Typing Lambda Terms in Elementary Logic with Linear Constraints

Paolo Coppola and Simone Martini

Dipartimento di Matematica e Informatica
Università di Udine
33100 Udine, Italy
{coppola,martini}@dimi.uniud.it

Abstract We present a type inference algorithm for λ-terms in Elementary Affine Logic using linear constraints. We prove that the algorithm is correct and complete.

Introduction

The optimal reduction of λ-terms ([9]; see [3] for a comprehensive account and references) is a graph-based technique for normalization in which a redex is never duplicated. To achieve this goal, the syntax tree of the term is transformed into a graph, with an explicit node (a *fan*) expressing the sharing of two common subterms (these subterms are always variables in the initial translation). Giving correct reduction rules for these graphs is a surprisingly difficult problem, first solved in [8,7]. One of the main issues is to decide how to reduce two meeting fans, for which a complex machinery and new nodes have to be added (the *oracle*). There is large class of (typed) terms, however, for which this decision is very simple, namely the terms typeable in Elementary Logic, both in the Linear [6] (ELL) and the Affine [1] (EAL) flavor. Indeed, any proof-net for ELL or EAL may be (optimally) reduced with a simple check for the matching of fans. This fact was first observed in [1] and then exploited in [2] to obtain a certain complexity result on optimal reduction, where (following [10]) we also showed that EAL-typed λ-terms are powerful enough to encode arbitrary computations of elementary bounded Turing machines. We did not know, however, of any systematic way to derive EAL-types for λ-terms, a crucial issue if we want to exploit in an optimal reducer the added benefits of this class of terms. This is what we present in this paper.

Main contribution of the paper is a type inference algorithm (Section 2 and Appendix), assigning EAL-types (formulas) to *type-free* λ-terms. A typing derivation of a λ-term M in EAL consists of a *skeleton* – given by the derivation of a type for M in the simple type discipline – together with a *box assignment*, essential because EAL allows contraction only on boxed terms. The algorithm tries to introduce all possible boxes by collecting integer linear constraints during the exploration of the syntax tree of M. At the end, the integer solutions (if any) to the constraints give specific box assignments (i.e., EAL-derivations) for

M (for other approaches to the boxing of intuitionistic derivations, see [5,13]). Correctness and completeness of the algorithm are proved with respect to a natural deduction system for EAL, introduced in Section 2.1 together with terms annotating the derivations. For such term calculus we prove the main standard properties, including subject reduction.

1 Elementary Affine Logic

Elementary Affine Logic [1] (Figure 1) is a system with unrestricted weakening, where contraction is allowed only for modal formulas. There is only one *exponential* rule for the modality ! (*of-course*, or *bang*), which is introduced at once on both sides of the turnstile. Cut-elimination may be proved for EAL in a standard way.

$$\frac{}{A \vdash_{\mathsf{EAL}} A}\ ax \qquad \frac{\Gamma \vdash_{\mathsf{EAL}} A \quad A, \Delta \vdash_{\mathsf{EAL}} B}{\Gamma, \Delta \vdash_{\mathsf{EAL}} B}\ cut$$

$$\frac{\Gamma \vdash_{\mathsf{EAL}} B}{\Gamma, A \vdash_{\mathsf{EAL}} B}\ weak \qquad \frac{\Gamma, !A, !A \vdash_{\mathsf{EAL}} B}{\Gamma, !A \vdash_{\mathsf{EAL}} B}\ contr$$

$$\frac{\Gamma, A \vdash_{\mathsf{EAL}} B}{\Gamma \vdash_{\mathsf{EAL}} A \multimap B}\ \multimap R \qquad \frac{\Gamma \vdash_{\mathsf{EAL}} A \quad B, \Delta \vdash_{\mathsf{EAL}} C}{\Gamma, A \multimap B, \Delta \vdash_{\mathsf{EAL}} C}\ \multimap L$$

$$\frac{A_1, \ldots, A_n \vdash_{\mathsf{EAL}} B}{!A_1, \ldots, !A_n \vdash_{\mathsf{EAL}} !B}\ !$$

Figure 1. (Implicational) Elementary Affine Logic

A simple inspection of the rules of EAL shows that any λ-term with an EAL type has also a simple type[1]. Indeed, the simple type (and the corresponding derivation) is obtained by forgetting the exponentials, which must be present in an EAL derivation because of contraction.

The idea underlying our type inference algorithm is simple:

1. finding all "maximal decorations";
2. solving sets of linear constraints.

We informally present the main point with an example on the term $two \equiv \lambda xy.(x(x\ y))$. One (sequent) simple type derivation for two is:

[1] However there are simply typed terms not typeable in EAL, see [2].

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{w{:}\alpha\vdash w{:}\alpha}\quad\dfrac{}{y{:}\alpha\vdash y{:}\alpha}}{x{:}\alpha\to\alpha,y{:}\alpha\vdash(x\ y){:}\alpha}\quad\dfrac{}{z{:}\alpha\vdash z{:}\alpha}}{x{:}\alpha\to\alpha,x{:}\alpha\to\alpha,y{:}\alpha\vdash(x(x\ y)){:}\alpha}}{x{:}\alpha\to\alpha,x{:}\alpha\to\alpha\vdash\lambda y.(x(x\ y)){:}\alpha\to\alpha}}{x{:}\alpha\to\alpha\vdash\lambda y.(x(x\ y)){:}\alpha\to\alpha}}{\vdash\lambda xy.(x(x\ y)){:}(\alpha\to\alpha)\to\alpha\to\alpha}$$

If we change every $\to$ in $\multimap$, the previous derivation can be viewed as the skeleton of an EAL derivation. To obtain a full EAL derivation (provided it exists), we need to decorate this skeleton with exponentials, and to check that the contraction is performed only on exponential formulas.

Let's produce first a *maximal decoration* of the skeleton, interleaving n ! introduction rules after each logical rule. For example

$$\dfrac{\dfrac{}{w{:}\alpha\vdash w{:}\alpha}\quad\dfrac{}{y{:}\alpha\vdash y{:}\alpha}}{x{:}\alpha\multimap\alpha,y{:}\alpha\vdash(x\ y){:}\alpha}$$

becomes

$$\dfrac{\dfrac{\dfrac{}{w{:}\alpha\vdash w{:}\alpha}}{!^{n_1}w{:}\alpha\vdash !^{n_1}w{:}\alpha}\;!^{n_1}\quad\dfrac{\dfrac{}{y{:}\alpha\vdash y{:}\alpha}}{!^{n_2}y{:}\alpha\vdash !^{n_2}y{:}\alpha}\;!^{n_2}}{x{:}!^{n_2}\alpha\multimap !^{n_1}\alpha,y{:}!^{n_2}\alpha\vdash(x\ y){:}!^{n_1}\alpha}$$

where n_1 and n_2 are fresh variables. We obtain in this way a meta-derivation representing all EAL derivations with $n_1, n_2 \in \mathbb{N}$.

Continuing to decorate the skeleton of *two* (i.e. to interleave n_i ! rules) we obtain

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{w{:}\alpha\vdash w{:}\alpha}}{w{:}!^{n_1}\alpha\vdash w{:}!^{n_1}\alpha}\;!^{n_1}\quad\dfrac{\dfrac{}{y{:}\alpha\vdash y{:}\alpha}}{y{:}!^{n_2}\alpha\vdash y{:}!^{n_2}\alpha}\;!^{n_2}}{x{:}!^{n_2}\alpha\multimap !^{n_1}\alpha,y{:}!^{n_2}\alpha\vdash(x\ y){:}!^{n_1}\alpha}}{x{:}!^{n_3}(!^{n_2}\alpha\multimap !^{n_1}\alpha),y{:}!^{n_2+n_3}\alpha\vdash(x\ y){:}!^{n_1+n_3}\alpha}\;!^{n_3}\quad\dfrac{\dfrac{}{z{:}\alpha\vdash z{:}\alpha}}{z{:}!^{n_4}\alpha\vdash z{:}!^{n_4}\alpha}\;!^{n_4}}{x{:}!^{n_1+n_3}\alpha\multimap !^{n_4}\alpha,x{:}!^{n_3}(!^{n_2}\alpha\multimap !^{n_1}\alpha),y{:}!^{n_2+n_3}\alpha\vdash(x(x\ y)){:}!^{n_4}\alpha}}{x{:}!^{n_5}(!^{n_1+n_3}\alpha\multimap !^{n_4}\alpha),x{:}!^{n_3+n_5}(!^{n_2}\alpha\multimap !^{n_1}\alpha),y{:}!^{n_2+n_3+n_5}\alpha\vdash(x(x\ y)){:}!^{n_4+n_5}\alpha}\;!^{n_5}}{x{:}!^{n_5}(!^{n_1+n_3}\alpha\multimap !^{n_4}\alpha),x{:}!^{n_3+n_5}(!^{n_2}\alpha\multimap !^{n_1}\alpha)\vdash\lambda y.(x(x\ y)){:}!^{n_2+n_3+n_5}\alpha\multimap !^{n_4+n_5}\alpha}}{x{:}!^{n_5+n_6}(!^{n_1+n_3}\alpha\multimap !^{n_4}\alpha),x{:}!^{n_3+n_5+n_6}(!^{n_2}\alpha\multimap !^{n_1}\alpha)\vdash\lambda y.(x(x\ y)){:}!^{n_6}(!^{n_2+n_3+n_5}\alpha\multimap !^{n_4+n_5}\alpha)}\;!^{n_6}}{x{:}!^{n_5+n_6}(!^{n_1+n_3}\alpha\multimap !^{n_4}\alpha)\vdash\lambda y.(x(x\ y)){:}!^{n_6}(!^{n_2+n_3+n_5}\alpha\multimap !^{n_4+n_5}\alpha)}$$

