules. This is solved by splitting the context into three parts, the *linear* context, the *modal* context, and the *parking* context. In particular the parking context is used to keep track of premises which must become modal in the future.

**Definition 4.1.**

- i) An **EAL** formula  $A$  is *modal* if  $A \equiv !B$  for some  $B$ , it is *linear* otherwise.

$\frac{\overline{\Gamma, x : A \mid \Delta \mid \Theta \vdash x : A} \quad A^L}{\Gamma, x : A \mid \Delta \mid \Theta \vdash M : B} \quad I_{\circ}^L$	$\frac{\overline{\Gamma \mid \Delta \mid x : A, \Theta \vdash x : A} \quad A^P}{\Gamma \mid \Delta, x : A \mid \Theta \vdash M : B} \quad I_{\circ}^L$
$\frac{\Gamma_1 \mid \Delta \mid \Theta \vdash M : A \multimap B \quad \Gamma_2 \mid \Delta \mid \Theta \vdash N : A}{\Gamma_1, \Gamma_2 \mid \Delta \mid \Theta \vdash M N : B} \quad E_{\circ}$	
$\frac{\Gamma_1 \mid \Delta_1 \mid \Theta_1 \vdash M : A}{\Gamma_2 \mid !\Gamma_1, !\Delta_1, !\Theta_1, \Delta_2 \mid \Theta_2 \vdash M : !A} \quad !$	

Table 3: The Elementary Type Assignment System (**ETAS**). Contexts with different names are intended to be disjoint.

- ii) A context is a set of pairs  $x : A$  where  $x$  is a variable and  $A$  is an **EA**-type, where all variables are disjoint. A context is *linear* if it assigns linear **EA**-types to variables, while it is *modal* if it assigns modal **EA**-types to variables. If  $\Phi$  is a context,  $\Phi^L$  and  $\Phi^I$  denote the linear and modal sub-contexts of  $\Phi$ , respectively.
- iii) The Elementary Type Assignment System (**ETAS**) proves statements like  $\Gamma \mid \Delta \mid \Theta \vdash M : A$  where  $\Gamma$  and  $\Theta$  are linear contexts and  $\Delta$  is a modal context. The contexts have disjoint variables. The rules of the system are shown in Table 3. In what follows,  $\Gamma$ ,  $\Delta$  and  $\Theta$  will range over linear, modal and parking contexts respectively.
- iv) A *typing judgement* for  $M$  is a statement of the kind  $\Gamma \mid \Delta \mid \Theta \vdash M : A$ . A term  $M \in \Lambda$  is *EA-typable* if there is a typing for it. Type derivations built according to rules in Table 3 will be denoted with greek letters like  $\pi$ ,  $\rho$  and  $\sigma$ . If  $\pi$  is a type derivation with conclusion  $\Gamma \mid \Delta \mid \Theta \vdash M : A$ , we write  $\pi : \Gamma \mid \Delta \mid \Theta \vdash M : A$ .

Rules  $A^L$  and  $A^P$  (see Table 3) are two variations on the classical axiom rule. Notice that a third axiom rule

$$\overline{\Gamma \mid x : !A, \Delta \mid \Theta \vdash x : !A} \quad A^I$$

is derivable. Abstractions cannot be performed on variables in the parking context. The rule  $E_{\circ}$  is the standard rule for application. Rule  $!$  is derived from the one traditionally found in sequent calculi and is weaker than the rule induced by **NEAL** via  $(\cdot)^*$ . Nevertheless, it is sufficient for our purposes and (almost) syntax-directed. The definition of an **EA**-typable term takes into account the auxiliary role of the parking context.

**Example 4.2.** Let us illustrate the rôles of the various **ETAS** rules by way of an example. Consider the Church's numeral  $\underline{2} \equiv \lambda x.\lambda y.x(xy)$ , let  $B$  be  $!(A \multimap A)$  and  $C$  be  $B \multimap B$ . A type derivation for  $\underline{2}$  is the following:

$$\frac{\overline{\emptyset \mid y : B \mid x : C \vdash x : C} \quad A^P \quad \frac{\overline{y : A \multimap A \mid \emptyset \mid \emptyset \vdash y : A \multimap A} \quad A^L}{\emptyset \mid y : B \mid x : C \vdash y : B} \quad !}{\emptyset \mid y : B \mid x : C \vdash xy : B} \quad E_{\circ}}{\overline{\emptyset \mid y : B \mid x : C \vdash xy : B} \quad E_{\circ}} \quad \frac{\overline{\emptyset \mid y : B \mid x : C \vdash x(xy) : B} \quad I_{\circ}^L}{\overline{\emptyset \mid \emptyset \mid x : C \vdash \lambda y.x(xy) : B \multimap B} \quad !} \quad \frac{\overline{\emptyset \mid x : !C \mid \emptyset \vdash \lambda y.x(xy) : !(B \multimap B)} \quad !}{\overline{\emptyset \mid \emptyset \mid \emptyset \vdash \lambda x.\lambda y.x(xy) : !C \multimap !C} \quad I_{\circ}^L} \quad E_{\circ}^P$$





(the other ones can be handled easily). Since  $N \in \mathcal{V}$ , the derivation for  $\Gamma_2 \mid \Delta \mid \Theta \vdash N : A$  must end with  $A^L$ ,  $A^P$ ,  $I_{\rightarrow}^L$  or  $I_{\leftarrow}^L$  (depending on the shape of  $N$ ), followed by exactly  $n$  instances of the  $!$  rule, being it the only non-syntax-directed rule. If the last rule used in  $\pi$  is  $E_{\rightarrow}$ , then  $\pi$  has the following shape:

$$\frac{\phi : \Gamma_3 \mid x : A, \Delta \mid \Theta \vdash L : D \multimap B \quad \psi : \Gamma_4 \mid x : A, \Delta \mid \Theta \vdash P : D}{\Gamma_1 \mid x : A, \Delta \mid \Theta \vdash M : B}$$

where  $\Gamma_1 \equiv \Gamma_3, \Gamma_4$  and  $M \equiv LP$ .  $\sigma$  can be written as follows:

$$\frac{\xi : \Gamma_5 \mid \Delta_1 \mid \Theta_1 \vdash N : C}{\Gamma_2 \mid \Delta \mid \Theta \vdash N : A}$$

where  $\Delta \equiv !\Gamma_5, !\Delta_1, !\Theta_1, \Delta_2$  and  $A \equiv !C$ . From  $\xi$  we can obtain a derivation  $\chi : \emptyset \mid \Delta \mid \Theta \vdash N : A$  and applying (two times) the induction hypothesis, we get  $\mu : \Gamma_3 \mid \Delta \mid \Theta \vdash L\{N/x\} : D \multimap B$  and  $\nu : \Gamma_4 \mid \Delta \mid \Theta \vdash P\{N/x\} : D$  from which we get the desired  $\rho$  by applying rule  $E_{\rightarrow}$  and Lemma 4.3. If the last rule used in  $\pi$  is  $!$ , then  $\pi$  has the following shape:

$$\frac{\phi : \Gamma_3 \mid \Delta_1 \mid \Theta_1 \vdash M : C}{\Gamma_1 \mid x : A, \Delta \mid \Theta \vdash M : B}$$

where  $x : A, \Delta \equiv !\Gamma_3, !\Delta_1, !\Theta_1, \Delta_3$  and  $B \equiv !C$ .  $\sigma$  can be written as follows:

$$\frac{\psi : \Gamma_4 \mid \Delta_2 \mid \Theta_2 \vdash N : D}{\Gamma_2 \mid \Delta \mid \Theta \vdash N : A}$$

where  $\Delta \equiv !\Gamma_4, !\Delta_2, !\Theta_2, \Delta_4$  and  $A \equiv !D$ . We now distinguish some cases:

- If  $x \in \text{dom}(\Delta_3)$ , then  $x \notin FV(M)$  and  $\rho$  is obtained easily from  $\phi$ .
- If  $x \in \text{dom}(\Delta_1)$ , then let  $\Delta_1 \equiv x : D, \Delta_5$ . By applying several times Lemma 4.3 and Lemma 4.4 we can obtain type derivations

$$\begin{aligned} \xi & : \emptyset \mid x : D, \Delta_2 \cup \Delta_5 \mid \Gamma_3 \cup \Gamma_4 \cup \Theta_1 \cup \Theta_2 \vdash M : C \\ \chi & : \emptyset \mid \Delta_2 \cup \Delta_5 \mid \Gamma_3 \cup \Gamma_4 \cup \Theta_1 \cup \Theta_2 \vdash N : D \end{aligned}$$

which have the same number of rule instances as  $\phi$  and  $\psi$ , respectively. By applying point ii) of this Lemma, we obtain

$$\mu : \emptyset \mid \Delta_2 \cup \Delta_5 \mid \Gamma_3 \cup \Gamma_4 \cup \Theta_1 \cup \Theta_2 \vdash M\{N/x\} : C$$

from which  $\rho$  can be easily obtained.

- If  $x \in \text{dom}(\Theta_1)$ , then let  $\Theta_1 \equiv x : D, \Theta_3$ . By applying several times Lemma 4.3 and Lemma 4.4 we can obtain type derivations

$$\begin{aligned} \xi & : \emptyset \mid \Delta_1 \cup \Delta_2 \mid x : D, \Gamma_3 \cup \Gamma_4 \cup \Theta_2 \cup \Theta_5 \vdash M : C \\ \chi & : \emptyset \mid \Delta_1 \cup \Delta_2 \mid \Gamma_3 \cup \Gamma_4 \cup \Theta_2 \cup \Theta_5 \vdash N : D \end{aligned}$$

which have the same number of rule instances as  $\phi$  and  $\psi$ , respectively. By applying the inductive hypothesis, we obtain

$$\mu : \emptyset \mid \Delta_1 \cup \Delta_2 \mid \Gamma_3 \cup \Gamma_4 \cup \Theta_2 \cup \Theta_5 \vdash M\{N/x\} : C$$

from which  $\rho$  can be easily obtained.

- If  $x \in \text{dom}(\Gamma_3)$ , then let  $\Gamma_3 \equiv x : D, \Gamma_5$ . By applying several times Lemma 4.3 and Lemma 4.4 we can obtain type derivations

$$\begin{aligned} \xi & : \emptyset \mid \Delta_1 \cup \Delta_2 \mid x : D, \Gamma_4 \cup \Gamma_5 \cup \Theta_1 \cup \Theta_2 \vdash M : C \\ \chi & : \emptyset \mid \Delta_1 \cup \Delta_2 \mid \Gamma_4 \cup \Gamma_5 \cup \Theta_1 \cup \Theta_2 \vdash N : D \end{aligned}$$

which have the same number of rule instances as  $\phi$  and  $\psi$ , respectively. By applying the inductive hypothesis, we obtain

$$\mu : \emptyset \mid \Delta_1 \cup \Delta_2 \mid \Gamma_4 \cup \Gamma_5 \cup \Theta_1 \cup \Theta_2 \vdash M\{N/x\} : C$$

from which  $\rho$  can be easily obtained.

This concludes the proof.  $\square$

**Theorem 4.6** (Call-by-Value Subject Reduction).  $\Gamma \mid \Delta \mid \Theta \vdash M : A$  and  $M \rightarrow_v N$  implies  $\Gamma \mid \Delta \mid \Theta \vdash N : A$ .

*Proof.* A redex is a term of the shape  $(\lambda x.M')N'$ , where  $N' \in \mathcal{V}$ . Then it can be the subject of a subderivation ending by an application of the rule  $(E_{-\circ})$  immediately preceded by an application of rule  $(I_{-\circ})$ . So the result follows by the Substitution Lemma.  $\square$

We are now going to prove that the set of typable  $\lambda$ -terms coincides with  $(\mathbf{NEAL})^\#$ . To do this we need the following lemma.

**Lemma 4.7** (Contraction Lemma).

- i) If  $\Gamma \mid \Delta \mid x : A, y : A, \Theta \vdash M : B$ , then  $\Gamma \mid \Delta \mid z : A, \Theta \vdash M\{z/x, z/y\} : B$
- ii) If  $\Gamma \mid x : A, y : A, \Delta \mid \Theta \vdash M : B$ , then  $\Gamma \mid z : A, \Delta \mid \Theta \vdash M\{z/x, z/y\} : B$

**Theorem 4.8.**

- i) If  $\Phi \vdash_{\mathbf{NEAL}} M : A$  then  $\Phi^L \mid \Phi^I \mid \emptyset \vdash (M)^\# : A$ .
- ii) If  $\Gamma \mid \Delta \mid \emptyset \vdash M : A$ , there is  $N \in \Lambda^{\mathbf{EA}}$  such that  $(N)^\# = M$  and  $\Gamma, \Delta \vdash_{\mathbf{NEAL}} N : A$ .

