A Feasible Algorithm for Typing in Elementary Affine Logic

Patrick Baillot⋆[1] and Kazushige Terui⋆⋆[2]

[1] Laboratoire d'Informatique de Paris-Nord / CNRS, Université Paris-Nord, France.
pb@lipn.univ-paris13.fr
[2] National Institute of Informatics, Tokyo, Japan.
terui@nii.ac.jp

Abstract. We give a new type inference algorithm for typing lambda-terms in Elementary Affine Logic (EAL), which is motivated by applications to complexity and optimal reduction. Following previous references on this topic, the variant of EAL type system we consider (denoted EAL*) is a variant where sharing is restricted to variables and without polymorphism. Our algorithm improves over the ones already known in that it offers a better complexity bound: if a simple type derivation for the term t is given our algorithm performs EAL* type inference in polynomial time in the size of the derivation.

1 Introduction

Linear logic (LL) has proved a fruitful logical setting in which computational complexity can be brought into the picture of the proofs-as-programs correspondence, since the early work [GSS92]. In particular Light linear logic ([Gir98]) and Soft linear logic ([Laf04]) are variants of LL in which all numerical functions programmed are polynomial time. Another system, Elementary linear logic (ELL, see [Gir98,DJ03]) corresponds to Kalmar elementary complexity.

Hence one can consider specific term calculi designed through the Curry-Howard correspondence and program directly in these languages with the guaranteed complexity bound ([Rov98,Ter01]). However this turns out in practice to be a difficult task, in particular because these languages require managing specific constructs corresponding to the logical modalities. Considering the *affine* variant (i.e. with unrestricted weakening) of these systems is an advantage ([Asp98]) but does not suppress the difficulty.

An alternative point of view is to keep ordinary lambda-calculus and use the logic as a type system: then if a program is well-typed the logic provides a way to execute it with the guaranteed complexity bound. The difficulty is then moved to the problem of type inference. This approach and the corresponding type inference problems have been studied in [CM01,CRdR03] for Elementary affine logic (EAL) and [Bai02,Bai04] for

⋆ Work partially supported by project CRISS ACI *Sécurité informatique* and project GEOCAL ACI *Nouvelles interfaces des mathématiques*.

⋆⋆ Work partially supported by Grant-in-Aid for Scientific Research, MEXT, Japan. This work was started during a visit of this author at Université Paris-Nord, in september 2004.

Light affine logic (LAL). It was shown that type inference in the propositional fragments of these systems is decidable.

Typing in EAL is actually also motivated by another goal (see [CM01,ACM00]): EAL-typed terms can be evaluated with the optimal reduction discipline much more easily than general terms, by using only the abstract part of Lamping's algorithm. Thus EAL has been advocated as a promising type system for performing efficient optimal reduction, using the following strategy: given a term, first try to infer an EAL type and if there is one then evaluate the term using Lamping's abstract algorithm. To succeed, this approach would require: an efficient type inference algorithm, evidence that the class of EAL-typable terms is large enough and includes interesting programs, and finally a proof that those terms are indeed evaluated in a faster way with Lamping's algorithm. Maybe intersection types could also be a useful tool in this direction ([Car04]).

However though the type inference problems for EAL and LAL have been shown decidable the algorithms provided, either for EAL or LAL, are not really efficient. They all run at least in exponential time, even if one considers as input a simply typed lambda-term. Our goal is to improve this state-of-the-art by providing more efficient and possibly simpler algorithms.

In this paper we propose a new algorithm for EAL typing, which is therefore a contribution to the perspective of EAL-driven optimal reduction discussed above. This is also a first step for designing an efficient inference procedure for Dual light affine logic (DLAL, [BT04a]) which is a simplification of LAL and corresponds to Ptime computation.

Contribution. Technically speaking the main difficulty with EAL typing is to find out *where* in the derivation to place !-rules and to determine *how many* of them to put. This corresponds in proof-nets terminology to placing *boxes*. The algorithms in [CM01] and [CRdR03] are based on two tactics for *first* placing abstract boxes and *then* working out their number using linear constraints. Our approach also uses linear constraints but departs from this point of view by determining the place of boxes *dynamically*, at the time of constraints solving. This method was actually already proposed in [Bai02] for LAL typing but with several conditions; in particular the term had to be in normal form. In the present work we show that in a system without sharing of subterms other than variables (like DLAL, but unlike LAL), this approach is considerably simplified. In particular it results that:

- one can use as intermediary syntax a very simple term calculus (introduced in [AR02]) instead of proof-nets like in [Bai02];
- the procedure can be run in polynomial time, if one considers as input a simply typed lambda-term (instead of an untyped lambda-term).

Outline. The paper will proceed as follows: in section 2 we introduce Elementary affine logic and the type system $\mathrm{EAL}^\star$ we consider for lambda-calculus; in section 3 we describe the term calculus (*pseudo-terms*, or *concrete syntax*) we will use to denote $\mathrm{EAL}^\star$ derivations and we prove a theorem (Theorem 8) on $\mathrm{EAL}^\star$ typability; finally in section 4 we give an $\mathrm{EAL}^\star$ decoration algorithm (based on Theorem 8), prove it can be run in polynomial time (4.2) and derive from it an $\mathrm{EAL}^\star$ type inference algorithm (4.3).

Acknowledgements. We are grateful to an anonymous referee who suggested important remarks about optimal reduction and possible improvement of the present work.

Notations. Given a lambda-term M we denote by $FV(M)$ the set of its free variables. Given a variable x we denote by $no(x,M)$ the number of occurrences of x in M. We denote by $|M|$ the structural size of a term M. We denote substitution (without capture of variable) by $M[N/x]$. When there is no ambiguity we will write $M[M_i/x_i]$ for $M[M_1/x_1,\ldots,M_n/x_n]$.

Notations for lists: ϵ will denote the empty list and pushing element a on list l will be denoted by $a :: l$. The prefix relation on lists will be denoted by $\leq$.

2 Typing in Elementary Affine Logic

The formulas of Intuitionistic multiplicative Elementary affine logic (Elementary affine logic for short, EAL) are given by the following grammar:

$$A, B ::= \alpha \mid A \multimap B \mid !A \mid \forall\alpha.A$$

We restrict here to propositional EAL (without quantification). A natural deduction presentation for this system is given on Figure 1.

$$\frac{}{A \vdash A}\ (\text{var}) \qquad \frac{\Gamma \vdash B}{\Gamma, A \vdash B}\ (\text{weak})$$

$$\frac{\Gamma_1 \vdash A \multimap B \quad \Gamma_2 \vdash A}{\Gamma_1, \Gamma_2 \vdash B}\ (\text{appl}) \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B}\ (\text{abst})$$

$$\frac{\Gamma \vdash !A \quad !A, \ldots, !A, \Delta \vdash B}{\Gamma, \Delta \vdash B}\ (\text{contr})$$

$$\frac{\Gamma_1 \vdash !A_1 \quad \cdots \quad \Gamma_n \vdash !A_n \quad A_1, \ldots, A_n \vdash B}{\Gamma_1, \ldots, \Gamma_n \vdash !B}\ (\text{prom})$$

Fig. 1. Natural deduction for EAL.