The last rule – contraction – is correct in EAL iff the types of x are unifiable and banged. In other words iff the following constraints are satisfied:

$$n_1,n_2,n_3,n_4,n_5,n_6\in\mathbb{N}\quad\wedge\quad n_5=n_3+n_5\quad\wedge\quad n_1+n_3=n_2\quad\wedge\quad n_4=n_1\quad\wedge\quad n_5+n_6\geq 1.$$

The second, third and fourth of these constraints come from unification; the last one from the fact that contraction is allowed only on exponential formulas. These constraints are equivalent to

$$n_1,n_5,n_6\in\mathbb{N}\quad\wedge\quad n_3=0\quad\wedge\quad n_1=n_2=n_4\quad\wedge\quad n_5+n_6\geq 1.$$

Since clearly these constraints admit solutions, we conclude the decoration procedure obtaining

$$\dfrac{\dfrac{\vdots}{x{:}!^{n_5+n_6}(!^{n_1}\alpha \multimap !^{n_1}\alpha) \vdash \lambda y.(x(x\ y)){:}!^{n_6}(!^{n_1+n_5}\alpha \multimap !^{n_1+n_5}\alpha)}}{\vdash \lambda xy.(x(x\ y)){:}!^{n_5+n_6}(!^{n_1}\alpha \multimap !^{n_1}\alpha) \multimap !^{n_6}(!^{n_1+n_5}\alpha \multimap !^{n_1+n_5}\alpha)}$$

Thus *two* has EAL types $!^{n_5+n_6}(!^{n_1}\alpha \multimap !^{n_1}\alpha) \multimap !^{n_6}(!^{n_1+n_5}\alpha \multimap !^{n_1+n_5}\alpha)$, for any n_1, n_5, n_6 solutions of

$$n_1, n_5, n_6 \in \mathbb{N} \quad \wedge \quad n_5+n_6 \geq 1.$$

We may display the full derivation in a more manageable way, representing the skeleton with the syntax tree of the lambda term with edges labelled with types and adding boxes representing the ! introduction rules, as in Figure 2.

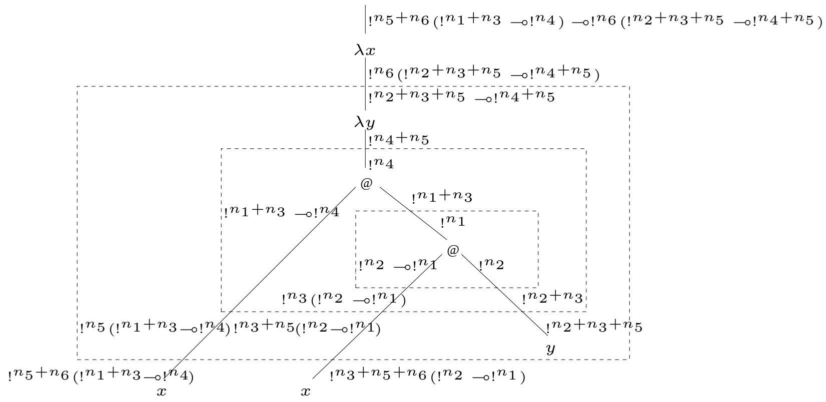


Figure 2. Meta EAL type derivation of *two*.

Finally notice that at the beginning of this section, we started with "*one (sequent) derivation*" for *two* (there are other derivations, building in a different way the application $x(xy)$)). If that derivation had produced an unsolvable set of constraint, the procedure should restart with another derivation. To avoid this problem, our search for maximal decorations (i.e., the collection of constraints) is not performed on sequent derivations, but on the syntax tree of the term. However, the fact that multiple derivations for a term and principal type scheme are possible, will surface again. It may happen that a solution to a set of constraints corresponds to more than one derivation (a *superposition of derivations*), with non compatible box-assignments. In this case, Lemma 6 ensures that compatible box assignments may be found.

2 Type Inference

A class of types for an EAL-typeable term can be seen as a decoration of a simple type with a suitable number of boxes. The main contribution of the paper is an algorithm collecting integer constraints whose solutions corresponds to proper box assignments.

Definition 1. *A* general EAL-type Θ *is generated from the following grammar:*

$$\Theta ::= o|\Theta \multimap \Theta|!^{n_1+\cdots+n_k}\Theta,$$

where $n_1, \ldots, n_k$ are variables ranging on integers $\mathbb{Z}$.

Definition 2 (Type Synthesis Algorithm). *Given a simply typeable lambda term and its principal type scheme $M : \sigma$, the* type synthesis algorithm $\mathscr{S}(M : \sigma)$ *returns a triple $\langle \Theta, B, A \rangle$, where Θ is a general EAL-type, B is a base (i.e. a multi-set of pairs variable, general EAL-type) and A is a set of linear constraints.*

The algorithm $\mathscr{S}(M : \sigma)$ is defined in the Appendix. One of the crucial issues is the localization of the points where derivations may differ for the presence or absence of boxes around some subterms. This is the role of *critical points*, managed by the boxing procedure, $\mathscr{B}$ (see A.5).

Proposition 1 (Termination). *Let M be a simply typed term and let σ be its most general type. $\mathscr{S}(M : \sigma)$ always terminates with a triple $\langle \Theta, B, A \rangle$.*

The algorithm is exponential in the size of the λ-term, because to investigate all possible derivations we need to (try to) box all possible combinations of critical points (see the clauses for the product union, $\Cup$, in A.6).

Correctness and completeness of $\mathscr{S}$ are much simpler if, instead of EAL, we formulate proofs and results with reference to an equivalent natural deduction formulation.

2.1 NEAL

The natural deduction calculus (NEAL) for EAL in given in Figure 3, after [4,1,12].

Lemma 1 (Weakening). *If $\Gamma \vdash_{\mathsf{NEAL}} A$ then $B, \Gamma \vdash_{\mathsf{NEAL}} A$.*

To annotate NEAL derivations, we use terms generated by the following grammar (*elementary terms*):

$$M ::= x \mid \lambda x.M \mid (M\ M) \mid !\,(M)\,[M/x, \ldots, M/x] \mid \|M\|^M_{x,x}$$

Observe that in $!\,(M)\,[M/x, \ldots, M/x]$, the $[M/x]$ is a kind of explicit substitution. To define ordinary substitution, define first the set of free variables of a term M, $\mathtt{FV}(M)$, inductively as follows:

- $\mathtt{FV}(x) = \{x\}$
- $\mathtt{FV}(\lambda x.M) = \mathtt{FV}(M) \smallsetminus \{x\}$
- $\mathtt{FV}(M_1\ M_2) = \mathtt{FV}(M_1) \cup \mathtt{FV}(M_2)$
- $\mathtt{FV}(!\,(M)\,[M_1/x_1, \ldots, M_n/x_n]) = \bigcup_{i=1}^n \mathtt{FV}(M_i)$
- $\mathtt{FV}(\|M\|^N_{x_1,x_2}) = (\mathtt{FV}(M) \smallsetminus \{x_1, x_2\}) \cup \mathtt{FV}(N)$

$$\frac{}{\Gamma, A \vdash_{\mathsf{NEAL}} A}\ ax \qquad \frac{\Gamma \vdash_{\mathsf{NEAL}} !A \quad \Delta, !A, !A \vdash_{\mathsf{NEAL}} B}{\Gamma, \Delta \vdash_{\mathsf{NEAL}} B}\ contr$$

$$\frac{\Gamma, A \vdash_{\mathsf{NEAL}} B}{\Gamma \vdash_{\mathsf{NEAL}} A \multimap B}\ (\multimap I) \quad \frac{\Gamma \vdash_{\mathsf{NEAL}} A \multimap B \quad \Delta \vdash_{\mathsf{NEAL}} A}{\Gamma, \Delta \vdash_{\mathsf{NEAL}} B}\ (\multimap E)$$

$$\frac{\Delta_1 \vdash_{\mathsf{NEAL}} !A_1 \cdots \Delta_n \vdash_{\mathsf{NEAL}} !A_n \quad A_1, \ldots, A_n \vdash_{\mathsf{NEAL}} B}{\Gamma, \Delta_1, \ldots, \Delta_n \vdash_{\mathsf{NEAL}} !B}\ !$$

Figure 3. Natural Elementary Affine Logic in sequent style notation

Ordinary substitution $N\{M/x\}$ of a term M for the free occurrences of x in N, is defined in the obvious way. The (pedantic) exponential cases are as follows:

1. $!\,(N)\,[P_1/x_1, \ldots, P_n/x_n]\,\{M/x\} =$
   $!\,(N\{y_1/x_1\} \cdots \{y_n/x_n\}\{M/x\})\,[P_1\{M/x\}/y_1, \ldots, P_n\{M/x\}/y_n]$
   if $x \notin \{x_1, \ldots, x_n\}$, where $y_1, \ldots, y_n$ are all fresh variables;
2. $!\,(N)\,[P_1/x_1, \ldots, P_n/x_n]\,\{M/x\} = !\,(N)\,[P_1\{M/x\}/x_1, \ldots, P_n\{M/x\}/x_n]$
   if $\exists i$ s.t. $x_i = x$;
3. $\|N\|^{P}_{y,z}\,\{M/x\} = \|N\{y'/y\}\{z'/z\}\{M/x\}\|^{P\{M/x\}}_{y',z'}$ if $x \notin \{y, z\}$, where y', z' are fresh variables;
4. $\|N\|^{P}_{y,z}\,\{M/x\} = \|N\|^{P\{M/x\}}_{y,z}$ if $x \in \{y, z\}$.

Elementary terms may be mapped to λ-terms, by forgetting the exponential structure:

- $x^* = x$
- $(\lambda x.M)^* = \lambda x.M^*$
- $(M_1\ M_2)^* = (M_1^*\ M_2^*)$
- $(!\,(M)\,[M_1/x_1, \ldots, M_n/x_n])^* = M^*\{M_1^*/x_1, \ldots, M_n^*/x_n\}$
- $(\|M\|^{N}_{x_1,x_2})^* = M^*\{N^*/x_1, N^*/x_2\}$

Definition 3 (Legal elementary terms). *The elementary terms are* legal *under the following conditions:*

1. *x is legal;*
2. *$\lambda x.M$ is legal iff M is legal;*
3. *$(M_1\ M_2)$ is legal iff M_1 and M_2 are both legal and $\mathtt{FV}(M_1) \cap \mathtt{FV}(M_2) = \emptyset$;*
4. *$!\,(M)\,[M_1/x_1, \ldots, M_n/x_n]$ is legal iff M and M_i are legal for any $i \quad 1 \leq i \leq n$ and $\mathtt{FV}(M) = \{x_1, \ldots, x_n\}$ and $(i \neq j \Rightarrow \mathtt{FV}(M_i) \cap \mathtt{FV}(M_j) = \emptyset)$;*
5. *$\|M\|^{N}_{x,y}$ is legal iff M and N are both legal and $\mathtt{FV}(M) \cap \mathtt{FV}(N) = \emptyset$.*

Proposition 2. *If M is a legal term, then every free variable $x \in \mathtt{FV}(M)$ is linear in M.*

Note 1. From now on we will consider only legal terms.

Notation. *Let* $\Gamma = \{x_1 : A_1, \ldots, x_n : A_n\}$ *be a basis.* $dom(\Gamma) = \{x_1, \ldots, x_n\}$*;* $\Gamma(x_i) = A_i$*;* $\Gamma \restriction V = \{x : A | x \in V \wedge A = \Gamma(x)\}$.

Legal terms are the ones induced by the Curry-Howard isomorphism applied to NEAL-derivations (see [11,12] for different approaches to Curry-Howard isomorphism for Linear and Light Linear Logic). The term assignment system is shown in Figure 4, where all bases in the premises of the contraction, $\multimap$ elimination and !-rule, have domains with empty intersection.

$$\frac{}{\Gamma, x : A \vdash x : A}\ ax \qquad \frac{\Gamma \vdash M :!A \quad \Delta, x :!A, y :!A \vdash N : B}{\Gamma, \Delta \vdash \|N\|^{M}_{x,y} : B}\ contr$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \multimap B}\ (\multimap I) \qquad \frac{\Gamma \vdash M : A \multimap B \quad \Delta \vdash N : A}{\Gamma, \Delta \vdash (M\ N) : B}\ (\multimap E)$$

$$\frac{\Delta_1 \vdash M_1 :!A_1 \cdots \Delta_n \vdash M_n :!A_n \quad x_1 : A_1, \ldots, x_n : A_n \vdash N : B}{\Gamma, \Delta_1, \ldots, \Delta_n \vdash !\,(N)\,[M_1/x_1, \ldots, M_n/x_n] :!B}\ !$$

Figure 4. Term Assignment System for Natural Elementary Affine Logic

Lemma 2. *1. If* $\Gamma \vdash_{\mathsf{NEAL}} M : A$ *then* $\mathtt{FV}(M) \subseteq dom(\Gamma)$*;*
2. if $\Gamma \vdash_{\mathsf{NEAL}} M : A$ *then* $\Gamma \restriction \mathtt{FV}(M) \vdash_{\mathsf{NEAL}} M : A$.

Lemma 3 (Substitution). *If* $\Gamma, x : A \vdash_{\mathsf{NEAL}} M : B$ *and* $\Delta \vdash_{\mathsf{NEAL}} N : A$ *and* $dom(\Gamma) \cap dom(\Delta) = \emptyset$ *then* $\Gamma, \Delta \vdash_{\mathsf{NEAL}} M\{N/x\} : B$.

Theorem 1 (Equivalence). $\Gamma \vdash_{EAL} A$ *if and only if* $\Gamma \vdash_{NEAL} A$.

Lemma 4 (Unique Derivation). *For any legal term* M *and formula* A*, if there is a valid derivation of the form* $\Gamma \vdash_{\mathsf{NEAL}} M : A$*, then such derivation is unique (up to weakening).*

Although we are not interested in this paper in the dynamics (i.e., normalization) of NEAL, a notion of reduction is needed to state and obtain our main result. We have first two *logical* reductions ($\rightarrow_\beta$ and $\rightarrow_{\mathsf{dup}}$) corresponding to the elimination of principal cuts in EAL. The other five reductions are permutation rules, allowing contraction to be moved out of a term.

