

# Optimizing Optimal Reduction: A Type Inference Algorithm for Elementary Affine Logic

Paolo Coppola

Università di Udine — Dip. di Matematica e Informatica

Simone Martini

Università di Bologna — Dip. di Scienze dell'Informazione

April 23, 2004

## Abstract

We propose a type inference algorithm for lambda terms in Elementary Affine Logic (EAL). The algorithm decorates the syntax tree of a simple typed lambda term and collects a set of linear constraints. The result is a parametric elementary type that can be instantiated with any solution of the set of collected constraints.

We point out that the typeability of lambda terms in EAL has a practical counterpart, since it is possible to reduce any EAL-typeable lambda terms with the Lamping's abstract algorithm obtaining a substantial increasing of performances.

We show how to apply the same techniques to obtain decorations of intuitionistic proofs into Linear Logic proofs.

## Introduction

The optimal reduction of  $\lambda$ -terms ([Lév80]; see [AG98] 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 (*fan*) expressing the sharing of two common subterms (these subterms are always variables in the initial translation of a  $\lambda$ -term). Giving correct reduction rules for these *sharing graphs* is a surprisingly difficult problem, first solved in [Kat90, Lam90]. 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 trivial, namely those  $\lambda$ -terms whose sharing graph is a proof-net of Elementary Logic, both in the Linear [Gir98] (ELL) and the Affine [Asp98] (EAL) flavor. This fact was first

observed in [Asp98] and then exploited in [ACM00] to obtain a certain complexity result on optimal reduction, where (following [Mai92]) we also showed that these *EAL-typed*  $\lambda$ -terms are powerful enough to encode arbitrary computations of elementary time-bounded Turing machines. We did not know, however, of any systematic way to derive EAL-types for  $\lambda$ -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.

The main contribution of the paper is a type inference algorithm (Section 2), assigning propositional EAL-types (formulas) to *type-free*  $\lambda$ -terms (more precisely: to sharing graphs corresponding to type-free  $\lambda$ -terms). We will see in Section 1 that a typing inference for a  $\lambda$ -term  $M$  in EAL consists of a *skeleton* – given by the assignment of a type to  $M$  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  $M$ . Correctness and completeness of the algorithm are proved with respect to a natural deduction system for EAL, introduced in Section 3.1 together with terms annotating the derivations. We remark that we consider only the propositional fragment of EAL; second order is necessary to make EAL complete for elementary time [DJ03].

The technique used in the paper, with minor modifications, can be used to obtain linear logic derivations as decorations of intuitionistic derivations, subsuming some of the results of [DJS95, Sch94]. In this way we may obtain linear derivations with a minimal number of boxes. We tackle this issue in Section 4.1.

A preliminary version of this work has already been published [CM01]. Besides giving more elaborated examples and technical details, several results are new. We prove that all EAL types can be obtained by applying the algorithm on the simple principal type schema; as a corollary, we may state the decidability of the type inference problem for EAL. We show how to use our technique to decorate full linear logic proofs. We show how the algorithm could be extended to allow arbitrary contractions.

In [CR03], the existence of a notion of principal type schema for EAL is investigated and established. Baillot [Bai02] gives a type-inference algorithm for Light Affine Logic, but it applies only to lambda terms in normal form. This restriction is eliminated in [Bai03], where the same author gives a LAL type-inference algorithm for pure lambda-terms, proving also the decidability of the LAL type inference problem for lambda-calculus, using the approach proposed in [CR03].

## 1 Elementary Affine Logic

Elementary Affine Logic [Asp98] is a system with unrestricted weakening, where contraction is allowed only for modal formulas. There is only one *exponential*

$$\boxed{
\begin{array}{c}
\frac{}{x : A \vdash x : A} \text{ ax} \qquad \frac{\Gamma \vdash N : A \quad x : A, \Delta \vdash M : B}{\Gamma, \Delta \vdash M\{N/x\} : B} \text{ cut} \\
\\
\frac{\Gamma \vdash M : B}{\Gamma, x : A \vdash M : B} \text{ weak} \qquad \frac{\Gamma, x_1 : !A, x_2 : !A \vdash M : B}{\Gamma, z : !A \vdash M\{z/x_1, z/x_2\} : B} \text{ contr} \\
\\
\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \multimap B} \multimap R \qquad \frac{\Gamma \vdash N : A \quad x : B, \Delta \vdash M : C}{\Gamma, f : A \multimap B, \Delta \vdash M\{(f N)/x\} : C} \multimap L \\
\\
\frac{x_1 : A_1, \dots, x_n : A_n \vdash M : B}{x_1 : !A_1, \dots, x_n : !A_n \vdash M : !B} !
\end{array}
}$$

Figure 1: (Implicational) Elementary Affine Logic

rule for the modality ! (*of-course*, or *bang*), which is introduced at once on both sides of the turnstile. The propositional fragment of the system is presented in Figure 1, where also  $\lambda$ -terms are added to the rules. We denote with  $M\{N/x\}$  the usual meta-notion of substitution of  $N$  for the free occurrences of  $x$  in  $M$ . In the contexts (or bases)  $(\Gamma, \Delta, \text{etc.})$  a variable can occur only once (they are *linear*). Observe that—according to most literature on optimal reduction but unlike Barendregt’s book—we always write parenthesis around an application and we assume that the scope of a  $\lambda$  is the minimal subterm following the dot; as a consequence, a term like  $(\lambda x.M N)$  should be parsed as  $((\lambda x.M)N)$ . Cut-elimination may be proved for EAL in a standard way.

Given the sharing graph of a type-free  $\lambda$ -term, we are interested in finding a derivation of a type for it, according to Figure 1. (There is a subtle point in this notion, which is relevant for the completeness of our algorithm; we will discuss this issue at the end of this section. For the time being we may remain informal).

A simple inspection of the rules of EAL shows that any  $\lambda$ -term with an EAL type has also a simple type. 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. Therefore, in looking for an EAL-type for a  $\lambda$ -term  $M$ , we can start from a simple type derivation for  $M$  and try to decorate this derivation (*i.e.*, add !-rules) to turn it into an EAL-derivation. Our algorithm implements this simple idea:

1. we find all “maximal decorations”;
2. these decorations correspond to well formed derivations only if certain linear constraints admit (integral) solutions.

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



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

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

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

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

While simple and appealing, the technique of maximal decoration cannot be applied directly. The first problem is that sequent derivations are too constrained. There are many different (simple type) derivations for the same  $\lambda$ -term, depending on the position of ( $\multimap L$ ) rules, contractions, cuts, etc. Given a  $\lambda$ -term, we should therefore produce all possible derivations, and then decorate them. The problem stems from the fact that sequent derivations are not driven by the syntax of the term. In fact, the standard simple type inference algorithm does *not* use a sequent-style presentation, but a natural deduction one, which is naturally syntax-driven. This is the solution we also follow in this paper—we decorate the  $\lambda$ -term. Unfortunately, it is well known (see Prawitz’s classical essay [Pra65]) that natural deduction for modal systems behave badly, since the obvious formulation for the modal rule (the one coinciding with rule  $!$  of the sequent presentation) does not enjoy a substitution lemma. As a result, there are EAL type inferences which cannot be obtained directly as decoration of simple type derivations *in natural deduction*. Consider, for instance, the following simple type derivation (in the obvious natural deduction presentation of implicational logic) for  $M = \lambda x y k. (x y) : (A \rightarrow B) \rightarrow A \rightarrow (C \rightarrow B)$ :

$$\frac{\frac{\frac{x : A \rightarrow B \vdash x : A \rightarrow B \quad y : A, k : C \vdash y : A}{x : A \rightarrow B, y : A, k : C \vdash (x y) : B}}{x : A \rightarrow B, y : A \vdash \lambda k. (x y) : C \rightarrow B}}{x : A \rightarrow B \vdash \lambda y k. (x y) : A \rightarrow (C \rightarrow B)}{\vdash \lambda x y k. (x y) : (A \rightarrow B) \rightarrow A \rightarrow (C \rightarrow B)}$$

It is not difficult to see that in the system of Figure 1 there is a derivation establishing  $\vdash M : (A \multimap !B) \multimap A \multimap !(C \multimap B)$ . But no interleaving of  $!$  rules into the derivation above can give this conclusion.

Indeed, to guarantee a substitution lemma, the modal rule for EAL in natural deduction must be formulated:

$$\frac{\Delta_1 \vdash !A_1 \quad \dots \quad \Delta_n \vdash !A_n \quad A_1, \dots, A_n \vdash B}{\Delta_1, \dots, \Delta_n, \vdash !B} \text{ box}$$

This rule, given a derivation of  $A_1, \dots, A_n \vdash B$  (*i.e.*, a  $\lambda$ -term  $M$  with the assignment of the type  $B$  from the basis  $A_1, \dots, A_n$ ): (i) “builds a box” around  $M$ ; (ii) allows the substitution of arbitrary terms for the free variables of  $M$ .

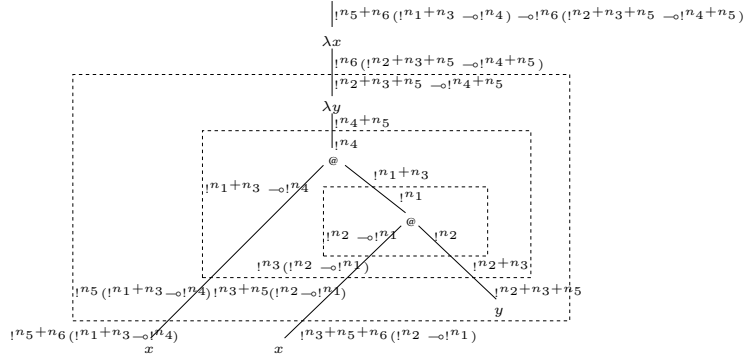


Figure 2: Meta EAL type derivation of *two*.

Our algorithm will start from a simple type derivation in natural deduction for a term  $M$  (*i.e.*, the syntax tree of the term decorated with simple types) and will try to insert (all possible) boxes around (suitable) subterms. We will sometimes use a graphical representation of this process. As an example, Figure 2 shows the decoration of the syntax tree of *two* we obtained in Section 1.

We are finally in the position to introduce formally the notion of EAL-typing for  $\lambda$ -terms. Recall that our main goal is to mechanically check whether a pure  $\lambda$ -term could be optimally reduced without the need of the oracle. While we lack a general characterization of this class of terms, we know that it contains any sharing graph coding the skeleton of a sequent proof in EAL. We already observed, however, that a single  $\lambda$ -term may correspond to more than one (sequent or natural deduction) proof. The position of the contraction is especially relevant in this context. Indeed, consider the term  $M = \lambda z x w.((x z) (x z) w)$ . Among the (infinite) EAL sequent derivations having  $M$  as a skeleton consider the following two fragments:

$$\begin{array}{c}
 \vdots \\
 \hline
 z_1 : a, z_2 : a, x_1 : a \multimap (b \multimap b), x_2 : a \multimap (b \multimap b), w : b \vdash ((x_1 z_1) ((x_2 z_2) w)) : b \\
 \hline
 z_1 : !a, z_2 : !a, x_1 : !(a \multimap (b \multimap b)), x_2 : !(a \multimap (b \multimap b)), w : !b \vdash ((x_1 z_1) ((x_2 z_2) w)) : !b \quad ! \\
 \hline
 \hline
 z : !a, x : !(a \multimap (b \multimap b)), w : !b \vdash ((x z) ((x z) w)) : !b \\
 \hline
 z : !a, x : !(a \multimap (b \multimap b)) \vdash \lambda w.((x z) ((x z) w)) : (!b \multimap !b) \multimap R \\
 \hline
 \end{array} \quad \text{contr, contr} \quad (1)$$

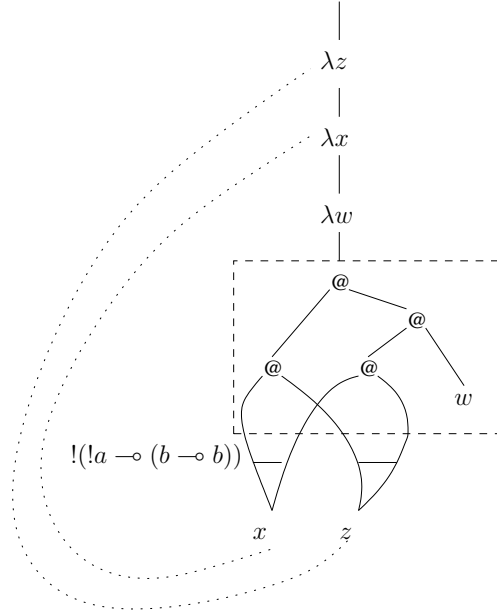


Figure 3: One decoration of  $\lambda z x w.((x z) ((x z) w))$ : the fan faces a lambda.

and

$$\frac{
 \frac{
 \frac{
 \vdots
 }{
 k_1, k_2 : b \multimap b \vdash \lambda w.(k_1 (k_2 w)) : b \multimap b
 }
 }{
 k_1, k_2 :!(b \multimap b) \vdash \lambda w.(k_1 (k_2 w)) :!(b \multimap b)
 }
 \text{!}
 }{
 z : a \vdash z : a \quad k :!(b \multimap b) \vdash \lambda w.(k (k w)) :!(b \multimap b)
 }
 \text{contr}
 }{
 z : a, x : a \multimap!(b \multimap b) \vdash \lambda w.((x z) ((x z) w)) :!(b \multimap b)
 }
 \multimap L
 \quad (2)$$

If we display these derivations as annotated syntax tree with explicit fan nodes for contraction (that is, as sharing graphs), we obtain Figure 3 for the derivation (1), and Figure 4 for (2).