*Proof.*

- i) By induction on the structure of the derivation for  $\Phi \vdash_{\mathbf{NEAL}} M : A$ . Let us focus on nontrivial cases.

If the last used rule is  $E_{-\circ}$ , the two premises are  $\Phi \vdash_{\mathbf{NEAL}} N : B \multimap C$  and  $\Phi_2 \vdash_{\mathbf{NEAL}} P : B$ , and  $M \equiv NP$ . By induction hypothesis,  $\Phi_1^L \mid \Phi_1^I \mid \emptyset \vdash (N)^\# : B \multimap C$ , and  $\Phi_2^L \mid \Phi_2^I \mid \emptyset \vdash (P)^\# : B$  and, by Weakening Lemma,  $\Phi_1^L \mid \Phi_1^I, \Phi_2^I \mid \emptyset \vdash (N)^\# : B \multimap C$ ,  $\Phi_2^L \mid \Phi_1^I, \Phi_2^I \mid \emptyset \vdash (P)^\# : B$  Rule  $E_{-\circ}$  leads to the thesis.

If the last used rule is  $C$ , the two premises are  $\Phi_1 \vdash_{\mathbf{NEAL}} N : !A$  and  $\Phi_2, x : !A, y : !A \vdash_{\mathbf{NEAL}} P : B$ . By induction hypothesis,  $\Phi_1^L \mid \Phi_1^I \mid \emptyset \vdash (N)^\# : !A$ ,  $\Phi_2^L \mid \Phi_2^I, x : !A, y : !A \mid \emptyset \vdash (P)^\# : B$ . By Contraction Lemma,  $\Phi_2^L \mid \Phi_2^I, z : !A \mid \emptyset \vdash (P)^\#\{z/x, z/y\} : B$  and so  $\Phi_2^L \mid \Phi_2^I \mid \emptyset \vdash \lambda z.(P)^\#\{z/x, z/y\} : !A \multimap B$  By rule  $E_{-\circ}$  and Weakening Lemma, we finally get  $\Phi_1^L, \Phi_2^L \mid \Phi_1^I, \Phi_2^I \mid \emptyset \vdash (\lambda z.(P)^\#\{z/x, z/y\})(N)^\# : B$ .

- ii) The following, stronger, statement can be proved by induction on  $\pi$ : if  $\pi : \Gamma \mid \Delta \mid x_1 : A_1, \dots, x_n : A_n \vdash M : A$ , then there is  $N \in \Lambda^{\mathbf{EA}}$  such that

$$M = (N)^\#\{x_1^1/y_1, \dots, x_1^{m_1}/y_1, \dots, x_n^1/y_n, \dots, x_n^{m_n}/y_n\}$$

and  $\Gamma, \Delta, y_1^1 : A_1, \dots, y_1^{m_1} : A_1, \dots, y_n^1 : A_n, \dots, y_n^{m_n} : A_n \vdash_{\mathbf{NEAL}} N : A$ .  $\square$

We have just established a deep *static* correspondence between  $\mathbf{NEAL}$  and the class of typable lambda terms. But what about *dynamics*? Unfortunately, the two systems are not bisimilar. Nevertheless, every call-by-value reduction-step in the lambda calculus corresponds to at least one normalization step in  $\Lambda^{\mathbf{EA}}$ . A normalization step in  $\Lambda^{\mathbf{EA}}$  is denoted by  $\rightarrow$ ;  $\rightarrow^+$  denotes the transitive closure of  $\rightarrow$ .

An *expansion* is a term in  $\Lambda^{\mathbf{EA}}$  that can be written either as  $!(M)[x_1/y_1, \dots, x_n/y_n]$  or as  $[N]_{z=x,y}$ , where  $N$  is itself an expansion.

**Lemma 4.9.** *If  $M$  is an expansion, then*

- $[L]_{M=x,y} \rightarrow^* P$ , where  $P^\# \equiv L\{M^\#/x, M^\#/y\}$ ;
- If  $M \equiv P_i$ , then  $!(L) [P_1/x_1, \dots, P_n/x_n] \rightarrow^* Q$  where

$$Q^\# \equiv (!(L) [P_1/x_1, \dots, P_{i-1}/x_{i-1}, P_{i+1}/x_{i+1}, \dots, P_n/x_n])^\# \{M^\#/x_i\}.$$

**Proposition 4.10.** *For every  $M \in \Lambda^{\text{EA}}$ , if  $\Gamma \vdash_{\text{NEAL}} M : A$  and  $(M)^\# \rightarrow_v N$ , then there is  $L \in \Lambda^{\text{EA}}$  such that  $(L)^\# = N$  and  $M \rightarrow^+ L$ .*

*Proof.* We can proceed by induction on the structure of  $M$ . If  $M$  is a variable, then  $M^\#$  is a variable, too, and so the premise is false. If  $M$  is an abstraction, then the thesis follows from the inductive hypothesis. If  $M$  is an application  $P Q$ , then we can assume  $P$  to be an abstraction  $\lambda x.R$  and  $N$  to be  $R^\#\{Q^\#/x\}$  (in all the other cases the thesis easily follows by induction hypothesis). It is easy to see that  $R^\#\{Q^\#/x\} \equiv (R\{Q/x\})^\#$  and so we can take  $R\{Q/x\}$  as our  $L$ . If  $M$  is  $[P]_{Q=x,y}$  and  $Q$  is not a variable (otherwise the thesis easily follows by inductive hypothesis), then  $M^\# = (\lambda z.P^\#\{z/x, z/y\})Q^\#$  and we can restrict to the case where  $N$  is  $P^\#\{Q^\#/x, Q^\#/y\}$ . First of all, we can observe that  $Q^\#$  must be an abstraction. This means that  $Q$  is an abstraction itself enclosed in one or more  $!(\cdot) [x_1/y_1, \dots, x_n/y_n]$  contexts and zero or more  $[\cdot]_{z=x,y}$  otherwise  $M$  cannot be typed in **EAL**. This means  $Q$  is an expansion and so, by Lemma 4.9, we know there must be a term  $R$  such that  $R^\# \equiv P^\#\{Q^\#/x, Q^\#/y\}$ , and  $M \rightarrow^* R$ , that is the thesis.  $!(P) [Q_1/x_1, \dots, Q_n/x_n]$  can be managed in a similar way.  $\square$

**Remark 4.11.** Notice that Proposition 4.10 is not a bisimulation result. In particular, there are normalization steps in **NEAL** that do not correspond to anything in **ETAS**. An example is the term  $(\lambda x.x)(yz)$ , which rewrites to  $yz$  in **NEAL** but is a (call-by-value) normal form as a pure lambda-term.

## 5. BOUNDS ON NORMALIZATION TIME

In order to prove elementary bounds on reduction sequences, we need to define a refined measure on lambda terms. We can look at a type derivation  $\pi : \Gamma \mid \Delta \mid \Theta \vdash M : A$  as a labelled tree, where every node is labelled by a rule instance. We can give the following definition:

**Definition 5.1.** Let  $\pi : \Gamma \mid \Delta \mid \Theta \vdash M : A$ .

- i) An occurrence of a subderivation  $\rho$  of  $\pi$  has *level*  $i$  if there are  $i$  applications of the rule  $!$  in the path from the root of  $\rho$  to the root of  $\pi$ .
- ii) An occurrence of a subterm  $N$  of  $M$  has *level*  $i$  in  $\pi$  if  $i$  is the maximum level of a subderivation of  $\pi$  corresponding to the particular occurrence of  $N$  under consideration (and thus having  $N$  as subject).
- iii) The level  $\partial(\pi)$  of  $\pi$  is the maximum level of subderivations of  $\pi$ .

Notice that the so defined level corresponds to the notion of box-nesting depth in proof-nets [1].

**Example 5.2.** Consider the derivation  $\pi(2, B)$  from Example 4.2. All the occurrences of rules  $A^P$  inside  $\pi(2, B)$  have level 1, since there is one instance of  $!$  in the path joining the root of  $\pi(2, B)$  to them. The occurrence of rule  $A^L$ , on the other hand, has level 2. As a consequence all occurrence of variables in  $\underline{2}$  have either level 1 or level 2 in  $\pi(2, B)$ . Now,

consider the (unique) occurrence of  $\lambda y.x(xy)$  in  $\underline{2}$ . There are two distinct subderivations corresponding to it, one with level 0, the other with level 1. As a consequence, the level of  $\lambda y.x(xy)$  in  $\pi(2, B)$  is 1. Since the maximum level of subderivations of  $\pi(2, B)$  is 2,  $\partial(\pi(2, B)) = 2$ .

The length  $L(M)$  of a typable lambda term  $M$  does not take into account levels as we have just defined them. The following definitions reconcile them, allowing  $L(M)$  to be “split” on different levels.

**Definition 5.3.** Let  $\pi : \Gamma \mid \Delta \mid \Theta \vdash M : A$ .

- i)  $S(\pi, i)$  is defined by induction on  $\pi$  as follows:
  - If  $\pi$  consists of an axiom, then  $S(\pi, 0) = 1$  and  $S(\pi, i) = 0$  for every  $i \geq 1$ ;
  - If the last rule in  $\pi$  is  $I_{\circ}^L$  or  $I_{\circ}^L$ , then  $S(\pi, 0) = S(\rho, 0) + 1$  and  $S(\pi, i) = S(\rho, i)$  for every  $i \geq 1$ , where  $\rho$  is the immediate subderivation of  $\pi$ ;
  - If the last rule in  $\pi$  is  $E_{\circ}$  then  $S(\pi, 0) = S(\rho, 0) + S(\sigma, 0) + 1$  and  $S(\pi, i) = S(\rho, i) + S(\sigma, i)$  for every  $i \geq 1$ , where  $\rho$  and  $\sigma$  are the immediate subderivations of  $\pi$ ;
  - If the last rule in  $\pi$  is  $!$ , then  $S(\pi, 0) = 0$  and  $S(\pi, i) = S(\rho, i - 1)$  for every  $i \geq 1$ , where  $\rho$  is the immediate subderivation of  $\pi$ .
- ii) Let  $n$  be the level of  $\pi$ . The *size* of  $\pi$  is  $S(\pi) = \sum_{i \leq n} S(\pi, i)$ .

**Example 5.4.** Consider again  $\pi(2, B)$ . By definition,  $S(\pi(2, B), 2) = 1$ ,  $S(\pi(2, B), 1) = 5$  and  $S(\pi(2, B), 0) = 1$ .

The following relates  $S(\pi)$  to the size of the term  $\pi$  types:

**Lemma 5.5.** *Let  $\pi : \Gamma \mid \Delta \mid \Theta \vdash M : A$ . Then,  $S(\pi) = L(M)$ .*

Substitution Lemma can be restated in the following way:

**Lemma 5.6** (Weakening Lemma, revisited). *If  $\pi : \Gamma_1 \mid \Delta_1 \mid \Theta_1 \vdash M : A$ , then there is  $\rho : \Gamma_1, \Gamma_2 \mid \Delta_1, \Delta_2 \mid \Theta_1, \Theta_2 \vdash M : A$ . such that  $S(\pi, i) = S(\rho, i)$  for every  $i$ .*

**Lemma 5.7** (Shifting Lemma, revisited). *If  $\pi : \Gamma, x : A \mid \Delta \mid \Theta \vdash M : B$ , then there is  $\rho : \Gamma \mid \Delta \mid x : A, \Theta \vdash M : B$  such that  $S(\pi, i) = S(\rho, i)$  for every  $i$ .*

**Lemma 5.8** (Substitution Lemma, revisited).

- i) *If  $\pi : \Gamma_1, x : A \mid \Delta \mid \Theta \vdash M : B$ ,  $\rho : \Gamma_2 \mid \Delta \mid \Theta \vdash N : A$  and  $N \in \mathcal{V}$ , then there is  $\sigma : \Gamma_1, \Gamma_2 \mid \Delta \mid \Theta \vdash M\{N/x\} : B$  such that  $S(\sigma, i) \leq S(\rho, i) + S(\pi, i)$  for every  $i$ .*
- ii) *If  $\pi : \Gamma \mid \Delta \mid x : A, \Theta \vdash M : B$ ,  $\rho : \emptyset \mid \Delta \mid \Theta \vdash N : A$  and  $N \in \mathcal{V}$ , then there is  $\sigma : \Gamma \mid \Delta \mid \Theta \vdash M\{N/x\} : B$  such that  $S(\sigma, i) \leq S(\pi, 0)S(\rho, i) + S(\pi, i)$  for every  $i$ .*
- iii) *If  $\pi : \Gamma_1 \mid \Delta, x : A \mid \Theta \vdash M : B$ ,  $\rho : \Gamma_2 \mid \Delta \mid \Theta \vdash N : A$  and  $N \in \mathcal{V}$ , then there is  $\sigma : \Gamma_1, \Gamma_2 \mid \Delta \mid \Theta \vdash M\{N/x\} : B$  such that  $S(\sigma, 0) \leq S(\pi, 0)$  and  $S(\sigma, i) \leq (\sum_{j \leq i} S(\pi, j))S(\rho, i) + S(\pi, i)$  for every  $i \geq 1$ .*