$$(\lambda x.M\ N) \quad \to_\beta \quad M\{N/x\}$$

$$\|N\|_{x,y}^{!(M)[M_1/x_1,\dots,M_n/x_n]} \quad \to_{\mathsf{dup}}$$
$$\left\|\cdots\left\|N\{{}^{!(M)\left[x'_1/x_1,\dots,x'_n/x_n\right]}/x\}\{{}^{!\left(M'\right)\left[y'_1/y_1,\dots,y'_n/y_n\right]}/y\}\right\|_{x'_1,y'_1}^{M_1}\cdots\right\|_{x'_n,y'_n}^{M_n}$$

$$!(M)[{}^{M_1}/x_1,\cdots,{}^{!(N)[P_1/y_1,\dots,P_m/y_m]}/x_i,\cdots,{}^{M_n}/x_n] \quad \to_{!-!}$$
$$!(M\{N/x_i\})[{}^{M_1}/x_1,\cdots,{}^{P_1}/y_1,\cdots,{}^{P_m}/y_m,\cdots{}^{M_n}/x_n]$$

$$(\|M\|_{x_1,x_2}^{M_1}\ N) \quad \to_{@-\mathsf{c}} \quad \|(M\{x'_1/x_1,x'_2/x_2\}\ N)\|_{x'_1,x'_2}^{M_1}$$

$$!(M)[{}^{M_1}/x_1,\cdots,{}^{\|M_i\|_{y,z}^N}/x_i,\cdots,{}^{M_n}/x_n] \quad \to_{!-c}$$
$$\left\|!(M)[{}^{M_1}/x_1,\cdots,{}^{M_i\{y'/y,z'/z\}}/x_i,\cdots,{}^{M_n}/x_n]\right\|_{y',z'}^N$$

$$\|M\|_{x_1,x_2}^{\|N\|_{y_1,y_2}^P} \quad \to_{\mathsf{c}-\mathsf{c}} \quad \left\|\|M\|_{x_1,x_2}^{N\{y'_1/y_1,y'_2/y_2\}}\right\|_{y'_1,y'_2}^P$$

$$\lambda x.\,\|M\|_{y,z}^N \quad \to_{\lambda-\mathsf{c}} \quad \|\lambda x.M\|_{y,z}^N \ \text{ where } x \notin \mathtt{FV}(N)$$

where M' in the $\to_{\mathsf{dup}}$-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 $\to_{@-\mathsf{c}}$-rule, y' and z' in the $\to_{!-c}$-rule and y'_1, y'_2 in the $\to_{\mathsf{c}-\mathsf{c}}$-rule are fresh variables.

Definition 4. *The reduction relation on legal terms $\rightsquigarrow$ is defined as the reflexive and transitive closure of the union of $\to_\beta, \to_{\mathsf{dup}}, \to_{!-!}, \to_{@-\mathsf{c}}, \to_{!-\mathsf{c}}, \to_{\mathsf{c}-\mathsf{c}}, \to_{\lambda-\mathsf{c}}$.*

Proposition 3. *Let $M \rightsquigarrow N$ and M be a legal term, then N is a legal term.*

Proposition 4. *Let $M \to_r N$ where r is not $\to_\beta$, then $M^* = N^*$.*

Lemma 5. *Let M be a well typed term in $\{\mathsf{dup}, !-!, @-\mathsf{c}, !-\mathsf{c}, \mathsf{c}-\mathsf{c}, \lambda-\mathsf{c}\}$-normal form, then*

1. *if $R = \|N\|_{x,y}^P$ is a subterm of M, then either $P = (P_1\ P_2)$ or P is a variable;*
2. *if $R =!\,(N)\,[P_1/x_1,\dots,P_k/x_k]$ is a subterm of M, then for any $i \in \{1,\dots,k\}$ either $P_i = (Q_i\ S_i)$ or P_i is a variable.*

Theorem 2 (Subject Reduction). *Let $\Gamma \vdash_{\mathsf{NEAL}} M : A$ and $M \rightsquigarrow N$, then $\Gamma \vdash_{\mathsf{NEAL}} N : A$.*

2.2 Properties of the Type Syntesis Algorithm

Lemma 6 (Superimposing of derivations). *Let $\mathscr{S}(M : \sigma) = \langle \Theta, B, A\rangle$ and let A be solvable. If there is a solution X_1 of A that instantiates two boxes* *belonging to two superimposed derivations that are not compatible, then there exists another solution X_2 where all the instantiated boxes belong to the same derivation.*

Moreover the instantiations Θ', B' of Θ, B using X_1 and the instantiations Θ'', B'' of Θ, B using X_2 are identical.

Proof. (sketch) We may think of boxes as levels; boxing a subterm can then be seen as raising that subterm, as in Figure 5, where also some types label the edges of the syntax tree of a simple term. In particular, the edge starting from the @-node and ending in x_0 has label $!^{n_2}(\alpha \multimap !^{n_1}(\beta \multimap \gamma))$ at level 0 (nearest to x_0) and has label $(\alpha \multimap !^{n_1}(\beta \multimap \gamma))$ at level n_2. This is the graphical counterpart of the !-rule

$$\frac{\ldots, x_0 : T, \ldots \vdash \ldots}{\ldots, x_0 : !^{n_2}T, \ldots \vdash \ldots}\ !^{n_2}$$

The complete decoration of Figure 5 can be produced in NEAL in two ways: by

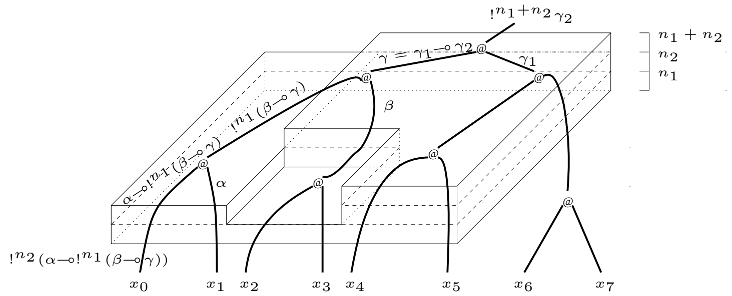


Figure 5. Boxes can be viewed as levels.

the instantiation of

$$!^{n_2}\left((((x_0\ x_1)y)((x_4\ x_5)w))\right)[(x_2\ x_3)/y, (x_6\ x_7)/w]$$

and[2]

$$!^{n_1}\left(((z(x_2\ x_3))((x_4\ x_5)w))\right)[(x_0\ x_1)/z, (x_6\ x_7)/w],$$

which are boxes belonging to two different derivations. Graphically such an instantiation can be represented as in the first row of Figure 6, where incompatibility is evident by the fact that the boxes are not well stacked, in particular the rectangular one covers a hole. To have a correct EAL-derivation it is necessary to find the equivalent, well stacked configuration (that corresponds to the subsequent application of boxes from the topmost to the bottommost).

[2] The correct legal terms should have all free variable inside the square brackets. We omit to write variables when they are just renamed, for readability reasons (compare the first elementary term above with the correct one $!^{n_2}\left((((x_0\ x_1)y)((x_4\ x_5)w))\right)[x_0'/x_0, x_1'/x_1, (x_2\ x_3)/y, x_4'/x_4, x_5'/x_5, (x_6\ x_7)/w]$).

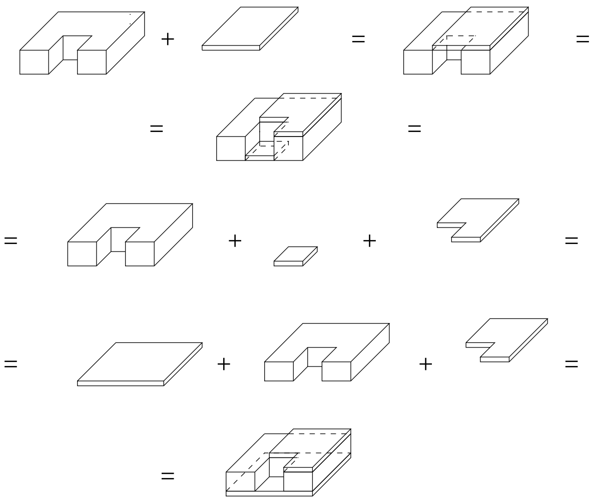


Figure 6. Equivalences of boxes.

The procedure by which we find the well stacked box configuration is visualized in Figure 6. The reader may imagine the boxes subject to gravity (the passage from the first to the second row of Figure 6) and able to fuse each other when they are at the same level (the little square in the third row fuse with the solid at its left in the passage from the third to the fourth row).

The "gravity operator" corresponds to finding the minimal common subterm of all the superimposed derivations and it is useful for finding the correct order of application of the ! rule. The "fusion operator" corresponds to the elimination of a cut between two exponential formulas. Moreover, the final configuration of Figure 6 corresponds to a particular solution of the set of constraints produced by the type synthesis algorithm, that instantiates the following boxes:

$$!^{n_1}\left(!^{n_2-n_1}\left(!^{n_1}\left(((z\ w)((x_4\ x_5)t))\right)[(x_0\ x_1)/z]\right)[(x_2\ x_3)/w]\right)(x_6\ x_7)/t]$$

Finally, notice that during the procedure all types labelling the boundary edges of the lambda-term never changes, i.e. the instantiations of the term type (the label of the topmost edge) and the base types (the labels of the edges at the bottom) remain unchanged.