Both graphs are legal EAL sharing graphs, but only the first is a possible initial translation of  $M$  as a sharing graph, since in initial translations the fan nodes are used to share (contract) only variables, before abstracting them. Although our technique could be extended to cope with arbitrary contractions (see Section 4), we present it as a type inference algorithm for initial translations of type-free  $\lambda$ -terms, according to our original aim to use it as a tool in an optimal reducer. This is the motivation for the following notion.

**Definition 1.** A type-free  $\lambda$ -term  $M$  has EAL type  $A$  from the basis  $\Gamma$  (write:  $\Gamma \vdash_{\text{EAL}} M : A$ ) iff there is a derivation of  $\Gamma \vdash M : A$  in the system of Figure 1 whose corresponding sharing graph does not have any fan node facing an application or an abstraction node.

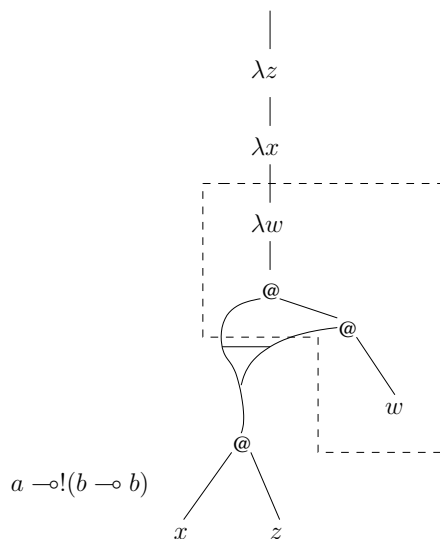


Figure 4: Another decoration of  $\lambda z x w.((x z) ((x z) w))$ : the fan faces an application.

**Remark 1.** *It is possible to formulate the previous definition directly in terms of sequent derivations, without any reference to the notion of sharing graph. It could be proved that  $\Gamma \vdash_{\text{EAL}} M : A$  iff there is a sequent derivation of  $\Gamma \vdash M : A$  where all contractions either are immediately followed by  $\multimap R$ , or are at the end of the derivation. However, the “only if” part is not trivial. In going from a sequent derivation to a sharing graph, in fact, we lose any information regarding the position of cuts and (to some extent) of  $\multimap L$ . Therefore, given a term  $M$  for which  $\Gamma \vdash_{\text{EAL}} M : A$  (that is, given a sharing graph that could be decorated with EAL-types and boxes) there are many sequent derivations corresponding to the skeleton coded by this sharing graph. Not all these derivations satisfy the constraint expressed by the “only if” part. It can be shown, however, that among these derivations there is one in which the constraint is satisfied. This could be obtained by using the notion of canonical form of an EAL derivation, introduced and exploited in [CR03].*

**Remark 2.** *There exist simply typeable terms without any EAL type. For instance the  $\lambda$ -term*

$$(\lambda n.(n \lambda y.(n \lambda z.y)) \lambda x.(x (x y)))$$

*has a simple type, but no EAL decoration (see Appendix A for an analysis).*

$$\begin{array}{c}
\frac{z : I_o \vdash z : I_o}{\vdash \lambda z.z : I_o} \\
\frac{n : I_o \rightarrow I_o \vdash n : I_o \rightarrow I_o \quad \frac{z : I_o \vdash z : I_o}{\vdash \lambda z.z : I_o}}{n : I_o \rightarrow I_o \vdash (n \lambda z.z) : I_o} \quad y : o \vdash y : o \\
\frac{n : I_o \rightarrow I_o, y : o \vdash ((n \lambda z.z) y) : o}{n : I_o \rightarrow I_o \vdash \lambda y.((n \lambda z.z) y) : I_o} \quad \frac{x : I_o \vdash x : I_o \quad \frac{w : o \vdash w : o}{\vdash \lambda w.w : I_o}}{x : I_o \vdash (x \lambda w.w) : I_o} \\
\frac{n : I_o \rightarrow I_o \vdash \lambda y.((n \lambda z.z) y) : I_o}{\vdash \lambda n.\lambda y.((n \lambda z.z) y) : (I_o \rightarrow I_o) \rightarrow I_o} \quad \frac{x : I_o \vdash (x \lambda w.w) : I_o}{\vdash \lambda x.(x \lambda w.w) : I_o \rightarrow I_o} \\
\hline
\vdash (\lambda n.\lambda y.((n \lambda z.z) y) \lambda x.(x \lambda w.w)) : o \rightarrow o
\end{array}$$

Figure 5: Simple type derivation of a term

## 2 Type inference

The type inference algorithm is given as a set of inference rules, specifying several functions. The complete set of rules is given in Section 2.2; the properties of the algorithm will be stated and proved in Section 3. We start in the next section with an example, where we discuss some of the main issues the algorithm deals with.

### 2.1 An example of 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.

**Definition 2.** A general EAL-type  $\Theta$  is generated by the following grammar:

$$\Theta ::= !^{n_1 + \dots + n_k} o \mid !^{n_1 + \dots + n_k} (\Theta \multimap \Theta)$$

where  $k \geq 0$  and  $n_1, \dots, n_k$  are variables ranging on  $\mathbb{N}$ .

We shall illustrate our algorithm on the term  $(\lambda n.\lambda y.((n \lambda z.z) y) \lambda x.(x \lambda w.w)) : o \rightarrow o$ , whose *simple* type derivation in natural deduction is given in Figure 5 ( $I_\alpha$  stands for  $\alpha \rightarrow \alpha$ ). We suggest to read this section in parallel with Section 2.2, where the official rules of the algorithm are given.

We will work out the example following a leftmost innermost strategy (the algorithm is correct under any bottom-up strategy). We localize the first subterm for which there is no assignment of an EAL-type yet and we build for it the “most decorated” type, adding boxes in all possible points. We obtain in this way a general type representing all possible decorations<sup>1</sup>. In this case, the first subterm to be dealt with is the variable  $n : (((o \rightarrow o) \rightarrow (o \rightarrow o)) \rightarrow (o \rightarrow o))$ . The core of the algorithm is the type synthesis function,  $\mathcal{S}$ , see Section 2.2.5. Given a term  $M$  of simple type  $\sigma$ ,  $\mathcal{S}(M : \sigma)$  returns a quadruple:

$\langle$ general EAL-type, base<sup>2</sup>  $\{x_i : \Theta_i\}_i$  of pairs (variable:general EAL-type), set of linear constraints, critical points<sup>3</sup> $\rangle$ .

<sup>1</sup>More precisely we obtain all possible decorations without exponential cuts and with some other properties listed in Theorem 3. Such decorations are sufficient for the completeness of the algorithm.

<sup>2</sup>A base here is a *multiset* where multiple copies of  $x : \Theta$  may be present.

<sup>3</sup>We will discuss critical points in a moment.

In the case of a variable, the rules for  $\mathcal{S}$  simply call the auxiliary function  $\mathcal{P}$  (rules (24) and (25)) to add modalities whenever possible, obtaining:

$$n : !^{p_1} (!^{p_2} (!^{p_3} (!^{p_4} o \multimap !^{p_5} o) \multimap !^{p_6} (!^{p_7} o \multimap !^{p_8} o)) \multimap !^{p_9} (!^{p_{10}} o \multimap !^{p_{11}} o)) \quad (3)$$

for any  $p_i \in \mathbb{N}, 1 \leq i \leq 11$ ; no constraints and no critical points are generated. In the following we will drop the “ $\in \mathbb{N}$ ” for any variable we introduce, as this constraint is implied by Definition 2.

**Notation 1.** We write  $(n \multimap m)$  instead of  $(!^n o \multimap !^m o)$  and  $n + m(k \multimap d)$  instead of  $!^n !^m (!^k o \multimap !^d o)$ , for a better reading.

Analogously,  $z : (o \rightarrow o)$  is typed  $z : p_{12}(p_{13} \multimap p_{14})$ . Next subterms to be handled are  $\lambda z.z$  (by rule (33)) and  $(n \lambda z.z) : (o \rightarrow o)$  (by rule (36)). No critical points are present. Therefore, the two rules simply add all possible boxes around the body of the lambda-abstraction (the first rule) and around the argument of the application (the second rule). The rule for application adds the constraint that the type of the term in functional position ( $n$  in this case) be indeed an arrow type—this is why the rule does not box also the function part—and calls then the unification algorithm  $\mathcal{U}$  (Section 2.2.1). Function  $\mathcal{U}$  generates constraints ensuring that the type of the argument matches with the argument part of the function. Observe that the implicational structure of the types is already correct, since we start from a simple type derivation. Therefore, unification only produces a set of constraints on the variables used to indicate boxes. In our example, we get:

$$\boxed{\begin{cases} p_1 = 0 \\ b_1 = p_2 \\ p_3 = p_6 = p_{12} \\ p_4 = p_7 = p_{13} \\ p_5 = p_8 = p_{14}. \end{cases}} \quad (4)$$

The resulting typing is

$$n : b_1(p_3(p_4 \multimap p_5) \multimap p_3(p_4 \multimap p_5)) \multimap p_9(p_{10} \multimap p_{11}) \vdash (n \lambda z.z) : p_9(p_{10} \multimap p_{11}), \quad (5)$$

and the corresponding decoration is shown in Figure 6.

Next step is the inference of a general EAL-type  $p_{15}$  for  $y : o$ . Then the algorithm starts to process  $((n \lambda z.z) y) : o$ . Being an application, we apply the same process we discussed before, obtaining the constraints

$$p_9 = 0 \quad \text{and} \quad \boxed{p_{10} = p_{15}}. \quad (6)$$

However, the present case is more delicate than the application we treated before, since the function part is already an application. Two consecutive applications in  $((n \lambda z.z) y)$  indicate that more than one decoration is possible. Indeed, there can be several derivations building the same term, that can be differently

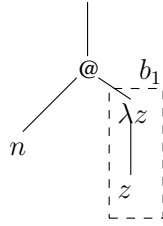


Figure 6: Decoration of  $(n \lambda z.z)$ .

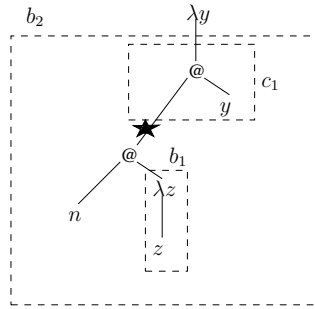


Figure 7: Critical point in the decoration of  $\lambda y.((n \lambda z.z)y)$ .

decorated. The issue is better appreciated if we look ahead for a moment and we consider the term  $\lambda y.((n \lambda z.z) y)$ . There are two (simple) sequent derivations for this term, both starting with the term  $(x y) : o$ , for  $x : o \rightarrow o, y : o$ . The first derivation, via a left  $\rightarrow$ -rule, obtains  $((n \lambda z.z) y) : o$ ; then it bounds  $y$ , giving  $\lambda y.((n \lambda z.z) y) : o \rightarrow (o \rightarrow o)$ . The second derivation permutes the rules: it starts by binding  $y$ , obtaining  $\lambda y.(x y)$  and only at this point substitutes  $(n \lambda z.z)$  for  $x$ , via the left  $\rightarrow$ -rule. When we add boxes to the two derivations, we see this is a *critical* situation. Indeed, in the first derivation we may box  $(x y)$ , then  $((n \lambda z.z) y)$  and finally  $\lambda y.((n \lambda z.z) y)$ . In the second, we box  $(x y)$ , then  $\lambda y.(x y)$  and finally the whole term. The two (incompatible) decorations are depicted in the two bottom trees of Figure 10. The critical edge—where the boxing radically differs—is the root of the subtree for  $((n \lambda z.z) y)$ , corresponding to the  $x$  that is substituted for in the left  $\rightarrow$ -rule. We will see in Lemma 6 and Lemma 8 that such critical points in the syntax tree are necessarily above applications nodes.