*Proof.* For each of the three claims, we can go by induction on the structure of  $\pi$ . Here, we do not concentrate on proving the existence of  $\sigma$  (it follows from lemma 4.5) but on proving that  $\sigma$  satisfies the given bounds. We implicitly use Lemma 5.6 and Lemma 5.7 without explicitly citing them. Let us first analyze the claim i). We will prove by induction on  $\pi$  that  $S(\sigma, i) \leq S(\rho, i) \min\{1, S(\pi, 0)\} + S(\pi, i)$  for every  $i$  (observe that  $S(\rho, i) \min\{1, S(\pi, 0)\} + S(\pi, i) \leq S(\rho, i) + S(\pi, i)$ ). If  $\pi$  is just an axiom, then  $\sigma$  is obtained from  $\rho$  by the weakening lemma and the bound holds. If the last rule in  $\pi$  is  $I_{\circ}^L$  ( $I_{\circ}^L$ ), then  $\sigma$  is obtained by using

the inductive hypothesis on the immediate premise  $\phi$  of  $\pi$  obtaining  $\psi$  and then applying  $I_{\rightarrow}^L$  ( $I_{\rightarrow}^I$ ) to  $\psi$ . In both cases

$$\begin{aligned} S(\sigma, 0) &= S(\psi, 0) + 1 \leq S(\rho, 0) \min\{1, S(\phi, 0)\} + S(\phi, 0) + 1 \\ &\leq S(\rho, 0) \min\{1, S(\pi, 0)\} + S(\pi, 0) \\ \forall i \geq 1. S(\sigma, i) &= S(\psi, i) \leq S(\rho, i) \min\{1, S(\phi, i)\} + S(\phi, i) \\ &\leq S(\rho, 0) \min\{1, S(\pi, i)\} + S(\pi, i) \end{aligned}$$

If the last rule in  $\pi$  is  $E_{\rightarrow}$ , then  $\sigma$  is obtained by using the inductive hypothesis on one of the immediate premises  $\phi$  of  $\pi$  obtaining  $\psi$ , applying  $E_{\rightarrow}$  to  $\psi$  and the other premise  $\xi$  of  $\pi$ . We have:

$$\begin{aligned} S(\sigma, 0) &= S(\psi, 0) + S(\xi, 0) + 1 \\ &\leq S(\rho, 0) \min\{1, S(\phi, 0)\} + S(\phi, 0) + S(\xi, 0) + 1 \\ &\leq S(\rho, 0) \min\{1, S(\pi, 0)\} + S(\pi, 0) \\ \forall i \geq 1. S(\sigma, i) &= S(\psi, i) + S(\xi, i) \\ &\leq S(\rho, i) \min\{1, S(\phi, i)\} + S(\phi, i) + S(\xi, i) \\ &\leq S(\rho, i) \min\{1, S(\pi, i)\} + S(\pi, i) \end{aligned}$$

If the last rule in  $\pi$  is  $!$ , then  $\sigma$  is just obtained from  $\pi$  by weakening lemma, because  $x$  cannot appear free in  $M$ . The inequality easily follows.

Let us now prove point ii). If  $\pi$  is just an axiom, we can proceed as previously. If the last rule in  $\pi$  is  $I_{\rightarrow}^L$  ( $I_{\rightarrow}^I$ ), then  $\rho$  is obtained as in point i) and, in both cases:

$$\begin{aligned} S(\sigma, 0) &= S(\psi, 0) + 1 \leq S(\rho, 0)S(\phi, 0) + S(\phi, 0) + 1 \\ &\leq S(\rho, 0)S(\pi, 0) + S(\pi, 0) \\ \forall i \geq 1. S(\sigma, i) &= S(\psi, i) \leq S(\rho, i)S(\phi, 0) + S(\phi, i) \\ &\leq S(\rho, i)S(\pi, 0) + S(\pi, i) \end{aligned}$$

If the last rule in  $\pi$  is  $E_{\rightarrow}$ , then  $\sigma$  is obtained by using the inductive hypothesis on both the immediate premises  $\phi$  and  $\psi$  of  $\pi$  obtaining  $\xi$  and  $\chi$  and applying  $E_{\rightarrow}$  to them. We obtain:

$$\begin{aligned} S(\sigma, 0) &= S(\xi, 0) + S(\chi, 0) + 1 \\ &\leq (S(\rho, 0)S(\phi, 0) + S(\phi, 0)) + (S(\rho, 0)S(\psi, 0) + S(\psi, 0)) + 1 \\ &\leq S(\rho, 0)(S(\phi, 0) + S(\psi, 0) + 1) + (S(\phi, 0) + S(\psi, 0) + 1) \\ &= S(\rho, 0)S(\pi, 0) + S(\pi, 0) \\ \forall i \geq 1. S(\sigma, i) &= S(\xi, i) + S(\chi, i) \\ &\leq (S(\rho, i)S(\phi, 0) + S(\phi, i)) + (S(\rho, i)S(\psi, 0) + S(\psi, i)) \\ &= S(\rho, i)(S(\phi, 0) + S(\psi, 0) + 1) + (S(\phi, i) + S(\psi, i)) \\ &= S(\rho, i)S(\pi, 0) + S(\pi, i) \end{aligned}$$

If the last rule in  $\pi$  is  $!$ , the  $\sigma$  is again obtained by  $\pi$  and the inequality follows.

Let us now prove claim iii). Notice that the last rule in  $\rho$  must be  $!$  rule, because  $A$  is modal and  $N$  is a value. If the last rule in  $\pi$  is  $I_{\rightarrow}^L$  ( $I_{\rightarrow}^I$ ), then  $\sigma$  is obtained in the usual way and:

$$\begin{aligned}
S(\sigma, 0) &= S(\psi, 0) + 1 \leq S(\phi, 0) + 1 = S(\pi, 0) \\
\forall i \geq 1. S(\sigma, i) &= S(\psi, i) \leq S(\rho, i) \left( \sum_{j \leq i} S(\phi, j) \right) + S(\phi, i) \\
&= S(\phi, i) \left( \sum_{j \leq i} S(\pi, j) \right) + S(\pi, i) \quad \forall i \geq 1
\end{aligned}$$

If the last rule in  $\pi$  is  $E_{\rightarrow}$ , then we apply the inductive hypothesis to the immediate premises  $\phi$  and  $\psi$  of  $\pi$  and to a type derivation which is structurally equivalent to  $\rho$ . We obtain  $\xi$  and  $\chi$  and apply  $E_{\rightarrow}$  to them, obtaining a type derivation which is structurally equivalent to the desired  $\sigma$ . Now we have:

$$\begin{aligned}
S(\sigma, 0) &= S(\xi, 0) + S(\chi, 0) + 1 \leq S(\phi, 0) + S(\psi, 0) + 1 = S(\pi, 0) \\
\forall i \geq 1. S(\sigma, i) &= S(\xi, i) + S(\chi, i) \\
&\leq S(\rho, i) \left( \sum_{j \leq i} S(\phi, j) \right) + S(\phi, i) + S(\rho, i) \left( \sum_{j \leq i} S(\psi, j) \right) + S(\psi, i) \\
&= S(\rho, i) \left( \sum_{j \leq i} (S(\phi, j) + S(\psi, j)) \right) + S(\phi, i) + S(\psi, i) \\
&= S(\rho, i) \left( \sum_{j \leq i} S(\pi, j) \right) + S(\pi, i)
\end{aligned}$$

If the last rule in  $\pi$  is  $!$ , then we can suppose the last rule in  $\rho$  to be a  $!$  and let  $\psi$  be the immediate premise of  $\rho$ . We first apply the induction hypothesis (or one of the other two claims) to the immediate premise  $\phi$  of  $\pi$  and to  $\psi$  obtaining  $\xi$ ; then, we apply rule  $!$  to  $\xi$  and we get  $\sigma$ . Clearly,  $S(\sigma, 0) = 0$  by definition. For every  $i \geq 0$ , we have that

$$S(\xi, i) \leq \left( \sum_{j \leq i} S(\phi, j) \right) S(\psi, i) + S(\phi, i)$$

independently on the way we get  $\xi$ . Indeed,

$$\begin{aligned}
\min\{1, \S(\phi, i)\} S(\psi, i) + S(\phi, i) &\leq \left( \sum_{j \leq i} S(\phi, j) \right) S(\psi, i) + S(\phi, i); \\
S(\phi, 0) S(\psi, i) + S(\phi, i) &\leq \left( \sum_{j \leq i} S(\phi, j) \right) S(\psi, i) + S(\phi, i).
\end{aligned}$$

As a consequence, for every  $i \geq 1$ ,

$$\begin{aligned}
S(\sigma, i) &= S(\xi, i-1) \leq \left( \sum_{j \leq i-1} S(\phi, j) \right) S(\psi, i-1) + S(\phi, i-1) \\
&\leq \left( \sum_{j \leq i} S(\pi, j) \right) S(\rho, i) + S(\pi, i)
\end{aligned}$$

This concludes the proof. □

The following can be thought of as a strengthening of subject reduction and is a corollary of Lemma 5.8.

**Proposition 5.9.** *If  $\pi : \Gamma \mid \Delta \mid \Theta \vdash M : A$ , and  $M \rightarrow_v N$  by reducing a redex at level  $i$  in  $\pi$ , then there is  $\rho : \Gamma \mid \Delta \mid \Theta \vdash N : A$  such that*

$$\begin{aligned} \forall j < i. S(\rho, j) &= S(\pi, j) \\ S(\rho, i) &< S(\pi, i) \\ \forall j > i. S(\rho, j) &\leq S(\pi, j) \left( \sum_{k \leq j} S(\pi, k) \right) \end{aligned}$$

*Proof.* Type derivation  $\rho$  is identical to  $\pi$  up to level  $i$ , so the equality  $S(\rho, j) = S(\pi, j)$  holds for all levels  $j < i$ . At levels  $j \geq i$ , the only differences between  $\rho$  and  $\pi$  are due to the replacement of a type derivation  $\phi$  for  $(\lambda x.L)P$  with another type derivation  $\psi$  for  $L\{P/x\}$ .  $\psi$  is obtained by Lemma 5.8. The needed inequalities follow from the ones in Lemma 5.8.  $\square$

If  $\pi$  is obtained from  $\rho$  by reducing a redex at level  $i$  as in Proposition 5.9, then we will write  $\pi \rightarrow_v^i \rho$ . Consider now a term  $M$  and a derivation  $\pi : \Gamma \mid \Delta \mid \Theta \vdash M : A$ . By Proposition 5.9, every reduction sequence  $M \rightarrow_v N \rightarrow_v L \rightarrow_v \dots$  can be put in correspondence with a sequence  $\pi \rightarrow_v^i \rho \rightarrow_v^j \sigma \rightarrow_v^k \dots$  (where  $\rho$  types  $N$ ,  $\sigma$  types  $L$ , etc.). The following result give bounds on the lengths of these sequences and on the possible growth during normalization.

**Proposition 5.10.** *For every  $d \in \mathbb{N}$ , there are elementary functions  $f_d, g_d : \mathbb{N} \rightarrow \mathbb{N}$  such that, for every sequence*

$$\pi_0 \rightarrow_v^{i_0} \pi_1 \rightarrow_v^{i_1} \pi_2 \rightarrow_v^{i_2} \dots$$

*it holds that*

- For every  $i \in \mathbb{N}$ ,  $\sum_{e \leq d} S(\pi_i, e) \leq f_d(S(\pi_0))$ ;
- There are at most  $g_d(S(\pi_0))$  reduction steps with indices  $e \leq d$ .