Theorem 3 (Soundness). *Let $\mathscr{S}(M : \sigma) = \langle \Theta, B, A\rangle$. For every X integer solution of A, and B', Θ' instantiations of B and Θ using X, there exists P elementary term such that $P^* = M$ and $B' \vdash P : \Theta'$ is derivable in NEAL.*

Proof. By induction on the structure of M, using the superimposing lemma.

Theorem 4 (Completeness). *Let $\Gamma \vdash_{\mathsf{NEAL}} P : \Psi$ and let P be in $\{!-!, @-\mathsf{c}, !-\mathsf{c}, \mathsf{c}-\mathsf{c}, \lambda-\mathsf{c}, \mathsf{dup}\}$-normal form with contraction only on variables ($\|R\|^{Q}_{x,y}$*

is a subterm of P only if Q is a variable). Let $\mathscr{S}(P^* : \overline{\Psi}) = \langle \Theta, B, A\rangle$, where $\overline{\Psi}$ is the erasure of Ψ, i.e. the simple type obtained from Ψ erasing all ! and converting $\multimap$ in $\rightarrow$, then there exist X integer solution of A such that the instantiation B' of B using X is a subset of Γ and Ψ is the instantiation of Θ using X and $B' \vdash_{\mathsf{NEAL}} P : \Psi$.

The request on the $\{!-!, @-\mathsf{c}, !-\mathsf{c}, \mathsf{c}-\mathsf{c}, \lambda-\mathsf{c}, \mathsf{dup}\}$-normal form is not a loss of generality, for the subject reduction lemma and Proposition 4. By Lemma 5, the only restriction is the exclusion of elementary terms with subterms of the form $\|R\|_{x,y}^{(Q_1\ Q_2)}$. In a sense, these terms "contract too much". Indeed, it could be the case that a term P is elementary thanks to the sharing of a β-redex (inside $(Q_1\ Q_2)$). However, the corresponding λ-term P^*, cannot share any redex – there is no sufficient syntax for this in the λ-calculus – hence P^* could be not elementary. As we discussed in the Introduction, our aim is to identify λ-terms that are reducible using optimal reduction without the *oracle* needed for the correct matching of fans. The NEAL terms excluded in the completeness theorem corresponds to EAL proof-nets which are not (the initial encoding of) λ-terms, since they contract an application.

Conclusions

We presented a complete algorithm to derive EAL-types for λ-terms. One of our main goals is the characterization of those terms that can be optimally reduced without the oracle, for which EAL-typeability is a sufficient condition. One should not see (N)EAL as a programming language; instead, it is a kind of intermediate language: if a λ-term is typeable in EAL, then we can compile it in a special manner with excellent performances during reduction, otherwise we compile it in the usual way, using the oracle. To get a more powerful (and, to a certain extent, flexible) language, a major development of this work would be the extension of our algorithm to second order EAL.

The same technique of this paper may be applied to Multiplicative Exponential Linear Logic. However, to treat dereliction, the number of constraints grows in an exponential way.

We believe techniques similar to those we used in this paper may be applied to type-inference for Light Linear (or Affine) Logic (LLL), a system characterizing polytime. A type-inference for LLL would be a uniform proof-technique to prove polynomiality of certain algorithms.

A puzzling open problem is whether there exist terms yielding constraints with only non integer solutions. Of course they have to be non EAL-typeable terms, in view of our completeness theorem. Our estensive experiments never produced such a scenario, yet we could not prove that the constraints have always integral solutions. Would there be any logical meaning for a term with a non integral number of boxes?

Acknowledgments Harry Mairson provided useful criticism and comments on the form and the substance of the paper.

References

1. Andrea Asperti. Light affine logic. In *Proc. of the 13-th Annual IEEE Symposium on Logic in Computer Science (LICS '98)*, pages 300–308, Indianapolis, U.S.A., 1998.
2. Andrea Asperti, Paolo Coppola, and Simone Martini. (Optimal) duplication is not elementary recursive. In *ACM POPL'00*, pages 96–107, Boston, Massachusetts, January 19–21, 2000.
3. Andrea Asperti and Stefano Guerrini. *The Optimal Implementation of Functional Programming Languages*, volume 45 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1998.
4. Nick Benton, Gavin Bierman, Valeria de Paiva, and Martin Hyland. A term calculus for intuitionistic linear logic. *TLCA'93*, volume 664 of *Lecture Notes in Computer Science*, pages 75–90, March 1993.
5. Vincent Danos, Jean-Baptiste Joinet and Harold Schellinx. On the linear decoration of intuitionistic derivations. *Archive for Mathematical Logic*, 33:387–412, 1995.
6. Jean-Yves Girard. Light linear logic. *Information and Computation*, 204:143–175, 1998.
7. Vinod K. Kathail. *Optimal Interpreters for Lambda-calculus Based Functional Programming Languages*. PhD thesis, MIT, May 1990.
8. John Lamping. An Algorithm for Optimal Lambda Calculus Reduction. In *ACM POPL '90*, pages 16–30, New York, NY, USA, 1990.
9. Jean-Jacques Lévy. Optimal reductions in the lambda-calculus. In Jonathan P. Seldin and J. Roger Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 159–191. Academic Press, London, 1980.
10. Harry G. Mairson. A simple proof of a theorem of Statman. *Theoretical Computer Science*, 103(2):387–394, September 1992.
11. Alberto Pravato and Luca Roversi. $\Lambda_!$ considered both as a paradigmatic language and a meta-language. In *Fifth Italian Conference on Theoretical Computer Science*, Salerno (Italy), 1995.
12. Luca Roversi. A Polymorphic Language which Is Typable and Poly-step. In *Proc. of the Asian Computing Science Conference (ASIAN'98)*, volume 1538 of *Lecture Notes in Computer Science*, pages 43 – 60, Manila (The Philippines), 1998.
13. Harold Schellinx. *The Noble Art of Linear Decorating*. PhD thesis, Institute for Logic, Language and Computation, University of Amsterdam, 1994.

A Appendix

In the following n, n_1, n_2 are always fresh variables, o is the base type. Moreover, we consider $!^{n_1}(!^{n_2}\Theta)$ syntactically equivalent to $!^{n_1+n_2}\Theta$.

A.1 Free variable occurrences: FVO

Definition 5. *The list of* free variable occurrences *of a lambda term M is defined in the obvious way:*

1. $\mathtt{FVO}(x) = [x]$;
2. $\mathtt{FVO}(\lambda x.M) = \mathtt{FVO}(M) - x$;
3. $\mathtt{FVO}((M_1\ M_2)) = \mathtt{FVO}(M_1) + \mathtt{FVO}(M_2)$ *(the concatenation of lists).*

A.2 Unification: $\mathscr{U}$

The *unification* of $h \geq 2$ general EAL-types produces a set of equations. Notice that in our algorithm, the erasure of general EAL-types in the unification clauses always gives structurally identical types. This is possible because we start our type synthesis procedure from a lambda term already simply typed.

$$\frac{}{\mathscr{U}\begin{pmatrix} !^{\sum n_{i_1}} o, \\ !^{\sum n_{i_2}} o, \\ \vdots \\ !^{\sum n_{i_h}} o \end{pmatrix} = \begin{array}{c} \sum n_{i_1} - \sum n_{i_2} = 0 \\ \sum n_{i_2} - \sum n_{i_3} = 0 \\ \vdots \\ \sum n_{i_{h-1}} - \sum n_{i_h} = 0 \end{array}} \; \mathrm{u}o \tag{1}$$

$$\frac{\mathscr{U}(\Theta_{1_1}, \ldots, \Theta_{1_h}) = A_1 \quad \mathscr{U}(\Theta_{2_1}, \ldots, \Theta_{2_h}) = A_2}{\mathscr{U}\begin{pmatrix} !^{\sum n_{i_1}}(\Theta_{1_1} \multimap \Theta_{2_1}), \\ !^{\sum n_{i_2}}(\Theta_{1_2} \multimap \Theta_{2_2}), \\ \vdots \\ !^{\sum n_{i_h}}(\Theta_{1_h} \multimap \Theta_{2_h}) \end{pmatrix} = \begin{array}{c} \sum n_{i_1} - \sum n_{i_2} = 0 \\ \sum n_{i_2} - \sum n_{i_3} = 0 \\ \vdots \\ \sum n_{i_{h-1}} - \sum n_{i_h} = 0 \\ A_1 \\ A_2 \end{array}} \; \mathrm{u} \multimap \tag{2}$$

A.3 Contraction: $\mathcal{C}$

The contraction of k general EAL-types produces a set of equations.