Let us then resume the discussion of the type inference for this term. At this stage we collect the *critical point*, marked with a star in Figure 7, indicating the presence of two possible derivations. When, in the future, it will be possible to add boxes, for example  $b_2$  in Figure 7 during the type inference of  $\lambda y.((n \lambda z.z) y)$ , the algorithm will consider the critical point as one of the closing points of such boxes,  $c_1$  in Figure 7, eventually modifying the constraint

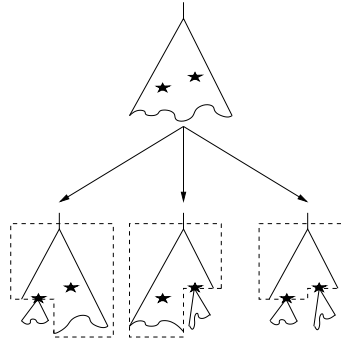


Figure 8: Combinations of two critical points.

in Equation (6) that imposes the type of  $(n \lambda z.z)$  to be functional and not exponential. Indeed, for completeness, the algorithm must take into account all possible derivations. When there will be more than one critical point, at every stage of the type inference, when it is possible to apply a  $!$  rule, the algorithm will compute all possible combinations of the critical points (see Figure 8, showing a schematic example with two critical points) eventually modifying some constraints. This is the role of functions  $\mathcal{B}$  and  $\mathbb{B}$ , see Section 2.2.3. We call *slices*<sup>4</sup> such combinations of critical points; they are the data maintained by the algorithm and indicated in the rules as *sls*. The task of combining the two lists of slices collected during the type inference of the function and argument part of an application is performed by  $\Psi$ , whose rules are given in Section 2.2.4.

**Definition 3.** *The list of free variable occurrences of a lambda term  $M$  is defined in the obvious way:  $\mathbf{FVO}(x) = [x]$ ;  $\mathbf{FVO}(\lambda x.M) = \mathbf{FVO}(M) - x$ ;  $\mathbf{FVO}((M_1 M_2)) = \mathbf{FVO}(M_1) :: \mathbf{FVO}(M_2)$  (the concatenation of lists).*

**Definition 4.** *A critical point is a pair (constraint, list of free variable occurrences).*

*A critical point*<sup>5</sup>  $(A^j, lv)$  is associated to an edge in the syntax tree of the term;  $lv$  are the free variable occurrences of the subterm rooted at this edge; this subterm is always an application (see Lemma 6 and Lemma 8).

*A slice is a set of critical points as in the following:*

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

*A slice corresponds to a combination of critical points.*

In our example the algorithm collects the slice  $\{(p_9 = 0, [n])\}$ .

The following property will be satisfied: a slice partitions the set of free variable occurrences in a derivation. Indeed, it marks the set of variable occurrences

<sup>4</sup>We thank Philippe Dague for useful discussions and suggestions on the calculation of critical points.

<sup>5</sup> $A^j$  means the  $j$ -th row of the matrix  $A$ , i.e., the  $j$ -th constraint.

whose types should not be modified when the box is added. This is the intuitive meaning of the set of free variable occurrences in the data structure we use.

**Notation 2.** •  $sl(x)$  indicates a slice having an occurrence of  $x$  in the list of variables of one of its critical points.

- $x \in sl$  if and only if there exists one critical point of  $sl$  whose list of variable occurrences contains the particular occurrence  $x$ .
- $A^j \in sl$  if and only if there exists one critical point of  $sl$  whose constraint is  $A^j$ .
- Let  $A^j$  be the constraint  $\pm n_{j_1} \pm \dots \pm n_{j_k} = 0$ ,  $A^j - n$  corresponds to the constraint  $\pm n_{j_1} \pm \dots \pm n_{j_k} - n = 0$ .

In the example, the application of the type-inference rule (38) gives thus:

$$\left\{ \begin{array}{l} n : b_1(p_3(p_4 \multimap p_5) \multimap p_3(p_4 \multimap p_5)) \\ \phantom{n} \phantom{b_1} \phantom{(p_3(p_4 \multimap p_5) \multimap p_3(p_4 \multimap p_5))} \multimap p_9(p_{10} \multimap p_{11}), \\ y : p_{10} \end{array} \right\} \vdash ((n \lambda z.z) y) : p_{11} \quad (7)$$

and slices of critical points  $sls = \{(p_9 = 0, [n])\}$ .

Typing  $\lambda y.((n \lambda z.z) y) : o \rightarrow o$  involves rule (33), the same we used for  $\lambda z.z$ , but now the boxing procedure (Section 2.2.3) is called on a subterm that is not a single variable, and, more important, with a non-empty set of slices. In this case, for any slice, rule (27) adds boxes “inside” the term—modifying the basis and the constraints—as it is schematically shown in Figure 8.

Therefore, rule (33) gives in our case:

$$\mathcal{S}(\lambda y.((n \lambda z.z) y) : o \rightarrow o) = \left\langle \begin{array}{l} b_2 + c_1 + p_{10} \multimap b_2 + c_1 + p_{11}, \\ \{n : b_2(b_1(p_3(p_4 \multimap p_5) \multimap p_3(p_4 \multimap p_5)) \multimap p_9(p_{10} \multimap p_{11}))\}, \\ \left\{ \begin{array}{l} \vdots \\ p_9 - c_1 = 0 \quad , \\ \vdots \end{array} \right. \\ \{(p_9 - c_1 = 0, [n])\} \end{array} \right\rangle \quad (8)$$

where  $p_9 - c_1 = 0$  is the unique constraint (Equation (6)) modified by the boxing. The decoration obtained is shown in Figure 7. Observe that, at this stage, the presence of incompatible derivations does not show up yet. It will be taken into account as soon as we will try to box a superterm of the one we just processed. If  $\lambda y.((n \lambda z.z) y)$  were the whole term, on the contrary, an additional call to the function  $\mathbb{B}$  would be performed, see rule (40) for function  $\mathcal{S}$ .

When the algorithm processes  $\lambda n.\lambda y.((n \lambda z.z) y) : ((o \rightarrow o) \rightarrow (o \rightarrow o)) \rightarrow (o \rightarrow o) \rightarrow (o \rightarrow o)$  it applies again rule (33). It adds  $c_2$  boxes passing through

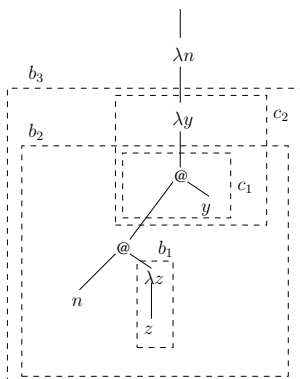


Figure 9:

the critical point and  $b_3$  boxes around the body of the function, obtaining:

$$\begin{aligned}
 & \mathcal{S}(\lambda n.\lambda y.((n \lambda z.z) y) : (((o \rightarrow o) \rightarrow (o \rightarrow o)) \rightarrow (o \rightarrow o)) \rightarrow (o \rightarrow o)) = \\
 & \quad b_3 + b_2(b_1(p_3(p_4 \multimap p_5) \multimap p_3(p_4 \multimap p_5)) \multimap p_9(p_{10} \multimap p_{11})) \\
 & \quad \quad \quad \multimap b_3 + c_2(b_2 + c_1 + p_{10} \multimap b_2 + c_1 + p_{11}) \quad , \\
 & \left\langle \begin{array}{c} \emptyset, \\ \vdots \\ \boxed{p_9 - c_1 - c_2 = 0} \\ \vdots \\ \emptyset \end{array} \right\rangle \quad (9)
 \end{aligned}$$

where  $p_9 - c_1 - c_2 = 0$  is the unique constraints modified at this stage of the type synthesis.

The slice  $\{(p_9 - c_2 - c_2 = 0, [n])\}$  is removed. Indeed, to bound  $n$ , the substitution of  $n(\lambda z.z)$  for  $x$  has to be already performed. It does not make sense to derive first  $\lambda n.\lambda y.(x y)$ , add boxes, and then substitute  $n(\lambda z.z)$  for  $x$ , since this would be a free-variable catching substitution.

Figure 9 shows the decoration obtained. Notice that the boxes  $c_2$  and  $b_2$  belong to the two *incompatible EAL-derivations* we already discussed before. The algorithm maintains at the same time these derivations guaranteeing (see Lemma 10) that if the final solution instantiates two incompatible derivations, we can always calculate an equivalent EAL-derivation (Figure 10 shows the two possible derivations for our example).

Going on with the type synthesis, the algorithm starts processing the left-most occurrence of  $x$  in  $(x (x \lambda w.w))$ . We use superscripts (1) and (2) to discriminate the right and left occurrence, respectively. The inference proceeds along the lines already discussed, obtaining  $x^{(2)} : p_{16}(p_{17}(p_{18} \multimap p_{19}) \multimap p_{20}(p_{21} \multimap p_{22}))$ ,  $x^{(1)} : p_{23}(p_{24}(p_{25} \multimap p_{26}) \multimap p_{27}(p_{28} \multimap p_{29}))$ , and  $\lambda w.w : p_{30} \multimap p_{30}$ . The

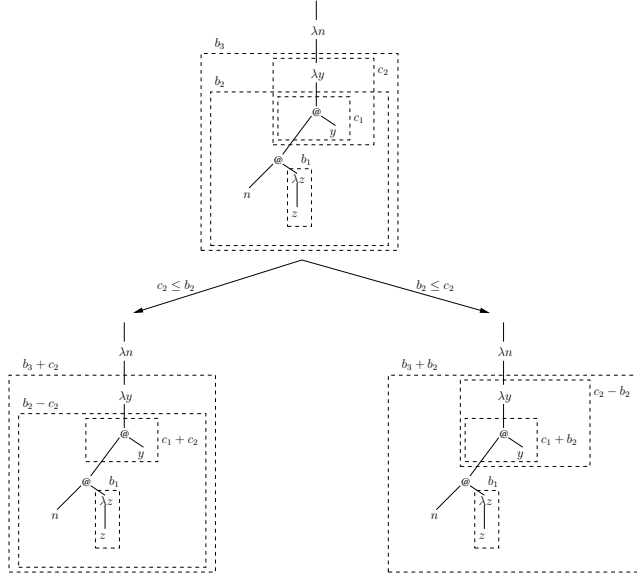


Figure 10: Superimposed derivations.

innermost application ( $x^{(1)} \lambda w.w$ ) is then typed  $p_{27}(p_{28} \multimap p_{29})$ , once we have boxed  $\lambda w.w$  with  $b_4$  boxes and we have collected the constraints

$$\left\{ \begin{array}{l} p_{23} = 0 \\ b_4 = p_{24} \\ p_{25} = p_{26} = p_{30}. \end{array} \right. \quad (10)$$

When the algorithm processes ( $x^{(2)} (x^{(1)} \lambda w.w)$ ), the presence of two consecutive applications reveals a new critical point. Usual boxing ( $b_5$  boxes around the argument) and unification produce the constraints

$$\left\{ \begin{array}{l} p_{16} = 0 \\ p_{17} = b_5 + p_{27} \\ p_{18} = p_{28} \\ p_{19} = p_{29}. \end{array} \right. \quad (11)$$

obtaining the derivation

$$\left\{ \begin{array}{l} x^{(1)} : b_5(b_4(p_{25} \multimap p_{25}) \multimap p_{27}(p_{18} \multimap p_{19})), \\ x^{(2)} : p_{17}(p_{18} \multimap p_{19}) \multimap p_{20}(p_{21} \multimap p_{22}) \end{array} \right\} \vdash (x^{(2)} (x^{(1)} \lambda w.w)) : p_{20}(p_{21} \multimap p_{22}) \quad (12)$$

We collect the new slice  $\{(p_{17} = b_5 + p_{27}, [x^{(1)}])\}$ . The decoration of the term is shown in Figure 11.

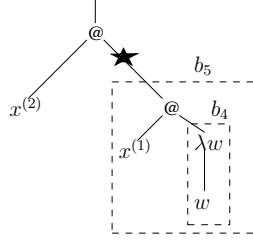


Figure 11:

For the type inference of  $\lambda x.(x^{(2)} (x^{(1)} \lambda w.w)) : ((o \rightarrow o) \rightarrow (o \rightarrow o)) \rightarrow (o \rightarrow o)$ , the algorithm applies the usual rule for abstractions (33), but in this case there are two instances of the bound variable  $x$ . Here comes to work the contraction function  $\mathcal{C}$  (Section 2.2.2), which introduces constraints ensuring that the types of the contracted variables have at least one “!”.

In our example, the algorithm adds  $c_3$  boxes passing through the critical point and  $b_6$  boxes around the body of the abstraction. The  $\mathcal{B}$  function (which adds boxes inside a term according to its critical points) modifies the second constraint in Equation (11):

$$\boxed{p_{17} = b_5 + p_{27} - c_3.} \quad (13)$$

Then the algorithm contracts the types of  $x$ :

$$\boxed{\begin{cases} b_6 + b_5 \geq 1 \\ b_5 = c_3 \\ b_4 = p_{17} \\ p_{18} = p_{19} = p_{21} = p_{22} = p_{25} \\ p_{20} = p_{27} \end{cases}} \quad (14)$$

Finally, it removes the slice  $\{(p_{17} = b_5 + p_{27} - c_3, [x^{(1)}])\}$ .