*Proof.* We go by induction on  $d$  and define  $f_d$  and  $g_d$  such that the given inequalities hold and, additionally,  $f_d(n) \geq n$  for each  $n \in \mathbb{N}$ .  $f_0$  and  $g_0$  are both the identity on natural numbers, because  $S(\pi_0, 0)$  can only decrease during reduction and it can do that at most  $S(\pi_0, 0)$  times. Consider now  $d \geq 1$ . Each time  $S(\pi_i, d)$  grows, its value goes from  $S(\pi_i, d)$  to at most  $S(\pi_i, d)(S(\pi_i, d) + f_{d-1}(S(\pi_0)))$ , because by Proposition 5.9 it can grow to, at most  $S(\pi_i, d)(\sum_{k \leq d} S(\pi_i, k))$  and, by inductive hypothesis

$$\sum_{k \leq d} S(\pi_i, k) = S(\pi_i, d) + \sum_{k \leq d-1} S(\pi_i, k) \leq S(\pi_i, d) + f_{d-1}(S(\pi_0)).$$

We claim that after having increased  $n$  times,  $S(\pi_i, d)$  is at most  $(f_{d-1}(S(\pi_0)) + n)^{2^{n+1}}$ . Indeed, initially

$$S(\pi_i, d) \leq S(\pi_0, d) \leq S(\pi_0) \leq (f_{d-1}(S(\pi_0)))^2$$

And, after  $n \geq 1$  increases,

$$\begin{aligned} S(\pi_i, d) &\leq (f_{d-1}(S(\pi_0)) + n - 1)^{2^n} ((f_{d-1}(S(\pi_0)) + n - 1)^{2^n} + f_{d-1}(S(\pi_0))) \\ &\leq (f_{d-1}(S(\pi_0)) + n)^{2^n} ((f_{d-1}(S(\pi_0)) + n - 1)^{2^n} + (f_{d-1}(S(\pi_0)) + n - 1)^{2^{n-1}}) \\ &\leq (f_{d-1}(S(\pi_0)) + n)^{2^n} ((f_{d-1}(S(\pi_0)) + n - 1 + 1)^{2^{n-1}})^2 \\ &= (f_{d-1}(S(\pi_0)) + n)^{2^{n+1}} \end{aligned}$$

From the above discussion, it follows that the functions

$$\begin{aligned} f_d(n) &= f_{d-1}(n) + (f_{d-1}(S(\pi_0)) + g_{d-1}(n))^{2^{g_{d-1}(n)+1}} \\ g_d(n) &= g_{d-1}(n) + \sum_{i=0}^{g_{d-1}(n)} (f_{d-1}(S(\pi_0)) + i)^{2^{i+1}} \end{aligned}$$

are elementary and satisfy the conditions above. This concludes the proof.  $\square$

**Theorem 5.11.** *For every  $d \in \mathbb{N}$  there are elementary functions  $p_d, q_d : \mathbb{N} \rightarrow \mathbb{N}$  such that whenever  $\pi : \Gamma \mid \Delta \mid \Theta \vdash M : A$ , the length of call-by-value reduction sequences starting from  $M$  is at most  $p_{\partial(\pi)}(L(M))$  and the length of any reduct of  $M$  is at most  $q_{\partial(\pi)}(L(M))$ .*

*Proof.* This is immediate from Proposition 5.10.  $\square$

## 6. TYPE INFERENCE

We prove, in a constructive way, that the type inference problem for **ETAS** is decidable. Namely a type inference algorithm is designed such that, for every lambda term  $M$  it produces a *principal typing* from which all and only its typings can be obtained by a suitable substitution. The substitution is a partial function, defined if it satisfies a set of linear constraints. If there is not a substitution defined on its principal typing, then  $M$  is not typable. We will also prove that the computational complexity of the type inference procedure is of the same order as the type inference for simple type assignment system.

The design of the algorithm is based on the following Generation Lemma.

**Lemma 6.1** (Generation Lemma). *Let  $\Gamma \mid \Delta \mid \Theta \vdash M : A$ .*

- i) *Let  $M \equiv x$ . If  $A$  is linear, then either  $\{x : A\} \subseteq \Gamma$  or  $\{x : A\} \subseteq \Theta$ . Otherwise,  $\{x : A\} \in \Delta$ .*
- ii) *Let  $M \equiv \lambda x.N$ . Then  $A$  is of the shape  $\underbrace{! \dots !}_n (B \multimap C)$  ( $n \geq 0$ ).*
  - *Let  $n = 0$ . If  $B$  is linear then  $\Gamma, x : B \mid \Delta \mid \Theta \vdash N : C$ , else  $\Gamma \mid \Delta, x : B \mid \Theta \vdash N : C$ .*
  - *Let  $n > 0$ . Then  $\emptyset \mid \Delta \mid \emptyset \vdash M : A$  and  $\Gamma' \mid \Delta' \mid \Theta' \vdash N : C$ , where  $\Delta = \underbrace{! \dots !}_n ((\Gamma' \cup \Delta' \cup \Theta') - \{x : B\})$ .*
- iii) *Let  $M \equiv PQ$ . Then  $A$  is of the shape  $\underbrace{! \dots !}_n B$  ( $n \geq 0$ ),  $\Gamma_1 \mid \Delta' \mid \Theta' \vdash P : C \multimap \underbrace{! \dots !}_m B$  and  $\Gamma_2 \mid \Delta' \mid \Theta' \vdash Q : C$ , for some  $m \leq n$ .*
  - (a) *If  $n - m = 0$ , then  $\Gamma = \Gamma_1 \cup \Gamma_2$ ,  $\Delta = \Delta'$  and  $\Theta = \Theta'$ .*
  - (b) *If  $n - m > 0$ , then  $\emptyset \mid \Delta \mid \emptyset \vdash M : A$ , where  $\Delta = \underbrace{! \dots !}_{n-m} (\Gamma_1 \cup \Gamma_2 \cup \Delta' \cup \Theta')$ .*

The principal typing is described through the notion of a *type scheme*, which is an extension of that one used in [11] in the context of  $\Lambda^{\text{EA}}$  and **NEAL**. Roughly speaking, a type scheme describes a class of types, i.e. it can be transformed into a type through a suitable notion of a substitution.

**Definition 6.2.**



i) *Linear schemes* and *schemes* are respectively defined by the grammars

$$\begin{aligned}\mu &::= \alpha \mid \sigma \multimap \sigma \\ \sigma &::= \mu \mid !^p \mu.\end{aligned}$$

where  $\alpha$  belongs to a countable set of scheme variables and the *exponential*  $p$  is defined by the grammar

$$p ::= a \mid p + p$$

where  $a$  ranges over a countable set of *literals*. Linear schemes are ranged over by  $\mu, \nu$ , schemes are ranged over by  $\sigma, \tau, \rho$ , exponentials are ranged over by  $p, q, r$  and literals are ranged over by  $a, b$ .

Note that the grammar does not generate nesting exponentials, i.e.,  $!^p!^q\alpha$  is not a correct scheme, while  $!^{p+q}\alpha$  is correct.

- ii) A *modality set* is a set of linear constraints in the form either  $p = q$  or  $p > 0$  or  $p = 0$ , where  $p$  and  $q$  are exponentials. Modality sets are ranged over by  $C$ .
- iii) A *type scheme* is denoted by  $\sigma \upharpoonright_C$ , where  $\sigma$  is a scheme and  $C$  is a modality set. Type schemes will be ranged over by  $\zeta, \theta$ . Let  $\mathcal{T}$  denote the set of type schemes.
- iv)  $\overline{\sigma \upharpoonright_C}$  denotes the simple type skeleton underlying the type scheme  $\sigma \upharpoonright_C$ , and it is defined as follows:

$$\begin{aligned}\overline{\alpha \upharpoonright_C} &= \alpha; \\ \overline{\sigma \multimap \tau \upharpoonright_C} &= \overline{\sigma \upharpoonright_C} \multimap \overline{\tau \upharpoonright_C} \\ \overline{!^p \sigma \upharpoonright_C} &= \overline{\sigma \upharpoonright_C}\end{aligned}$$

- v) A *scheme substitution*  $S$  is a partial function from type schemes to types, replacing scheme variables by types and literals by natural numbers, in such a way that constraints in  $C$  are satisfied. If  $p$  is an exponential, let  $S(p)$  be the result of applying the scheme substitution  $S$  on all the literals in  $p$ , e.g. if  $p$  coincides with  $a_1 + \dots + a_n$ , then  $S(p)$  is  $S(a_1) + \dots + S(a_n)$ .  $C$  is satisfied by  $S$  if, for every constraint  $p = q$  ( $p > 0, p = 0$ ) in  $C$ ,  $S(p) = S(q)$  ( $S(p) > 0, S(p) = 0$ ). Clearly the solvability of a set of linear constraints is a decidable problem. The application of a substitution  $S$  satisfying  $C$  to a type scheme  $\sigma \upharpoonright_C$  is defined inductively as follows:

$$\begin{aligned}S(\alpha \upharpoonright_C) &= A && \text{if } [\alpha \mapsto A] \in S; \\ S(\sigma \multimap \tau \upharpoonright_C) &= S(\sigma \upharpoonright_C) \multimap S(\tau \upharpoonright_C) && ; \\ S(!^p \mu \upharpoonright_C) &= \underbrace{! \dots !}_n S(\mu \upharpoonright_C) && \text{if } p = a_1 + \dots + a_m \\ &&& n = S(a_1) + \dots + S(a_m).\end{aligned}$$

If  $C$  is not satisfied by  $S$ , then  $S(\sigma \upharpoonright_C)$  is undefined.

Binary relation  $\equiv$  is extended to denote the syntactical identity between both types, schemes and type schemes. Making clear what we said before, a type scheme can be seen as a description of the set of all types that can be obtained from it through a scheme substitution defined on it.

A substitution is a total function from type schemes to type schemes mapping scheme variables to schemes, and generating some constraints. A substitution is denoted by a pair

$\frac{}{U(\alpha \upharpoonright_C, \alpha \upharpoonright_{C'}) = \langle \llbracket, \emptyset \rrbracket \rangle} \quad (U1)$
$\frac{\alpha \text{ does not occur in } \sigma}{U(\alpha \upharpoonright_C, \sigma \upharpoonright_{C'}) = \langle [\alpha \mapsto \sigma], \emptyset \rangle} \quad (U2)$
$\frac{\alpha \text{ does not occur in } \sigma}{U(\sigma \upharpoonright_C, \alpha \upharpoonright_{C'}) = \langle [\alpha \mapsto \sigma], \emptyset \rangle} \quad (U3)$
$\frac{U(\sigma \multimap \tau \upharpoonright_C, \nu \upharpoonright_{C'}) = \langle s, C'' \rangle}{U(\sigma \multimap \tau \upharpoonright_C, !^q \nu \upharpoonright_{C'}) = \langle s, C'' \cup \{q = 0\} \rangle} \quad (U4)$
$\frac{U(\mu \upharpoonright_C, \sigma \multimap \tau \upharpoonright_{C'}) = \langle s, C'' \rangle}{U(!^p \mu \upharpoonright_C, \sigma \multimap \tau \upharpoonright_{C'}) = \langle s, C'' \cup \{p = 0\} \rangle} \quad (U5)$
$\frac{U(\mu \upharpoonright_C, \nu \upharpoonright_{C'}) = \langle s, C'' \rangle}{U(!^p \mu \upharpoonright_C, !^q \nu \upharpoonright_{C'}) = \langle s, C'' \cup \{p = q\} \rangle} \quad (U6)$
$\frac{U(\sigma_1 \upharpoonright_C, \tau_1 \upharpoonright_{C'}) = \langle s_1, C_1 \rangle \quad U(\langle s_1, C_1 \rangle(\sigma_2 \upharpoonright_C), \langle s_1, C_1 \rangle(\tau_2 \upharpoonright_{C'})) = \langle s_2, C_2 \rangle}{U(\sigma_1 \multimap \sigma_2 \upharpoonright_C, \tau_1 \multimap \tau_2 \upharpoonright_{C'}) = \langle s_1, C_1 \rangle \circ \langle s_2, C_2 \rangle} \quad (U7)$
<p><math>\langle s_1, C_1 \rangle \circ \langle s_2, C_2 \rangle</math> is the substitution such that  <math>\langle s_1, C_1 \rangle \circ \langle s_2, C_2 \rangle(\sigma \upharpoonright_C) = \langle s_2, C_2 \rangle(\langle s_1, C_1 \rangle(\sigma \upharpoonright_C))</math>.</p>

Table 4: The unification algorithm  $U$ 

$\langle s, C \rangle$ , where  $s$  is a function from scheme variables to schemes and  $C$  is a modality set. Substitutions will be ranged over by  $t$ . The behaviour of  $\langle s, C \rangle$  is defined as follows.

$$\begin{aligned}
\langle s, C \rangle(\alpha \upharpoonright_{C'}) &= \sigma \upharpoonright_{C \cup C'} && \text{if } [\alpha \mapsto \sigma] \in s; \\
\langle s, C \rangle(\sigma \multimap \tau \upharpoonright_{C'}) &= \sigma' \multimap \tau' \upharpoonright_{C''} && \text{if } \langle s, C \rangle(\sigma \upharpoonright_{C'}) = \sigma' \upharpoonright_{C''} \\
&&& \text{and } \langle s, C''' \rangle(\tau \upharpoonright_{C'}) = \tau' \upharpoonright_{C''}; \\
\langle s, C \rangle(!^p \mu \upharpoonright_{C'}) &= !^p \nu \upharpoonright_{C''} && \text{if } \langle s, C \rangle(\mu \upharpoonright_{C'}) = \nu \upharpoonright_{C''}; \\
\langle s, C \rangle(!^p \mu \upharpoonright_{C'}) &= !^r \nu \upharpoonright_{C'' \cup \{r=p+q\}} && \text{if } \langle s, C \rangle(\mu \upharpoonright_{C'}) = !^q \nu \upharpoonright_{C''}.
\end{aligned}$$

Note that the last rule is necessary in order to preserve the correct syntax of schemes, where the nesting of exponentials is not allowed.