The first rule of contraction is just a trick allowing a common treatment of two cases in the type synthesis algorithm. In particular it states that the contraction of a single general EAL-type produces no constraint.

$$\frac{}{\mathcal{C}(\Theta) = \emptyset} \; \mathrm{c}\emptyset \tag{3}$$

Two formulas may be contracted in EAL only if they are equal and both are banged (exponential). Hence the contraction rule produces the same set of equations of unification (A) plus the constraint imposing that the outer number of bangs must be at least one ($n_1 + \cdots + n_h \geq 1$).

$$\frac{\mathscr{U}(!^{n_1+\cdots+n_h}\Theta_1, \Theta_2, \ldots, \Theta_k) = A}{\mathcal{C}(!^{n_1+\cdots+n_h}\Theta_1, \ldots, \Theta_k) = \begin{array}{c} n_1 + \cdots + n_h \geq 1 \\ A \end{array}} \; \mathrm{c} \tag{4}$$

Where Θ_1 is either $\Gamma \multimap \Delta$ or o.

A.4 Type processing: $\mathscr{P}$

The type processing rule produces a pair $\langle \Theta, A \rangle$ where Θ is a general EAL-type and A is a set of constraints $n_i \geq 0$. This pair can be seen as the "most general EAL-type" from a simple type. The set of constraints resulting from this procedure states that every time $\mathscr{P}$ adds n bangs, n should be positive or zero.

Notice that a processed type is always banged, hence it is ready to be contracted.

$$\frac{}{\mathscr{P}(o) = \langle !^n o, n \geq 0 \rangle}\ \mathrm{p}o \tag{5}$$

$$\frac{\mathscr{P}(\sigma) = \langle \Theta, A_1 \rangle \quad \mathscr{P}(\tau) = \langle \Gamma, A_2 \rangle}{\mathscr{P}(\sigma \to \tau) = \left\langle !^n(\Theta \multimap \Gamma), \begin{array}{c} n \geq 0 \\ A_1 \\ A_2 \end{array} \right\rangle}\ \mathrm{p}\to \tag{6}$$

A.5 Boxing: $\mathscr{B}$

The boxing procedure superimposes all boxes due to the existence of critical points. Every time there are two possible EAL-derivations for the unique simple type, there is a critical point. For example during the type synthesis of

$$\frac{\vdots}{x : \alpha \to \alpha, x : \alpha \to \alpha, y : \alpha \vdash (x(x\ y)) : \alpha}$$

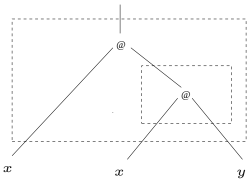
we need to try all possible decorations of

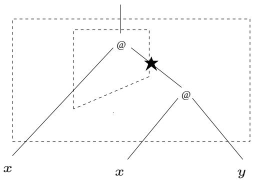
$$\frac{\dfrac{\vdots}{x : \alpha \to \alpha, y : \alpha \vdash (x\ y) : \alpha} \quad \dfrac{}{z : \alpha \vdash z : \alpha}}{x : \alpha \to \alpha, x : \alpha \to \alpha, y : \alpha \vdash (x(x\ y)) : \alpha}$$

but also of

$$\frac{\dfrac{}{y : \alpha \vdash y : \alpha} \quad \dfrac{\vdots}{z : \alpha, x : \alpha \to \alpha \vdash (xz) : \alpha}}{x : \alpha \to \alpha, x : \alpha \to \alpha, y : \alpha \vdash (x(x\ y)) : \alpha}$$

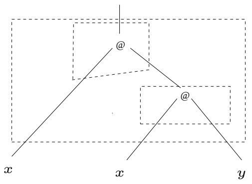
In our graphical notation, the two decorations appear as follows (the one corresponding to the first derivation is at the left; the star indicates the critical point):





When $\mathscr{B}$ is called during the synthesis of $(x(x\ y))$, the base B is something like $\{x :!^{n_1}\alpha \multimap !^{n_2}\alpha, x :!^{n_6}(!^{n_3}\alpha \multimap !^{n_4}\alpha), y :!^{n_5+n_6}\alpha\}$, where the first x is the leftmost in the figure, the type Γ of $(x(x\ y))$ is $!^{n_2}\alpha$, the set of critical points *cpts* is $\{(n_4+n_6-n_1 = 0, [x, y])\}$ and the set of constraints A is $\{n_4+n_6-n_1 = 0; n_5 - n_3 = 0\}$(see inference rules in A.7).

At that stage of the type synthesis procedure, the decoration corresponds to the first one (the two constraints are needed for the correct type matching between the two occurrences of x and the respective arguments). $\mathscr{B}$ superimposes the second derivation, adding n_7 boxes as in the second figure, obtaining the superimposed decoration



and modifying the base B in $\{x\ :!^{n_7}(!^{n_1}\alpha\ \multimap !^{n_2}\alpha), x\ :!^{n_6}(!^{n_3}\alpha\ \multimap !^{n_4}\alpha), y\ :\ !^{n_5+n_6}\alpha\}$ and the set of constraints A in $\{n_4 + n_6 - n_1 - n_7 = 0; n_5 - n_3 = 0\}$.

Definition 6. *A* slice *is a set of critical points, i.e. pairs (constraint, list of free variable occurrences) as in the following:*

$$sl = \left\{(A^{j_1}, [y_{1_1}, \ldots, y_{1_h}]), \ldots, (A^{j_k}, [y_{k_1}, \ldots, y_{k_h}])\right\}$$

A slice corresponds to a combination of critical points.

Notation.
- *$sl(x)$ means that x is an element of every list of variables in $sl(x)$.*
- *$x \in sl$ if and only if there exists one element of sl whose list of variables contains x.*
- *$A^j \in sl$ if and only if there exists one element of sl whose constraint is A^j.*
- *Being A^j the constraint $\pm n_{j_1} \pm \cdots \pm n_{j_k} = 0$, $A^j - n$ corresponds to the constraint $\pm n_{j_1} \pm \cdots \pm n_{j_k} - n = 0$.*

$\mathscr{B}$ without critical points produces no effect:

$$\frac{}{\mathscr{B}(B, \Gamma, \emptyset, A) = \langle B, \Gamma, A\rangle}\ \mathrm{b}\emptyset \tag{7}$$

For every slice sl, $\mathscr{B}$ adds n boxes around all the term but the set of subterms under the critical points of sl, hence modifies all types of variables in the base but not in sl. Moreover $\mathscr{B}$ modifies all constraints of sl, the final type Γ and then goes on with the other slices.

$$\frac{\begin{array}{c}\mathscr{B}\left(B_1, !^n\Gamma, cpts, \begin{array}{c}A_2\\ n \geq 0\end{array}\right) = \langle B, \Delta, A_1\rangle\\ B_1 = \left\{x_i : \left\{\begin{array}{ll}!^n\Theta_i & x_i \notin sl\\ \Theta_i & x_i \in sl\end{array}\right\}\right\}_i\\ A_2 = \left(\left\{\begin{array}{ll}A^j & A^j \notin sl\\ A^j - n & A^j \in sl\end{array}\right)\right)_j\end{array}}{\mathscr{B}(\{x_i : \Theta_i\}_i, \Gamma, \{sl\} \cup cpts, A) = \langle B, \Delta, A_1\rangle}\ \mathrm{b}sl \tag{8}$$

A.6 Product union: $\Cup$

Here resides the exponential complexity of the algorithm. In order to investigate all possible derivations we need to box all possible combinations of critical points.

$$\frac{}{\emptyset \Cup X = X}\ \emptyset\Cup \qquad \frac{}{X \Cup \emptyset = X}\ \Cup\emptyset \tag{9}$$

$$\frac{\{sl_{2_1}, \ldots, sl_{n_1}\} \Cup \{sl_{1_2}, \ldots, sl_{n_2}\} = X}{\begin{Bmatrix} sl_{1_1} \\ \vdots \\ sl_{n_1} \end{Bmatrix} \Cup \begin{Bmatrix} sl_{1_2} \\ \vdots \\ sl_{n_2} \end{Bmatrix} = \{sl_{1_1}, sl_{1_1} \cup sl_{1_2}, \ldots, sl_{1_1} \cup sl_{n_2}\} \cup X}\ \Cup \tag{10}$$

A.7 Type synthesis: $\mathcal{S}$

The type synthesis of $x : \sigma$ returns the processed type Θ, no critical points and constraints of the kind $n_j \geq 0$ for every n_j introduced during the processing of σ.