The derivation obtained, whose decoration is shown in Figure 12, is:

$$\vdash \lambda x.(x (x \lambda w.w)) : b_6 + b_5(b_4(p_{18} \multimap p_{18}) \multimap p_{20}(p_{18} \multimap p_{18})) \multimap b_6 + b_5 + p_{20}(p_{18} \multimap p_{18}). \quad (15)$$

The algorithm process now the whole term  $(\lambda n.\lambda y.((n \lambda z.z) y) \lambda x.(x (x \lambda w.w))) : o \rightarrow o$ . It adds  $b_7$  boxes around the argument of the application and unifies the EAL-types for the application to be correct:

$$\boxed{\begin{cases} b_7 = b_3 + b_2 \\ b_1 = b_6 + b_5 \\ b_4 = p_3 = p_{20} \\ p_4 = p_5 = p_{10} = p_{11} = p_{18} \\ p_9 = b_6 + b_5 + p_{20} \end{cases}} \quad (16)$$

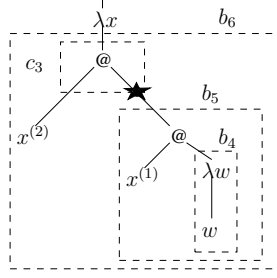


Figure 12:

Since this is the complete term, the final step of the algorithm is a single call to the function  $\mathcal{S}$  (Section 2.2.6), which in this case simply adds  $b_8$  boxes around the term. Therefore, the simply typed lambda term

$$(\lambda n. \lambda y. ((n \lambda z. z) y) \lambda x. (x (x \lambda w. w))) : o \rightarrow o \quad (17)$$

has EAL-type

$$!^{b_8+b_3+c_2} (!^{b_2+c_1+p_4} o \multimap !^{b_2+c_1+p_4} o) \quad (18)$$

for any  $p_1, \dots, p_{30}, b_1, \dots, b_8, c_1, c_2, c_3 \in \mathbb{N}$  solutions of the set of constraints<sup>6</sup> in equations (4)–(16):

$$\left\{ \begin{array}{l} b_6 + b_5 \geq 1 \\ b_7 = b_3 + b_2 \\ b_1 = p_2 = b_6 + b_5 \\ b_5 = c_3 \\ p_1 = p_{16} = p_{23} = 0 \\ p_9 = c_1 + c_2 = b_6 + b_5 + b_4 \\ p_{17} = b_5 + p_{27} - c_3 \\ b_4 = p_3 = p_6 = p_{12} = p_{17} = p_{20} = p_{24} = p_{27} \\ p_4 = p_5 = p_7 = p_8 = p_{10} = p_{11} = p_{13} = p_{14} = p_{15} = p_{18} \\ p_4 = p_{19} = p_{21} = p_{22} = p_{25} = p_{26} = p_{28} = p_{29} = p_{30}. \end{array} \right. \quad (19)$$

The final decoration is shown in Figure 13. Considering the set of constraints in Equation (19) and the incompatibility of  $c_2$  and  $b_2$  stated above, the simply typed term

$$(\lambda n. \lambda y. ((n \lambda z. z) y) \lambda x. (x (x \lambda w. w))) : o \rightarrow o$$

can be typed in EAL either:

1. for any  $n_1, \dots, n_6 \in \mathbb{N}, n_1 \geq 1$  with EAL-type  $!^{n_3+n_5} (!^{n_1+n_2+n_4+n_6} o \multimap !^{n_1+n_2+n_4+n_6} o)$  and decoration shown in Figure 14, or
2. for any  $m_1, \dots, m_7 \in \mathbb{N}, m_1 \geq 1 \wedge m_2 + m_3 = m_1 + m_5$  with EAL-type  $!^{m_3+m_4+m_6} (!^{m_2+m_7} o \multimap !^{m_2+m_7} o)$  and decoration shown in Figure 15.

<sup>6</sup>We have boxed the constraints which were not modified by  $\mathcal{B}$  until the end of the type

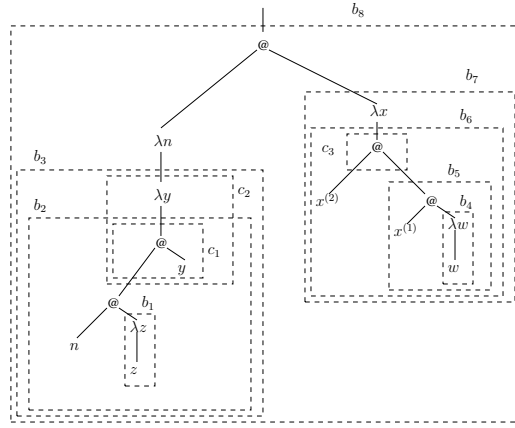


Figure 13: Final superimposed decoration.

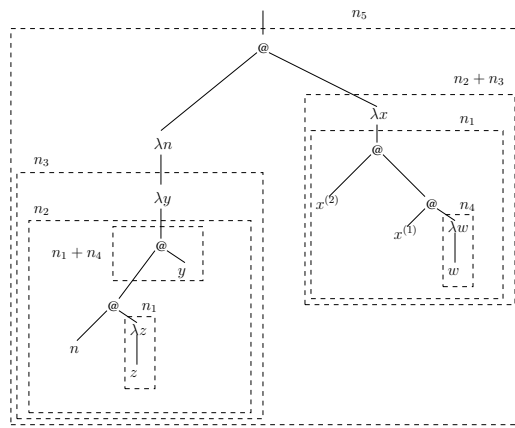


Figure 14: Final decoration.

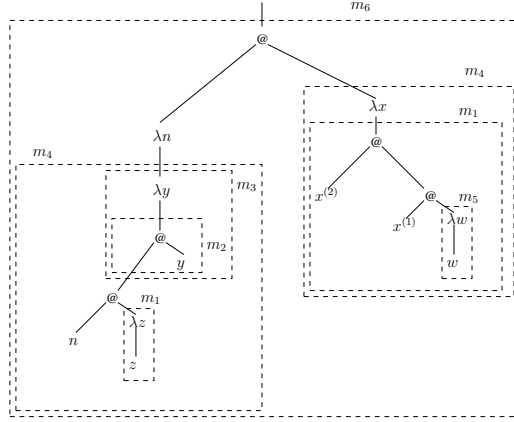


Figure 15: Another possible final decoration.

## 2.2 The full algorithm

We define in this section the formal rules for the algorithm. An almost complete trace of its application to a simply typed term with no EAL type can be found in the Appendix.

**Definition 5.** (Type Synthesis Algorithm) *Given a simply typeable lambda term  $M : \sigma$ , the type synthesis algorithm  $\mathcal{S}(M : \sigma)$  returns a triple  $\langle \Theta, B, A \rangle$ , where  $\Theta$  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.*

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  $!^{n_1+n_2}\Theta$ .

**Notation 3.** *Given a set of linear constraints  $A$  and a solution  $X$  of  $A$ , for any general EAL-type  $\Theta$  and for any base  $B = \{x_1 : \Theta_1, \dots, x_n : \Theta_n\}$ , we denote with  $X(\Theta)$  the instantiation of  $\Theta$  with  $X$  and with  $X(B)$  the instantiation of  $B$  with  $X$ , i.e.,  $X(B) = \{x_1 : X(\Theta_1), \dots, x_n : X(\Theta_n)\}$ .*

### 2.2.1 Unification: $\mathcal{U}$

Unification takes a set of  $h \geq 2$  general EAL-types having the same underlying intuitionistic shape and returns a set of linear equations  $A$  such that for any solution  $X$  of  $A$ , the instantiations of the  $h$  general EAL-types are syntactically

---

inference process in the exposition above. They are now all collected in the set of constraints (19).

identical.

$$\overline{\mathcal{W}(!\sum^{n_{i_1}} o, \dots, !\sum^{n_{i_h}} o)} = \left\{ \begin{array}{c} \sum n_{i_1} - \sum n_{i_2} = 0, \\ \vdots \\ \sum n_{i_{h-1}} - \sum n_{i_h} = 0 \end{array} \right\} \quad (20)$$

$$\frac{\mathcal{W}(\Theta_{1_1}, \dots, \Theta_{1_h}) = A_1 \quad \mathcal{W}(\Theta_{2_1}, \dots, \Theta_{2_h}) = A_2}{\mathcal{W}\left(\begin{array}{c} !\sum^{n_{i_1}}(\Theta_{1_1} \multimap \Theta_{2_1}), \\ \vdots \\ !\sum^{n_{i_h}}(\Theta_{1_h} \multimap \Theta_{2_h}) \end{array}\right)} = \left\{ \begin{array}{c} \sum n_{i_1} - \sum n_{i_2} = 0, \\ \vdots \\ \sum n_{i_{h-1}} - \sum n_{i_h} = 0 \end{array} \right\} \cup A_1 \cup A_2 \quad (21)$$

### 2.2.2 Contraction (C) and Type Processing (P)

Contraction in EAL is allowed only for exponential formulas. Thus, given  $k$  general EAL-types,  $\mathcal{C}$  returns the same set of constraints of  $\mathcal{W}$  with the additional constraint that the number of external  $!$  must be greater than zero.

$$\overline{\mathcal{C}(\Theta)} = \emptyset \quad (22)$$

$$\frac{\mathcal{W}(!^{n_1+\dots+n_h}\Theta_1, \Theta_2, \dots, \Theta_k) = A}{\mathcal{C}(!^{n_1+\dots+n_h}\Theta_1, \dots, \Theta_k) = \{n_1 + \dots + n_h \geq 1\} \cup A} \quad (23)$$

Given a simple type  $\tau$ ,  $\mathcal{P}$  returns the corresponding most general EAL-type obtained by adding everywhere  $p$  exponentials (every  $p$  is a fresh variable).

$$\overline{\mathcal{P}(o)} = !^p o \quad (24)$$

$$\frac{\mathcal{P}(\sigma) = \Theta \quad \mathcal{P}(\tau) = \Xi}{\mathcal{P}(\sigma \rightarrow \tau) = !^p(\Theta \multimap \Xi)} \quad (25)$$

### 2.2.3 Boxing: B and B

The boxing procedure  $\mathcal{B}$  superimposes all boxes due to the presence of critical points. Recall the notion of slice (Definition 4) and Notation 2.  $\mathcal{B}$  has no effect if there is no critical point:

$$\overline{\mathcal{B}(B, \Xi, \emptyset, A)} = \langle B, \Xi, A \rangle \quad (26)$$

For any slice  $sl$ ,  $\mathcal{B}$  adds  $c$  boxes (where  $c$  is fresh) around the subterm above the critical points belonging to  $sl$ :

$$\begin{aligned} \mathcal{B}(B_1, !^c \Xi, sls, A_2) &= \langle B, \Delta, A_1 \rangle \\ B_1 &= \left\{ x_i : \left\{ \begin{array}{ll} !^c \Theta_i & \text{if } x_i \notin sl \\ \Theta_i & \text{if } x_i \in sl \end{array} \right\} \right\}_i \\ A_2 &= \left( \left\{ \begin{array}{ll} A^j & \text{if } A^j \notin sl \\ A^j - c & \text{if } A^j \in sl \end{array} \right\} \right)_j \\ \overline{\mathcal{B}(\{x_i : \Theta_i\}_i, \Xi, \{sl\} \cup sls, A)} &= \langle B, \Delta, A_1 \rangle \end{aligned} \quad (27)$$

Function  $\mathbb{B}$  is the wrapper for  $\mathcal{B}$ . It calls  $\mathcal{B}$  and then adds  $b$  external boxes ( $b$  is fresh):

$$\overline{\mathbb{B}(x, B, \Xi, sls, A)} = \langle B, \Xi, A \rangle \quad (28)$$

$$\frac{\mathcal{B}(B, \Xi, sls, A) = \langle B_1, \Xi_1, A_1 \rangle}{\overline{\mathbb{B}(M, B, \Xi, sls, A)} = \langle !^b B_1, !^b \Xi_1, A_1 \rangle} \quad (29)$$

**Proposition 1.** *Let  $b, c_1, \dots, c_k$  be the fresh variables introduced by  $\mathbb{B}(M, B, \Xi, sls, A) = \langle !^b B_1, !^b \Xi_1, A_1 \rangle$  and let  $X$  be a solution of  $A$ , then*

1.  $X_1 = (X, b = 0, c_1 = 0, \dots, c_k = 0)$  is a solution of  $A_1$ ;
2.  $X_1(\Xi_1) = X(\Xi)$ ;
3.  $X_1(B_1) = X(B)$ .

*Proof.* 1. By Equations (27) and (26), for every variable  $c_i$  introduced by  $\mathcal{B}$ , there is a constraint  $\pm n_{i_1} \pm \dots \pm n_{i_{k_i}} = 0$  changed in  $\pm n_{i_1} \pm \dots \pm n_{i_{k_i}} - c_i = 0$ , hence trivially, if the first one is solvable, then the second one is solvable too imposing  $c_i = 0$ . Moreover, by Equation (29),  $b$  is not added to the set of constraint, hence the thesis.

2. By Equations (27) and (26)  $\Xi_1 = !^{c_1 + \dots + c_k} \Xi$ .

3. By Equations (27) and (26) if  $B = \{x_i : \Theta_i\}_i$  then  $B_1 = \{x_i : !^{\sum_{j \in J_i} c_j} \Theta_i\}_i$  where  $J_i \subseteq \{1, \dots, k\}$ .