We call *erasure* A^- of an EAL formula A the simple type defined inductively by:

$$\alpha^- = \alpha,\ (!A)^- = A^-,\ (A \multimap B)^- = A^- \to B^-.$$

Conversely, given a simple type T we say that an EAL formula A is a *decoration* of T if we have $A^- = T$.

We will use EAL as a type system for lambda-terms, but in a way more constrained than that allowed by this natural deduction presentation:

Definition 1. Let M be a lambda-term; we say M is typable in EAL* with type $\Gamma \vdash M : A$ if there is a derivation of this judgment in the system from Figure 2.

Notice that the rule (contr) is restricted and an affinity condition is imposed on the rule (prom). The effect is that it does not allow *sharing* of subterms other than variables. This comes in contrast with the computational study of ELL carried out for instance in [DJ03] but is motivated by several points:

– With our restrictions, terms and derivations correspond more closely to each other. For instance, the size of a typed term M is always linear in the length (i.e. the number of typing rules) of its type derivation.
– This approach where sharing is restricted to variables (and not arbitrary subterms) is enough to define Dual Light Affine Logic (DLAL) typing ([BT04a]) which is sufficient to capture polynomial time computation.
– It is not hard to see that our notion of EAL*-typability precisely coincides with the EAL-typability for lambda-terms considered by Coppola and Martini in [CM01] (see [BT04b]). As argued in their paper [CM01], sharing-free derivations are necessary to be able to use EAL for optimal reduction with the abstract part of Lamping's algorithm.
– Finally: using sharing of arbitrary subterms would make type inference more difficult ...

$$\frac{}{x : A \vdash x : A}\ (\text{var}) \qquad \frac{\Gamma \vdash M : B}{\Gamma, x : A \vdash M : B}\ (\text{weak})$$

$$\frac{\Gamma_1 \vdash M_1 : A \multimap B \quad \Gamma_2 \vdash M_2 : A}{\Gamma_1, \Gamma_2 \vdash (M_1)M_2 : B}\ (\text{appl}) \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \multimap B}\ (\text{abst})$$

$$\frac{x_1 : !A, \ldots, x_n : !A, \Delta \vdash M : B}{x : !A, \Delta \vdash M[x/x_1, \ldots, x_n] : B}\ (\text{contr})$$

$$\frac{\Gamma_1 \vdash M_1 : !A_1 \quad \cdots \quad \Gamma_n \vdash M_n : !A_n \quad x_1 : A_1, \ldots, x_n : A_n \vdash M : B}{\Gamma_1, \ldots, \Gamma_n \vdash M[M_i/x_i] : !B}\ (\text{prom})$$

In the rule (prom), each x_i occurs at most once in M.

Fig. 2. Typing rules for EAL*.

3 Concrete syntax and box reconstruction

3.1 Pseudo-terms

In order to describe the structure of type derivations we need a term calculus more informative than lambda-calculus. We will use the language introduced in [AR02] (called *concrete syntax* in this paper), which is convenient because it has no explicit construct neither for boxes, nor for contractions. It was stressed in this reference that this syntax is not faithful for LAL: several type derivations (LAL proofs) correspond to the same term. However it is faithful for EAL*, precisely because sharing is restricted to variables and there is no ambiguity on the placement of contractions.

Let us introduce *pseudo-terms*:

$$t, u ::= x \mid \lambda x.t \mid (t)u \mid !t \mid \bar{!}t$$

The basic idea is that ! constructs correspond to main doors of boxes in *proof-nets* ([Gir87,AR02]) while $\bar{!}$ constructs correspond to auxiliary doors of boxes. But note that there is no information in the pseudo-terms to link occurrences of ! and $\bar{!}$ corresponding to the same box.

There is a natural erasure map $(.)^-$ from pseudo-terms to lambda-terms consisting in removing all occurrences of ! and $\bar{!}$. When $t^- = M$, t is called a *decoration* of M.

For typing pseudo-terms the rules are the same as in Definition 1 and Figure 2, but for (prom):

$$\frac{\Gamma_1 \vdash t_1 : !A_1 \quad \cdots \quad \Gamma_n \vdash t_n : !A_n \quad x_1 : A_1, \ldots, x_n : A_n \vdash t : B}{\Gamma_1, \ldots, \Gamma_n \vdash !t\,[\bar{!}t_i/x_i] : !B} \text{ (prom)}$$

We want to give an algorithm to determine if a pseudo-term can be typed in EAL*: this can be seen as a kind of correctness criterion allowing to establish if boxes can be reconstructed in a suitable way; this issue will be examined in 3.2.

Actually, when searching for EAL* type derivations for (ordinary) lambda-terms it will be interesting to consider a certain subclass of derivations. A type derivation in EAL* is *restricted* if in all applications of the rule (prom),

(i) the subject M of the main premise $x_1 : A_1, \ldots, x_n : A_n \vdash M : B$ is not a variable, and
(ii) the last rules to derive auxiliary premises $\Gamma_i \vdash M_i :!A_i$ ($1 \leq i \leq n$) are either (var) or (appl).

A pseudo-term is *restricted* if it is obtained by the following grammar:

$$\begin{aligned} a &::= x \mid \lambda x.t \mid (t)t \\ t &::= !^m a, \end{aligned}$$

where m is an arbitrary value in $\mathbb{Z}$ and $!^m a$ is defined by:

$$!^m a = \underbrace{! \cdots !}_{m\ times}\ a \quad \text{if } m \geq 0; \quad !^m a = \underbrace{\bar{!} \cdots \bar{!}}_{-m\ times}\ a \quad \text{if } m < 0.$$

We then have the following proposition (see [BT04b] for the proof):

Proposition 2.

1. *(For lambda-terms) if $\Gamma \vdash M : A$ has a type derivation, then it also has a restricted type derivation.*
2. *(For pseudo-terms) Every restricted derivation yields a restricted pseudo-term.*

As a consequence, when a lambda-term M is typable in EAL* one can always find a decoration of M (of the same type) in the set of restricted pseudo-terms.

3.2 Box reconstruction

We will consider words over the language $\mathcal{L} = \{!, \bar{!}\}^\star$.

