Principal Typing in Elementary Affine Logic

Paolo Coppola[1] and Simona Ronchi della Rocca[2]

[1] Dipartimento di Matematica e Informatica
Università di Udine
via delle Scienze 206, I-33100 Udine, Italy
coppola@dimi.uniud.it
[2] Dipartimento di Informatica
Università di Torino
Corso Svizzera 185, I-10149 Torino, Italy
ronchi@di.unito.it

Abstract. Elementary Affine Logic (EAL) is a variant of the Linear Logic characterizing the computational power of the elementary bounded Turing machines. The EAL Type Inference problem is the problem of automatically assign to terms of λ-calculus EAL formulas as types. The problem is solved by showing that every λ-term which is typeable has a finite set of *principal typing schemata*, from which all and only its typings can be derived, through suitable operations. An algorithm is showed, that gives as output, for every λ-term, either a negative answer or the set of its principal typing schemata.

1 Introduction

In a communication computing setting, for example in writing communication protocols, it can be necessary to produce tools whose execution is limited in time [10]. Starting from a seminal idea of Girard, which designed the first variant of Linear Logic normalizable in polynomial time, the Light Linear Logic [7], some logical systems have been proposed in the literature, characterizing different complexity classes. From each one of these logics a λ-calculus like language can be defined, using the Curry-Howard isomorphism. In [2] a language has been designed, Λ^{EA}, starting from (the implicational fragment of) Elementary Affine Logic (EAL), whose computational power is that one of elementary bounded Turing machines. The language is untyped, and it can be assigned types which are formulas of EAL through a type assignment system in natural deduction style (NEAL), proving statements (typings) of the shape $\Gamma \vdash_{\mathsf{NEAL}} M : A$, where the context Γ assigns types to variables. However the language, as almost all designed in this way, is quite complex, both as syntax and as reduction rules. Indeed, its syntax reflects the inference rules, and its reduction rules reflect all the normalization steps, which are quite involved. So it cannot be proposed as paradigm for a real programming language. It would be interesting to have a decidable type assignment system which directly assigns EAL formulas as types to terms of the standard λ-calculus: in this way the programmer can use an easy

and well known language, while the compiler, through the type inference algorithm, can automatically control the time complexity of the programs. Moreover typability in EAL has an interesting and useful side effect. In fact, from a practical point of view, if we are interested in the efficient implementation of functional languages, EAL-typeable terms are wonderful candidates. In fact they can be reduced with the *abstract* subset of Lamping's optimal reduction algorithm [9] obtaining excellent performances. Hence a type inference algorithm in EAL can be used in an optimal implementation of a functional language as a check: if it returns true than *compile* the term for the abstract Lamping's algorithm else *compile* as usual.

A type assignment system for λ-calculus cannot be designed directly, since the only way of defining a EAL typing for the λ-calculus is the following: a term $M \in \Lambda$ can have a typing $\Gamma \vdash_{\mathsf{NEAL}} M : A$ if and only if there is a term M' of Λ^{EA} such that it *corresponds* to M, and $\Gamma \vdash_{\mathsf{NEAL}} M' : A$. The correspondence between Λ^{EA} and Λ is realized by performing the substitutions that are implicit in terms of Λ^{EA}. Moreover, in order to use typability as a sufficient condition for reducing terms with the Lamping's abstract algorithm, we restrict ourselves to consider just terms of Λ^{EA} wich are *simple*. To be simple means that the corresponding proofnet *contracts at most variables*, i.e. terms of Λ^{EA} sharing non trivial subterms are not taken into account. In fact Lamping's proofnets corresponding to the initial translation of lambda terms do not share subterms. The number of terms of Λ^{EA} corresponding to a λ-term is in general infinite (thanks to the possibility of adding boxes just replacing variables by variables), and so at a first look the problem of the EAL type inference for the λ-calculus seems semi-decidable.

However, in [2] it is proved that such a problem is decidable. More precisely a procedure has been designed for assigning to a λ-term, that can be typed in the Curry's type assignment system [3, 4, 5], a set of types of EAL: the procedure takes as input a pair of a term and a Curry's type that can be assigned to it, and produces as output either a negative result or a typing, assigning to the term itself a set of EAL types related to the Curry type in input, in the sense that it express the same functionality behavior.

In this paper we give a stronger solution of the problem, by designing a modular type inference algorithm for λ-calculus. Namely the algorithm generates, for every λ-term, either a negative answer or a finite set of *principal typing schemata*, from which all and only its typings can be derived by certain operations. The solution is given in two steps. First we introduce the language of abstract EA-terms, to which EAL types can be assigned through a suitable type assignment system, An abstract EA-terms codifies an infinite set of terms of Λ^{EA}, and in particular to every term M of Λ^{EA} an abstract term in normal form (its canonical form $\mathscr{C}an(M)$) can be assigned. In particular, every typing of M are typings of $\mathscr{C}an(M)$, and every typing of $\mathscr{C}an(M)$ is a typing of at least one of the infinite set of terms it represents. We prove that every abstract term has a principal typing schemata, from which all and only the typings for

it can be derived, through suitable operations. An algorithm $PT(\ )$, generating the principal typing scheme of an abstract EA-term, is showed.

Secondly we prove that the result can be extended to Λ^{EA}, by designing a procedure $\mathcal{C}(\ )$ such that, for every pure λ-term M, $\mathcal{C}(M)$ generates all and only the canonical forms of the terms of Λ^{EA} corresponding to M. Since $\mathcal{C}(M)$ generates always a finite set, the composition of $\mathcal{C}(\ )$ and $PT(\ )$ gives the desired type inference algorithm for pure λ-terms.

The notion of principal typing for the EAL type assignment system is reminiscent of the corresponding notion for intersection types [12, 11]. In fact, while in the seminal case of Curry's type assignment types are obtained from the schemata using substitutions [8], in both cases also other operations are involved, which, for EAL typing, include to solve a system of linear constraints.

Related works are the results of Schellinx *et al* [13, 6] in the field of linear decorations of intuitionistic derivations and the type checking algorithm of Baillot [1] for lambda terms in LAL.

The paper is organized as follows. In Section 2 the λ^{EA}-calculus and the type assignment system $\vdash_{\mathsf{NEAL}}$ are recalled. Section 3 presents the abstract EA-terms, the type assignment system $\vdash_{abs}$, and the notion of canonical forms. In Section 4 the principal typing property for abstract EA-terms is proved. In Section 5 the principal typing property for the λ-calculus is proved. The appendix contains the grammars generating EA-terms and simple EA-terms respectively, and the sketches of some technical proofs.

2 The λ^{EA}-calculus

In this section we will briefly recall the λ^{EA}-calculus, as defined in [2], its typing rules, its relation with the λ-calculus, and the definition of EAL-typability of λ-calculus.

Definition 1. *i) The set of terms of the λ^{EA}-calculus (Λ^{EA}) is generated by the grammar:*

$$M ::= x \mid \lambda x.M \mid (M\ M) \mid !(M)\left[{}^{M}/x, \ldots, {}^{M}/x\right] \mid [M]_{M=x,y}$$

where all variables occur once. Λ^{EA} is ranged over by M, N, P, Q.

ii) EA-types *are formulas of Elementary Affine Logic, and they are generated by the following grammar:*

$$A ::= a \mid A \multimap A \mid !A$$

where a belongs to a countable set of basic type constants. EA-types will be ranged over by A, B, C.

iii) The type assignment system $\vdash_{\mathsf{NEAL}}$ assigns EA-types to EA-terms, starting from a context, assigning EA-types to variables. The rules of the system are given in Table 1.