Notice that, for the property of $\mathscr{P}$, Θ is always of type $!^n\Theta'$.

$$\frac{\mathscr{P}(\sigma) = \langle \Theta, A \rangle}{\mathcal{S}(x : \sigma) = \langle \Theta, x : \Theta, A, \emptyset \rangle}\ \mathrm{s}x \tag{11}$$

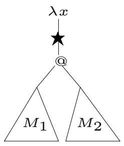
For the type synthesis of the abstraction, we need to examine four different cases according to whether the body of the abstraction is an application (if it is not we need to add a box all around the body, whilst in the other case the box is already added by the rule of the application) or the the abstracted variable appears in the body (if it does, we eventually need to contract all the different types of the variable, while if it does not, we need to process the type).

If the body of the abstraction is an application and the bound variable occurs at least one time in the body, the algorithm erases all the occurrences of the bound variable from the base, eventually contracts all types of x introducing A_2 new constraints (notice that by definition of $\mathcal{C}$, if $h = 1$ then A_2 is empty) and finally erases all slices that contain x (after the abstraction of x they do not represent any more possible derivations).

$$\frac{\mathcal{S}((M_1\ M_2) : \tau) = \left\langle \Gamma, B \uplus \begin{Bmatrix} x : \Theta_1 \\ \vdots \\ x : \Theta_h \end{Bmatrix}, A_1, cpts \cup \begin{Bmatrix} sl_1(x) \\ \vdots \\ sl_k(x) \end{Bmatrix} \right\rangle \qquad \mathcal{C}(\Theta_1, \ldots, \Theta_h) = A_2}{\mathcal{S}(\lambda x.(M_1\ M_2) : \sigma \to \tau) = \left\langle \Theta_1 \multimap \Gamma, B, \begin{matrix} A_1 \\ A_2 \end{matrix}, cpts \right\rangle}\ \mathrm{s}\lambda x@ \tag{12}$$

Where $h \geq 1$.

If the body is an application and the bound variable doesn't appear in it, we just need to process the type of x and add a critical point between the lambda and the application.



The added critical point represents derivations of the form:

$$\dfrac{\dfrac{\vdots}{\vdash M : \alpha} \qquad \dfrac{\vdots}{y : \beta \vdash \lambda x.y : \gamma \to \beta}}{z : \alpha \to \beta \vdash \lambda x.(z\ M) : \gamma \to \beta}$$

where the application is introduced after the abstraction of x.

$$\dfrac{\begin{array}{l} cpts = cpts_1 \cup \{(\sum n_i - n = 0, \mathtt{FVO}(M_1\ M_2))\} \\ \mathscr{P}(\sigma) = \langle \Theta, A_2 \rangle \\ \mathcal{S}((M_1\ M_2) : \tau) = \langle !^{\sum n_i} \Gamma, B, A_1, cpts_1 \rangle \end{array}}{\mathcal{S}(\lambda x.(M_1\ M_2) : \sigma \to \tau) = \left\langle \Theta \multimap !^n \Gamma, B, \begin{array}{c} A_1 \\ A_2 \\ n \geq 0 \\ \sum n_i - n = 0 \end{array}, cpts \right\rangle} \ \mathrm{s}\lambda@ \tag{13}$$

Where Γ is not exponential.

If the body of the abstraction is not an application and the bound variable occurs in the body, we need to add all the boxes for the critical points with $\mathscr{B}$, after then eventually contract all types of x and finally add n boxes all around the body of the term.

$$\dfrac{\begin{array}{l} \mathcal{C}(!^n \Theta_1, \ldots, !^n \Theta_h) = A_2 \\ \mathscr{B}(B_1, \Gamma_1, \{sl_i(x)\}_i \cup cpts, A) = \left\langle B \uplus \left\{ \begin{array}{c} x : \Theta_1 \\ \vdots \\ x : \Theta_h \end{array} \right\}, \Gamma, A_1 \right\rangle \\ \mathcal{S}(M : \tau) = \langle \Gamma_1, B_1, A, \{sl_i(x)\}_i \cup cpts \rangle \end{array}}{\mathcal{S}(\lambda x.M : \sigma \to \tau) = \left\langle \Theta_1 \multimap !^n \Gamma, !^n B, \begin{array}{c} A_1 \\ A_2 \\ n \geq 0 \end{array}, cpts \right\rangle} \ \mathrm{s}\lambda x \tag{14}$$

Where $h \geq 1$. In the case of $h = 1$, $A_2 = \emptyset$.

Finally, if the body of the abstraction is not an application and the bound variable does not occur in it, as in the previous case we need to box the body with $\mathscr{B}$ and finally with n boxes, but now there are no types to contract but a type (σ) to process. In this case, as there are no occurrences of x, the base and the set of critical points are left unchanged.

$$\dfrac{\begin{array}{l}\mathscr{P}(\sigma) = \langle \Theta, A_3 \rangle \\ \mathscr{B}(B_1, \Gamma_1, cpts, A_1) = \langle B, \Gamma, A_2 \rangle \\ \mathcal{S}(M : \tau) = \langle \Gamma_1, B_1, A_1, cpts \rangle\end{array}}{\mathcal{S}(\lambda x.M : \sigma \to \tau) = \left\langle \Theta \multimap !^n \Gamma, !^n B, \begin{array}{c} A_2 \\ A_3 \\ n \geq 0 \end{array}, cpts \right\rangle} \text{s}\lambda \tag{15}$$

For the type synthesis of the application we need to examine six different cases, according to whether the argument is an application itself (as in the rules of the abstraction, if it is not an application we need to call $\mathscr{B}$ and add n boxes all around of it and moreover we need to add a new critical point) and the functional part is a variable (in this case we need to process its type), or an application (and we need to add a critical point and impose the outermost number of bangs of the type to be equals to 0 otherwise we can't perform the application) or an abstraction.

If the functional part is an abstraction and the argument is not an application, we need to add boxes around the argument ($\mathscr{B}(B_2, \Theta_2, cpts_2, A_2)$ plus n_1), unify the types for a correct application ($\mathscr{U}(!^{n_1}\Theta_3, \Theta_1)$), and finally add boxes around the whole application. The final set of critical points is the product union $\Cup$ of the set of critical point of function and argument.

$$\dfrac{\begin{array}{l}\mathscr{B}\left(B_1 \uplus !^{n_1} B_3, \Gamma_1, cpts_1 \Cup cpts_2, \begin{array}{c} A_1 \\ A_3 \\ A_4 \\ n_1 \geq 0 \end{array} \right) = \langle B, \Gamma, A \rangle \\ \mathscr{U}(!^{n_1}\Theta_3, \Theta_1) = A_4 \\ \mathscr{B}(B_2, \Theta_2, cpts_2, A_2) = \langle B_3, \Theta_3, A_3 \rangle \\ \mathcal{S}(N : \sigma) = \langle \Theta_2, B_2, A_2, cpts_2 \rangle \\ \mathcal{S}(\lambda x.M : \sigma \to \tau) = \langle \Theta_1 \multimap \Gamma_1, B_1, A_1, cpts_1 \rangle\end{array}}{\mathcal{S}((\lambda x.M\ N) : \tau) = \left\langle !^n \Gamma, !^n B, \begin{array}{c} A \\ n \geq 0 \end{array}, cpts_1 \Cup cpts_2 \right\rangle} \text{s@}\lambda \tag{16}$$

If the argument is an application itself, we don't need to add boxes around it, instead we need to add a new critical point $((A_3^1, \texttt{FVO}((N_1\ N_2))))$.

$$\dfrac{\begin{array}{l}\mathscr{B}\left(B_1 \uplus !^{n_1} B_2, \Gamma_1, cpts_1 \Cup cpts_3, \begin{array}{c} A_1 \\ A_2 \\ A_3 \\ n_1 \geq 0 \end{array} \right) = \langle B, \Gamma, A \rangle \\ cpts_3 = cpts_2 \cup \{(A_3^1, \texttt{FVO}((N_1\ N_2)))\} \\ \mathscr{U}(!^{n_1}\Theta_2, \Theta_1) = A_3 \\ \mathcal{S}((N_1\ N_2) : \sigma) = \langle \Theta_2, B_2, A_2, cpts_2 \rangle \\ \mathcal{S}(\lambda x.M : \sigma \to \tau) = \langle \Theta_1 \multimap \Gamma_1, B_1, A_1, cpts_1 \rangle\end{array}}{\mathcal{S}((\lambda x.M\ (N_1\ N_2)) : \tau) = \left\langle !^n \Gamma, !^n B, \begin{array}{c} A \\ n \geq 0 \end{array}, cpts_1 \Cup cpts_3 \right\rangle} \text{s@}\lambda\text{@} \tag{17}$$