In order to define the principal typing, we will use a unification algorithm for type schemes, which is a variant of that defined in [11]. Let  $=_e$  be the relation between type schemes such that  $\sigma \upharpoonright_{C=e} \tau \upharpoonright_{C'}$  if  $\overline{\sigma \upharpoonright_C} \equiv \overline{\tau \upharpoonright_C}$ .

The unification algorithm, which we will present in Table 4, in Structured Operational Semantics style, is a function  $U$  from  $\mathcal{T} \times \mathcal{T}$  to substitutions.

The following lemma proves that the function  $U$  supplies a more general unifier for two type schemes, in two steps: the substitution it generates is the most general unifier

with respect to the relation  $=_e$ , and moreover there is a most general scheme substitution unifying the two type schemes modulo the syntactic equivalence  $\equiv$ .

**Lemma 6.3.**

- i) (*correctness*)  $U(\sigma \upharpoonright_C, \tau \upharpoonright_{C'}) = \langle s, C'' \rangle$  implies  $\langle s, C'' \rangle(\sigma \upharpoonright_C) = \sigma' \upharpoonright_{C''}$ ,  $\langle s, C'' \rangle(\tau \upharpoonright_{C'}) = \tau' \upharpoonright_{C''}$  where  $\sigma' \upharpoonright_{C''} =_e \tau' \upharpoonright_{C''}$ . Moreover for every scheme substitution  $S$ , defined on both the type schemes,  $S(\sigma' \upharpoonright_{C''}) \equiv S(\tau' \upharpoonright_{C''})$ .
- ii) (*completeness*)  $S(\sigma \upharpoonright_C) \equiv S(\tau \upharpoonright_{C'})$  implies  $U(\sigma \upharpoonright_C, \tau \upharpoonright_{C'}) = \langle s, C'' \rangle$  and there is  $S'$  such that, for every type scheme  $\zeta$ ,  $S(\zeta) = S'(\langle s, C'' \rangle(\zeta))$ , for some  $S'$ .

*Proof.*

- i) Easy, by induction on the rules defining  $U(\sigma \upharpoonright_C, \tau \upharpoonright_{C'})$ .
- ii) By induction on the pair  $(n, m)$ , where  $n$  and  $m$  are respectively the number of scheme variables occurring in both  $\sigma$  and  $\tau$  and the total number of symbols of  $\sigma$  and  $\tau$ .

Let  $\sigma \equiv \alpha$ , and let  $S(\sigma \upharpoonright_C) \equiv S(\tau \upharpoonright_{C'}) \equiv A$ ; clearly either  $\tau \equiv \alpha$  or  $\alpha$  cannot occur as proper subterm of  $\tau$ . In the first case  $U(\alpha \upharpoonright_C, \alpha \upharpoonright_{C'}) = \langle [], \emptyset \rangle$ , and  $S = S'$ . In the second case  $U(\sigma \upharpoonright_C, \tau \upharpoonright_{C'}) = \langle [\alpha \mapsto \tau], \emptyset \rangle$ . Then every scheme substitution  $S'$ , solving both  $C$  and  $C'$  and acting as  $S$  on all the scheme variables occurring in  $\sigma$  and  $\tau$  but  $\alpha$  does the desired job.

Let  $\sigma \equiv \sigma_1 \multimap \sigma_2$  and  $\tau \equiv \tau_1 \multimap \tau_2$ . So  $S(\sigma_1 \multimap \sigma_2 \upharpoonright_C) \equiv S(\sigma_1 \upharpoonright_C) \multimap S(\sigma_2 \upharpoonright_C)$ , and  $S(\tau_1 \multimap \tau_2 \upharpoonright_{C'}) \equiv S(\tau_1 \upharpoonright_{C'}) \multimap S(\tau_2 \upharpoonright_{C'})$ . So by induction  $U(\sigma_1 \upharpoonright_C, \tau_1 \upharpoonright_{C'}) = \langle s_1, C_1 \rangle$ ,  $\langle s_1, C_1 \rangle(\sigma_1 \upharpoonright_C) = \sigma' \upharpoonright_{C'_1}$  and  $\langle s_1, C_1 \rangle(\tau_1 \upharpoonright_{C'}) = \tau' \upharpoonright_{C''_1}$  where  $\sigma' \upharpoonright_{C'_1} =_e \tau' \upharpoonright_{C''_1}$ . Moreover there is  $S_1$  such that  $S(\sigma_2 \upharpoonright_C) \equiv S_1(\langle s_1, C_1 \rangle(\sigma_2 \upharpoonright_C))$  and  $S(\tau_2 \upharpoonright_{C'}) \equiv S_1(\langle s_1, C_1 \rangle(\tau_2 \upharpoonright_{C'}))$ . In case  $s_1$  is not empty, the number of scheme variables in both the type schemes is less than in  $\sigma$  and  $\tau$ ; otherwise their total number of symbols is less than the one in  $\sigma$  and  $\tau$ . In both cases we can apply the induction hypothesis and conclude the proof.

All the other cases follow directly from the induction hypothesis.  $\square$

The principal type scheme of a term is a pair in the form  $\langle \Sigma; \zeta \rangle$ , where  $\Sigma$  is a scheme context (i.e., a set of assignments of the shape  $x : \theta$ , where no variable is repeated), and  $\zeta$  is a type scheme.

In order to simplify the text of the algorithm, we will use the following conventions:

- Let  $\sigma$  be a scheme.  $!^p \sigma$  denotes  $!^p \mu$  in case  $\sigma \equiv \mu$ ,  $!^r \mu$ , where  $r = p + q$  in case  $\sigma \equiv !^q \mu$ ; if  $\zeta \equiv \sigma \upharpoonright_C$ , then  $!^p \zeta$  denotes  $!^p \sigma \upharpoonright_C$ ;
- Let  $\Sigma$  be a scheme context.  $!^p \Sigma$  denotes the scheme context  $\{x : !^p \zeta \mid x : \zeta \in \Sigma\}$ .

The principal type scheme algorithm is defined in Table 5.

**Theorem 6.4** (Type Inference).

- i) (*correctness*) If  $PT(M) = \langle \Sigma, \zeta \rangle$  then for every scheme substitution  $S$  defined on all the type schemes occurring in  $PT(M)$ ,  $\exists \Gamma, \Delta, \Theta$  partitioning  $S(\Sigma)$  s.t.  $\Gamma \mid \Delta \mid \Theta \vdash M : S(\zeta)$ .
- ii) (*completeness*) If  $\Gamma \mid \Delta \mid \Theta \vdash M : A$  then  $PT(M) = \langle \Sigma, \zeta \rangle$  and there exists a scheme substitution  $S$  defined on all the type schemes occurring in  $PT(M)$  such that  $S(\Sigma) \subseteq \Gamma \cup \Delta \cup \Theta$  and  $A = S(\zeta)$ .

*Proof.*

- i) By induction on  $M$ .
  - $M \equiv x$ . Then  $PT(M) = \langle \{x : !^a \alpha \upharpoonright_\emptyset\}; !^a \alpha \upharpoonright_\emptyset \rangle$ . Every scheme substitution  $S$  satisfies the empty set of constraints.

<ul style="list-style-type: none"> <li>• <math>PT(x) = \langle \{x : !^a \alpha \uparrow \emptyset\}, !^a \alpha \uparrow \emptyset \rangle</math></li> <li>• <math>PT(\lambda x.M) = \text{let } PT(M) = \langle \Sigma, \sigma \uparrow_C \rangle \text{ in}</math>            if <math>x</math> doesn't occur free in <math>M</math>            <math>\langle !^b \Sigma, !^b (!^a \alpha \multimap \sigma) \uparrow_C \rangle</math>            else            let <math>\Sigma = \{x : \tau \uparrow_{C'}\} \cup \Sigma'</math> in            if <math>x</math> occurs multiple times in <math>M</math>            let <math>U(\tau \uparrow_{C'}, !^a \alpha \uparrow_{\{a&gt;0\}}) = \langle s_1, C_1 \rangle</math> in            let <math>\langle s_1, C_1 \rangle(\tau \uparrow_{C'}) = \tau' \uparrow_{C''}</math> and            <math>\langle s_1, C_1 \rangle(\sigma \uparrow_C) = \sigma' \uparrow_{C'''}</math> in            <math>\langle \langle s_1, C_1 \rangle(!^b \Sigma'), !^b (\tau' \multimap \sigma') \uparrow_{C'' \cup C'''} \rangle</math>            else            <math>\langle !^a \Sigma', !^a (\tau \multimap \sigma) \uparrow_{C \cup C'} \rangle</math></li> <li>• <math>PT(M_1 M_2) = \text{let } PT(M_1) = \langle \Sigma_1, \sigma_1 \uparrow_{C_1} \rangle, PT(M_2) = \langle \Sigma_2, \sigma_2 \uparrow_{C_2} \rangle</math>            and let them be disjoint in            let <math>\{x_1, \dots, x_k\} = \text{dom}(\Sigma_1) \cap \text{dom}(\Sigma_2)</math> and            <math>\forall 1 \leq i \leq k \forall 1 \leq j \leq 2 x_i : \zeta_j^i \in \Sigma_j</math> and            <math>U(\sigma_1 \uparrow_{C_1}, (\sigma_2 \multimap !^a \alpha) \uparrow_{C_2}) = \langle s', C' \rangle</math> and            <math>U(\zeta_1^1, \zeta_2^1) = t_1</math> and            <math>U(t_{i-1}(\zeta_1^i), t_{i-1}(\zeta_2^i)) = t_i</math> and            let <math>t = t_1 \circ t_2 \circ \dots \circ t_k</math> in            <math>\langle !^b (t(\Sigma_1) \cup t(\Sigma_2)), !^b (t(!^a \alpha) \uparrow_{C'}) \rangle</math></li> </ul> <p><math>\alpha, a, b</math> are fresh variables.</p>
--

Table 5: The type inference algorithm  $PT$ .

If  $S(!^a \alpha \uparrow \emptyset)$  is linear, then take  $\Delta = \emptyset$  and either  $\Gamma = \{x : S(!^a \alpha \uparrow \emptyset)\}$  and  $\Theta = \emptyset$  or  $\Theta = \{x : S(!^a \alpha \uparrow \emptyset)\}$  and  $\Gamma = \emptyset$ . Otherwise choose  $\Gamma = \Theta = \emptyset$ , and  $\Delta = \{x : S(!^a \alpha \uparrow \emptyset)\}$ .

- $M \equiv \lambda x.P$ . This case follows directly by induction.
- $M \equiv M_1 M_2$ . Then  $PT(M_1) = \langle \Sigma_1, \theta_1 \rangle$ ,  $PT(M_2) = \langle \Sigma_2, \theta_2 \rangle$ , and  $PT(M) = \langle \Sigma', \theta' \rangle$ , where  $\theta'$  is defined as in Table 5.

Let  $S$  be a scheme substitution defined on all the type schemes occurring in  $PT(M)$ : note that, by the definition of the function  $U$  this implies that  $S$  is defined on both  $PT(M_1)$  and  $PT(M_2)$ . Moreover, by construction of  $PT$ ,  $x : \zeta_i \in \Sigma_i$  ( $1 \leq i \leq 2$ ) implies  $U(\zeta_1, \zeta_2)$  is defined, and  $S(\zeta_1) \equiv S(\zeta_2)$ , by Lemma 6.3.