If t is a pseudo-term and x is an occurrence of variable (either free or bound) in t, we define $t\langle x \rangle$ as the word of $\mathcal{L}$ obtained by listing the occurrences of !, $\bar{!}$ holding x in their scope. More formally:

$$\begin{array}{lll} x\langle x \rangle &= \epsilon, & (!t)\langle x \rangle = ! :: (t\langle x \rangle), \\ (\lambda y.t)\langle x \rangle &= t\langle x \rangle, \ (y \text{ might be equal to } x) & (\bar{!}t)\langle x \rangle = \bar{!} :: (t\langle x \rangle), \\ ((t_1)t_2)\langle x \rangle &= t_i\langle x \rangle \ \text{ where } t_i \text{ is the subterm containing } x. & \end{array}$$

We define a map: $s : \mathcal{L} \to \mathbb{Z}$ by:

$$s(\epsilon) = 0, \quad s(! :: l) = 1 + s(l), \quad s(\bar{!} :: l) = -1 + s(l).$$

We call $s(l)$ the *sum* associated to l.

Let t be a pseudo-term. We say that t satisfies the *bracketing condition* if

– for any occurrence of variable x in t:

$$\forall l \leq t\langle x\rangle,\ s(l) \geq 0,$$

– moreover if x is an occurrence of free variable:

$$s(t\langle x\rangle) = 0.$$

That is to say: if ! is seen as an opening bracket and $\bar{!}$ as a closing bracket, in $t\langle x\rangle$ any $\bar{!}$ matches a ! (we will say that $t\langle x\rangle$ is *weakly well-bracketed*) and if x is free $t\langle x\rangle$ is well-bracketed.

We say t satisfies the *scope condition* if: for any subterm $\lambda x.v$ of t, for any occurrence x_i of x in v, $v\langle x_i\rangle$ is well-bracketed:

– $\forall l \leq v\langle x_i\rangle, \quad s(l) \geq 0$,
– and $s(v\langle x_i\rangle) = 0$.

It is obvious that:

Lemma 3. *If t is a pseudo-term which satisfies the scope condition, then any subterm of t also satisfies this condition.*

Proposition 4. *If t is an EAL* typed pseudo term, then t satisfies the bracketing and scope conditions.*

Proof. By induction on the EAL* type derivations.

For instance the two following pseudo-terms are *not* EAL* typable:

$$!\lambda f.!((\bar{!}f)(\bar{!}f)\bar{!}^3 x), \qquad !\lambda f.!((\bar{!}^2 f)(\bar{!}f)\bar{!}^2 x),$$

the first one because it does not satisfy bracketing, and the second one because it does not satisfy the scope condition (because of the first occurrence of f).

Now, we can observe the following property:

Lemma 5 (Boxing). *If $!u$ is a pseudo-term which satisfies the bracketing and scope conditions then there exist v, $u_1, \ldots, u_n$ unique (up to renaming of v's free variables) such that:*

– $FV(v) = \{x_1, \ldots, x_n\}$ *and for* $1 \leq i \leq n$, $no(x_i, v) = 1$,
– $!u = !v[\bar{!}u_1/x_1, \ldots, \bar{!}u_n/x_n]$,
– v *and* u_i*, for* $1 \leq i \leq n$*, satisfy the bracketing condition.*

Proof. We denote by $!_0$ the first occurrence of $!$ in the term considered: $!_0 u$. Denote by $\bar{!}_1, \ldots, \bar{!}_n$ the occurrences of $\bar{!}$ matching $!_0$ in the words $!u\langle x \rangle$, where x ranges over the occurrences of variables in $!u$. Let u_i, with $1 \leq i \leq n$, be the subterms of $!u$ such that $\bar{!}_i u_i$ is a subterm of $!u$, for $1 \leq i \leq n$. Then it is clear that no u_i is a subterm of a u_j, for $i \neq j$.

Let now v be the pseudo-term obtained from u by replacing each $\bar{!}_i u_i$ by a distinct variable x_i. Let us show that inside t, no occurrence of variable in u_i can be bound by a λ in v. Indeed assume it was the case for an occurrence y in u_i and let $\lambda y.w$ denote the subterm of t starting with λy. Then $\lambda y.w$ would be of the form $\lambda y.w'\{\bar{!}u_i/x_i\}$, where $v_1\{v_2/x\}$ denotes the syntactic substitution of x by v_2 in v_1 (i.e. possibly with variable capture). One can check that the scope condition for t would then be violated, hence a contradiction.

Therefore we have $!u = !v[\bar{!}u_1/x_1, \ldots, \bar{!}u_n/x_n]$ (without variable capture), and by definition of $\bar{!}_i$ we know that for $1 \leq i \leq n$, $v\langle x_i \rangle$ is well-bracketed.

Finally let us assume x is an occurrence of free variable in v distinct from x_i, for $1 \leq i \leq n$. Then x is an occurrence of free variable in $!u$, and as $!u$ is well-bracketed we have that $s(!u\langle x \rangle) = 0$, hence x is in the scope of a $\bar{!}_0$ matching $!_0$. Then $\bar{!}_0$ must be one of the $\bar{!}_i$, for $1 \leq i \leq n$, hence x is in u_i and thus does not occur in v, which gives a contradiction. Therefore we have $FV(v) = \{x_1, \ldots, x_n\}$.

Let us show that v satisfies bracketing. Let y be an occurrence of variable in v. If y is free we already know that $v\langle y \rangle$ is well-bracketed. If y is bound then $!v\langle y \rangle = !u\langle y \rangle$. So if $l \leq !v\langle y \rangle$ and $l \neq \epsilon$, then $s(l) \geq 1$, therefore $\forall l \leq v\langle y \rangle,\ s(l) \geq 0$. So v satisfies the bracketing condition. It is easy to check that the u_is also satisfy the bracketing condition.

Given a pseudo-term t we call *EAL type assignment* for t a map Γ from the variables of t (free or bound) to EAL formulas. EAL type assignments are simply called *assignments* when there is no danger of confusion. This map Γ is extended to a partial map from subterms of t to EAL formulas by the following inductive definition:

$$\begin{array}{lll}
\Gamma(!u) & = !A, & \text{if } \Gamma(u) = A, \\
\Gamma(\bar{!}u) & = A, & \text{if } \Gamma(u) = !A, \text{ undefined otherwise}, \\
\Gamma(\lambda x.u) & = A \multimap B, & \text{if } \Gamma(x) = A, \Gamma(u) = B, \\
\Gamma((u_1)u_2) & = B, & \text{if } \Gamma(u_2) = A \text{ and } \Gamma(u_1) = A \multimap B, \text{ undefined otherwise.}
\end{array}$$

Given a pair (t, Γ) of a pseudo-term t and an assignment Γ (we omit Γ if it is natural from the context) we say that (t, Γ) satisfies the *typing condition* if:

- $\Gamma(t)$ is defined (so in particular each subterm of t of the form $(u_1)u_2$ satisfies the condition above),
- for any variable x of t which has at least 2 occurrences we have: $\Gamma(x)$ is of the form $!B$ for some formula B.