If the function is a variable and the argument is not an application, we need to add boxes around the argument and also to impose the type of x to be without external bangs ($\sum n_{i_j} = 0$) in order to perform the application.

$$\dfrac{\begin{array}{l}\mathscr{B}\left(B_1 \uplus !^{n_1} B_3, \mathit{\Gamma}_1, cpts_1 \Cup cpts_2, \begin{array}{c}\sum n_{i_j} = 0\\ A_1 \\ A_3 \\ A_4 \\ n_1 \geq 0\end{array}\right) = \langle B, \mathit{\Gamma}, A\rangle \\ \mathscr{U}(!^{n_1}\mathit{\Theta}_3, \mathit{\Theta}_1) = A_4 \\ \mathscr{B}(B_2, \mathit{\Theta}_2, cpts_2, A_2) = \langle B_3, \mathit{\Theta}_3, A_3\rangle \\ \mathcal{S}(N : \sigma) = \langle \mathit{\Theta}_2, B_2, A_2, cpts_2\rangle \\ \mathcal{S}(x : \sigma \to \tau) = \langle !^{\sum n_{i_j}}(\mathit{\Theta}_1 \multimap \mathit{\Gamma}_1), B_1, A_1, cpts_1\rangle\end{array}}{\mathcal{S}((x\ N) : \tau) = \left\langle !^n \mathit{\Gamma}, !^n B, \begin{array}{c}A\\ n \geq 0\end{array}, cpts_1 \Cup cpts_2\right\rangle}\ \mathrm{s@}x \tag{18}$$

If the function is a variable and the argument is an application, similar to the previous cases, we need to add a new critical point $((A_3^1, \mathtt{FVO}((N_1\ N_2))))$ and the constraint stating that the type of x is not exponential ($\sum n_{i_j} = 0$).

$$\dfrac{\begin{array}{l}\mathscr{B}\left(B_1 \uplus !^{n_1} B_2, \mathit{\Gamma}_1, cpts_1 \Cup cpts_3, \begin{array}{c}\sum n_{i_j} = 0\\ A_1 \\ A_2 \\ A_3 \\ n_1 \geq 0\end{array}\right) = \langle B, \mathit{\Gamma}, A\rangle \\ cpts_3 = cpts_2 \cup \{(A_3^1, \mathtt{FVO}((N_1\ N_2)))\} \\ \mathscr{U}(!^{n_1}\mathit{\Theta}_2, \mathit{\Theta}_1) = A_3 \\ \mathcal{S}((N_1\ N_2) : \sigma) = \langle \mathit{\Theta}_2, B_2, A_2, cpts_2\rangle \\ \mathcal{S}(x : \sigma \to \tau) = \langle !^{\sum n_{i_j}}(\mathit{\Theta}_1 \multimap \mathit{\Gamma}_1), B_1, A_1, cpts_1\rangle\end{array}}{\mathcal{S}((x\ (N_1\ N_2)) : \tau) = \left\langle !^n \mathit{\Gamma}, !^n B, \begin{array}{c}A\\ n \geq 0\end{array}, cpts_1 \Cup cpts_3\right\rangle}\ \mathrm{s@}x\mathrm{@} \tag{19}$$

If the function is an application, we need to impose its type to be not exponential ($\sum n_{i_j} = 0$) and to add a critical point $((\sum n_i = 0, \mathtt{FVO}((M_1\ M_2))))$. If the argument is not an application we need to add boxes as in the previous cases.

$$\dfrac{\begin{array}{l}\mathscr{B}\left(B_1 \uplus !^{n_1} B_3, \mathit{\Gamma}_1, cpts_3 \Cup cpts_2, \begin{array}{c}\sum n_{i_j} = 0\\ A_1 \\ A_3 \\ A_4 \\ n_1 \geq 0\end{array}\right) = \langle B, \mathit{\Gamma}, A \rangle \\ \mathscr{U}(!^{n_1}\mathit{\Theta}_3, \mathit{\Theta}_1) = A_4 \\ cpts_3 = cpts_1 \cup \{(\sum n_i = 0, \mathtt{FVO}((M_1\ M_2)))\} \\ \mathscr{B}(B_2, \mathit{\Theta}_2, cpts_2, A_2) = \langle B_3, \mathit{\Theta}_3, A_3 \rangle \\ \mathcal{S}(N : \sigma) = \langle \mathit{\Theta}_2, B_2, A_2, cpts_2 \rangle \\ \mathcal{S}((M_1\ M_2) : \sigma \to \tau) = \langle !^{\sum n_{i_j}}(\mathit{\Theta}_1 \multimap \mathit{\Gamma}_1), B_1, A_1, cpts_1 \rangle\end{array}}{\mathcal{S}((M_1\ M_2)\ N) : \tau) = \left\langle !^n \mathit{\Gamma}, !^n B, \begin{array}{c} A \\ n \geq 0\end{array}, cpts_3 \Cup cpts_2 \right\rangle}\ \mathrm{s@@} \tag{20}$$

Finally, if both function and argument are applications, we need to add two new critical points $((A_3^1, \mathtt{FVO}((N_1\ N_2)))$ for the argument and $(\sum n_i = 0, \mathtt{FVO}((M_1\ M_2)))$ for the function) and impose the function's type not to be exponential.

$$\dfrac{\begin{array}{l}\mathscr{B}\left(B_1 \uplus !^{n_1} B_2, \mathit{\Gamma}_1, cpts_3 \Cup cpts_4, \begin{array}{c}\sum n_{i_j} = 0\\ A_1 \\ A_2 \\ A_3 \\ n_1 \geq 0\end{array}\right) = \langle B, \mathit{\Gamma}, A \rangle \\ cpts_4 = cpts_2 \cup \{(A_3^1, \mathtt{FVO}((N_1\ N_2)))\} \\ \mathscr{U}(!^{n_1}\mathit{\Theta}_2, \mathit{\Theta}_1) = A_3 \\ cpts_3 = cpts_1 \cup \{(\sum n_i = 0, \mathtt{FVO}((M_1\ M_2)))\} \\ \mathcal{S}((N_1\ N_2) : \sigma) = \langle \mathit{\Theta}_2, B_2, A_2, cpts_2 \rangle \\ \mathcal{S}((M_1\ M_2) : \sigma \to \tau) = \langle !^{\sum n_{i_j}}(\mathit{\Theta}_1 \multimap \mathit{\Gamma}_1), B_1, A_1, cpts_1 \rangle\end{array}}{\mathcal{S}((M_1\ M_2)\ (N_1\ N_2)) : \tau) = \left\langle !^n \mathit{\Gamma}, !^n B, \begin{array}{c} A \\ n \geq 0\end{array}, cpts_3 \Cup cpts_4 \right\rangle}\ \mathrm{s@@@} \tag{21}$$

A.8 Type synthesis algorithm: $\mathscr{S}$

$\mathscr{S}$ simply adds n boxes, eventually contracts types (remember that contraction produces an empty set of constraints if there is only one argument) and finally forgets the list of critical points *cpts* that are of no interest at this stage.

$$\frac{\begin{array}{l}\mathcal{C}(!^n\Theta_{1_h},\dots,!^n\Theta_{k_h}) = A_h\\ \vdots\\ \mathcal{C}(!^n\Theta_{1_1},\dots,!^n\Theta_{k_1}) = A_1\\ \mathcal{S}(M:\sigma) = \left\langle \Theta, \left\{\begin{array}{c} x_1:\Theta_{1_1},\dots,x_1:\Theta_{k_1},\\ \vdots\\ x_h:\Theta_{1_h},\dots x_h:\Theta_{k_h}\end{array}\right\}, A, cpts \right\rangle\end{array}}{\mathscr{S}(M:\sigma) = \left\langle !^n\Theta, \left\{\begin{array}{c} x_1 :!^n\Theta_{1_1}\\ \vdots\\ x_h :!^n\Theta_{1_h}\end{array}\right\}, \begin{array}{c} A\\ A_1\\ \vdots\\ A_h\\ n\geq 0\end{array} \right\rangle} \tag{22}$$