By induction  $\Gamma_i \mid \Delta_i \mid \Theta_i \vdash M_i : S(\theta_i)$  ( $1 \leq i \leq 2$ ). Note that every type derivation for  $M$  ends with an application of the rule  $(E^\circ)$ , followed by a sequence, may be empty, of rule  $(!)$ . Since in rule  $(E^\circ)$  the two linear contexts are disjoint, we can build the partition of the contexts in the following way:

$$\Gamma_1 = \{x : S(\zeta) \mid (\{x : S(\zeta)\} \subseteq S(\Sigma_1)) \wedge (x \in FV(M_1) \wedge x \notin FV(M_2)) \wedge (S(\zeta) \text{ is linear})\},$$

$$\Gamma_2 = \{x : S(\zeta) \mid \{x : S(\zeta)\} \subseteq S(\Sigma_2) \wedge (x \in FV(M_2) \wedge x \notin FV(M_1)) \wedge (S(\zeta) \text{ is linear})\},$$

$$\Delta_i = \{x : S(\zeta) \mid (\{x : S(\zeta)\} \subseteq S(\Sigma_i)) \wedge (S(\zeta) \text{ is modal})\} \text{ and}$$

$$\Theta_i = \Sigma_i - (\Gamma_i \cup \Delta_i) \quad (1 \leq i \leq 2).$$

By the weakening Lemma, we have that  $\Gamma_i \mid \Delta_1, \Delta_2 \mid \Theta_1, \Theta_2 \vdash M_i : S(\theta_i)$  ( $1 \leq i \leq 2$ ).

Since  $S(\theta_1) \equiv S(\theta_2) \multimap S(!^a \alpha \uparrow_{C_2})$  (by Lemma 6.3.i), the proof follows by rule  $(E_{\multimap})$ .

- ii) By induction on the derivation proving  $\Gamma \mid \Delta \mid \Theta \vdash M : A$ .

Let the last used rule be  $(A^L)$ . Then  $M \equiv x$ , and  $\{x : A\} \subseteq \Gamma$ . By definition,  $PT(x) = \langle \{x : !^a\alpha \uparrow_\emptyset\}; !^a\alpha \uparrow_\emptyset \rangle$ , and the proof follows easily.

The case  $(A^P)$  is similar.

The cases  $(I_{\circ}^L)$  and  $(I_{\circ}^L)$  both follow by induction and weakening lemma.

Let us consider the case when the last used rule is  $(E_{\circ})$ . Then  $M \equiv M_1M_2$ , and  $\Gamma_1 \mid \Delta \mid \Theta \vdash M_1 : B \multimap A$  and  $\Gamma_2 \mid \Delta \mid \Theta \vdash M_2 : B$ , for some  $B$ , where  $\Gamma_1$  and  $\Gamma_2$  are disjoint. By induction  $PT(M_i) = \langle \Sigma_i, \zeta_i \rangle$  and there is  $S_i$  defined on all type schemes in  $PT(M_i)$  such that  $S_1(\zeta_1) \equiv B \multimap A$ , and  $S_2(\zeta_2) \equiv B$ .

Moreover  $S_i(\Sigma_i) \subseteq \Gamma_i \cup \Delta \cup \Theta$  ( $1 \leq i \leq 2$ ). Since by construction  $PT(M_1)$  and  $PT(M_2)$  are disjoint, there is a well defined scheme substitution  $S$  acting as both  $S_1$  and  $S_2$  and such that  $S(!^a\alpha \uparrow_\emptyset) \equiv A$ , for  $\alpha, a$  fresh. So if  $\zeta_2 \equiv \sigma \uparrow_C$ ,  $S(\zeta_1) \equiv S(\sigma \multimap !^{p_1}\alpha \uparrow_C)$ , and  $S$  is defined on all the type schemes in  $PT(M_i)$  ( $1 \leq i \leq 2$ ). Then by Lemma 6.3.ii),  $U(\zeta_1, \sigma \multimap !^{p_1}\alpha \uparrow_C) = \langle s', C' \rangle$ . Since  $S$  satisfies all the constraints in  $PT(M_i)$  ( $1 \leq i \leq 2$ ), if  $x : \theta_1$  and  $x : \theta_2$  belong to  $\Sigma_1$  and  $\Sigma_2$  respectively, then  $S(\theta_1) \equiv S(\theta_2)$ , and so, by Lemma 6.3.ii), they can be unified. So  $PT(M_1M_2)$  is defined, and by induction it enjoys the desired properties.

Let the last used rule be  $(!)$ . Then  $A \equiv !A'$  and the premise of the rule is  $\Gamma' \mid \Delta' \mid \Theta' \vdash M : A'$ , where  $!\Gamma' \cup !\Delta' \cup !\Theta' \subseteq \Delta$ . By induction  $PT(M) = \langle \Sigma, \zeta \rangle$  and there is a scheme substitution  $S'$  such that  $S'(\Sigma) \subseteq \Gamma' \cup \Delta' \cup \Theta'$ . Let  $S$  be such that  $S$  is defined on all the type schemes in  $PT(M)$ , and such that  $S(\zeta) \equiv !S'(\zeta)$ , for all type scheme  $\zeta$ : it is easy to check that, if  $S'$  is defined, i.e., it satisfies all the constraints in  $PT(M)$ , than  $S$  is well defined too, and so it does the right job.  $\square$

The complexity of the type inference algorithm  $PT$  is of the same order as the type inference algorithm for simple types. In order to prove this, we need some notations. If  $A(n)$  is an algorithm running on a datum  $n$ , let us denote by  $|A(n)|$  its asymptotic complexity. Moreover, if  $\sigma$  is a scheme, let  $|\sigma|$  be the number of symbols in it. Let  $TA(M)$  be the type inference algorithm for simple types running on a term  $M$ . By abuse of notation, we assume that, for every type scheme  $\sigma$ ,  $\bar{\sigma}$  denotes a simple type: in fact the syntax of type schemes, when erasing exponentials and constraints, coincides with that of simple types.

**Theorem 6.5** (Complexity).  $PT(M) \in O(L(M) + |TA(M)|)$ .

*Proof.* First of all, let us observe that the the unification algorithm  $U$  coincide with the Robinson unification, when it is applied to two type schemes whose set of constraints is empty, so, if  $RU$  denotes the Robinson's unification,  $|U(\sigma \uparrow_\emptyset, \tau \uparrow_\emptyset)| = |RU(\bar{\sigma}, \bar{\tau})|$ . Then  $|U(\sigma \uparrow_C, \tau \uparrow_{C'})| \leq |RU(\bar{\sigma}, \bar{\tau})| + |\sigma| + |\tau|$ . Remember that Robinson unification is equivalent to the principal simple type assignment. Moreover it is easy to see that, if  $PT(M) = \langle \Sigma, \sigma \uparrow_C \rangle$ , then  $TA(M) = \langle \bar{\Sigma}, \bar{\sigma} \rangle$ , when  $\bar{\Sigma}$  denotes the context obtained from  $\Sigma$  by applying the function  $\bar{[\cdot]}$  to all types in it.

The proof is by induction on  $M$ . If  $M$  is a constant the proof is trivial. If  $M \equiv \lambda x.N$ , then  $|PT(M)| = |PT(N)| + |U(\tau \uparrow_C, !^a\alpha \uparrow_{a>0})| = |PT(N)| + k$ , for a constant  $k$ , since  $\alpha$  is a scheme variable. Then the result follows by induction. Let  $M \equiv PQ$ . Then  $|PT(PQ)| = |PT(P)| + |PT(Q)| + |U(\sigma \uparrow_C, (\tau \multimap !^a\alpha) \uparrow_{C'})| + |U(\Sigma_1, \Sigma_2)|$  if  $PT(P) = \langle \Sigma_1, \sigma \uparrow_C \rangle$  and  $PT(Q) = \langle \Sigma_2, \tau \uparrow_{C'} \rangle$ .  $U(\Sigma_1, \Sigma_2)$  is an abbreviation for the unification of the two scheme contexts  $\Sigma_1$  and  $\Sigma_2$ , as specified in the algorithm. By induction  $PT(PQ) = |TA(P)| + k_1 + |TA(Q)| + k_2 + |RU(\bar{\sigma}, \bar{\tau})| + |\sigma| + |\tau| + |RU(\bar{\Sigma}_1, \bar{\Sigma}_2)| + |\Sigma_1| + |\Sigma_2|$ . Remembering that  $|TA(PQ)| = |TA(P)| + |TA(Q)| + |RU(\bar{\sigma}, \bar{\tau})| + |RU(\bar{\Sigma}_1, \bar{\Sigma}_2)|$ , the result follows.  $\square$

**Example 6.6.** Let us illustrate the application of the type inference algorithm to the term:

$$\underline{2} \ \underline{3} \equiv (\lambda xy.x(xy))\lambda xy.x(x(xy)).$$

Let  $a, b, c, d, e, f, g, h, k, i, m, n, p, q, r$  be literals and  $\alpha, \beta, \gamma, \delta, \epsilon$  be scheme variables. First we will build  $PT(\underline{2})$ . Starting from

$$PT(x) = \langle \{x : !^a \alpha \uparrow \emptyset\}, !^a \alpha \uparrow \emptyset \rangle \text{ and } PT(y) = \langle \{y : !^b \beta \uparrow \emptyset\}, !^b \beta \uparrow \emptyset \rangle,$$

due to  $U(!^a \alpha \uparrow \emptyset, !^b \beta \multimap !^c \gamma \uparrow \emptyset) = \langle [\alpha \mapsto !^b \beta \multimap !^c \gamma], \{a = 0\} \rangle$ , we obtain:

$$PT(xy) = \langle \{x : !^{d+a} (!^b \beta \multimap !^c \gamma) \uparrow_{C_0}, y : !^{d+b} \beta \uparrow_{C_0}\}, !^{c+d} \gamma \uparrow_{C_0} \rangle$$

where  $C_0 = \{a = 0\}$ .

Now a fresh version of  $PT(x)$  is needed, so consider  $\langle \{x : !^{a_1} \alpha_1 \uparrow \emptyset\}, !^{a_1} \alpha_1 \uparrow \emptyset \rangle$ . The rule for application allows us to perform certain unifications. First we obtain

$$U(!^{a_1} \alpha_1 \uparrow \emptyset, (!^{c+d} \gamma \multimap !^e \delta) \uparrow_{C_0}) = \langle [\alpha_1 \mapsto !^{c+d} \gamma \multimap !^e \delta], C_1 \rangle$$

where  $C_1 = \{a_1 = 0\}$ . A second unification is necessary for unifying the two premises on  $x$  in the first component of  $PT(xy)$  and  $PT(x)$ , respectively:

$$U(!^{a_1} \alpha_1 \uparrow \emptyset, !^{d+a} (!^b \beta \multimap !^c \gamma) \uparrow_{C_0}) = \langle [\alpha_1 \mapsto !^b \beta \multimap !^c \gamma], C_2 \rangle$$

where  $C_2 = \{a_1 = a + d\}$ . By composing the two substitutions, we have  $\langle [\gamma \mapsto \beta, \gamma \mapsto \delta], \{c + d = b, e = c\} \rangle$ . So

$$PT(x(xy)) = \langle \{x : !^{h+a_1} (!^b \beta \multimap !^e \beta) \uparrow_{C_3}, y : !^{h+b+d} \beta \uparrow_{C_3}\}, !^{h+e} \beta \uparrow_{C_3} \rangle$$

where  $C_3 = C_0 \cup C_1 \cup C_2 \cup \{c + d = b, c = e\}$ . By applying the rule for the abstraction, we obtain:

$$PT(\lambda y.x(xy)) = \langle \{x : !^f (!^b \beta \multimap !^e \beta) \uparrow_{C_3}\}, !^k (!^{h+b+d} \beta \multimap !^{h+e} \beta) \uparrow_{C_3} \rangle$$

and

$$PT(\lambda xy.x(xy)) = \langle \emptyset; !^i (!^f (!^b \beta \multimap !^e \beta) \multimap !^k (!^{h+b+d} \beta \multimap !^{h+e} \beta)) \uparrow_{C_4} \rangle$$

where  $C_4 = C_3 \cup \{f = k + h + a_1, f > 0\}$ .