Given an EAL* type derivation for a pseudo-term t there is a natural assignment Γ obtained from this derivation: the value of Γ on free variables is obtained from the environment of the final judgment and its value on bound variables from the type of the variable in the premise of the abstraction rule in the derivation.

Proposition 6. *If t is an EAL* typed pseudo-term and Γ is an associated assignment then (t, Γ) satisfies the typing condition.*

Moreover it is easy to observe that:

Lemma 7. *If (t, Γ) satisfies the typing condition and u is a subterm of t, then (u, Γ) also satisfies the typing condition.*

Now, the conditions on pseudo-terms we have listed up to now are sufficient to ensure that t is an EAL* typed pseudo-term:

Theorem 8. *If t is a pseudo-term and Γ an assignment such that:*

- *t satisfies the bracketing and scope conditions,*
- *(t, Γ) satisfies the typing condition,*

then t is typable in EAL* with a judgment $\Delta \vdash t : A$ such that: $\Gamma(t) = A$ and Δ is the restriction of Γ to the free variables of t.

Proof. Let us use the following enumeration for the conditions:
(i) bracketing, (ii) scope, (iii) typing.

The proof proceeds by structural induction on the pseudo-term t. Let us just deal here with the case $t = !u$. The complete proof can be found in [BT04b].

By the Boxing Lemma 5, t can be written as $t = !v[\bar{!}u_1/x_1, \dots, \bar{!}u_n/x_n]$ where $FV(v) = \{x_1, \dots, x_n\}$ and each $v\langle x_i \rangle$ is well-bracketed. By Lemma 5 again, each u_i satisfies (i).

By Lemmas 3 and 7 as t satisfies (ii) and (iii), u_i also satisfies (ii) and (iii). Therefore by induction hypothesis we get that there exists an EAL* derivation of conclusion:

$$\Delta_i \vdash u_i : A_i,$$

where $A_i = \Gamma(u_i)$, for $1 \leq i \leq n$.

Let us now examine the conditions for v. As t satisfies the bracketing condition and by the Boxing Lemma 5, we get that v satisfies (i). By the Boxing Lemma again we know that all free variables of v have exactly one occurrence. It is easy to check that as t satisfies the scope condition (ii), so does v.

Consider now the typing condition. Let $\tilde{\Gamma}$ be defined as Γ but $\tilde{\Gamma}(x_i) = \Gamma(\bar{!}u_i)$ for $1 \leq i \leq n$. If y has several occurrences in v then it has several occurrences in t, hence $\Gamma(y) = !B$, so $\tilde{\Gamma}(y) = !B$. If $(v_1)v_2$ is a subterm of v then $(v_1')v_2'$, where $v_i' = v_i[\bar{!}u_1/x_1, \dots, \bar{!}u_n/x_n]$, is a subterm of t and $\tilde{\Gamma}(v_i') = \Gamma(v_i)$. Therefore as (t, Γ) satisfies the typing condition, then so does $(v, \tilde{\Gamma})$.

As $\Gamma(u_i) = A_i$ and $\Gamma(\bar{!}u_i)$ is defined we have $A_i = !B_i$ and $\tilde{\Gamma}(x_i) = B_i$. Finally as v satisfies conditions (i)–(iii), by i.h. there exists an EAL* derivation of conclusion:

$$x_1 : B_1, \dots, x_n : B_n \vdash v : C,$$

where $C = \tilde{\Gamma}(v)$.

If u_i and u_j for $i \neq j$ have a free variable y in common then as t satisfies the typing condition we have $\Gamma(y) = !B$. We rename the free variables common to several of the u_is, apply a (prom) rule to the judgments on u_i and the judgment on v, then some (contr) rules and get a judgment: $\Delta' \vdash t : !C$. Hence the i.h. is valid for t.

4 A decoration algorithm

4.1 Decorations and instantiations

We consider the following *decoration problem*:

Problem 9 (decoration). Let $x_1 : A_1, \dots, x_n : A_n \vdash M : B$ be a simply typed term; does there exist EAL decorations A'_i of the A_i for $1 \leq i \leq n$ and B' of B such that $x_1 : A'_1, \dots, x_n : A'_n \vdash M : B'$ is a valid EAL* judgment for M?

For that we will need to find out the possible concrete terms corresponding to M. Actually following section 3.1 and Prop. 2 it is sufficient to search for a suitable term in the set of restricted pseudo-terms, instead of considering the whole set of pseudo-terms. To perform this search we will use *parameters*: $\mathbf{n}, \mathbf{m}, \mathbf{k}, \dots$. The *parameterized pseudo-terms* are defined by the following grammar:

$$a ::= x \mid \lambda x.t \mid (t)t, \qquad t ::= !^{\mathbf{n}}a,$$

where $\mathbf{n}$ is a parameter (and not an integer).

To parameterize types, we will also use *linear combinations of parameters* $\mathbf{c}, \mathbf{d}, \dots$ defined by:

$$\mathbf{c} ::= 0 \mid \mathbf{n} \mid \mathbf{n} + \mathbf{c}.$$

The *parameterized types* are defined by:

$$A ::= !^{\mathbf{c}}\alpha \mid !^{\mathbf{c}}(A \multimap A).$$

Given a parameterized pseudo-term t, a *parameterized type assignment* Σ for t is a map from the variables of t (free or bound) to the parameterized types.

We denote by $par(t)$ ($par(A)$, resp.) the set of parameters occurring in t (A, resp.), and by $par(\Sigma)$ the union of $par(\Sigma(x))$ with x ranging over all the variables of t.

An *instantiation* ϕ for t is a map $\phi : par(t) \rightarrow \mathbb{Z}$. It allows to define a restricted pseudo-term $\phi(t)$ obtained by substituting the integer $\phi(\mathbf{n})$ for each parameter $\mathbf{n}$. Similarly, an *instantiation* ϕ for (t, Σ) is a map $\phi : par(t) \cup par(\Sigma) \rightarrow \mathbb{Z}$. The map ϕ is naturally extended to the linear combinations of parameters. If A is a parameterized type such that $par(A) \subseteq par(\Sigma)$ and moreover $\phi(\mathbf{c})$ is *non-negative* whenever $!^{\mathbf{c}}B$ occurs in A, one can obtain an EAL type $\phi(A)$ by substituting $\phi(\mathbf{n})$ for each parameter $\mathbf{n}$ as above. For instance, $\phi(!^{\mathbf{n}}(!^0\alpha \multimap !^{\mathbf{n}+\mathbf{n}}\alpha)) = !^3(\alpha \multimap !^6\alpha)$ when $\phi(\mathbf{n}) = 3$. An EAL type assignment $\phi\Sigma$ for $\phi(t)$ is then obtained by $\phi\Sigma(x) = \phi(\Sigma(x))$ when $\phi(\Sigma(x))$ is defined for all variables x of t.