The following definition relates terms of Λ^{EA} with the terms of the classical λ-calculus.

$$\frac{}{\Gamma, x : A \vdash_{\mathsf{NEAL}} x : A}\ ax \qquad \frac{\Gamma \vdash_{\mathsf{NEAL}} M : !A \quad \Delta, x : !A, y : !A \vdash_{\mathsf{NEAL}} N : B}{\Gamma, \Delta \vdash_{\mathsf{NEAL}} [N]_{M=x,y} : B}\ contr$$

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

$$\frac{\Delta_1 \vdash_{\mathsf{NEAL}} M_1 : !A_1 \quad \cdots \quad \Delta_n \vdash_{\mathsf{NEAL}} M_n : !A_n \quad x_1 : A_1, \ldots, x_n : A_n \vdash_{\mathsf{NEAL}} N : B}{\Gamma, \Delta_1, \ldots, \Delta_n \vdash_{\mathsf{NEAL}} !\,(N)\left[{}^{M_1}/_{x_1}, \ldots, {}^{M_n}/_{x_n}\right] : !B}\ !$$

Table 1. Type assignment system for EA-terms.

Definition 2. *i) The set of terms of the λ-calculus (Λ) are defined by the following grammar:*

$$M ::= x \mid MM \mid \lambda x.M$$

By abuse of notation, Λ will be ranged over by M, N, P, Q, as Λ^{EA}, being the different meaning clear from the context.

ii) The erasure function $()^* : \Lambda^{EA} \to \Lambda$ is defined by induction on the structure of the EA-term as follows:

$$\begin{aligned}
(x)^* &= x \\
(\lambda x.M)^* &= \lambda x.M^* \\
((M_1\ M_2))^* &= (M_1^*\ M_2^*) \\
(!(M)[{}^{M_1}/_{x_1}, \cdots, {}^{M_n}/_{x_n}])^* &= M^*\{M_1^*/x_1, \ldots, M_n^*/x_n\} \\
([M]_{N=x_1,x_2})^* &= M^*\{N^*/x_1, N^*/x_2\}
\end{aligned}$$

In order to extend to Λ the notion of type assignment with EA-types, let us introduce a particular class of EA-terms, the simple ones, which correspond, if translated into proofnets, to contracting at most variables. We discard EA-terms contracting non trivial subterms otherwise we could loose the possibility of reducing lambda terms with the abstract Lamping's algorithm.

Definition 3. *i) The* length *$L(M)$ of an EA-term M is defined inductively as follows:*

$$\begin{aligned}
L(x) &= 0 \\
L(\lambda x.M) &= 1 + L(M) \\
L((M\ N)) &= 1 + L(M) + L(N) \\
L(!\,(M)\left[{}^{M_1}/_{x_1}, \ldots, {}^{M_n}/_{x_n}\right]) &= L(M) + \sum_{i=1}^{n} L(M_i) \\
L([M]_{N=x,y}) &= L(M) + L(N).
\end{aligned}$$

ii) An EA-term M is simple *if and only if*

1. *M has no subterm of the form $[M_1]_{M_2=x,y}$ where $(M_2)^*$ is not a variable,*
2. *$L(M) = L((M)^*)$*

Definition 4. *Let $M \in \Lambda$. $\Gamma \vdash_{\mathsf{NEAL}} M : A$ if and only if $\Gamma \vdash_{\mathsf{NEAL}} P : A$ and $(P)^* = M$, for some $P \in \Lambda^{EA}$, P simple.*

Let $M \in \Lambda$ and $N \in \Lambda^{EA}$ be *correspondent* if $(N)^* = M$. The simple EA-terms corresponding to a given λ-terms can be infinite, so at a first look the problem of EA typability for λ-terms seems to be semi-decidable.

3 Abstract EA-terms and Canonical Forms

In this section the notion of abstract EA-terms, and a type assignment for assigning EA-types to them are introduced. As will be clarified in the sequel, a term of Abs^{EA} is a term in a meta-language, representing an infinite set of terms of Λ^{EA}.

Definition 5. *The set Abs^{EA} of* abstract EA-terms *is generated by the following grammar:*

$$M ::= x \mid \lambda x.M \mid (M\ M) \mid\ [M]_{N_1 \to (\widehat{x_1}),\ldots,N_k \to (\widehat{x_k})} \mid\ \nabla(M)[{}^{N_1}/x_1, \cdots, {}^{N_k}/x_n]$$

with the condition that every variable occur just once in a term.

On abstract EA-terms we define a reduction rule, $\to_{\mathscr{C}an}$, whose role is to limit the number of nested occurrences of subterms of the shape either $[M]_{N_1 \to (\widehat{x_1}),\ldots,N_k \to (\widehat{x_k})}$ or $\nabla(M)[{}^{N_1}/x_1, \cdots, {}^{N_k}/x_n]$. The definition of the sets of free variables $\mathtt{FV}$ for Abs^{EA} and the reduction rules $\to_{\mathscr{C}an}$ are given in Figures 1 and 2 respectively.

Lemma 1. *$\to_{\mathscr{C}an}$ is Church-Rosser.*

The type assignment system assigning formulas of EAL to terms of Abs^{EA} is shown in Table 2. Note that the main difference between this system and that one for Λ^{EA} is that the rule $(!^m)$ is parametric, in the sense that it can assign a type having any number $m > 0$ of modalities. In fact a derivation in the type assignment system $\vdash_{abs}$ can represent an infinite number of derivations in the system $\vdash_{\mathsf{NEAL}}$, by contracting in one application of the rule $(!^m)$ a sequence of $m > 0$ application of the rule $(!)$ and in one application of the rule $(contr)$ a sequence of the applications of the rule with the same name, acting on disjoint pairs of variables.

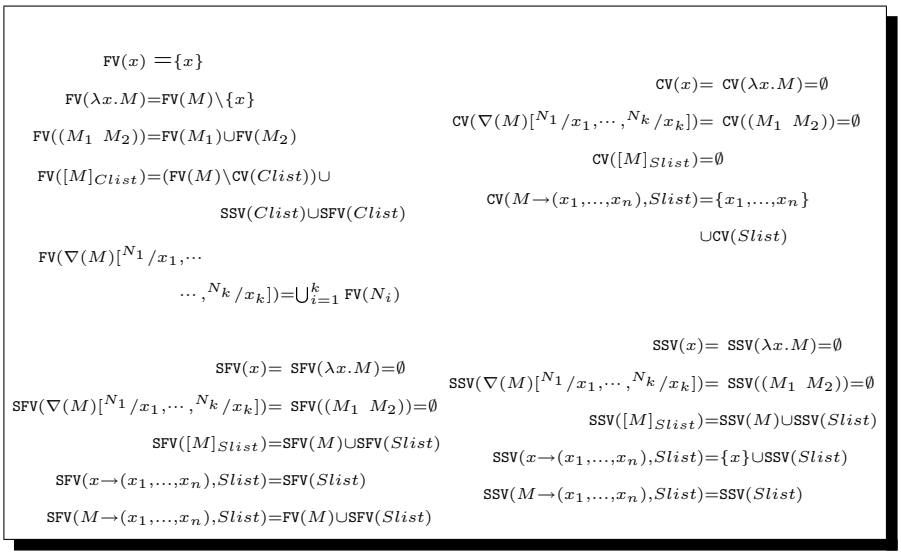
Lemma 2 (Subject reduction). *Let $M \in Abs^{EA}$ and $M \to^*_{\mathscr{C}an} N$, then*

$$\Gamma \vdash_{abs} M : A \Rightarrow \Gamma \vdash_{abs} N : A.$$

Proof. By induction on the length of the reduction. By cases on the last step of the reduction.