Due to the particular form of  $PT(\lambda xy.x(xy))$ , we can deduce that the term  $\lambda xy.x(xy)$  can be assigned, among others, the following types

$$!(A \multimap A) \multimap !(A \multimap A), !(A \multimap A) \multimap !A \multimap !A, !! (A \multimap A) \multimap !(A \multimap A).$$

In particular, the scheme substitution that replaces  $\beta$  by  $!(A \multimap A)$ , furthermore  $b, e, h, d$  and  $a_1$  by 0, and both  $k$  and  $f$  by 1, satisfies the constraints and generates the typing  $\emptyset \mid \emptyset \mid \emptyset \vdash \underline{2} : !(A \multimap A) \multimap !(A \multimap A)$ , whose derivation is shown in Example 4.2.

In order to build the principal type scheme of  $\underline{3}$ , we need to start from two fresh copies of  $PT(x)$  and  $PT(x(xy))$ , let

$$\langle \{x : !^n \epsilon \uparrow \emptyset\}, !^n \epsilon \uparrow \emptyset \rangle \text{ and } \langle \{x : !^{h'+a'_1} (!^{b'} \alpha \multimap !^{e'} \alpha) \uparrow_{C'_3}, y : !^{h'+b'+d'} \alpha \uparrow_{C'_3}\}, !^{h'+e'} \alpha \uparrow_{C'_3} \rangle$$

where  $C'_3 = \{a' = 0, a'_1 = 0, a'_1 = d' + a', b' = c' + d', e' = c'\}$ .

By applying the rule for application, we obtain, for the first unification:

$$U(!^n \epsilon \uparrow \emptyset, !^{h'+e'} \alpha \multimap !^{p'} \eta \uparrow_{C'_3}) = \langle [\epsilon \mapsto !^{h'+e'} \alpha \multimap !^{p'} \eta], C'_5 \rangle$$

where  $C'_5 = \{n = 0\}$ . By unifying the two premises on  $x$ , we have

$$U(!^n \epsilon \uparrow \emptyset, !^{h'+a'_1} (!^{b'} \alpha \multimap !^{e'} \alpha) \uparrow_{C'_3}) = \langle [\epsilon \mapsto !^{b'} \alpha \multimap !^{e'} \alpha], C'_6 \rangle$$

where  $C_6 = \{n = h' + a'_1\}$ . So, composing the two substitutions:

$$PT(x(x(xy))) = \langle \{x : !^{q+h'+a'_1}(!^{b'}\alpha \multimap !^{e'}\alpha) \upharpoonright_{C_7}, y : !^{q+h'+b'+d'}\alpha \upharpoonright_{C_7}, !^{p+q}\alpha \upharpoonright_{C_7} \rangle$$

where  $C_7 = C'_3 \cup C_5 \cup C_6 \cup \{b' = h' + e', e' = p\}$ . So, by applying the rules for abstraction, we have:

$$PT(\lambda y.x(x(xy))) = \langle \{x : !^{r+q+h'+a'_1}(!^{b'}\alpha \multimap !^{e'}\alpha) \upharpoonright_{C_7}, !^r(!^{q+h'+b'+d'}\alpha \multimap !^{p+q}\alpha) \upharpoonright_{C_7} \rangle$$

and

$$PT(\lambda xy.x(x(xy))) = \langle \emptyset, !^s(!^g(!^{b'}\alpha \multimap !^{e'}\alpha) \multimap !^r(!^{q+h'+b'+d'}\alpha \multimap !^{p+q}\alpha)) \upharpoonright_{C_8} \rangle$$

where  $C_8 = C_7 \cup \{g = r + q + h' + a'_1, g > 0\}$ .

It is easy, but boring, to check that the typings for  $\underline{3}$  are the same that the ones for  $\underline{2}$ , by inspecting the modality set. Now, in order to build  $PT(\underline{2} \underline{3})$ , we need to unify the two type schemes:

$$\sigma \equiv !^i(!^{k+h+a_1}(!^b\beta \multimap !^e\beta) \multimap !^k(!^{h+b+d}\beta \multimap !^{h+e}\beta)) \upharpoonright_{C_4}$$

and

$$\tau \equiv !^s(!^g(!^{b'}\alpha \multimap !^{e'}\alpha) \multimap !^r(!^{q+h'+b'+d'}\alpha \multimap !^{p+q}\alpha)) \multimap !^t\gamma \upharpoonright_{C_8}$$

obtaining:

$$U(\sigma, \tau) = \langle [\beta \mapsto !^{b'}\alpha \multimap !^{e'}\alpha, \gamma \mapsto !^{h+b+d}\beta \multimap !^{h+e}\beta], C_9 \rangle$$

where  $C_9 = \{i = 0, k + h + a_1 = s, t = k, s = k + h + a_1, b = g, e = r, b' = q + h' + b' + d', e = p + q\}$ . So

$$PT(\underline{2} \underline{3}) = \langle \emptyset, !^t(!^{h+b+d}(!^{b'}\alpha \multimap !^{e'}\alpha) \multimap !^{h+e}(!^{b'}\alpha \multimap !^{e'}\alpha)) \upharpoonright_{C_{10}} \rangle$$

where  $C_{10} = C_8 \cup C_9$ .

Finally, the scheme substitution that replaces  $\alpha$  by  $A$ , furthermore  $b, e, b'$  and  $e'$  by 0, and both  $t$  and  $h$  by 1, satisfies the constraints and generates the typing  $\emptyset \mid \emptyset \mid \emptyset \vdash \underline{2} \underline{3} : !(!(A \multimap A) \multimap !(A \multimap A))$ , whose derivation is shown in Example 4.2.

## 7. ACHIEVING COMPLETENESS

The type-system we introduced in this paper is not complete for the class of elementary time functions, at least if we restrict to uniform encodings. Indeed, simply typed lambda-calculus *without constants* is itself incomplete with respect to any reasonable complexity class (see, for example, [12]). In order to achieve completeness, two different extensions of the system can be built, one adjoining basic types and constants, and the other adjoining second order types. In this section we will briefly discuss these two solutions.

**7.1. Basic Types and Constants.** Let us fix a finite set of free algebras  $\mathcal{A} = \{\mathbb{A}_1, \dots, \mathbb{A}_n\}$ . The constructors of  $\mathbb{A}_i$  will be denoted as  $c_{\mathbb{A}_i}^1, \dots, c_{\mathbb{A}_i}^{k(\mathbb{A}_i)}$ . The arity of constructor  $c_{\mathbb{A}_i}^j$  will be denoted as  $\mathcal{R}_{\mathbb{A}_i}^j$ . The algebra  $\mathbb{U}$  of unary integers has two constructors  $c_{\mathbb{U}}^1, c_{\mathbb{U}}^2$ , where  $\mathcal{R}_{\mathbb{U}}^1 = 1$  and  $\mathcal{R}_{\mathbb{U}}^2 = 0$ . The languages of types, terms and values are extended by the the following productions

$$\begin{aligned} A & ::= \mathbb{A}_j \\ M & ::= \text{iter}_{\mathbb{A}_j} \mid \text{cond}_{\mathbb{A}_j} \mid c_{\mathbb{A}_j}^i \\ V & ::= \text{iter}_{\mathbb{A}_j} \underbrace{V \dots V}_k \text{ times} \mid \text{cond}_{\mathbb{A}_j} \underbrace{V \dots V}_k \text{ times} \mid c_{\mathbb{A}_j}^i \underbrace{V \dots V}_l \text{ times} \end{aligned}$$

where  $\mathbb{A}_j$  ranges over  $\mathcal{A}$ ,  $i$  ranges over  $\{1, \dots, k(\mathbb{A}_j)\}$  and  $k$  ranges over  $\{0, \dots, k(\mathbb{A}_j)\}$  and  $l$  ranges over  $\{0, \dots, \mathcal{R}_{\mathbb{A}_j}^i\}$ . If  $t$  is a term of the free algebra  $\mathbb{A}$  and  $M_1, \dots, M_{k(\mathbb{A})}$  are terms, then  $t\{M_1, \dots, M_{k(\mathbb{A})}\}$  is defined by induction on  $t$ :  $(c_{\mathbb{A}}^i t_1 \dots t_{\mathcal{R}_{\mathbb{A}}^i})\{M_1, \dots, M_{k(\mathbb{A})}\}$  will be

$$M_i(t_1\{M_1, \dots, M_{k(\mathbb{A})}\}) \dots (t_{\mathcal{R}_{\mathbb{A}}^i}\{M_1, \dots, M_{k(\mathbb{A})}\}).$$

The new constants receive the following types in any context:

$$\begin{aligned} \text{iter}_{\mathbb{A}} & : \mathbb{A} \multimap \underbrace{(A \multimap \dots \multimap A)}_{\mathcal{R}_{\mathbb{A}}^1 \text{ times}} \multimap A \multimap \dots \multimap \underbrace{(A \multimap \dots \multimap A)}_{\mathcal{R}_{\mathbb{A}}^{k(\mathbb{A})} \text{ times}} \multimap A \multimap A \\ \text{cond}_{\mathbb{A}} & : \mathbb{A} \multimap \underbrace{(A \multimap \dots \multimap A)}_{\mathcal{R}_{\mathbb{A}}^1 \text{ times}} \multimap A \multimap \dots \multimap \underbrace{(A \multimap \dots \multimap A)}_{\mathcal{R}_{\mathbb{A}}^{k(\mathbb{A})} \text{ times}} \multimap A \multimap A \\ c_{\mathbb{A}}^i & : \underbrace{A \multimap \dots \multimap A}_{\mathcal{R}_{\mathbb{A}}^i \text{ times}} \multimap A \end{aligned}$$

New (call-by-value) reduction rules are the following ones:

$$\begin{aligned} \text{iter}_{\mathbb{A}} t V_1 \dots V_{k(\mathbb{A})} & \rightarrow_v t\{V_1 \dots V_{k(\mathbb{A})}\} \\ \text{cond}_{\mathbb{A}} c_{\mathbb{A}}^i (t_1 \dots t_{\mathcal{R}_{\mathbb{A}}^i}) V_1 \dots V_{k(\mathbb{A})} & \rightarrow_v V_i t_1 \dots t_{\mathcal{R}_{\mathbb{A}}^i} \end{aligned}$$

It is easy to check that Lemma 5.8 holds in the presence of the new constants. Moreover:

**Proposition 7.1.** *Every typable closed normal form is a value.*

*Proof.* By induction on the structure of a normal form  $M$ :

- A variable is not closed.
- If  $M$  is an abstraction, then it is a value by definition.
- If  $M$  is a constant, then it is a value by definition.
- If  $M$  is an application  $NL$ , then  $\vdash N : A \multimap B$  and  $\vdash L : A$ . By induction,  $N$  and  $L$  are both values and, as a consequence,  $N$  cannot be a variable nor an abstraction. So,  $N$  must be obtained from one of the new productions for values; let us distinguish some cases:
  - If  $N \equiv \text{iter}_{\mathbb{A}_j} V_1 \dots V_k$  with  $k < k(\mathbb{A}_j)$ , then  $M$  is a value itself.
  - If  $N \equiv \text{iter}_{\mathbb{A}_j} V_1 \dots V_{k(\mathbb{A}_j)}$ , then  $M$  is a redex, because  $V_1$  is a closed value with type  $\mathbb{A}_j$ .
  - If  $N \equiv \text{cond}_{\mathbb{A}_j} V_1 \dots V_k$  with  $k < k(\mathbb{A}_j)$ , then  $M$  is a value itself.
  - If  $N \equiv \text{cond}_{\mathbb{A}_j} V_1 \dots V_{k(\mathbb{A}_j)}$ , then  $M$  is a redex, because  $V_1$  is a closed value with type  $\mathbb{A}_j$ .
  - If  $N \equiv c_{\mathbb{A}_j}^i V_1 \dots V_k$ , then  $k < \mathcal{R}_{\mathbb{A}_j}^i$  because  $N$  has an arrow type. As a consequence,  $M$  is a value.