□

### 2.2.4 Product union: $\cup$

Product union computes all possible combinations of critical points. It is the culprit for the exponential complexity of the algorithm.

$$\overline{\emptyset \cup sls = sls \cup \emptyset = sls} \quad (30)$$

$$\frac{\left\{ \begin{array}{c} sl_{1,2} \\ \vdots \\ sl_{1,n_1} \end{array} \right\} \cup \left\{ \begin{array}{c} sl_{2,1} \\ \vdots \\ sl_{2,n_2} \end{array} \right\} = sls}{\left\{ \begin{array}{c} sl_{1,1} \\ \vdots \\ sl_{1,n_1} \end{array} \right\} \cup \left\{ \begin{array}{c} sl_{2,1} \\ \vdots \\ sl_{2,n_2} \end{array} \right\} = \{sl_{1,1}, sl_{1,1} \cup sl_{2,1}, \dots, sl_{1,1} \cup sl_{2,n_2}\} \cup sls} \quad (31)$$

### 2.2.5 Type synthesis: $\mathcal{S}$

$\mathcal{S}$  is the main function of the algorithm. It is defined by cases on the structure of the  $\lambda$ -term. Its main cases have already been discussed in Section 2.1. Define  $\neg app(M)$  iff the term  $M$  is not an application.

Variable case:

$$\frac{\mathcal{P}(\sigma) = \Theta}{\mathcal{S}(x : \sigma) = \langle \Theta, \{x : \Theta\}, \emptyset, \emptyset \rangle} \quad (32)$$

First abstraction case: in  $\lambda x.M$ ,  $x \in \mathbf{FV}(M)$ :

$$\frac{\begin{array}{l} h \geq 1 \quad x \in \mathbf{FV}(M) \\ \mathcal{C}(\Theta_1, \dots, \Theta_h) = A_3 \\ \mathbb{B} \left( M, B_1, \Xi_1, sls \cup \left\{ \begin{array}{c} sl_1(x) \\ \vdots \\ sl_k(x) \end{array} \right\}, A_1 \right) = \left\langle B \cup \left\{ \begin{array}{c} x : \Theta_1 \\ \vdots \\ x : \Theta_h \end{array} \right\}, \Xi, A_2 \right\rangle \\ \mathcal{S}(M : \tau) = \langle \Xi_1, B_1, A_1, sls \cup \{sl_1(x), \dots, sl_k(x)\} \rangle \end{array}}{\mathcal{S}(\lambda x.M : \sigma \rightarrow \tau) = \langle \Theta_1 \multimap \Xi, B, A_2 \cup A_3, sls \rangle} \quad (33)$$

Notice that the premises of the rule should be read bottom-up and that when we write  $X \cup \{x_1, \dots, x_n\}$  for a given set or multiset, we mean that  $\forall 1 \leq i \leq n \ x_i \notin X$ . We use such a notation to put in evidence a subset that is deleted by the rule.

Second abstraction case: in  $\lambda x.M$ ,  $x \notin \mathbf{FV}(M)$  and  $M$  is an application:

$$\frac{\begin{array}{l} x \notin \mathbf{FV}((M_1 \ M_2)) \\ sls = sls_1 \cup \{(\sum n_i - n = 0, \mathbf{FV}0(M_1 \ M_2))\} \\ \mathcal{P}(\sigma) = \Theta \\ \mathbb{B}((M_1 \ M_2), B_1, \Xi_1, sls_1, A_1) = \langle B, !\sum n_i \Xi, A \rangle \\ \mathcal{S}((M_1 \ M_2) : \tau) = \langle \Xi_1, B_1, A_1, sls_1 \rangle \end{array}}{\mathcal{S}(\lambda x.(M_1 \ M_2) : \sigma \rightarrow \tau) = \langle \Theta \multimap !^n \Xi, B, A \cup \{\sum n_i - n = 0\}, sls \rangle} \quad (34)$$

Third abstraction case: in  $\lambda x.M$ ,  $x \notin \text{FV}(M)$  and  $M$  is not an application:

$$\begin{array}{l}
\neg \text{app}(M) \\
x \notin \text{FV}(M) \\
\mathcal{P}(\sigma) = \Theta \\
\mathbb{B}(M, B_1, \Xi_1, \text{sl}s, A_1) = \langle B, \Xi, A \rangle \\
\mathcal{S}(M : \tau) = \langle \Xi_1, B_1, A_1, \text{sl}s \rangle \\
\hline
\mathcal{S}(\lambda x.M : \sigma \rightarrow \tau) = \langle \Theta \multimap \Xi, B, A, \text{sl}s \rangle
\end{array} \tag{35}$$

There are four cases for the application, differing in the way they handle the set of slices.

First application case: in  $(M N)$ , neither  $M$  nor  $N$  are applications:

$$\begin{array}{l}
\neg \text{app}(M) \wedge \neg \text{app}(N) \\
\mathcal{U}(\Theta_1, \Theta_3) = A_4 \\
\mathbb{B}(N, B_2, \Theta_2, \text{sl}s_2, A_2) = \langle B_3, \Theta_3, A_3 \rangle \\
\mathcal{S}(N : \sigma) = \langle \Theta_2, B_2, A_2, \text{sl}s_2 \rangle \\
\mathcal{S}(M : \sigma \rightarrow \tau) = \langle !\sum^{n_i}(\Theta_1 \multimap \Xi), B_1, A_1, \text{sl}s_1 \rangle \\
\hline
\mathcal{S}((M N) : \tau) = \langle \Xi, B_1 \cup B_3, A_1 \cup A_3 \cup A_4 \cup \{\sum n_i = 0\}, \text{sl}s_1 \uplus \text{sl}s_2 \rangle
\end{array} \tag{36}$$

Second application case: in  $(M N)$ ,  $M$  is not an application:

$$\begin{array}{l}
\neg \text{app}(M) \\
\text{sl}s = \text{sl}s_1 \uplus (\text{sl}s_2 \cup \{\{(A_4^1, \text{FV}0((N_1 N_2))\}\}\}) \\
\mathcal{U}(\Theta_3, \Theta_1) = A_4 \\
\mathbb{B}((N_1 N_2), B_2, \Theta_2, \text{sl}s_2, A_2) = \langle B_3, \Theta_3, A_3 \rangle \\
\mathcal{S}((N_1 N_2) : \sigma) = \langle \Theta_2, B_2, A_2, \text{sl}s_2 \rangle \\
\mathcal{S}(M : \sigma \rightarrow \tau) = \langle !\sum^{n_i}(\Theta_1 \multimap \Xi), B_1, A_1, \text{sl}s_1 \rangle \\
\hline
\mathcal{S}((M (N_1 N_2)) : \tau) = \langle \Xi, B_1 \cup B_3, A_1 \cup A_3 \cup A_4 \cup \{\sum n_i = 0\}, \text{sl}s \rangle
\end{array} \tag{37}$$

Notice that  $A_4^1$  indicates the equality constraints between the outermost number of ! in the type of  $(N_1 N_2)$  and in the function part of the type of  $M$ .

Third application case: in  $(M N)$ ,  $N$  is not an application:

$$\begin{array}{l}
\neg \text{app}(N) \\
\text{sl}s = (\text{sl}s_1 \cup \{\{(\sum n_i = 0, \text{FV}0((M_1 M_2))\}\}) \uplus \text{sl}s_2 \\
\mathcal{U}(\Theta_1, \Theta_3) = A_4 \\
\mathbb{B}(N, B_2, \Theta_2, \text{sl}s_2, A_2) = \langle B_3, \Theta_3, A_3 \rangle \\
\mathcal{S}(N : \sigma) = \langle \Theta_2, B_2, A_2, \text{sl}s_2 \rangle \\
\mathcal{S}((M_1 M_2) : \sigma \rightarrow \tau) = \langle !\sum^{n_i}(\Theta_1 \multimap \Xi), B_1, A_1, \text{sl}s_1 \rangle \\
\hline
\mathcal{S}((M_1 M_2) N) : \tau) = \langle \Xi, B_1 \cup B_3, A_1 \cup A_3 \cup A_4 \cup \{\sum n_i = 0\}, \text{sl}s \rangle
\end{array} \tag{38}$$

Fourth application case: in  $(M\ N)$ , both  $M$  and  $N$  are applications:

$$\begin{aligned}
sls_4 &= sls_2 \cup \{\{(A_4^1, \text{FV}0((N_1\ N_2)))\}\} \\
sls_3 &= sls_1 \cup \{\{(\sum n_i = 0, \text{FV}0((M_1\ M_2)))\}\} \\
\mathcal{U}(\Theta_3, \Theta_1) &= A_4 \\
\mathbb{B}((N_1\ N_2), B_2, \Theta_2, sls_2, A_2) &= \langle B_3, \Theta_3, A_3 \rangle \\
\mathcal{S}((N_1\ N_2) : \sigma) &= \langle \Theta_2, B_2, A_2, sls_2 \rangle \\
\mathcal{S}((M_1\ M_2) : \sigma \rightarrow \tau) &= \langle \sum n_i (\Theta_1 \multimap \Xi), B_1, A_1, sls_1 \rangle \\
\hline
\mathcal{S}((M_1\ M_2)\ (N_1\ N_2) : \tau) &= \langle \Xi, B_1 \cup B_3, A_1 \cup A_3 \cup A_4 \cup \{\sum n_i = 0\}, sls_3 \uplus sls_4 \rangle
\end{aligned} \tag{39}$$

### 2.2.6 Type synthesis algorithm: $\mathcal{S}$

$\mathcal{S}$  is the top level call for the algorithm. It invokes  $\mathcal{S}$ , takes the result, boxes the term, forgets the critical points and eventually contracts the common variables in the base.

$$\begin{aligned}
&\mathcal{C}(\Theta_{1,1}, \dots, \Theta_{1,k_1}) = A_1 \quad \dots \quad \mathcal{C}(\Theta_{h,1}, \dots, \Theta_{h,k_h}) = A_h \\
\mathbb{B}(M, B_1, \Theta_1, sls, A') &= \left\langle \left\{ \begin{array}{l} x_1 : \Theta_{1,1}, \dots, x_1 : \Theta_{1,k_1}, \\ \vdots \\ x_h : \Theta_{h,1}, \dots, x_h : \Theta_{h,k_h} \end{array} \right\}, \Theta, A \right\rangle \\
\mathcal{S}(M : \sigma) &= \langle \Theta_1, B_1, A', sls \rangle \\
\hline
\mathcal{S}(M : \sigma) &= \left\langle \Theta, \{x_1 : \Theta_{1,1}, \dots, x_h : \Theta_{h,1}\}, A \cup \bigcup_{i=1}^h A_i \right\rangle
\end{aligned} \tag{40}$$

## 3 Correctness and completeness

We will prove in this section that our algorithm  $\mathcal{S}$  is complete with respect to the notion of  $\Gamma \vdash_{\text{EAL}} M : A$  introduced in Definition 1. Correctness and completeness of  $\mathcal{S}$  are much simpler if, instead of EAL, we formulate proofs and results with reference to an equivalent natural deduction formulation, discussed in the following subsection. Before, we state the obvious fact that our algorithm does not loop, since any rule  $\mathcal{S}$  decreases the structural size of the  $\lambda$ -term  $M$ , any rule  $\mathcal{U}$  decreases the size of the type  $\Theta$  and any rule  $\mathcal{B}$  and  $\mathcal{U}$  decreases the size of the set of slices of critical points  $sls$ .

**Proposition 2 (Termination).** *Let  $M$  be a simply typed term and let  $\sigma$  its simple type.  $\mathcal{S}(M : \sigma)$  always terminates with a triple  $\langle \Theta, B, A \rangle$ .*

The algorithm is exponential in the size of the  $\lambda$ -term, because to investigate all possible derivations we need to (try to) box all possible combinations of critical points (see the rules for the product union,  $\mathcal{U}$ , in Section 2.2.4), that are roughly bounded by the size of the term.