- if $[x]_{\ldots,N \to (\ldots,x,\ldots),\ldots} \to_{\mathscr{C}an} N$ than the type assignment is

$$\dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash_{abs} N :!A'\end{array} \qquad \begin{array}{c}\vdots\\ \Delta_1, x :!A', \Delta_2 \vdash_{abs} x :!A'\end{array}}{\Delta_1, \Delta_2, \Gamma \vdash_{abs} [x]_{\ldots,N \to (\ldots,x,\ldots),\ldots} :!A'}\ contr$$



$$\mathtt{FV}(x) = \{x\}$$

$$\mathtt{FV}(\lambda x.M) = \mathtt{FV}(M) \setminus \{x\}$$

$$\mathtt{FV}((M_1\ M_2)) = \mathtt{FV}(M_1) \cup \mathtt{FV}(M_2)$$

$$\mathtt{FV}([M]_{Clist}) = (\mathtt{FV}(M) \setminus \mathtt{CV}(Clist)) \cup \mathtt{SSV}(Clist) \cup \mathtt{SFV}(Clist)$$

$$\mathtt{FV}(\nabla(M)[{}^{N_1}/_{x_1}, \cdots, {}^{N_k}/_{x_k}]) = \bigcup_{i=1}^{k} \mathtt{FV}(N_i)$$

$$\mathtt{CV}(x) = \mathtt{CV}(\lambda x.M) = \emptyset$$

$$\mathtt{CV}(\nabla(M)[{}^{N_1}/_{x_1}, \cdots, {}^{N_k}/_{x_k}]) = \mathtt{CV}((M_1\ M_2)) = \emptyset$$

$$\mathtt{CV}([M]_{Slist}) = \emptyset$$

$$\mathtt{CV}(M \to (x_1, \ldots, x_n), Slist) = \{x_1, \ldots, x_n\} \cup \mathtt{CV}(Slist)$$

$$\mathtt{SFV}(x) = \mathtt{SFV}(\lambda x.M) = \emptyset$$

$$\mathtt{SFV}(\nabla(M)[{}^{N_1}/_{x_1}, \cdots, {}^{N_k}/_{x_k}]) = \mathtt{SFV}((M_1\ M_2)) = \emptyset$$

$$\mathtt{SFV}([M]_{Slist}) = \mathtt{SFV}(M) \cup \mathtt{SFV}(Slist)$$

$$\mathtt{SFV}(x \to (x_1, \ldots, x_n), Slist) = \mathtt{SFV}(Slist)$$

$$\mathtt{SFV}(M \to (x_1, \ldots, x_n), Slist) = \mathtt{FV}(M) \cup \mathtt{SFV}(Slist)$$

$$\mathtt{SSV}(x) = \mathtt{SSV}(\lambda x.M) = \emptyset$$

$$\mathtt{SSV}(\nabla(M)[{}^{N_1}/_{x_1}, \cdots, {}^{N_k}/_{x_k}]) = \mathtt{SSV}((M_1\ M_2)) = \emptyset$$

$$\mathtt{SSV}([M]_{Slist}) = \mathtt{SSV}(M) \cup \mathtt{SSV}(Slist)$$

$$\mathtt{SSV}(x \to (x_1, \ldots, x_n), Slist) = \{x\} \cup \mathtt{SSV}(Slist)$$

$$\mathtt{SSV}(M \to (x_1, \ldots, x_n), Slist) = \mathtt{SSV}(Slist)$$

Fig. 1. Free, Contracted, Shared Free, and Single Shared Variables for EA-terms.

$$\frac{}{\Gamma, x : A \vdash_{abs} x : A}\ ax$$

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

$$\frac{\begin{array}{cc} \Gamma_1 \vdash_{abs} N_1 : !A_1 & x_1^1 : !A_1, \ldots, x_{n_1}^1 : !A_1, \\ \vdots & \vdots \\ \Gamma_k \vdash_{abs} N_k : !A_k & \Delta, x_1^k : !A_k, \ldots, x_{n_k}^k : !A_k \vdash_{abs} M : B \end{array}}{\Gamma_1, \ldots, \Gamma_k, \Delta \vdash_{abs} [M]_{N_1 \to (x_1^1, \cdots, x_{n_1}^1), \ldots, N_k \to (x_1^k, \cdots, x_{n_k}^k)} : B}\ contr$$

$$\frac{\begin{array}{c} \Delta_1 \vdash_{abs} N_1 : \overbrace{!\ldots!}^{m} A_1 \\ \vdots \\ \Delta_k \vdash_{abs} N_k : \overbrace{!\ldots!}^{m} A_k \end{array} \qquad x_1 : A_1, \ldots, x_k : A_k \vdash_{abs} M : B \quad m > 0}{\Gamma, \Delta_1, \ldots, \Delta_k \vdash_{abs} \nabla(M)[{}^{N_1}/_{x_1}, \cdots, {}^{N_k}/_{x_n}] : \underbrace{!\ldots!}_{m} B}\ !^m$$

Table 2. Type assignment system for Abs^{EA}-terms.

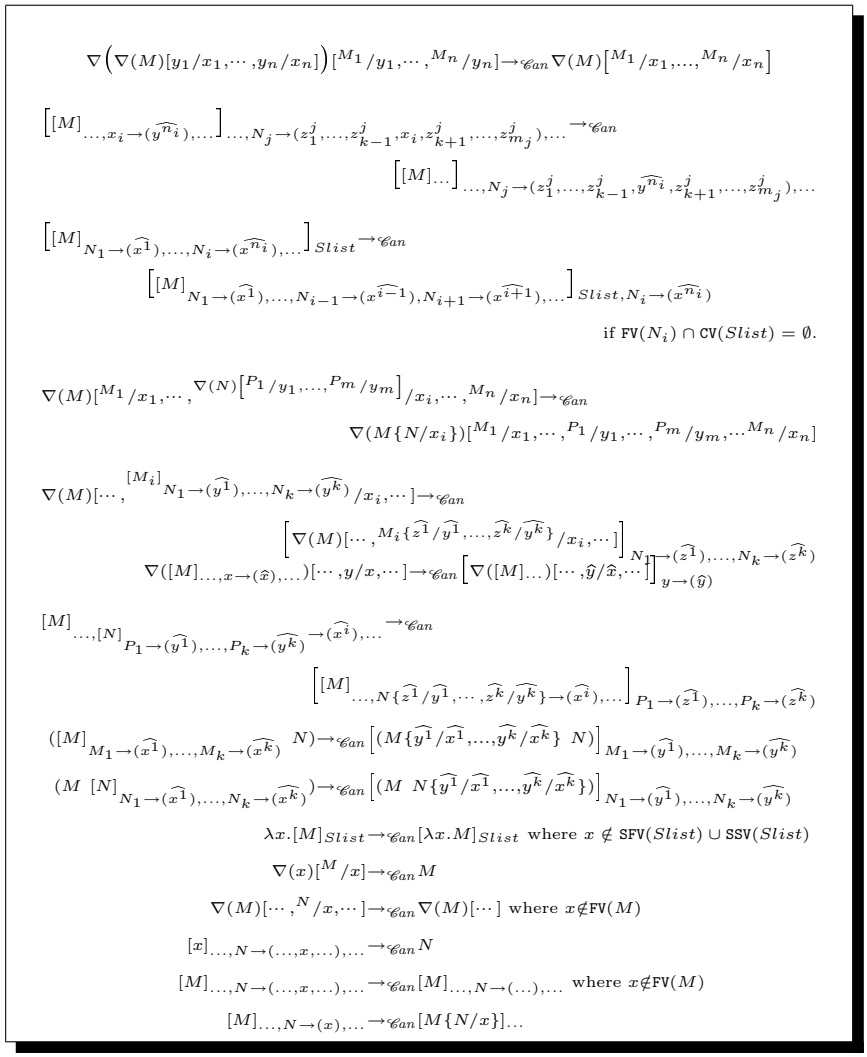


Fig. 2. Reduction relation $\to_{\mathscr{C}an}$ for Abs^{EA} terms.

hence, trivially, $\Delta_1, \Delta_2, \Gamma \vdash_{abs} N :!A'$.

– if $\nabla(x)[^N/x] \rightarrow_{\mathscr{C}an} N$ than the type assignment is

$$\frac{\Gamma \vdash_{abs} N : \overbrace{!...!}^{m} A \quad x : A \vdash_{abs} x : A \quad m > 0}{\Delta, \Gamma \vdash_{abs} \nabla(x)[^N/x] : \underbrace{!...!}_{m} A}\ !^m$$

hence, trivially, $\Delta, \Gamma \vdash_{abs} N : \underbrace{!...!}_{m} A$.