This concludes the proof. □

We can prove the following theorem:

**Theorem 7.2.** *There is a finite set of free algebras  $\mathcal{A}$  including the algebra  $\mathbb{U}$  of unary integers such that for every elementary function  $f : \mathbb{N} \rightarrow \mathbb{N}$ , there is a term  $M_f : \mathbb{U} \rightarrow !^k \mathbb{U}$  such that  $M_f [u] \rightarrow_v^* [f(u)]$  (where  $[n]$  is the term in  $\mathbb{U}$  corresponding to the natural number  $n$ ).*



*Proof.* We will show that if  $f : \mathbb{N} \rightarrow \mathbb{N}$  is computable by a Turing Machine  $\mathcal{M}$  running in elementary time, then there is a term  $M_f$  representing that same function. First of all,  $\mathcal{A}$  will contain a free algebra  $\mathbb{C}$  with four constructors  $c_{\mathbb{C}}^1, c_{\mathbb{C}}^2, c_{\mathbb{C}}^3, c_{\mathbb{C}}^4$  having arities  $\mathcal{R}_{\mathbb{C}}^1 = 4, \mathcal{R}_{\mathbb{C}}^2 = 1, \mathcal{R}_{\mathbb{C}}^3 = 1, \mathcal{R}_{\mathbb{C}}^4 = 0$ . Constructors  $c_{\mathbb{C}}^2, c_{\mathbb{C}}^3, c_{\mathbb{C}}^4$  can be used to build binary strings and a configuration will correspond to a term  $c_{\mathbb{C}}^1 t_1 t_2 t_3 t_4$  where  $t_1$  represent the current state,  $t_2$  represents the current symbol,  $t_3$  represents the left-hand side of the tape and  $t_4$  represents the right-hand side of the tape. A closed term  $trans : \mathbb{C} \multimap \mathbb{C}$  encoding the transition function can be built using, in particular, the new constant  $cond_{\mathbb{C}}$ . Iteration, on the other hand, helps when writing  $init : \mathbb{U} \multimap !\mathbb{C}$  (whose purpose is to translate a unary integer  $t$  into the initial configuration of  $\mathcal{M}$  for  $t$ ) and  $final : \mathbb{C} \multimap !\mathbb{U}$  (which extracts a unary integer from the final configuration of  $M$ ). In this way, the so-called qualitative part of the encoding can be done. The quantitative part, on the other hand, can be encoded as follows. We will show there are terms  $tower^n : \mathbb{U} \multimap !^{2^n} \mathbb{U}$  such that  $tower^n [m] \rightarrow_v^* [2_n(m)]$  where  $2_0(m) = m$  and  $2_{n+1}(m) = 2^{2^n(m)}$  for every  $n \geq 0$ . We will prove the existence of such terms by induction on  $n$ .  $tower^0 : \mathbb{U} \rightarrow \mathbb{U}$  is simply the identity  $\lambda x.x$ . Consider now the term

$$exp \equiv \lambda x.iter_{\mathbb{U}} x (\lambda y \lambda z.y(yz)) (\lambda y.c_{\mathbb{U}}^1 y) : \mathbb{U} \rightarrow !!(\mathbb{U} \rightarrow \mathbb{U})$$

We now prove that for every  $m \in \mathbb{N}$ ,  $exp[m] \rightarrow_v^* V_m$  where  $V_m$  is a value such that  $V_m[p] \rightarrow_v^* [2^m + p]$  for every  $p \in \mathbb{N}$ . We go by induction on  $m$ . If  $m = 0$ , then

$$exp[m] \rightarrow_v^* (\lambda x.c_{\mathbb{U}}^1 x)$$

and  $(\lambda x.c_{\mathbb{U}}^1 x)[p] \rightarrow_v^* [1 + p] \equiv [2^m + p]$ . If  $m > 0$ , then

$$exp[m] \rightarrow_v^* (\lambda x.\lambda y.x(xy)) V_{m-1} \rightarrow_v \lambda y.V_{m-1}(V_{m-1}y)$$

and

$$\lambda y.V_{m-1}(V_{m-1}y)[p] \rightarrow_v^* V_{m-1}[2^{m-1} + p] \rightarrow_v^* [2^{m-1} + 2^{m-1} + p] \equiv [2^m + p]$$

$tower^n$  is

$$\lambda x.(\lambda y.tower^{n-1}y)((\lambda z.z[0])(exp x))$$

Finally, we need terms  $coerc^n : \mathbb{U} \rightarrow !^n \mathbb{U}$  such that  $coerc^n [m] \rightarrow_v^* [m]$ .  $coerc^0$  is simply the identity, while  $coerc^n$  is  $\lambda x.iter_{\mathbb{U}} x c_{\mathbb{U}}^1 (\lambda x.c_{\mathbb{U}}^2 x)$  for every  $n \geq 1$ . We can suppose there is  $d$  such that  $\mathcal{M}$  performs at most  $2_d(n)$  steps processing any input of length  $n$ . The term  $M_f$  encoding  $f$  will then be:

$$\lambda x.(\lambda y.final y)((\lambda z.\lambda v.iter_{\mathbb{U}} z trans (init v))(coerc^{2_d} x)(tower^d x))$$

This concludes the proof.  $\square$

For the extended system a principal type property can be proved, extending the algorithm defined in Table 5 in order to take into account the new constants. Clearly, the system can further be extended with other constants without losing its nice properties, provided Lemma 5.8 is satisfied.

**7.2. Restricted Second Order Quantification.** If we had the full power of second-order quantification in **ETAS**, we would easily found a counterexample to Substitution Lemma and, as a consequence, to Subject Reduction. Consider the following type derivation:

$$\frac{x : \forall\alpha.\alpha \mid \emptyset \mid \emptyset \vdash x : \forall\alpha.\alpha}{x : \forall\alpha.\alpha \mid \emptyset \mid \emptyset \vdash x : !\beta}$$

This shows we would be able to give type  $!\beta$  to the variable  $x$ , but without using any instance of rule  $!$ . This undermines any hope to prove subject reduction in presence of types like  $\forall\alpha.\alpha$ . The same holds when we have types in the form  $\forall\alpha.!A$ . Restricting second order quantification to arrow types (and, recursively, to second-order types) allows us to preserve all results we proved in sections 4 and 5.

Formally, a subclass of formulae can be defined by the following two productions

$$S ::= A \multimap A \mid \forall\alpha.S$$

and the following rules are added to the type system:

$$\frac{\Gamma \mid \Delta \mid \Theta \vdash M : S \quad \alpha \notin FV(\Gamma) \cup FV(\Delta) \cup FV(\Theta)}{\Gamma \mid \Delta \mid \Theta \vdash M : \forall\alpha.S} I_{\forall} \quad \frac{\Gamma \mid \Delta \mid \Theta \vdash M : \forall\alpha.S}{\Gamma \mid \Delta \mid \Theta \vdash M : S\{A/\alpha\}} E_{\forall}$$

As can be easily checked, Theorem 4.6 and Theorem 5.11 still hold.

Type inference in presence of second order is at least problematic [17]. We conjecture that, even if second order quantification is the restricted one described here, decidability of the type inference is lost.

## 8. EXTENSIONS TO OTHER LOGICS

We believe the approach described in this paper to be applicable to other logics besides Elementary Affine Logic. Two examples are Light Affine Logic [1] and Soft Affine Logic [5]. Light Affine Logic needs an additional modality, denoted with  $\S$ . So, there will be two modal rules:

$$\frac{\frac{\Gamma_1 \mid \Delta_1 \mid \emptyset \vdash M : A \quad |\Gamma_1| + |\Delta_1| \leq 1}{\Gamma_2 \mid !\Gamma_1, !\Delta_1, \Delta_2 \mid \Theta_2 \vdash M : !A} !}{\Gamma_1, \Gamma_2 \mid \Delta_1, \Delta_2 \Delta_3 \mid \Theta_1 \Theta_2 \vdash M : A} \S$$

$$\frac{\S\Gamma_1, \S\Delta_1, \Gamma_4 \mid !\Gamma_2, !\Delta_2, !\Theta_1, \Delta_4 \mid \S\Theta_2, \S\Delta_3, \Theta_4 \vdash M : \S A}{\S\Gamma_1, \S\Delta_1, \Gamma_4 \mid !\Gamma_2, !\Delta_2, !\Theta_1, \Delta_4 \mid \S\Theta_2, \S\Delta_3, \Theta_4 \vdash M : \S A} \S$$

Notice that we do not need an additional context for the new paragraph modality, since contraction on formulae like  $\S A$  is not allowed.

Soft Affine Logic is even simpler than elementary affine logic: there is just one modality and the context is split into just two sub-contexts. The  $!$  rule becomes:

$$\frac{\Gamma_1 \mid \Delta_1 \vdash M : A}{!\Gamma_1, \Gamma_2 \mid !\Delta_1, \Delta_2 \vdash M : !A} !$$

However, the contraction in **SAL** is deeply different from the one in **EAL** and **LAL**. In particular the formula  $!A \multimap !A \otimes !A$  is not provable anymore and is “replaced” by

$$!A \multimap \underbrace{A \otimes \dots \otimes A}_{n \text{ times}}$$

In the type system the axiom above is reflected by the following two rules:

$$\frac{\Gamma \mid \Delta, x : A \vdash M : B}{\Gamma \mid \Delta, x :!A \vdash M : B} \epsilon_1 \quad \frac{\Gamma \mid x : A, \Delta \vdash M : B}{\Gamma, x :!A \mid \Delta \vdash M : B} \epsilon_2$$

Notice that the analogous of Shifting Lemma (Lemma 4.4 of Section 4) stating that every formula in left context can be shifted to the right one holds in this case too.

## 9. CONCLUSIONS

We presented a type system for the call-by-value lambda-calculus, called **ETAS**, designed from Elementary Affine Logic and enjoying subject reduction and elementary time normalization. Inference of principal types can be done in polynomial time thanks to the fact that the type system is *almost* syntax directed. We believe the approach to be extendible to other systems besides **EAL**, in particular to Light Affine Logic and Soft Affine Logic (as sketched in Section 8). Moreover, we show that adding constants for iteration makes the system (extensionally) complete for elementary time, without altering its good properties. We briefly discuss also the problem of extending the system with second order.

## REFERENCES

- [1] Andrea Asperti. Light affine logic. In *Proceedings of the 13th IEEE Symposium on Logic in Computer Science*, pages 300–308, 1998.
- [2] Andrea Asperti and Luca Roversi. Intuitionistic light affine logic. *ACM Transactions on Computational Logic*, 3(1):137–175, 2002.
- [3] Vincent Atassi, Patrick Baillot, and Kazushige Terui. Verification of Ptime reducibility for system F terms via dual light affine logic. *Logical Methods in Computer Science*, 3(4), 2007.
- [4] Patrick Baillot. Checking polynomial time complexity with types. In *Proceedings of the 2nd IFIP International Conference on Theoretical Computer Science*, pages 370–382, 2002.
- [5] Patrick Baillot and Virgile Mogbil. Soft lambda-calculus: a language for polynomial time computation. In *Proceedings of the 7th International Conference on Foundations of Software Science and Computational Structures*, pages 27–41, 2004.
- [6] Patrick Baillot and Kazushige Terui. Light types for polynomial time computation in lambda-calculus. In *Proceedings of the 19th IEEE Symposium on Logic in Computer Science*, pages 266–275, 2004.
- [7] Patrick Baillot and Kazushige Terui. A feasible algorithm for typing in elementary affine logic. In *Proceedings of the 8th International Conference on Typed Lambda-Calculi and Applications*, pages 55–70, 2005.
- [8] Corrado Böhm and Alessandro Berarducci. Automatic synthesis of typed lambda-programs on term algebras. *Theoretical Computer Science*, 39:135–154, 1985.
- [9] Paolo Coppola, Ugo Dal Lago, and Simona Ronchi Della Rocca. Elementary affine logic and the call by value lambda calculus. In Pawel Urzyczyn, editor, *TLCA '05*, volume 3461 of *Lecture Notes in Computer Science*, pages 131–145. Springer, 2005.
- [10] Paolo Coppola and Simone Martini. Typing lambda terms in elementary logic with linear constraints. In *Proceedings of the 6th International Conference on Typed Lambda-Calculi and Applications*, pages 76–90, 2001.
- [11] Paolo Coppola and Simona Ronchi della Rocca. Principal typing in elementary affine logic. In *Proceedings of the 6th International Conference on Typed Lambda-Calculi and Applications*, pages 90–104, 2003.
- [12] Ugo Dal Lago and Patrick Baillot. Light affine logic, uniform encodings and polynomial time. *Mathematical Structures in Computer Science*, 16(4):713–733, 2006.
- [13] Jean-Yves Girard. Light linear logic. *Information and Computation*, 143(2):175–204, 1998.

- [14] Yves Lafont. Soft linear logic and polynomial time. *Theoretical Computer Science*, 318(1-2):163–180, 2004.
- [15] Kazushige Terui. *Light logic and polynomial time computation*. PhD thesis, Keio University, 2002.
- [16] Kazushige Terui. Light affine lambda calculus and polytime strong normalization. *Archive for Mathematical Logic*, 46(3-4):253–280, 2007.
- [17] Joe Wells. Typability and type checking in the second-order lambda -calculus are equivalent and undecidable. In *Proceedings of the 9th IEEE Symposium on Logic in Computer Science*, pages 176–185, 1994.