### 3.1 NEAL

The natural deduction calculus (NEAL) for EAL is given in Figure 16, after [Asp98, BBdPH93, Rov98].

$$\boxed{
\begin{array}{c}
\frac{}{\Gamma, A \vdash_{\text{NEAL}} A} \text{ ax} \qquad \frac{\Gamma \vdash_{\text{NEAL}} !A \quad \Delta, !A, !A \vdash_{\text{NEAL}} B}{\Gamma, \Delta \vdash_{\text{NEAL}} B} \text{ contr} \\
\frac{\Gamma, A \vdash_{\text{NEAL}} B}{\Gamma \vdash_{\text{NEAL}} A \multimap B} (\multimap I) \qquad \frac{\Gamma \vdash_{\text{NEAL}} A \multimap B \quad \Delta \vdash_{\text{NEAL}} A}{\Gamma, \Delta \vdash_{\text{NEAL}} B} (\multimap E) \\
\frac{\Delta_1 \vdash_{\text{NEAL}} !A_1 \cdots \Delta_n \vdash_{\text{NEAL}} !A_n \quad A_1, \dots, A_n \vdash_{\text{NEAL}} B}{\Gamma, \Delta_1, \dots, \Delta_n \vdash_{\text{NEAL}} !B} !
\end{array}
}$$

Figure 16: Natural Elementary Affine Logic in sequent style notation

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

To annotate NEAL derivations, we use terms generated by the following grammar (*elementary affine terms*  $\Lambda^{EA}$ ):

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

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

- $\text{FV}(x) = \{x\}$
- $\text{FV}(\lambda x.M) = \text{FV}(M) \setminus \{x\}$
- $\text{FV}(M_1 M_2) = \text{FV}(M_1) \cup \text{FV}(M_2)$
- $\text{FV}(!(M) [^{M_1}/x_1, \dots, ^{M_n}/x_n]) = \bigcup_{i=1}^n \text{FV}(M_i) \cup \text{FV}(M) \setminus \{x_1, \dots, x_n\}$
- $\text{FV}([M]_{N=x_1, x_2}) = (\text{FV}(M) \setminus \{x_1, x_2\}) \cup \text{FV}(N)$

Ordinary substitution  $N\{M/x\}$  of a term  $M$  for the free occurrences of  $x$  in  $N$ , is defined in the obvious way (we omit the cases for the standard  $\lambda$ -calculus):

1.  $!(N) [^{P_1}/x_1, \dots, ^{P_n}/x_n] \{M/x\} =$   
 $!(N\{y_1/x_1\} \cdots \{y_n/x_n\}\{M/x\}) [^{P_1\{M/x\}}/y_1, \dots, ^{P_n\{M/x\}}/y_n]$   
if  $x \notin \{x_1, \dots, x_n\}$ , where  $y_1, \dots, y_n$  are all fresh variables;
2.  $!(N) [^{P_1}/x_1, \dots, ^{P_n}/x_n] \{M/x\} = !(N) [^{P_1\{M/x\}}/x_1, \dots, ^{P_n\{M/x\}}/x_n]$   
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  $x \in \{y, z\}$ .

$$\boxed{
\begin{array}{c}
\frac{}{\Gamma, x : A \vdash_{\text{NEAL}} x : A} \text{ax} \qquad \frac{\Gamma \vdash_{\text{NEAL}} M : !A \quad \Delta, x : !A, y : !A \vdash_{\text{NEAL}} N : B}{\Gamma, \Delta \vdash_{\text{NEAL}} [N]_{M=x,y} : B} \text{contr} \\
\frac{\Gamma, x : A \vdash_{\text{NEAL}} M : B}{\Gamma \vdash_{\text{NEAL}} \lambda x. M : A \multimap B} (\multimap I) \qquad \frac{\Gamma \vdash_{\text{NEAL}} M : A \multimap B \quad \Delta \vdash_{\text{NEAL}} N : A}{\Gamma, \Delta \vdash_{\text{NEAL}} (M N) : B} (\multimap E) \\
\frac{\Delta_1 \vdash_{\text{NEAL}} M_1 : !A_1 \cdots \Delta_n \vdash_{\text{NEAL}} M_n : !A_n \quad x_1 : A_1, \dots, x_n : A_n \vdash_{\text{NEAL}} N : B}{\Gamma, \Delta_1, \dots, \Delta_n \vdash_{\text{NEAL}} ! (N) [^{M_1}/x_1, \dots, ^{M_n}/x_n] : B} !
\end{array}
}$$

Figure 17: Term Assignment System for Natural Elementary Affine Logic

Elementary terms may be mapped to  $\lambda$ -terms, by forgetting the exponential structure:

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

**Lemma 2.** 1. For any elementary terms  $M$  and  $N$ ,  $(M\{N/x\})^* \equiv M^*\{N^*/x\}$ .  
2. For any elementary term  $M$ ,  $\text{FV}(M) = \text{FV}(M^*)$ .

**Definition 6.** (Legal elementary terms) An elementary term  $M$  is legal if and only if every variable in  $M$  (either free or bound) occurs linearly.

**Note 1.** From now on we will consider only legal terms. Notice that this is not a limitation in the sense that if there is an elementary term  $M$  with type  $A$  then there is a legal elementary term  $M' =_{\alpha} M$  with type  $A$ .

**Notation 4.** Let  $\Gamma = \{x_1 : A_1, \dots, x_n : A_n\}$  be a basis.  $\text{dom}(\Gamma) = \{x_1, \dots, x_n\}$ ;  $\Gamma(x_i) = A_i$ ;  $\Gamma \upharpoonright V = \{x : A \mid x \in V \wedge A = \Gamma(x)\}$ .

The term assignment system is shown in Figure 17, where all bases in the premises of the contraction,  $\multimap$  elimination and  $!$ -rule, have domains with empty intersection.

**Lemma 3.**

1. If  $\Gamma \vdash_{\text{NEAL}} M : A$  then  $\text{FV}(M) \subseteq \text{dom}(\Gamma)$ ;
2. if  $\Gamma \vdash_{\text{NEAL}} M : A$  then  $\Gamma \upharpoonright \text{FV}(M) \vdash_{\text{NEAL}} M : A$ .

**Lemma 4 (Substitution).** If  $\Gamma, x : A \vdash_{\text{NEAL}} M : B$  and  $\Delta \vdash_{\text{NEAL}} N : A$  and  $\text{dom}(\Gamma) \cap \text{dom}(\Delta) = \emptyset$  then  $\Gamma, \Delta \vdash_{\text{NEAL}} M\{N/x\} : B$ .

*Proof.* Recalling that both  $M$  and  $N$  are legal terms, by easy induction on the size of  $M$ .  $\square$

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

*Proof.* (if) By induction, using the cut rule. It is also possible to prove, by an easy inspection of the cut-elimination theorem for EAL, that it is possible to eliminate just the exponential cuts, leaving the logical ones.

(only if) The only interesting case is  $(\multimap L)$ . The proof is identical to the case of intuitionistic logic.  $\square$

**Lemma 5 (Unique Derivation).** *For any legal term  $M$  and formula  $A$ , if there is a valid derivation of the form  $\Gamma \vdash_{NEAL} M : A$ , then such derivation is unique (up to weakening).*

A notion of reduction is needed to state and obtain completeness of the type inference algorithm. We define two *logical* reductions ( $\rightarrow_\beta$  and  $\rightarrow_{\text{dup}}$ ) 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.

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

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

**Definition 7.** *The reduction relation on legal terms  $\rightsquigarrow$  is defined as the reflexive and transitive closure of the union of  $\rightarrow_\beta, \rightarrow_{\text{dup}}, \rightarrow_{!-!}, \rightarrow_{@-c}, \rightarrow_{!-c}, \rightarrow_{c-c}, \rightarrow_{\lambda-c}$ .*

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

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

*Proof.* By induction on the size of  $M$  using Lemma 2. □

**Lemma 6.** *Let  $M$  be a well typed term in  $\{\text{dup}, !-, @-c, !-c, c-c, \lambda-c\}$ -normal form, then*

1. *if  $R = [N]_{P=x,y}$  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.*

Point 2 of Lemma above intuitively justifies why we collect critical points on the root of applications during the type synthesis in Section 2.2.

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

### 3.2 Properties of the Type Inference Algorithm

We start with a property of the slides collected by the algorithm. At any point, in a single slice there are never two critical points on the same path from the root to a leaf.

**Lemma 7.** *Let  $\mathcal{S}(M : \sigma) = \langle \Theta, B, A, sls \rangle$ . For any slice  $sl$  in  $sls$ ,  $sl = \{cpt_1, \dots, cpt_k\}$ , for every path from the root of the syntax tree of  $M$  to any leaf, there exists at most one  $cpt_i$  in the path.*

*Proof.* Notice that if  $cpt_i$  and  $cpt_j$  are in the same path from the root to any leaf of the syntax tree of  $M$  then the intersection of the free variable occurrences sets of  $cpt_i$  and  $cpt_j$  cannot be empty. Now, by induction on the derivation, if  $\mathcal{S}(M : \sigma) = \langle \Theta, B, A, sls \rangle$  then we prove that

1.  $\forall sl_i \in sls \quad \forall cpt_j, cpt_k \in sl_i$  the set of free variable occurrences of  $cpt_j$  and  $cpt_k$  are disjoint ( $\text{FVO}(cpt_j) \cap \text{FVO}(cpt_k) = \emptyset$ );
2.  $\forall sl_i \in sls \quad \forall cpt_j \in sl_i$  the set of free variable occurrences of  $cpt_j$  is a subset of  $\text{FVO}(M)$ .

Actually by inspecting the rules of  $\mathcal{S}$  the set of slices of critical points either: (i) shrinks, by deleting the slices containing the bounded variable (equation (33)); (ii) or grows, by a new slice of a single critical point (equation (34))—in these cases the thesis trivially holds; (iii) or grows by product union of set of slices (equations (36)–(39)). If  $M$  is an application  $(M' N)$ ,  $sls_1$  is the set of slices of  $M'$  and  $sls_2$  is the set of slices of  $N$ , then, with abuse of notation,  $\text{FVO}(sls_1) \cap \text{FVO}(sls_2) = \emptyset$ , since  $\text{FVO}(M') \cap \text{FVO}(N) = \emptyset$ . Moreover by induction on the size of  $sls_1$  and by definition of product union (equations (30), (31)), if  $\text{FVO}(sls_1) \cap$

$\text{FVO}(sls_2) = \emptyset$  and  $\forall sl_i \in sls_1 \cup sls_2 \quad \forall cpt_j, cpt_k \in sl_i \quad \text{FVO}(cpt_j) \cap \text{FVO}(cpt_k) = \emptyset$   
then  $\forall sl_i \in sls_1 \uplus sls_2 \quad \forall cpt_j, cpt_k \in sl_i \quad \text{FVO}(cpt_j) \cap \text{FVO}(cpt_k) = \emptyset$ . Hence the thesis.  $\square$

The following Lemma illustrates the relation between the set of critical points calculated by the algorithm for a given term  $M$  and a particular class of decompositions of  $M$ .

**Lemma 8.** *Let  $\mathcal{S}(M : \sigma) = \langle \Theta, B, A, sls \rangle$ .*

1.  $\forall \{cpt_1, \dots, cpt_k\} = sl \in sls$  there exist  $P, (N_{1,1} N_{2,1}), \dots, (N_{1,k} N_{2,k})$  such that  $P$  is not a variable,  $x_1, \dots, x_k \in \text{FV}(P)$  and  $M = P\{(N_{1,1} N_{2,1})/x_1, \dots, (N_{1,k} N_{2,k})/x_k\}$ ;
2.  $\forall P, (N_{1,1} N_{2,1}), \dots, (N_{1,k} N_{2,k})$  such that  $P$  is not a variable,  $x_1, \dots, x_k \in \text{FV}(P)$  and  $M = P\{(N_{1,1} N_{2,1})/x_1, \dots, (N_{1,k} N_{2,k})/x_k\}$ , there exists  $\{cpt_1, \dots, cpt_k\} = sl \in sls$  such that  $cpt_i$  is the critical point at the root of  $(N_{1,i} N_{2,i})$ .

*Proof.* By structural induction on  $M$ .

1.
  - If  $M$  is a variable, the thesis trivially holds being  $sls = \emptyset$ .
  - If  $M = \lambda x.M'$ , either  $sl$  is the slice collected by the rule (34) and consists of a single critical point corresponding to the root of  $M'$ , then  $P = \lambda x.y$ , or  $sl$  is a slice of  $M'$ , then by inductive hypothesis there exists  $P'$  s.t. the thesis holds for  $M'$ . We take  $P = \lambda x.P'$ .
  - If  $M = (M_1 M_2)$ , by rules (36)–(39) and by definition of product union (31), if  $sl$  is a slice of  $M_1$  or  $M_2$  or it is the union of two slices in  $M_1$  and  $M_2$  than the thesis holds by inductive hypothesis. Otherwise  $sl$  is a slice with a critical point  $cpt_1$  corresponding to the root of  $M_1$  or a critical point  $cpt_2$  corresponding to the root of  $M_2$ , collected by the algorithm in one of the rules (37), (38) or (39). If  $sl = \{cpt_1, cpt_2\}$  then  $P = (x y)$ , if  $sl = \{cpt_1\}$  then  $P = (x M_2)$ , analogously for  $sl = \{cpt_2\}$  and finally, if  $sl = \{cpt_1\} \cup sl_2$ , by Lemma 7 all the other critical points in  $sl_2$  belong to  $M_2$ , then by inductive hypothesis there exists  $P_2$  and we take  $P = (x P_2)$ . The cases  $sl = sl_1 \cup \{cpt_2\}$  is analogous.
2.
  - If  $M$  is a variable then there exists no  $P$  and the thesis trivially holds.
  - If  $M = \lambda x.M'$  then  $P = \lambda x.P'$ .  
If  $P'$  is a variable, then the slice to consider is the one containing only the critical point corresponding to the root of  $M'$ . Such a slice has been added to  $sls$  in the rule (34).  
Otherwise  $P', (N_{1,1} N_{2,1}), \dots, (N_{1,k} N_{2,k})$  is a decomposition of  $M'$  too. Then by inductive hypothesis there exists  $sl$  in  $sls'$  collected by  $\mathcal{S}(M' \dots)$ . Notice that  $x \notin sl$  since  $\lambda x.M' = \lambda x.P'\{(N_{1,1} N_{2,1})/x_1, \dots, (N_{1,k} N_{2,k})/x_k\}$ , hence  $sl$  is a slice in  $sls$  too. Hence the thesis.
  - Finally if  $M = (M_1 M_2)$ , then  $P = (P_1 P_2)$ .