Other cases are equally easy. □

The relation between Λ^{EA} and Abs^{EA} can be formalized by two *embedding functions*, the function *emb* associating to every term of Λ^{EA} a term of Abs^{EA} (its abstract representation) and the function *bme*, associating to every term of Abs^{EA} a set of terms in $\mathscr{P}(\Lambda^{EA})$ (the set of EA-terms which are represented by it).

emb is inductively defined as follows:

$$\begin{aligned} emb(x) &= x \\ emb(\lambda x.M) &= \lambda x.emb(M) \\ emb((M_1\ M_2) &= (emb(M_1)\ emb(M_2)) \\ emb([M]_{N=x,y}) &= [emb(M)]_{emb(N)\rightarrow(x,y)} \\ emb(!(M)[^{N_1}/x_1, \cdots, ^{N_k}/x_k]) &= \nabla(emb(M))[^{emb(N_1)}/x_1, \cdots, ^{emb(N_k)}/x_k] \end{aligned}$$

Lemma 3. $\forall M \in \Lambda^{EA} \quad \Gamma \vdash_{\mathsf{NEAL}} M : A \Rightarrow \Gamma \vdash_{abs} emb(M) : A$

Proof. By structural induction on M, the thesis holds trivially for $M = x$ and by hypothesis for $M = \lambda x.M'$, $M = (M_1\ M_2)$ and $M = [M_1]_{N=x,y}$. For $M =!(M_1)[^{N_1}/x_1, \cdots, ^{N_k}/x_k]$ the thesis holds by the observation that !-rule in the assignment system for Λ^{EA} is simply a particular case of $!^m$-rule in the assignment system for Abs^{EA}. □

In particular, to every term in Λ^{EA} a *canonical form* can be assigned, which is a term of Abs^{EA} in normal form. The canonical form of $M \in \Lambda^{EA}$ can be obtained by reducing to normal form (with respect to $\rightarrow_{\mathscr{C}an}$) the term $emb(M) \in Abs^{EA}$. Thanks to the Lemma 1 the canonical form of a term is unique: let us call it $\mathscr{C}an(M)$.

Lemma 4. *$\forall M \in \Lambda^{EA}$. $\Gamma \vdash_{\mathsf{NEAL}} M : A$ then $\Gamma \vdash_{abs} \mathscr{C}an(M) : A$.*

Proof. By Lemma 3 and Lemma 2. □

The function bme, from Abs^{EA} to $\mathscr{P}(\Lambda^{EA})$, can be defined inductively in the following way:

$$\begin{aligned}
bme(x) &= \{x\}\\
bme(\lambda x.M) &= \{\lambda x.M' | M' \in bme(M)\}\\
bme((M_1\ M_2) &= \{(M_1'\ M_2') | M_1' \in bme(M_1) \wedge M_2' \in bme(M_2)\}\\
bme\big(\nabla(M)[^{N_1}/x_1, \dots {}^{N_k}/x_k]\big) &= \{\overbrace{!(\cdots!(}^{n} M')[z_1^1/x_1, \dots, z_k^1/x_k]\overbrace{)\cdots)}^{n-1}[^{N_1'}/z_1^n, \dots\\
&\qquad \dots, {}^{N_k'}/z_k^n]\ | M' \in bme(M) \wedge N_i' \in bme(N_i) \wedge n > 0\}\\
bme([M]_{N\to(x_1^1,x_1^2,\dots,x_1^{n_1}),\dots}) &= bme\left(\left[[M]_{y\to(x_1^1,x_1^2)}\right]_{N\to(y,x_1^3,\dots,x_1^{n_1}),\dots}\right)
\end{aligned}$$

y fresh variable

$$\begin{aligned}
bme([M]_{N_1\to(x_1^1,x_1^2),N_2\to(\widehat{x_2}),\dots}) &= bme\left(\left[[M]_{N_1\to(x_1^1,x_1^2)}\right]_{N_2\to(\widehat{x_2}),\dots}\right)\\
bme([M]_{N\to(x_1,x_2)}) &= \{[M']_{N'=x_1,x_2}\ | M' \in bme(M) \wedge N' \in bme(N)\}
\end{aligned}$$

Lemma 5. $\forall M \in Abs^{EA}$

$$\Gamma \vdash_{abs} M : A \Rightarrow \exists M' \in bme(M)\ \Gamma \vdash_{\mathsf{NEAL}} M' : A.$$

Example 1. i) The abstract term
$\lambda x.[\lambda y.\nabla((x_1\, x_2\, t))[^y/t, {}^{z_1}/x_1, {}^{z_2}/x_2]]_{x\to(z_1,z_2)}$ is the canonical form of the infinite set of terms:

$$\begin{aligned}
\{\lambda x.\big[\lambda z_3^{m+1}.\underbrace{!(\dots!(!}_{m+1}(x_1\, x_2\, t))[^{z_1^1}/x_1, {}^{z_2^1}/x_2, {}^{z_3^1}/t]\big)\\
\cdots)[^{z_1^{m+1}}/z_1^m, {}^{z_2^{m+1}}/z_2^m, {}^{z_3^{m+1}}/z_3^m]\big]_{x\to(z_1^{m+1},z_2^{m+1})} \mid m \in \omega\}.
\end{aligned}$$

ii) The EA term:

$$\lambda x.\Bigg[\big[!((x_1\, y_1\, y_2\, y_3\, y_4))[^x/x_1, {}^{z_1}/y_1,\\
{}^{z_2}/y_2, {}^{z_3}/y_3, {}^{z_4}/y_4]\big]_{!(\lambda x.x)[\,]=z_1,z_2}\Bigg]_{!(\lambda x\, y.x)[\,]=z_3,z_4}$$

has the following canonical form:

$$\lambda x.\Bigg[\nabla((x_1\, y_1\, y_2\, y_3\, y_4))[^x/x_1, {}^{z_1}/y_1, {}^{z_2}/y_2,\\
{}^{z_3}/y_3, {}^{z_4}/y_4]\Bigg]_{\nabla(\lambda x.x)[\,]\to(z_1,z_2),\nabla(\lambda x\, y.x)[\,]\to(z_3,z_4)}$$

4 Principal Typing for $\vdash_{abs}$

In this section we will prove that $\vdash_{abs}$ enjoys the principal typing property, i.e., every typing can be derived from a particular typing schemata by means of suitable operations. First of all, let us introduce the notion of *type scheme*. A type scheme represents an infinite set of types, namely all these ones that can be obtained from it through a special kind of substitution, called *scheme substitution*.

Definition 6. *i)* Type schemata *are defined by the following grammar:*

$$\sigma ::= \alpha \mid \sigma \multimap \sigma \mid !^p(\sigma)$$

where the exponent *p is defined by the following grammar:*

$$p ::= n \mid p + p$$

where α belongs to a countable set of scheme variables, ranged over by α, β, γ, and n belongs to a countable set of literals; type schemata are ranged over by σ, τ, ρ and exponentials are ranged over by p, q, r. Let $\mathscr{T}$ denote the set of type schemata;

ii) *A* scheme substitution *is a function from type schemata to types, denoted by a pair of substitutions $< S, X >$, where S replaces scheme variables by types and X replaces literals by natural numbers ≥ 1. The application of $< S, X >$ to a type scheme is defined inductively as follows:*

$$< S, X > (\alpha) = S(\alpha);$$
$$< S, X > (\sigma \multimap \tau) =< S, X > (\sigma) \multimap < S, X > (\tau);$$
$$< S, X > (!^{n_1 + \ldots + n_i}\sigma) = \underbrace{!\ldots!}_{q} < S, X > (\sigma),$$

where $q = X(n_1) + \ldots + X(n_i)$.

$\equiv$ denotes the syntactical identity between both types and type schemes.

In order to define the principal typing, we need a unification algorithm for type schemes. But first some technical definitions are necessary.

First we introduce a function F collapsing all consecutive $!^p$ in a type scheme:

$$F(\alpha) = \alpha;$$
$$F(\sigma \multimap \tau) = F(\sigma) \multimap F(\tau);$$
$$F(!^p(\sigma)) =!^p F(\sigma) \text{ if } F(\sigma) \text{ is not of the form } !^q\tau.$$
$$F(!^p(\sigma)) =!^{p+q}(\tau) \text{ if } F(\sigma) =!^q\tau.$$

In the following we assume that type schemes are already transformed by F, i.e. it is not the case that a subterm $!^p(!^q(\sigma))$ occur in a type scheme.