We define the *size* $|A|$ of a parameterized formula A as the structural size of its underlying simple type (so the sum of the number of $\multimap$ connectives and atomic subtypes), and $|\Sigma|$ as the maximum of $|\Sigma(x)|$ for all variables x. The *erasure map* $(.)^-$ is defined for parameterized pseudo-terms and parameterized types analogously to those for pseudo-terms and EAL types.

It is clear that given a lambda-term M there exists a parameterized pseudo-term t such that $t^- = M$ and all occurrences of parameter in t are distinct. We denote t, which is unique up to renaming of its parameters, by $\overline{M}$ and call it the *free decoration* of M. Note that the size of $\overline{M}$ is linear in the size of M. Given a simple type T, its *free decoration* $\overline{T}$ is defined by:

$$\overline{\alpha} = !^{\mathbf{n}}\alpha, \qquad \overline{A \multimap B} = !^{\mathbf{n}}(\overline{A} \multimap \overline{B}),$$

where in the second case we have taken $\overline{A}$ and $\overline{B}$ with disjoint sets of parameters and $\mathbf{n}$ a fresh parameter. Finally, a *simple type assignment* Θ for M is a map from the variables of M to the simple types. Its *free decoration* $\overline{\Theta}$ is defined pointwise, by taking $\overline{\Theta}(x) = \overline{\Theta(x)}$, where all these decorations are taken with disjoint parameters.

The following picture illustrates the relationship among various notions introduced so far:

$$\boxed{\begin{array}{c}\text{pseudo-terms}\\ \text{EAL types}\\ \text{EAL typ. assign.}\end{array}} \xleftarrow{\text{instantiation}} \boxed{\begin{array}{c}\text{param. pseudo-terms}\\ \text{param. types}\\ \text{param. typ. assign.}\end{array}} \underset{\text{free decoration}}{\overset{\text{erasure}}{\rightleftarrows}} \boxed{\begin{array}{c}\text{lambda-terms}\\ \text{simple types}\\ \text{simple typ. assign.}\end{array}}$$

Given a simple type derivation of $x_1 : T_1, \dots, x_n : T_n \vdash M : T$, one can naturally obtain a simple type assignment Θ for M. Furthermore, it is automatic to build a parameterized pseudo-term $\overline{M}$ and a parameterized type assignment $\overline{\Theta}$ for $\overline{M}$. If ϕ is an instantiation for $(\overline{M}, \overline{\Theta})$ such that $\phi(\overline{T_i})$ and $\phi(\overline{T})$ are defined (i.e. $\phi(\mathbf{n}) \geq 0$ for all $\mathbf{n} \in par(\overline{T_1}) \cup \cdots par(\overline{T_n}) \cup par(\overline{T})$), then $\phi(\overline{T_i})$ is a decoration of T_i for $1 \leq i \leq n$ and $\phi(\overline{T})$ is a decoration of T. Conversely, any decorations of T_i's and T are obtained through some instantiations for $(\overline{M}, \overline{\Theta})$. Therefore, the decoration problem boils down to the following *instantiation problem*:

Problem 10 (instantiation). Given a parameterized pseudo-term t and a parameterized type assignment Σ for it: does there exist an instantiation ϕ for (t, Σ) such that $\phi(t)$ has an EAL* type derivation associated to $\phi\Sigma$?

To solve this problem we will use Theorem 8 to find suitable instantiations ϕ if there exists any. For that we will need to be able to state the conditions of this theorem on parameterized pseudo-terms; they will yield linear constraints. We will speak of *linear inequations*, meaning in fact both linear equations and linear inequations.

We will consider lists over parameters $\mathbf{n}$. Let us denote by $\mathcal{L}'$ the set of such lists.

As for pseudo-terms we define for t a parameterized pseudo-term and x an occurrence of variable in t, a list $t\langle x \rangle$ in $\mathcal{L}'$ by:

$$\begin{array}{lcl} x\langle x \rangle &=& \epsilon, \qquad ((t_1)t_2)\langle x \rangle = t_i\langle x \rangle \ \text{ where } t_i \text{ is the subterm containing } x, \\ (!^{\mathbf{n}}a)\langle x \rangle &=& \mathbf{n} :: (a\langle x \rangle), \qquad (\lambda y.t)\langle x \rangle = t\langle x \rangle \ \ (y \text{ might be equal to } x). \end{array}$$

The sum $s(l)$ of an element l of $\mathcal{L}'$ is a linear combination defined by:

$$s(\epsilon) = 0, \quad s(\mathbf{n} :: l) = \mathbf{n} + s(l).$$

Let t be a parameterized pseudo-term. We define the *boxing constraints* for t as the set of linear inequations $\mathcal{C}^b(t)$ obtained from t in the following way:

– bracketing: for any occurrence of variable x in t and any prefix l of $t\langle x \rangle$, add the inequation: $s(l) \geq 0$; moreover if x is an occurrence of free variable add the equation $s(t\langle x \rangle) = 0$.

– scope: for any subterm $\lambda x.v$ of t, for any occurrence x_i of x in v, add similarly the inequations expressing the fact that $v\langle x_i \rangle$ is well-bracketed.

It is then straightforward that:

Proposition 11. *Given an instantiation ϕ for t, we have: $\phi(t)$ satisfies the bracketing and scope conditions iff ϕ is a solution of $\mathcal{C}^b(t)$.*

Note that the number of inequations in $\mathcal{C}^b(t)$ is polynomial in the size of t (hence also in the size of t^-).

In the sequel, we will need to unify parameterized types. For that, given 2 parameterized types A and B we define their unification constraints $U(A, B)$ by:

$$U(!^{\mathbf{c}}\alpha, !^{\mathbf{d}}\alpha) = \{\mathbf{c} = \mathbf{d}\}$$
$$U(!^{\mathbf{c}}(A_1 \multimap A_2), !^{\mathbf{d}}(B_1 \multimap B_2)) = \{\mathbf{c} = \mathbf{d}\} \cup U(A_1, B_1) \cup U(A_2, B_2)$$

and $U(A, B) = \{\text{FALSE}\}$ (unsolvable constraint) in the other cases.