- If  $P_1$  and  $P_2$  are both variables, then by definition of rule (39) there is a slice  $sl_1$  corresponding to the root of  $M_1$  and a slice  $sl_2$  corresponding to the root of  $M_2$ . We take  $sl = sl_1 \cup sl_2$  generated by the product union.
- If  $P_1$  is a variable, but  $P_2$  is not, by inductive hypothesis on  $M_2$  there exists  $sl_2 \in sls_2$ . Then we take  $sl_1$  corresponding to the root of  $M_1$  and by definition of product union (31)  $sl = sl_1 \cup sl_2$ . The specular case is analogous.
- Finally, if both  $P_1$  and  $P_2$  are not a variable, then by inductive hypothesis there are  $sl_1 \in sls_1$  and  $sl_2 \in sls_2$ . Then the thesis holds by definition of product union (31) taking  $sl = sl_1 \cup sl_2$ .

□

Consider the *length*  $L(M)$  of an EAL-term  $M$  defined inductively:

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

**Definition 8.** An EAL-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.  $L(M) = L((M)^*)$

**Fact 1.** A simple EAL-term contracts at most variables.

**Definition 9.** The set of candidate EAL-terms is the set of all EAL-terms  $P$  such that

1.  $P$  is in  $\{!-, @-, !-, c-, \lambda-, \text{dup}\}$ -normal form;
2.  $P$  is simple;
3. if  $[R]_{Q=x,y}$  is a subterm of  $P$ , then  $x, y \in \text{FV}(R)$ ;
4. if  $!(R) [^{Q_1}/x_1, \dots, ^{Q_k}/x_k]$  is a subterm of  $P$ , then  $R$  is not a variable.

**Definition 10.** Given a general EAL-type  $\Theta$  we define its erasure  $\bar{\Theta}$  as the simple type obtained by  $\Theta$  erasing all the exponentials “!” and changing  $\multimap$  into  $\rightarrow$ .

**Lemma 9.** For any  $\Theta$  general EAL-type there exists a solution  $X$  s.t.  $X(\mathcal{P}(\bar{\Theta})) = \Theta$ .

**Theorem 3 (Completeness).** *Let  $\Gamma \vdash_{\text{NEAL}} P : \Psi$  and let  $P$  be a candidate EAL-term. Let  $\mathcal{S}(P^* : \overline{\Psi}) = \langle \Theta, B, A \rangle$ , then there exists  $X$  integer solution of  $A$  such that  $X(B) \subseteq \Gamma$ ,  $\Psi = X(\Theta)$  and  $X(B) \vdash_{\text{NEAL}} P : X(\Theta)$ .*

*Proof.* By induction on  $P$ .

- If  $\Gamma, x : \Psi \vdash_{\text{NEAL}} x : \Psi$  then  $\mathcal{S}(x : \overline{\Psi}) = \langle \mathcal{P}(\overline{\Psi}), \{x : \mathcal{P}(\overline{\Psi})\}, \emptyset \rangle$  and the thesis holds by Lemma 9 being any  $X$  solution of the empty set of constraints.
- If the type derivation ends with

$$\frac{\Gamma \vdash_{\text{NEAL}} x : !\Phi \quad \Delta, y : !\Phi, z : !\Phi \vdash_{\text{NEAL}} N : \Psi}{\Gamma, \Delta \vdash_{\text{NEAL}} [N]_{x=y, z} : \Psi}$$

then the thesis holds by inductive hypothesis on  $\Delta, y : !\Phi, z : !\Phi \vdash_{\text{NEAL}} N : \Psi$ , since the set of constraints generated by  $\mathcal{S}(N^* : \overline{\Psi})$  is the same generated by  $\mathcal{S}([N]_{x=y, z}^* : \overline{\Psi})$  but the additional constraint due to the contraction of types of  $y$  and  $z$ . However, if  $X$  is a solution such that  $X(B)(y) = X(B)(z) = !\Phi$  then it is also a solution of the additional constraint.

- If  $P$  is an abstraction then the type derivation is

$$\frac{\Gamma, x : \Psi \vdash_{\text{NEAL}} M : \Phi}{\Gamma \vdash_{\text{NEAL}} \lambda x. M : \Psi \multimap \Phi}$$

By inductive hypothesis  $\mathcal{S}(M^* : \overline{\Phi}) = \langle \Theta', B', A' \rangle$  and  $X'$  is a solution of  $A$ . Moreover, by definition of  $\mathcal{S}$  and rules (33)–(35) and by Lemma 9,  $X'$  is a solution of the set of constraints generated by  $\mathcal{S}((\lambda x. M)^* : \overline{\Psi \multimap \Phi})$  too. Hence, by Proposition 1,  $X = (X', b = 0, c_1 = 0, \dots, c_k = 0)$ , where  $b, c_1, \dots, c_k$  are the fresh variables introduced by  $\mathbb{B}$  in the premise of the rule for  $\mathcal{S}((\lambda x. M)^* : \overline{\Psi \multimap \Phi})$ , is the solution needed to prove the thesis.

- If  $P$  is an application

$$\frac{\Gamma \vdash_{\text{NEAL}} M : \Phi \multimap \Psi \quad \Delta \vdash_{\text{NEAL}} N : \Phi}{\Gamma, \Delta \vdash_{\text{NEAL}} (M N) : \Psi}$$

By inductive hypothesis there are solutions  $X_1$  for  $M$  and  $X_2$  for  $N$ . Now, by definition of  $\mathcal{S}$  and by rules (36)–(39),  $X_1$  is such that all fresh variables introduced by  $\mathbb{B}(M^*, \dots)$  are equal to 0, otherwise the type of  $M$  cannot be functional. Moreover, by inductive hypothesis,  $X_1(\Theta_1) = \Phi$  and  $X_2(\Theta_2) = \Phi$ , hence  $X_1 \cup X_2$  is a solution of  $\mathcal{U}(\Theta_1, \Theta_2)$ . Notice that the union of solutions is well defined since the set of variables in the constraints for  $\mathcal{S}(M \dots)$  is distinct from the set of variables in the constraints for  $\mathcal{S}(N \dots)$ . Finally, by Proposition 1 again, the thesis holds with  $X = (X_1 \cup X_2, b = 0, c_1 = 0, \dots, c_k = 0)$  where  $b, c_1, \dots, c_k$  are the fresh variables introduced by  $\mathbb{B}$  in the premise of the rule for  $\mathcal{S}((M N)^* : \overline{\Psi})$ .

- Finally, if the derivation is

$$\frac{\Delta_1 \vdash_{\text{NEAL}} M_1 : !\Phi_1 \cdots \Delta_n \vdash_{\text{NEAL}} M_n : !\Phi_n \quad x_1 : \Phi_1, \dots, x_n : \Phi_n \vdash_{\text{NEAL}} N : \Psi}{\Gamma, \Delta_1, \dots, \Delta_n \vdash_{\text{NEAL}} !(N) [^{M_1/x_1, \dots, M_n/x_n}] : !\Psi}$$

then by Lemma 6 either  $M_i$  is a variable or an application.

If all  $M_i$  are variables, then  $N^* =_{\alpha} (!(N) [^{M_1/x_1, \dots, M_n/x_n}])^*$  and the thesis holds getting the solution of the inductive hypothesis and increasing the variable  $b$  introduced by  $\mathbb{B}(N^* \dots)$  by one.

If there is at least one  $M_i$  that is an application, then by Lemma 8 there is a critical point  $\text{cpt}_i$  collected by the algorithm  $\mathcal{S}$  at the root of  $M_i$ . Moreover  $N^* \neq (!(N) [^{M_1/x_1, \dots, M_n/x_n}])^*$  and the set of constraints generated by  $\mathcal{S}(!(N) [^{M_1/x_1, \dots, M_n/x_n}])^* : !\Psi$  is different by the union of the set of constraints for  $N$  and the various  $M_i$ 's. This is due to the  $\mathbb{B}$  function and to the fact that the set of slices collected by the algorithm is different for  $N^*$  and  $(!(N) [^{M_1/x_1, \dots, M_n/x_n}])^*$ . However we can build the solution  $X$  starting from the union of the solutions obtained by inductive hypothesis for the  $M_i$ 's that are not variables  $\bigcup X_i$  and modifying the solution for  $N^*$  as follows: for any  $b$  in  $X'$  introduced by  $\mathbb{B}$  during the type synthesis of  $N^*$ , if there is no corresponding  $b$  in  $X$  than there exists at least one  $c_j$  in  $X$ , corresponding to a slice containing the critical point  $\text{cpt}_i$ , introduced by  $\mathcal{B}$  during the type synthesis of  $(!(N) [^{M_1/x_1, \dots, M_n/x_n}])^*$ , and we can set it to the same value of  $b$ . By Proposition 1 all the additional variables introduced by  $\mathbb{B}$  are set to 0. Finally the variable  $c_k$  introduced by  $\mathbb{B}(!(N) [^{M_1/x_1, \dots, M_n/x_n}])^*, \dots$  and corresponding to the slice associated to the decomposition  $N^*, M_1^*, \dots, M_n^*$  is set to 1.

□

In the statement of the previous theorem, the request on the  $\{!-, @-, \text{c}, !-, \text{c}, \text{c}-\text{c}, \lambda-\text{c}, \text{dup}\}$ -normal form is not a loss of generality, for the subject reduction lemma and Proposition 4. By Lemma 6, the only restriction induced by the request of contracting at most variable is the exclusion of elementary terms with subterms of the form  $[R]_{(Q_1 \ Q_2)=x,y}$  or  $!(R)[^{P_1/x_1, \dots, (Q_1 \ Q_2)/x, \dots, P_n/x_n}]$  with  $[S]_{x=y,z}$  subterm of  $R$ . Recalling the discussion at the end of Section 1, we see that these terms, in a sense, “contract too much”—in the sharing graph of the corresponding  $\lambda$ -term  $P^*$ , there would be fan nodes corresponding to non-variable contractions. We also do not take into account elementary affine terms with “false contractions”. This is not a limitation by Lemma 1 and Theorem 2. Finally we discard terms as  $!(x)[^M/x]$ . Again this is not a limitation, in fact  $(!(x)[^M/x])^* = M^*$  and  $\Gamma \vdash_{\text{NEAL}} !(x)[^M/x] : !\Psi$  if and only if  $\Gamma \vdash_{\text{NEAL}} M : !\Psi$ .

**Notation 5.** *We use*

$$\frac{\Gamma \vdash M : !^n A \quad x : A \vdash N : B}{\Gamma \vdash !^n (N) [^M/x] : !^n B}$$

as a shorthand for

$$\begin{array}{c}
\frac{x_1 : !A \vdash x_1 : !A \quad x : A \vdash N : B}{x_1 : !A \vdash ! (N)^{[x_1/x]} : !B} \\
\hline
x_2 : !!A \vdash x_2 : !!A \quad \vdots \\
\vdots \\
\frac{\Gamma \vdash M : \overbrace{! \dots !}^n A \quad x_{n-1} : \overbrace{! \dots !}^{n-1} : A \vdash \overbrace{! (\dots ! (N)^{[x_1/x]} \dots)^{[x_{n-1}/x_{n-2}]} : \overbrace{! \dots !}^{n-1} B}{\Gamma \vdash \overbrace{! (\dots ! (N)^{[x_1/x]} \dots)^{[M/x_{n-1}]} : \overbrace{! \dots !}^n B}
\end{array}$$

**Lemma 10 (Superimposing of derivations).** *Let  $\mathcal{S}(M : \sigma) = \langle \Theta, B, A \rangle$  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  $X_1(\Theta) = X_2(\Theta)$  and  $X_1(B) = X_2(B)$ .

*Proof.* If  $X_1$  instantiates two incompatible boxes, i.e. two overlapped boxes, then one of those boxes corresponds to a variable  $c_i$  introduced by  $\mathcal{B}$  when processing a slice  $sl$ . Actually all boxes corresponding to variables  $b$  introduced by  $\mathbb{B}$  cannot overlap since they are all around the subterm. Moreover if the variable  $c_i$  has been introduced while  $\mathcal{S}$  were processing  $M_1$  and the second box corresponds to a variable  $x$  introduced by  $\mathcal{S}$  during the synthesis of  $M_2$ , then either  $M_2$  is a subterm of  $M_1$  or  $M_2 \equiv M_1$  otherwise the boxes cannot overlap. Let  $sl_x$  the slice corresponding to the second box—the case of  $x$  introduced by  $\mathbb{B}$  can be treated uniformly taking  $sl_x = \emptyset$ —and let  $sl_{c_i}$  be the slice corresponding to the first box. Since box  $c_i$  overlaps box  $x$ , there exists a critical point  $cpt \in sl_{c_i}$  in the path from the root of  $M_2$  to the leaves and there is no critical point in  $sl_x$  between the root of  $M_2$  and  $cpt$ . By Lemma 8 there exists a decomposition of  $M_2 = M_2' \{N_1/x_1, \dots, N_k/x_k\}$  corresponding to  $sl_x$  and a decomposition of  $M_1 = M_1' \{P_1/y_1, \dots, P_h/y_h\}$ .