Moreover, let $=_e$ be the relation between type scheme defined as follows: $\alpha =_e \alpha$; $\sigma =_e \mu$ and $\tau =_e \nu$ imply $\sigma \multimap \tau =_e \mu \multimap \nu$; $\sigma =_e \tau$ implies $!^p\sigma =_e !^q\tau$.

Roughly speaking, two type schemes are $=_e$ if and only if they are identical modulo the exponentials.

The unification algorithm, which we will present in SOS style, is a function U from $\mathscr{T} \times \mathscr{T}$ to pairs of the shape $< C, s >$, where C (the *modality* set) is a set of natural linear constraints, in the form $p = q$, where p and q are exponentials, and s is a substitution, replacing scheme variables by type schemes. A set C of linear constraints is *solvable* if there is a substitution from literals to natural numbers such that, for every constraint $n_1 + ... + n_i = m_1 + ... + m_j$ in C, $X(n_1) + ... + X(n_i) = X(m_1) + ... + X(m_j)$. Clearly the solvability of a set of linear constraints is a decidable problem.

$$\frac{}{U(\alpha, \alpha) =< \emptyset, [\,] >}$$

$$\frac{\alpha \text{ is a scheme variable not occurring in } \tau}{U(\alpha, \tau) =< \emptyset, [\alpha \mapsto \tau] >}$$

$$\frac{\alpha \text{ is a scheme variable not occurring in } \sigma}{U(\sigma, \alpha) =< \emptyset, [\alpha \mapsto \sigma] >}$$

$$\frac{U(\rho, \mu) =< C, s >}{U(!^p\rho, !^q\mu) =< C \cup \{p = q\}, s >}$$

$$\frac{U(\sigma_1, \tau_1) =< C_1, s_1 > U(s_1(\sigma_2), s_1(\tau_2)) =< C_2, s_2 >}{U(\sigma_1 \to \sigma_2, \tau_1 \multimap \tau_2) =< C_1 \cup C_2, s_1 \circ s_2 >}$$

Note that U is a partially defined function: for example both $U(\alpha, \alpha \multimap \beta)$ and $U(!^p\alpha, \sigma \multimap \tau)$ are undefined.

The following lemma proves that U is the most general unifier for type schemes, with respect to $=_e$.

Lemma 6. *i) (correctness)* $U(\sigma, \tau) =< C, s >$ *implies* $s(\sigma) =_e s(\tau)$.
ii) (completeness) $s(\sigma) =_e s(\tau)$ *implies* $U(\sigma, \tau) =< C, s' >$ *and* $s = s' \circ s''$, *for some* s''.

Proof. (correctness) By induction on the rules deriving $U(\sigma, \tau) =< c, s >$.

(Completeness) By induction on the pair (number of symbols of σ, number of scheme variables occurring in σ). □

Lemma 7. *Let* $< S, X >$ *be a scheme substitution such that* $< S, X > (\sigma) \equiv < S, X > (\tau)$. *Then* $U(\sigma, \tau) =< C, s >$, *and there is* $< S', X' >$ *such that* X' *is a solution of* C *and* $< S', X' > (s(\sigma)) \equiv < S, X > (\sigma)$.

Proof. By induction on the pair (number of symbols of σ, number of scheme variables occurring in σ). □

Let

$$\begin{aligned} U(\sigma_1, \sigma_2, \ldots, \sigma_n) = \ & \texttt{let } U(\sigma_1, \sigma_2) =< C_1, s_1 > \\ & \quad \texttt{and let } U(s_1 \circ \cdots \circ s_i(\sigma_{i+1}), s_1 \circ \cdots \circ s_i(\sigma_{i+2})) =< C_{i+1}, s_{i+1} > \\ & \qquad \texttt{in } \ < C_1 \cup \cdots \cup C_n, s_1 \circ \cdots \circ s_n > . \end{aligned}$$

The following function $PT(\ )$ takes as input a term of Abs^{EA} and gives as output a *typing scheme* , i.e., a triple of a scheme context, a type scheme and a set of constraints, where a scheme context is a finite set of scheme assignments to variables. $PT(M)$ is designed by induction on the structure of M, and it is based on the property that the type assignment $\vdash_{abs}$ is syntax directed. Note that, if restricted to terms of λ-calculus, PT coincides with the well known principal type algorithm designed by Hindley for Curry's type assignment [8]. PT is defined modulo names of type variables.

- $PT(x) =< \{x : \alpha\}, \alpha, \emptyset >$, where α is fresh;
- $PT(\lambda x.M) =$ `let` $PT(M) =< B, \sigma, C >$ `in`
  `if` $B = B' \cup \{x : \tau\}$ `then` $< B', \tau \multimap \sigma, C >$
  `else` $< B, \alpha \multimap \sigma, C >$ where α is fresh;
- $PT(M\ N) =$ `let` $PT(M) =< B_1, \sigma_1, C_1 >$ `and` $PT(N) =< B_2, \sigma_2, C_2 >$
  `and let` they be disjoint `in let` $U(\sigma_1, \sigma_2 \multimap \alpha) =< C, s >$ (α fresh)
  `in` $< s(B_1 \cup B_2), s(\alpha), C \cup C_1 \cup C_2 >$;
- $PT([M]_{N_1 \to (\widehat{x_1}),\ldots,N_k \to (\widehat{x_k})}) =$ `let` $PT(M) =< B, \sigma, C >$ `and`
  $PT(N_i) =< B_i, \sigma_i, C_i >$ (all disjoint and disjoint from $PT(M)$)
  `and let` $U(B(x_1^1), \ldots, B(x_{n_1}^1), \sigma_1, !^{m_1}\alpha_1) =< C_1', s_1 >$
  `and` $U(s_1 \circ \cdots \circ s_i(B(x_1^{i+1})), \ldots, s_1 \circ \cdots \circ s_i(B(x_{n_{i+1}}^{i+1})),$
  $s_1 \circ \cdots \circ s_i(\sigma_{i+1}), !^{m_{i+1}}\alpha_{i+1}) =< C_{i+1}', s_{i+1} >$
  `and` $s = s_1 \circ \cdots \circ s_k$
  `in`$< s(B/\widehat{x_1}, ..., \widehat{x_k} \cup \bigcup_{1 \le j \le k} B_j), s(\sigma), C \cup \bigcup_{1 \le j \le k} C_j \cup \bigcup_{1 \le j \le k} C_j' >$.
  (where α_i and m_i are fresh and $B/\widehat{x_1}, ..., \widehat{x_k}$ denotes the context obtained from B by deleting the assignments to variables in $\widehat{x_i}$ $(1 \le i \le k)$)
- $PT(\nabla(M)[N_1/x_1, \ldots, N_m/x_m]) =$ `let` $PT(M) =< B, \sigma, C >$
  `and` $PT(N_i) =< B_i, \sigma_i, C_i >$ (all disjoint and disjoint from $PT(M)$)
  `and let` $U(!^n B(x_1), \sigma_1) =< C_1', s_1 >$
  `and` $U(!^n(s_1 \circ \cdots \circ s_i(B(x_{i+1}))), s_1 \circ \cdots \circ s_i(\sigma_{i+1})) =< C_{i+1}', s_{i+1} >$
  `and` $s = s_1 \circ \cdots \circ s_m$
  `in` $< s(\bigcup_{1 \le j \le m} B_j), !^n(s(\sigma)), C \cup \bigcup_{1 \le j \le m} C_j \cup \bigcup_{1 \le j \le m} C_j' >$ (n fresh).
  where x_j^i denotes the j-th component of $\widehat{x_i}$.

Let Θ be a scheme context: $< S, X > (\Theta)$ is an abbreviation for the scheme context $\{x :< S, X > (\sigma) \mid x : \sigma \in \Theta\}$.

From the typing scheme of a term all and only its typings can be derived, through scheme substitutions, as proved in the next theorem.

Theorem 1 (Principal typing for Abs^{EA}).

(correctness) $PT(M) =< \Theta, \sigma, C >$ implies, for all scheme substitution $< S, X >$ such that X satisfies C, $< S, X > (\Theta) \vdash_{abs} M :< S, X > (\sigma)$.