Let Σ be a parameterized type assignment for t. Then we extend Σ to a partial map from the subterms of t to parameterized types in the following way:

$$\begin{array}{lll}
\Sigma(!^{\mathbf{n}}a) & = !^{\mathbf{n}+\mathbf{c}}A & \text{if } \Sigma(a) = !^{\mathbf{c}}A, \\
\Sigma(\lambda x.u) & = !^0(A \multimap B) & \text{if } \Sigma(x) = A, \Sigma(u) = B, \\
\Sigma((u_1)u_2) = B, & & \text{if } \Sigma(u_1) = !^{\mathbf{c}}(A \multimap B), \text{ undefined otherwise.}
\end{array}$$

We define the *typing constraints* for (t, Σ) as the set of linear inequations $\mathcal{C}^{typ}(t, \Sigma)$ obtained from t, Σ as follows. When $\Sigma(t)$ is not defined, then $\mathcal{C}^{typ}(t, \Sigma) = \{\text{FALSE}\}$. Otherwise:

(*applications*) for any subterm of t of the form $(u_1)u_2$ with $\Sigma(u_1) = !^{\mathbf{c}}(A_1 \multimap B_1)$ and $\Sigma(u_2) = A_2$ add the constraints $U(A_1, A_2) \cup \{\mathbf{c} = 0\}$.

(*bangs*) for any subterm of t of the form $!^{\mathbf{n}}u$ with $\Sigma(u) = !^{\mathbf{c}}A$, add the constraint $\mathbf{n} + \mathbf{c} \geq 0$.

(*contractions*) for any variable x of t which has at least 2 occurrences and $\Sigma(x) = !^{\mathbf{c}}A$, add the constraint $\mathbf{c} \geq 1$.

(*variables*) for any $\mathbf{c}$ such that $!^{\mathbf{c}}B$ is a subtype of $\Sigma(x)$ for some variable x of t, add the constraint $\mathbf{c} \geq 0$.

We then have:

Proposition 12. *Let t be a parameterized pseudo-term and Σ be a parameterized type assignment for t. Given an instantiation ϕ for (t, Σ), we have: $\phi\Sigma$ is defined and $(\phi(t), \phi\Sigma)$ satisfies the typing condition iff ϕ is a solution of $\mathcal{C}^{typ}(t, \Sigma)$.*

Note that the number of inequations in $\mathcal{C}^{typ}(t, \Sigma)$ is polynomial in $(|t| + |\Sigma|)$.

We define $\mathcal{C}(t, \Sigma) = \mathcal{C}^b(t) \cup \mathcal{C}^{typ}(t, \Sigma)$. Using the two previous Propositions and Theorem 8 we get the following result, which solves the instantiation problem:

Theorem 13. *Let t be a parameterized pseudo-term, Σ be a parameterized type assignment for t, and ϕ be an instantiation for (t, Σ). The two following conditions are equivalent:*

– *$\phi\Sigma$ is defined and $\phi(t)$ is typable in EAL* with a judgment $\Delta \vdash \phi(t) : A$ such that $\phi\Sigma(\phi(t)) = A$ and Δ is the restriction of $\phi\Sigma$ to the free variables of $\phi(t)$,*
– *ϕ is a solution of $\mathcal{C}(t, \Sigma)$.*

Moreover the number of inequations in $\mathcal{C}(t, \Sigma)$ is polynomial in $(|t| + |\Sigma|)$.

Finally, we obtain the following result, which solves the decoration problem:

Theorem 14. *Let $x_1 : A_1, \ldots, x_n : A_n \vdash M : B$ be a simply typed term and let Θ be the associated simple type assignment. There exist decorations A'_i of the A_i for $1 \leq i \leq n$ and B' of B such that $x_1 : A'_1, \ldots, x_n : A'_n \vdash M : B'$ is a valid EAL* judgment iff there is a solution ϕ of $\mathcal{C}(\overline{M}, \overline{\Theta})$.*

In this case each solution ϕ gives a suitable EAL* judgment $x_1 : A'_1, \ldots, x_n : A'_n \vdash M : B'$. Moreover the number of inequations and the number of parameters in $\mathcal{C}(\overline{M}, \overline{\Theta})$ are polynomial in $(|M| + |\Theta|)$.

4.2 Solving the constraints

Now we turn our attention to the constraints and their solutions. Let t be a parameterized pseudo-term and Σ be an assignment. We consider instead of the previous instantiation maps with values in $\mathbb{Z}$, maps with rational numbers as values: $\psi : par(t) \cup par(\Sigma) \to \mathbb{Q}$. If ψ is such a map and k is a non-negative integer we defined the map $k\psi$ by: $(k\psi)(\mathbf{n}) = k.\psi(\mathbf{n})$, for any parameter $\mathbf{n}$.

Lemma 15. *If ψ is a solution of $\mathcal{C}(t, \Sigma)$ and k is a strictly positive integer then $k\psi$ is also a solution of $\mathcal{C}(t, \Sigma)$.*

Proof. It is enough to observe that for any inequation of $\mathcal{C}^b(t)$ and $\mathcal{C}^{typ}(t, \Sigma)$ if ψ is a solution then so is $k\psi$:

– all inequations from $\mathcal{C}^b(t)$ and all those from $\mathcal{C}^{typ}(t, \Sigma)$ except the contraction cases are homogeneous (no constant element in combinations) and as $k \geq 0$ the inequalities are preserved when multiplying both members by k;
– the inequations coming from the contraction cases in $\mathcal{C}^{typ}(t, \Sigma)$ are of the form $\mathbf{m} \geq 1$, so as $k \geq 1$ we have: if $\psi(\mathbf{m}) \geq 1$ holds then so does $k\psi(\mathbf{m}) \geq 1$.

Recall that the problem of finding if a linear system of inequations $\mathcal{C}$ admits a solution in $\mathbb{Q}$ can be solved in polynomial time in the size of $\mathcal{C}$ and its number of variables. Hence we have:

Proposition 16. *The problem of whether the system $\mathcal{C}(t, \Sigma)$ admits a solution with values in $\mathbb{Z}$ can be solved in time polynomial in $(|t| + |\Sigma|)$.*

Proof. As the number of inequations and the number of parameters in $\mathcal{C}^{typ}(t, \Sigma)$ are polynomial in $(|t| + |\Sigma|)$ and by the result we recalled above we have: one can decide if $\mathcal{C}^{typ}(t, \Sigma)$ admits a solution with values in $\mathbb{Q}$ in time polynomial in $(|t| + |\Sigma|)$.

Then, if there is no solution in $\mathbb{Q}$ there is no solution in $\mathbb{Z}$. Otherwise if ψ is a solution in $\mathbb{Q}$ take for k the least multiple of the denominators of $\psi(\mathbf{n})$, for all parameters $\mathbf{n}$. Then by Lemma 15, $k\psi$ is a solution in $\mathbb{Z}$.