(completeness) $\Gamma \vdash_{abs} M : A$ implies $PT(M) =< \Theta, \sigma, C >$ and there is a scheme substitution $< S, X >$ such that X satisfies C, $\Gamma \supseteq < S, X > (\Theta)$ and $A \equiv < S, X > (\sigma)$.

Proof. (correctness) By induction on M, using the fact that the derivations are syntax directed.
(compl.) By induction on the derivation of $\Gamma \vdash_{abs} M : A$, using Lemma 7. □

Example 2. i) Let M be the abstract term

$$\lambda x.\big[\lambda y.\nabla(x_3\,(x_4\,y_1))[{}^{x_1}/_{x_3}, {}^{x_2}/_{x_4}, {}^{y}/_{y_1}]\big]_{x\to(x_1,x_2)}.$$

Then $PT(M) = \langle \emptyset, !^n(\alpha \multimap \alpha) \multimap !^n\alpha \multimap !^n\alpha, \emptyset\rangle$.

ii) Let N be the abstract term

$$\lambda x.\big[\nabla((x_1\,y_1\,y_2\,y_3\,y_4))[{}^{x}/_{x_1}, {}^{z_1}/_{y_1}, {}^{z_2}/_{y_2}, {}^{z_3}/_{y_3}, {}^{z_4}/_{y_4}]\big]_{\nabla(\lambda x.x)[\,]\to(z_1,z_2),\nabla(\lambda x.\lambda y.x)[\,]\to(z_3,z_4)}$$

Then $PT(N) = \langle \emptyset, !^n\sigma \multimap !^n\sigma \multimap !^m\tau \multimap !^m\tau \multimap !^p\alpha, \{n = p, n = m\}\rangle$, where $\sigma \equiv \beta \multimap \beta$ and $\tau \equiv \gamma \multimap \epsilon \multimap \gamma$.

5 Type Inference for λ-calculus

In this section we prove that if a term of the λ-calculus is typeable in the type assignment system $\vdash_{\mathsf{NEAL}}$, than it has a finite set of principal typing schemes. First of all, we can associate, to every λ-term, the set of canonical forms of all terms of Λ^{EA} corresponding to it, according with the following definition.

Definition 7. *Let $M \in \Lambda$. The set of canonical forms of the terms of Λ^{EA} corresponding to the λ-termM is $C(M) = \{N \mid \exists R \in \Lambda^{EA}$ such that $(R)^* = M$, R is simple and $N = \mathscr{C}an(R)\}$.*

The following lemma proves the key property that assures us the decidability.

Lemma 8. *For every $M \in \Lambda$, $C(M)$ is finite.*

We will show an algorithm $\mathcal{C}$ such that, for every $M \in \Lambda$, $\mathcal{C}(M)$ gives either $C(M)$ or a negative answer. $\mathcal{C}$ is correct and complete.

Let $\mathbb{L}(M)$ be the *linearization* of M with respect to all its free variables and let $\mathbb{L}_x(M)$ be the set of fresh variables generated by $\mathbb{L}$ during the linearization of x in M. I.e. let $M = (x\ (x\ y\ y))$ then $\mathbb{L}(M) = (x_1\ (x_2\ y_1\ y_2))$ and $\mathbb{L}_x(M) = \{x_1, x_2\}$, $\mathbb{L}_y(M) = \{y_1, y_2\}$ and $\mathbb{L}_z(M) = \emptyset$ for any other variable z.

$\mathcal{C}$ has three sub-procedures:

- $\mathbb{T}$ building the *linear* part of the canonical forms;
- $\mathbb{F}$ and $\mathbb{F}'$ building the boxes, i.e. the sharable parts of the canonical forms.

The algorithm is defined by the equations below. For space and readability reasons we use meta notations. For example $x \in_{i>1} A$ iff x occurs in A more than once, $\lambda x.(A)$ represents the set $\{\lambda x.M \mid M \in A\}$, $A@B$ stands for $\{(M\ N) \mid M \in A \wedge N \in B\}$.

$$\begin{aligned}\mathcal{C}(M) = \ & \texttt{if}\ \ \exists x_1, \ldots, x_k \in_{i>1} \mathtt{FV}(M)\\ & \texttt{then}\ \ \big[\mathbb{T}(\mathbb{L}(M)) \cup \mathbb{F}(\mathbb{L}(M))\big]_{x_1\to(\mathbb{L}_{x_1}(M)),\ldots,x_k\to(\mathbb{L}_{x_k}(M))}\\ & \texttt{else}\ \ \mathbb{T}(M) \cup \mathbb{F}(M)\end{aligned}$$

$$
\begin{aligned}
\mathbb{T}(x) &= \{x\}\\
\mathbb{T}(\lambda x.M) &= \texttt{if } x \in_{i>1} \texttt{FV}(M) \texttt{ then } \quad \lambda x.\big[\mathbb{T}(\mathbb{L}(M)) \cup \mathbb{F}(\mathbb{L}(M))\big]_{x\to(\mathbb{L}_x(M))}\\
&\qquad \texttt{else } \quad \lambda x.(\mathbb{T}(M) \cup \mathbb{F}(M))\\
\mathbb{T}((M\ N)) &= \mathbb{T}(M)@\Big(\mathbb{T}(N) \cup \mathbb{F}(N)\Big)\\
\mathbb{F}(x) &= \mathbb{F}'(x) = \emptyset\\
\mathbb{F}(M \neq x) &= \mathbb{F}'(M)\ \cup\ \nabla\big(\mathbb{F}'(M\{\widehat{z}/\texttt{FV}(M)\}) \cup \mathbb{T}(M\{\widehat{z}/\texttt{FV}(M)\})\big)[\texttt{FV}(M)/\widehat{z}]\\
\mathbb{F}'(M \neq x) &= \forall A_1,\dots,A_n,P,n>0 \text{ s.t. } M =_\alpha P\{A_1/y_1,\cdots,A_n/y_n\}\\
&\qquad \{y_1,\dots,y_{n+k}\} = \texttt{FV}(P)\ P \neq x\ \forall 1 \leq i \leq n\ A_i = (A_{i_1}\ A_{i_2})\\
&\quad \nabla\big(\mathbb{T}(P\{\widehat{z_1^k}/\widehat{y_{n+1}^{n+k}}\}) \cup (\mathbb{F}'(P\{\widehat{z_1^k}/\widehat{y_{n+1}^{n+k}}\})\big)\big[\mathbb{T}(A_1)/y_1,\cdots,\mathbb{T}(A_n)/y_n,\widehat{y_{n+1}^{n+k}}/\widehat{z_1^k}\big]
\end{aligned}
$$

Where $\widehat{z}$ and $\widehat{z_1^k}$ are fresh variables. $\widehat{z_1^k}/\widehat{y_{n+1}^{n+k}}$ stands for $z_1/y_{n+1},\cdots,z_k/y_{n+k}$, $\widehat{z}/\texttt{FV}(M)$ stands for the complete renaming of free variables of M width fresh ones, and $\texttt{FV}(M)/\widehat{z}$ stands for the inverse substitution.

Example 3. Let M be the λ-term $\lambda x.\lambda y.(x(x\,y))$. The set $\mathcal{C}(M)$ contains 24 elements and only six of them are typeable in EAL. They are:

- $\lambda x.[\lambda y.\nabla((x_3\,(x_4\,y_1)))[{}^{x_1}/_{x_3},{}^{x_2}/_{x_4},{}^{y}/_{y_1}]]_{x\to(x_1,x_2)}$ with $PT = \langle\emptyset, !^n(\alpha \multimap \alpha) \multimap !^n\alpha \multimap !^n\alpha, \emptyset\rangle$;
- $\lambda x.[\nabla(\lambda y.(x_3\,(x_4\,y)))[{}^{x_1}/_{x_3},{}^{x_2}/_{x_4}]]_{x\to(x_1,x_2)}$ with $PT = \langle\emptyset, !^n(\alpha \multimap \alpha) \multimap !^n(\alpha \multimap \alpha), \emptyset\}\rangle$;
- $\lambda x.[\nabla(\lambda y.\nabla((x_5\,(x_6\,y_1)))[{}^{x_3}/_{x_5},{}^{x_4}/_{x_6},{}^{y}/_{y_1}])[{}^{x_1}/_{x_3},{}^{x_2}/_{x_4}]]_{x\to(x_1,x_2)}$ with $PT = \langle\emptyset, !^{n_1}(\alpha \multimap \alpha) \multimap !^{n_2}(!^{n_3}\alpha \multimap !^{n_3}\alpha), \{n_1 = n_2 + n_3\}\rangle$;
- $\nabla(\lambda x.[\lambda y.\nabla((x_3\,(x_4\,y_1)))[{}^{x_1}/_{x_3},{}^{x_2}/_{x_4},{}^{y}/_{y_1}]]_{x\to(x_1,x_2)})[\,]$ with $PT = \langle\emptyset, !^m(!^n(\alpha \multimap \alpha) \multimap !^n\alpha \multimap !^n\alpha), \emptyset\rangle$;
- $\nabla(\lambda x.[\nabla(\lambda y.(x_3\,(x_4\,y)))[{}^{x_1}/_{x_3},{}^{x_2}/_{x_4}]]_{x\to(x_1,x_2)})[\,]$ with $PT = \langle\emptyset, !^m(!^n(\alpha \multimap \alpha) \multimap !^n(\alpha \multimap \alpha)), \emptyset\rangle$;
- $\nabla(\lambda x.[\nabla(\lambda y.\nabla((x_5\,(x_6\,y_1)))[{}^{x_3}/_{x_5},{}^{x_4}/_{x_6},{}^{y}/_{y_1}])[{}^{x_1}/_{x_3},{}^{x_2}/_{x_4}]]_{x\to(x_1,x_2)})[\,]$ with $PT = \langle\emptyset, !^m(!^{n_1}(\alpha \multimap \alpha) \multimap !^{n_2}(!^{n_3}\alpha \multimap !^{n_3}\alpha)), \{n_1 = n_2 + n_3\}\rangle$;

Fact 1. $\forall \mathbb{C} \in \mathcal{C}(M) \qquad (\mathbb{C})^* = M$

Theorem 2 (Soundness and Completeness of $\mathcal{C}$). $\forall M \in \Lambda$

1. $\mathcal{C}(M) \subseteq C(M)$;
2. $N \in C(M)$ *and* $\exists \Gamma, A$ *s.t.* $\Gamma \vdash_{abs} N : A \Rightarrow N \in \mathcal{C}(M)$.

Proof. See Appendix A.4.

Now we are able to prove the existence of principal typings for terms of λ-calculus.

Theorem 3 (Principal typing for Λ in EAL). *$\forall M \in \Lambda$, $\Gamma \vdash_{\mathsf{NEAL}} M : A$ if and only if $PT(N) =< \Theta, \sigma, C >$ and $\Gamma \supseteq < S, X > (\Theta)$, $A =< S, X > (\sigma)$, for some scheme substitution $< S, X >$ such that X satisfies C and for some $N \in \mathcal{C}(M)$.*

Proof. **(If)** Let be $PT(N) =< \Theta, \sigma, C >$, then by Theorem 1 $\Gamma \vdash_{abs} N : A$ and by Lemma 5 $\exists N' \in bme(N)$ $\Gamma \vdash_{\mathsf{NEAL}} N' : A$, hence the thesis.

(Only if) $\Gamma \vdash_{\mathsf{NEAL}} M : A$ then by definition $\exists R \in \Lambda^{EA}$ $\Gamma \vdash_{\mathsf{NEAL}} R : A$ and R is simple. Then $\Gamma \vdash_{abs} \mathscr{C}an(R) : A$ by Lemma 4. Moreover $\mathscr{C}an(R) \in C^{EA}$ by Lemma 9 and then, being $\mathscr{C}an(R)$ typeable, $\mathscr{C}an(R) \in CC^{EA}$ by Lemma 14. This is sufficient to prove that $\mathscr{C}an(R) \in \mathcal{C}(M)$ (by Lemma 4). The thesis holds by principal typing for Abs^{EA}. □

References

[1] Patrick Baillot. Checking polynomial time complexity with types. Linear Logic Workshop.

[2] Paolo Coppola and Simone Martini. Typing Lambda Terms in Elementary Logic with Linear Constraints. In Samson Abramsky, editor, *Proc. of Typed Lambda Calculi and Applications, 5th International Conference, TLCA 2001*, volume 2044 of *Lecture Notes in Computer Science*, pages 76–90. Springer, may 2001.

[3] H. B. Curry. Functionality in combinatory logic. In *Proc. Nat. Acad. Science USA*, volume 20, pages 584–590, 1934.

[4] H. B. Curry and R. Feys. *Combinatory Logic, Volume I.* Studies in Logic and the Foundations of Mathematics. North-Holland, Amsterdam, 1958.

[5] H. B. Curry, J. R. Hindley, and J. P. Seldin. *Combinatory Logic, Volume II.* Studies in Logic and the Foundations of Mathematics. North-Holland, Amsterdam, 1972.

[6] Vincent Danos, Jean-Baptiste Joinet, and Harold Schellinx. On the linear decoration of intuitionistic derivations. In *Archive for Mathematical Logic*, volume 33, pages 387–412, 1995.

[7] Jean-Yves Girard. Light linear logic. *Information and Computation*, 204(2):143–175, 1998.

[8] J. R. Hindley. The principal type-scheme of an object in combinatory logic. *Transactions of the American Mathematical Society*, 146:29–60, 1969.

[9] J. Lamping. An algorithm for optimal lambda calculus reduction. In ACM, editor, *POPL '90. Proceedings of the seventeenth annual ACM symposium on Principles of programming languages, January 17–19, 1990, San Francisco, CA*, pages 16–30, New York, NY, USA, 1990. ACM Press.

[10] P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *Proceedings of the 5th ACM Conference on Computer and Communications Security*, pages 112–121, San Francisco, California, United States, nov 1998. ACM Press.

[11] S. Ronchi Della Rocca. Principal Type Scheme and Unification for Intersection Type Discipline. *Theoretical Computer Science*, 59:181–209, 1988.

[12] S. Ronchi Della Rocca and B. Venneri. Principal Type Schemes for an Extended Type Theory. *Theoretical Computer Science*, 28:151–169, 1984.

[13] Harold Schellinx. *The Noble Art of Linear Decorating.* PhD thesis, Institute for Logic, Language and Computation, University of Amsterdam, 1994.

A Appendix

A.1 Canonical forms

Definition 8. *The set* C^{EA} *of* canonical forms *is generated by the following grammar (*C *is the starting symbol):*

$$
\begin{aligned}
C &::= S \mid NS \\
S &::= [LB]_{Slist} \text{ where } \mathtt{CV}(Slist) \subseteq \mathtt{FV}(LB) \\
&\quad \mid [S]_{Slist} \text{ where } \mathtt{CV}(Slist) \cap \mathtt{SSV}(S) = \emptyset \\
&\qquad \text{and } \mathtt{CV}(Slist) \subseteq \mathtt{FV}(S) \text{ and } \forall S_i \in \mathtt{ST}(S)\ \mathtt{CV}(Slist) \cap \mathtt{FV}(S_i) \neq \emptyset \\
Slist &::= NS \to (\widehat{x}) \mid NS \to (\widehat{x}), Slist \text{ where } |\widehat{x}| \geq 2 \\
NS &::= LB \mid x \\
LB &::= L \mid B \\
L &::= (NS\ NS) \mid \lambda x.NS \mid \lambda x.S \text{ where } S = [C]_{NS \to (\widehat{y})} \text{ and } x \in \mathtt{FV}(NS) \\
DSL &::= D \mid S \mid L \\
B &::= D \mid \nabla(DSL)[\widehat{x}/\widehat{y}] \text{ where } \mathtt{SBV}(\widehat{x}/\widehat{y}) \cap \mathtt{SSV}(DSL) = \emptyset \\
&\qquad \text{and } \mathtt{BV}(\widehat{x}/\widehat{y}) = \mathtt{FV}(DSL) \\
D &::= \nabla(DSL)[Blist] \text{ where } \mathtt{SBV}(Blist) \cap \mathtt{SSV}(DSL) = \emptyset \\
&\qquad \text{and } \mathtt{BV}(Blist) = \mathtt{FV}(DSL) \\
Blist &::= L/x \mid L/x, Blist \mid x/y, Blist
\end{aligned}
$$