It then follows that:

Theorem 17. *The decoration problem of Theorem 14 can be solved in time polynomial in $(|M| + |\Theta|)$.*

4.3 Type inference

The procedure for EAL* decoration we have given can be extended to a type inference procedure for EAL* in the way used in [CM01]: given an ordinary term M,

– compute the principal assignment Θ for M (giving the principal simple type),
– use the procedure of Theorem 14 to find if M, Θ admits a suitable EAL* decoration.

It follows from a result of [CRdR03] that:

Proposition 18. *if M is EAL* typable and admits as principal simple type judgment $\Delta \vdash M : A$, then M admits an EAL* type judgment which is a decoration of this judgment.*

See [BT04b] for a self-contained proof of this proposition.

As a consequence, searching for an EAL* decoration of the principal type judgment of M is sufficient to decide if M is EAL* typable. It then follows from Theorem 17 that our EAL* type inference algorithm applied to a term M can be executed in time bounded by a polynomial in $(|M| + |\Theta|)$ where Θ is the principal (simple type) assignment of M.

Note, however, that this does not mean that the overall algorithm is polynomial time in $|M|$, as the principal simple type assignment for M can have a size exponential in $|M|$. Still, type inference in simples types can be performed in polynomial time if one uses a representation with sharing for the types. Further work is needed to examine if using a shared representation for types one can design an algorithm for EAL typing polynomial w.r.t. the size of the untyped term.

4.4 Example

Let us consider a small example to illustrate our method: take $M = \lambda y.\lambda z.(y)(y)z$ (the Church integer 2). The decoration $\overline{M}$ is given by:

$$\overline{M} = !^{\mathbf{m_1}}\lambda y.!^{\mathbf{m_2}}\lambda z.!^{\mathbf{m_3}}[\ (!^{\mathbf{m_4}}y_1)\ !^{\mathbf{m_5}}[\ (!^{\mathbf{m_6}}y_2)!^{\mathbf{m_7}}z\ ]\ ]$$

(we have distinguished the 2 occurrences of y in y_1 and y_2). We get for the boxing constraints:

$$\mathcal{C}^b(\overline{M}) = \begin{cases} \mathbf{m_1} \geq 0\ (1) & \mathbf{m_2} \geq 0\ \ (8) \\ \mathbf{m_1}+\mathbf{m_2} \geq 0\ (2) & \mathbf{m_2}+\mathbf{m_3} \geq 0\ \ (9) \\ \mathbf{m_1}+\mathbf{m_2}+\mathbf{m_3} \geq 0\ (3) & \mathbf{m_2}+\mathbf{m_3}+\mathbf{m_4} = 0\ (10) \\ \mathbf{m_1}+\mathbf{m_2}+\mathbf{m_3}+\mathbf{m_4} \geq 0\ (4) & \mathbf{m_2}+\mathbf{m_3}+\mathbf{m_5} \geq 0\ (11) \\ \mathbf{m_1}+\mathbf{m_2}+\mathbf{m_3}+\mathbf{m_5} \geq 0\ (5) & \mathbf{m_2}+\mathbf{m_3}+\mathbf{m_5}+\mathbf{m_6} = 0\ (12) \\ \mathbf{m_1}+\mathbf{m_2}+\mathbf{m_3}+\mathbf{m_5}+\mathbf{m_6} \geq 0\ (6) & \mathbf{m_3} \geq 0\ (13) \\ \mathbf{m_1}+\mathbf{m_2}+\mathbf{m_3}+\mathbf{m_5}+\mathbf{m_7} \geq 0\ (7) & \mathbf{m_3}+\mathbf{m_5} \geq 0\ (14) \\ & \mathbf{m_3}+\mathbf{m_5}+\mathbf{m_7} = 0\ (15) \end{cases}$$

where (1)–(7) express bracketing, (8)–(12) scope for λy and (13)–(15) scope for λz.

Now let us examine the typing constraints. We consider the principal typing assignment: $\Theta(y) = \alpha \to \alpha$, $\Theta(z) = \alpha$, which yields $\Theta(M) = (\alpha \to \alpha) \to (\alpha \to \alpha)$. Thus we have: $\overline{\Theta}(y) = !^{\mathbf{p_1}}(!^{\mathbf{p_2}}\alpha \multimap !^{\mathbf{p_3}}\alpha)$, $\overline{\Theta}(z) = !^{\mathbf{p_4}}\alpha$. We get for instance:

$$
\begin{array}{ll}
\overline{\Theta}(!^{\mathbf{m_7}}z) & = !^{\mathbf{m_7}+\mathbf{p_4}}\alpha \\
\overline{\Theta}(!^{\mathbf{m_6}}y_2) & = !^{\mathbf{m_6}+\mathbf{p_1}}(!^{\mathbf{p_2}}\alpha \multimap !^{\mathbf{p_3}}\alpha) \\
\overline{\Theta}((!^{\mathbf{m_6}}y_2)!^{\mathbf{m_7}}z) & = !^{\mathbf{p_3}}\alpha \\
\overline{\Theta}(!^{\mathbf{m_5}}((!^{\mathbf{m_6}}y_2)!^{\mathbf{m_7}}z)) & = !^{\mathbf{m_5}+\mathbf{p_3}}\alpha \\
\overline{\Theta}(!^{\mathbf{m_4}}y_1) & = !^{\mathbf{m_4}+\mathbf{p_1}}(!^{\mathbf{p_2}}\alpha \multimap !^{\mathbf{p_3}}\alpha) \\
\overline{\Theta}((!^{\mathbf{m_4}}y_1)\ !^{\mathbf{m_5}}[\ (!^{\mathbf{m_6}}y_2)!^{\mathbf{m_7}}z\ ]) & = !^{\mathbf{p_3}}\alpha \\
\overline{\Theta}(\overline{M}) & = !^{\mathbf{m_1}}(!^{\mathbf{p_1}}(!^{\mathbf{p_2}}\alpha \multimap !^{\mathbf{p_3}}\alpha) \multimap !^{\mathbf{m_2}}(!^{\mathbf{p_4}}\alpha \multimap !^{\mathbf{m_3}+\mathbf{p_3}}\alpha))
\end{array}
$$

We obtain the following typing constraints (omitting some obvious constraints):

$$
\mathcal{C}^{typ}(\overline{M}) = \left\{
\begin{array}{llll llll}
\mathbf{m_7}+\mathbf{p_4} & \geq & 0 & (16) & \mathbf{m_4}+\mathbf{p_1} & \geq & 0 & (21) \\
\mathbf{m_6}+\mathbf{p_1} & \geq & 0 & (17) & \mathbf{m_4}+\mathbf{p_1} & = & 0 & (22) \\
\mathbf{m_6}+\mathbf{p_1} & = & 0 & (18) & \mathbf{p_2} & = & \mathbf{m_5}+\mathbf{p_3} & (23) \\
\mathbf{p_2} & = & \mathbf{m_7}+\mathbf{p_4} & (19) & \mathbf{p_1},\dots,\mathbf{p_4} & \geq & 0 & (24) \\
\mathbf{m_5}+\mathbf{p_3} & \geq & 0 & (20) & \mathbf{p_1} & \geq & 1 & (25)
\end{array}
\right.
$$

Putting $\mathcal{C}^b(\overline{M})$ and $\mathcal{C}^{typ}(\overline{M})$ together we get that $\mathcal{C}(\overline{M})$ is equivalent to:

$$
\begin{array}{l}
\{\mathbf{m_1},\mathbf{m_2},\mathbf{m_3} \geq 0;\ \mathbf{m_2}+\mathbf{m_3} = -\mathbf{m_4} = -\mathbf{m_6} = \mathbf{p_1} \geq 1; \\
\ \mathbf{m_5} = 0;\ \mathbf{m_3}+\mathbf{m_7} = 0;\ \mathbf{p_2} = \mathbf{p_3} \geq 0;\ \mathbf{p_4} = \mathbf{p_2}+\mathbf{m_3}\}.
\end{array}
$$

This finally gives the following (informally written) parameterized term and type with constraints, which describe all solutions to this decoration problem:

$$
\left\{
\begin{array}{l}
\overline{M} = !^{\mathbf{m_1}}\lambda y.!^{\mathbf{m_2}}\lambda z.!^{\mathbf{m_3}}[\ (\bar{!}^{\mathbf{m_2}+\mathbf{m_3}}y_1)\ [\ (\bar{!}^{\mathbf{m_2}+\mathbf{m_3}}y_2)\bar{!}^{\mathbf{m_3}}z\ ]\ ] \\
!^{\mathbf{m_1}}(!^{\mathbf{m_2}+\mathbf{m_3}}(!^{\mathbf{p_2}}\alpha \multimap !^{\mathbf{p_2}}\alpha) \multimap !^{\mathbf{m_2}}(!^{\mathbf{p_2}+\mathbf{m_3}}\alpha \multimap !^{\mathbf{p_2}+\mathbf{m_3}}\alpha)) \\
\qquad \text{constraints: } \{\mathbf{m_1},\mathbf{m_2},\mathbf{m_3},\mathbf{p_2} \geq 0, \mathbf{m_2}+\mathbf{m_3} \geq 1\}.
\end{array}
\right.
$$

Observe that this representation corresponds to several canonical forms (6 in this particular example) in the approach of Coppola and Ronchi della Rocca (see [CRdR03]).

5 Conclusion

We have given a new type inference algorithm for EAL* which is more efficient and we think simpler than the previous ones. It generates a set of constraints which consists of two parts: one which deals with placing suitable (potential) boxes and the other one with typing the boxed term obtained. We believe the first part is not specific to EAL* typing and could be used for typing with other Linear logic systems which require determining boxes; what would need to be adapted to each case is the second (typing) part. This was already stressed by Coppola and Martini for their EAL type inference procedure ([CM04]). In particular we plan to study in this way second-order EAL typing (assuming a system F type given) and DLAL typing ([BT04a]).

We have shown that the set of constraints needed in our algorithm is polynomial in the size of the term and its simple type assignment. Finally we have also shown that by using resolution of linear inequations over rationals our algorithm can be executed in polynomial time with respect to the size of the initial term and its principal simple type assignment.

References

[ACM00] A. Asperti, P. Coppola, and S. Martini. (Optimal) duplication is not elementary recursive. In *Proceedings of POPL'00*, pages 96–107, 2000.

[AR02] A. Asperti and L. Roversi. Intuitionistic light affine logic. *ACM Transactions on Computational Logic*, 3(1):1–39, 2002.

[Asp98] A. Asperti. Light affine logic. In *Proceedings of LICS'98*, pages 300–308, 1998.

[Bai02] P. Baillot. Checking polynomial time complexity with types. In *Proceedings of IFIP TCS'02*, pages 370–382, Montreal, 2002.

[Bai04] P. Baillot. Type inference for light affine logic via constraints on words. *Theoretical Computer Science*, 328(3):289–323, 2004.

[BT04a] P. Baillot and K. Terui. Light types for polynomial time computation in lambda-calculus. In *Proceedings of LICS'04*, pages 266–275, 2004. Long version available at http://arxiv.org/abs/cs.LO/0402059.

[BT04b] P. Baillot and K. Terui. A feasible algorithm for typing in elementary affine logic (long version). Technical Report cs.LO/0412028, arXiv, 2004. Available from http://arxiv.org/abs/cs.LO/0412028.

[Car04] D. de Carvalho. Intersection types for light affine lambda calculus. In *Proceedings of 3rd Workshop on Intersection Types and Related Systems (ITRS'04)*, 2004. To appear in ENTCS.

[CM01] P. Coppola and S. Martini. Typing lambda-terms in elementary logic with linear constraints. In *Proceedings of TLCA'01*, volume 2044 of *LNCS*, pages 76–90, 2001.

[CM04] P. Coppola and S. Martini. Optimizing optimal reduction. A type inference algorithm for elementary affine logic. *ACM Transactions on Computational Logic*, 2004. To appear.

[CRdR03] P. Coppola and S. Ronchi della Rocca. Principal typing in Elementary Affine Logic. In *Proceedings of TLCA'03*, volume 2701 of *LNCS*, pages 90–104, 2003.

[DJ03] V. Danos and J.-B. Joinet. Linear logic & elementary time. *Information and Computation*, 183(1):123–137, 2003.

[DJS95] V. Danos, J.-B. Joinet, and H. Schellinx. On the linear decoration of intuitionistic derivations. *Archive for Mathematical Logic*, 33(6):387–412, 1995.

[Gir87] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[Gir98] J.-Y. Girard. Light linear logic. *Information and Computation*, 143:175–204, 1998.

[GSS92] J.-Y. Girard, A. Scedrov, and P. Scott. Bounded linear logic: A modular approach to polynomial time computability. *Theoretical Computer Science*, 97:1–66, 1992.

[Laf04] Y. Lafont. Soft linear logic and polynomial time. *Theoretical Computer Science*, 318(1–2):163–180, 2004.

[Rov98] L. Roversi. A polymorphic language which is typable and poly-step. In *Proceedings of the Asian Computing Science Conference (ASIAN'98)*, volume 1538 of *LNCS*, pages 43–60, 1998.

[Ter01] K. Terui. Light affine lambda-calculus and polytime strong normalization. In *Proceedings of LICS'01*, pages 209–220, 2001. Full version to appear in *Archive for Mathematical Logic*.