all variables are linear, $\widehat{x}$ *stands for* $x_1, \ldots, x_k$ *and*

- *the set of* shared terms $\mathtt{ST}$*,*
- *the set of* banged variables $\mathtt{BV}$ *and*
- *the set of* single banged variables $\mathtt{SBV}$

are defined as follows:

$$
\begin{aligned}
\mathtt{ST}(NS) &= \emptyset & \mathtt{ST}([LB]_{Slist}) &= \mathtt{ST}(Slist) \\
\mathtt{ST}([S]_{Slist}) &= \mathtt{ST}(Slist) & \mathtt{ST}(NS \to (x_1, \ldots, x_n)) &= \{NS\}
\end{aligned}
$$

$$
\mathtt{ST}(NS \to (x_1, \ldots, x_n), Slist) = \{NS\} \cup \mathtt{ST}(Slist)
$$

$$
\begin{aligned}
\mathtt{BV}(M/x) &= \{x\} & \mathtt{BV}(M/x, Blist) &= \{x\} \cup \mathtt{BV}(Blist) \\
\mathtt{SBV}(y/x) &= \{x\} & \mathtt{SBV}(L/x) &= \emptyset \\
\mathtt{SBV}(y/x, Blist) &= \{x\} \cup \mathtt{SBV}(Blist) & \mathtt{SBV}(L/x, Blist) &= \mathtt{SBV}(Blist)
\end{aligned}
$$

A canonical form is a term in a meta-language, representing an infinite set of terms of Λ^{EA}.

Lemma 9 (Soundness of $\mathscr{C}an$). $\forall M \in \Lambda^{EA}\ \mathscr{C}an(M) \in C^{EA}$

Lemma 10. $\forall M \in C^{EA}$*,* M *is in* $\to_{\mathscr{C}an}$ *normal form.*

A.2 Simple Canonical Forms

Definition 9 (CC^{EA}). *The set of* canonical EAL-terms contracting at most variables CC^{EA} *is generated by the following grammar (CC is the starting symbol):*

$$\begin{aligned}
CC &::= [K]_{Clist} \text{ where } \mathtt{CV}(Clist) \subseteq \mathtt{FV}(K) \mid K \\
Clist &::= y \to (\widehat{x}) \mid y \to (\widehat{x}), Clist \text{ where } |\widehat{x}| \geq 2 \\
K &::= \nabla(B)[\widehat{x}/\widehat{y}] \text{ where } \mathtt{BV}(\widehat{x}/\widehat{y}) = \mathtt{FV}(B) \mid B \mid x \\
B &::= \nabla\,(B)\,[L] \text{ where } \mathtt{BV}(L) = \mathtt{FV}(B) \mid R \\
L &::= A/x \mid y/x, L \mid A/x, L \\
R &::= \lambda x.[K]_{x\to(x_1,\ldots,x_n)} \text{ where } \{x_1,\ldots,x_n\} \subseteq \mathtt{FV}(K) \mid \lambda x.K \mid A \\
A &::= (R\ K) \mid (x\ K)
\end{aligned}$$

where all variables are linear, $\widehat{x}$ *stands for* $x_1, \ldots, x_n$ *and* $n > 0$.

Notice that side condition $|\widehat{x}| \geq 2$ in the production of $Clist$ implies $[x]_{Clist}$ is not a possible term in CC^{EA} by side condition $\mathtt{CV}(Clist) \subseteq \mathtt{FV}(K)$ in production of CC and by $\{x_1, \ldots, x_n\} \subseteq \mathtt{FV}(K)$ in production of R.

Lemma 11.

$$CC^{EA} \subseteq C^{EA}$$

Fact 2. *Each term in* CC^{EA} *contracts at most variables.*

Lemma 12. *If* $M \in C^{EA}$ *is simple then* $\nexists N$ *subterm of* M *such that* N *has the form* $\big[[N']_{Slist_1}\big]_{Slist_2}$.

Lemma 13. *If* $M \in C^{EA}$ *is simple then* $\nexists N$ *subterm of* M *such that* N *has the form* $\nabla\big([N']_{Slist}\big)[Blist]$.

Lemma 14. $\forall M \in C^{EA}$, *M simple,*

$$\Gamma \vdash_{abs} M : A \Rightarrow M \in CC^{EA}.$$

A.3 Properties of the Canonical Forms Algorithm

Lemma 15. *Let* $\mathscr{L}_X$ *be the language generated by the grammar of Definition 9 with starting element X, for* $X \in \{R, A, B, K\}$*, then the following hold:*

1. $\mathbb{T}(M) \subseteq \mathscr{L}_R \cup \{x\}$
   - 1.1) $\mathbb{T}(M \neq x) \subseteq \mathscr{L}_R$
   - 1.2) $\mathbb{T}((M_1\ M_2)) \subseteq \mathscr{L}_A$
2. $\mathbb{F}'(M) \subseteq \mathscr{L}_B$
3. $\mathbb{F}(M) \subseteq \mathscr{L}_k$

Lemma 16. *Let $\mathscr{L}_{CC}$ the language of canonical forms generated by the grammar of Definition 9, then*

$$\forall M \in \Lambda \quad \mathcal{C}(M) \subseteq \mathscr{L}_{CC}$$

Lemma 17. *1.* $M \in \mathscr{L}_R \cup \{x\} \Rightarrow M \in \mathbb{T}((M)^*)$
2. $M \in \mathscr{L}_A \Rightarrow (M)^* = (M_1\ M_2)\ \wedge\ M \in \mathbb{T}((M_1\ M_2))$
3. $M \in \mathscr{L}_B \Rightarrow M \in \mathbb{F}'((M)^*) \cup \mathbb{T}((M)^*)$
4. $M \in \mathscr{L}_K \Rightarrow M \in \mathbb{F}((M)^*) \cup \mathbb{T}((M)^*)$

Lemma 18.

$$M \in \mathscr{L}_{CC} \Rightarrow M \in \mathcal{C}((M)^*)$$

Theorem 4.

$$\bigcup_{M \in \Lambda} \mathcal{C}(M) = \mathscr{L}_{CC}$$

A.4 Proof of Theorem 2

Theorem 2 $\forall M \in \Lambda$

1. $\mathcal{C}(M) \subseteq C(M)$;
2. $N \in C(M)$ and $\exists \Gamma, A$ s.t. $\Gamma \vdash_{abs} N : A \Rightarrow N \in \mathcal{C}(M)$.

Proof. We recall the definition of $C(M)$:

$$C(M) = \{N \mid \exists R \in \Lambda^{EA} \text{ s.t. } (R)^* = M, R \text{ is simple and } N = \mathscr{C}an(R)\}$$

1. By Lemma 16 for any $\mathbb{C} \in \mathcal{C}(M)$ we have $\mathbb{C} \in CC^{EA}$ and hence, by Lemma 11, $\mathbb{C} \in C^{EA}$. Moreover $(\mathbb{C})^* = M$ by Fact 1. Then $\mathbb{C}$ is in $C(M)$ because it exists $R \in bme(\mathbb{C})$ s.t. $(R)^* = M$, R is simple and it is sufficient to notice that $emb(bme(\mathbb{C}))$ is either equal to $\mathbb{C}$ (that is in $\rightarrow_{\mathscr{C}an}$ normal form) or there are a set of $\rightarrow_{\mathscr{C}an}$ redexes after firing them we get $\mathbb{C}$ again.
2. By Lemma 9 $\mathscr{C}an(R) = N \in C^{EA}$. R is simple by hypothesis and then N is simple too. Moreover $\Gamma \vdash_{abs} N : A$, hence, by Lemma 14, $N \in CC^{EA}$ and then $N \in \mathcal{C}(M)$ by Lemma 18. □