Since  $cpt$  is inside the box of  $M_2$  then there could be some critical points of  $sl_x$  in the path from  $cpt$  to the leaves and then there exists eventually a decomposition, without loss of generality, of  $P_h = P_h' \{N_r/x_r, \dots, N_s/x_s\}$ . Hence it is possible to decompose  $M_1 = M_1'' \{P_1/y_1, \dots, P_{h-1}/y_{h-1}, N_r/x_r, \dots, N_s/x_s\}$  where  $M_1'' = M_1' \{P_h'/y_h\}$ . Again by Lemma 8 there is a slice  $sl_{c_j}$  in  $sls_1$  of  $M_1$  corresponding to such decomposition. Notice that  $cpt$  is not a critical point of the box corresponding to  $c_j$  since it is at the root of  $P_h$ .

Analogously it is possible to prove that there is a decomposition of  $M_2 = M_2'' \{N_1/x_1, \dots, N_{k-r}/x_{k-r}, P_h/y_h\}$  and a slice  $sl_{c_k}$  in  $sls_2$  of  $M_2$ . Notice that  $cpt$  is a critical point of this box.

- If  $x \leq c_i$  then we set

$$\begin{aligned} X_2(c_j) &= X_1(x) + X_1(c_j) \\ X_2(c_i) &= X_1(c_i) - X_1(x) \\ X_2(x) &= 0 \\ X_2(c_k) &= X_1(x) + X_1(c_k). \end{aligned}$$

Then, if  $\Theta_1$  is the type of  $M_1$ ,

$$\begin{aligned} X_1(\Theta_1) &= !^{X_1(c_j)+X_1(c_i)} X_1(\Theta'_1) = \\ & \quad !^{X_1(c_j)+X_1(x)-X_1(x)+X_1(c_i)} X_2(\Theta'_1) = \\ & \quad \quad !^{X_2(c_j)+X_2(c_i)} X_2(\Theta'_1) = X_2(\Theta_1). \end{aligned}$$

Analogously for the type of  $M_2$ .

Moreover, by decomposition above, it is possible to partition the set of free variable occurrences of  $M_1$  in four subsets, depending on how the type is modified by boxes  $x, c_i, c_j$  and  $c_k$ . If  $y : !^{x+c_i+c_j+c_k} \Theta_y$  is a variable occurrence, then  $X_1(!^{x+c_i+c_j+c_k} \Theta_y) = X_2(!^{x+c_i+c_j+c_k} \Theta_y)$  and analogously for  $y : !^{c_i+c_j} \Theta_y, y : !^{x+c_j} \Theta_y$  and variables not affected by boxes  $x, c_i, c_j$  and  $c_k$ . The same considerations apply to free variable occurrences of  $M_2$ . Analogously we can partition the set of critical points of  $M_1$  and  $M_2$  in five subsets, depending on how the constraint of the critical point is modified by boxes  $x, c_i, c_j$  and  $c_k$ . If the constraint is of the form  $A^h - x - c_i - c_j - c_k = 0$ , then  $X_2$  satisfies the constraints since  $X_2(x) + X_2(c_i) + X_2(c_j) + X_2(c_k) = 0 + X_1(c_i) - X_1(x) + X_1(x) + X_1(c_j) + X_1(x) + X_1(c_k) = X_1(x) + X_1(c_i) + X_1(c_j) + X_1(c_k)$ . Analogously if the constraint is of the form  $A^h - c_j - c_i = 0$  then  $X_2(c_i) + X_2(c_j) = X_1(c_i) - X_1(x) + X_1(x) + X_1(c_j) = X_1(c_i) + X_1(c_j)$ . If the constraint is of the form  $A^h - c_i - c_k = 0$  then  $X_2(c_i) + X_2(c_k) = X_1(c_i) - X_1(x) + X_1(x) + X_1(c_k) = X_1(c_i) + X_1(c_k)$ . If the constraint is of the form  $A^h - c_j - x = 0$  then  $X_2(x) + X_2(c_j) = 0 + X_1(x) + X_1(c_j)$ . Finally if  $x, c_i, c_j$  and  $c_k$  doesn't appear in the constraint, then it is trivially satisfied by  $X_2$ .

- If  $x > c_i$  then we set  $X_2(c_j) = X_1(c_i) + X_1(c_j)$ ,  $X_2(c_i) = 0$ ,  $X_2(x) = X_1(x) - X_1(c_i)$  and  $X_2(c_k) = X_1(c_i) + X_1(c_k)$ . Following analogous reasonings to the previous case, the thesis holds.

In both cases either  $X_2(c_i)$  or  $X_2(x)$  is equal to 0 and then the boxes don't overlap anymore. Moreover notice that the box around the biggest subterm,  $c_j$ , overlaps another box  $x'$  if and only if  $x'$  either overlaps with  $c_i$  or with  $x$ . Then, in order to remove all overlappings, we start from the largest overlapping box, i.e. the box around the biggest subterm, and remove the overlapping with the procedure above. Since there is a finite number of boxes, in a finite number of steps, we obtain a box with no overlaps with the corresponding variable strictly increased. Since the sum of the variables doesn't change with the procedure above, in a finite number of steps we obtain the compatible decoration.  $\square$

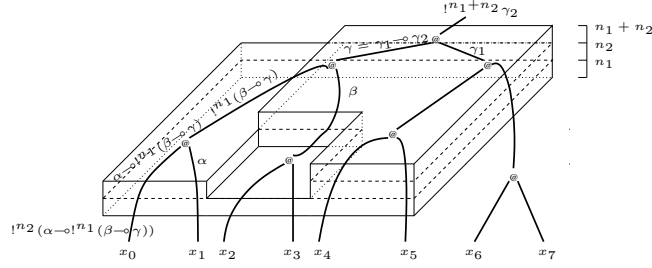


Figure 18: Boxes as levels.

The proof of the previous Lemma can be easily understood if we follow the intuition explained below with an example.

We may think of boxes as levels; boxing a subterm can then be seen as raising that subterm, as in Figure 18, 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

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

The complete decoration of Figure 18 can be produced in NEAL in two ways: by the instantiation of

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

and<sup>7</sup>

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

which are boxes belonging to two different derivations. Graphically such an instantiation can be represented as in the first row of Figure 19, 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).

The procedure by which we find the well stacked box configuration is visualized in Figure 19. The reader may imagine the boxes subject to gravity (the passage from the first to the second row of Figure 19) and able to fuse each other when they are at the same level (the little square in the third row fuses 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

<sup>7</sup>The correct legal terms should have all free variables inside 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  $!^{n_2} (((x_0 \ x_1)y)((x_4 \ x_5)w)) [x'_0/x_0, x'_1/x_1, (x_2 \ x_3)/y, x'_4/x_4, x'_5/x_5, (x_6 \ x_7)/w]$ ).

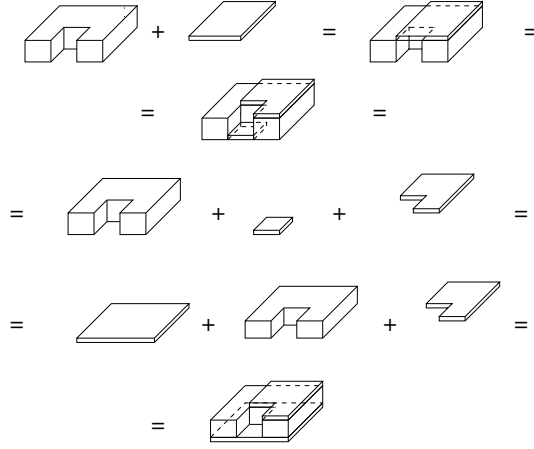


Figure 19: Equivalences of boxes.

application of the  $!$  rule. The “fusion operator” corresponds to the elimination of a cut between two exponential formulas. Moreover, the final configuration of Figure 19 corresponds to a particular solution of the set of constraints produced by the type synthesis algorithm, that instantiates the following boxes:

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

Finally, notice that during the procedure all types labeling 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 4 (Soundness).** *Let  $\mathcal{S}(M : \sigma) = \langle \Theta, B, A \rangle$ . For every  $X$  integer solution of  $A$ , there exists  $P$  candidate EAL-term such that  $P^* = M$  and  $X(B) \vdash_{\text{NEAL}} P : X(\Theta)$ .*

*Proof.* By induction on the structure of  $M$ , using the superimposing lemma. We first need a definition:

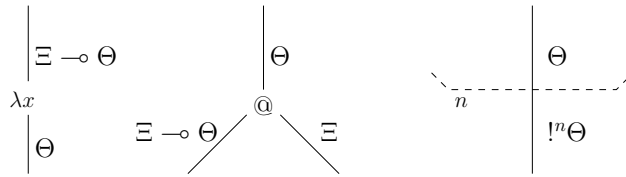


Figure 20: Type labels for decorated syntax trees.

**Definition 11.** *A syntax tree  $T$  is correctly decorated if the edges of the graph are labeled according to Figure 20 (in the rightmost picture,  $\Theta$  is inside  $n$  boxes).*

Moreover all edges connecting a variable  $x$  occurring multiple, are labeled with the same type  $!^n\Xi$ . In the case the variable is abstracted, the type label of variable is syntactically identical to the argument part of the type label of the edge at the root of the abstraction.

Given a correctly decorated syntax tree, and an instantiation  $X$  for the general EAL-types labeling its edges such that the number of exponentials for types of multiple variables is greater than 1, it is easy to build the corresponding NEAL derivation. Just use the Curry-Howard isomorphism and eventually apply a contraction before the  $\multimap$  introduction for binded variables and at the end of the derivation for free variables.

Thus, in order to prove soundness of our algorithm, it is sufficient to prove by structural induction on  $M$  that we can build a correctly decorated syntax tree. If the solution taken into account instantiates two overlapping boxes we use Lemma 10. Hence without loss of generality we can consider  $X$  such that all boxes are compatible. The only interesting part of the proof is the checking

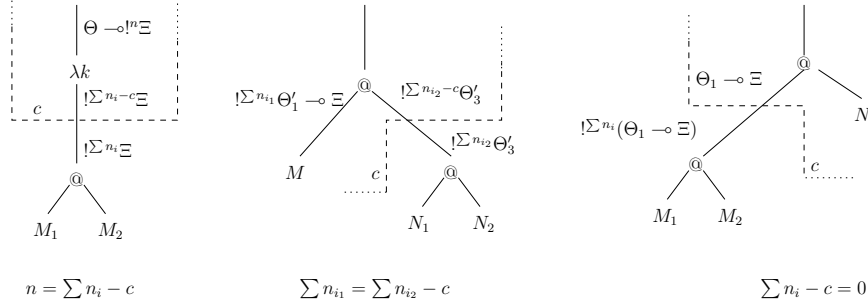


Figure 21: Decorations given by  $\mathbb{B}$ .

of the rules for  $\mathbb{B}$ . In Figure 21 it is shown how to build a correctly decorated syntax tree when the solution  $X$  instantiates a box passing through a critical point (all three cases of critical points are depicted).

Finally we need to prove that  $P$  is a candidate EAL-term. Points 2 and 3 of Definition 9 hold by construction of the NEAL derivation from the correctly decorated syntax tree, which also guarantees that  $P$  is in  $\{\textcircled{\lambda} - c, ! - c, c - c, \lambda - c, \text{dup}\}$ -normal form. Point 4 holds by definition of  $\mathbb{B}$ , and, finally, if  $P$  is not in  $! - !$ -normal, then by subject reduction exists  $P'$  such that the thesis holds.  $\square$

**Theorem 5 (Main theorem).** *Let  $M$  be a simply typeable  $\lambda$ -term. For any basis  $\Gamma$  and EAL formula  $C$ :*

$\Gamma \vdash_{\text{EAL}} M : C$  iff  $\mathcal{S}(M : \overline{C}) = \langle \Theta, B, A \rangle$  and  $A$  admits an integral solution  $X$  such that  $X(B) \subseteq \Gamma$  and  $C = X(\Theta)$ .

*Proof.*  $(\Rightarrow)$   $\Gamma \vdash_{\text{EAL}} M : C$  is established by a sharing graph where no fan node faces the root of a subgraph. It is ready to see that the corresponding EAL-term is a candidate EAL-term. Theorem 3 allows to conclude.

( $\Leftarrow$ ) By Theorem 4, there is an EAL-term  $P$  such that  $P^* = M$  and  $X(B) \vdash_{\text{NEAL}} P : X(\Theta)$ . The NEAL-term  $P$  codes a sharing graph establishing  $X(B) \vdash_{\text{EAL}} P^* : X(\Theta)$ .  $\square$

**Lemma 11.** *Let  $M$  be a simply typeable  $\lambda$ -term; let  $\sigma$  be its principal type schema, and let  $\tau$  be any other type for  $M$ . If  $\mathcal{S}(M : \tau) = \langle \Theta, B, A \rangle$  and  $A$  admits a solution  $X$ , then  $\mathcal{S}(M : \sigma) = \langle \Theta', B', A' \rangle$  and there exists  $X'$  solution of  $A'$ .*

*Proof.* We have to show that it is not the case that  $A$  admits a solution and  $A'$  is unsolvable. Constraints are added only by contraction (23) or unification (21). The former constraints depend only on the structure of the syntax tree of the term and hence they are not affected by the type change. As for the latter, changing  $\tau$  into  $\sigma$  makes some unification constraints disappear. In fact, it is possible to decompose  $\Theta$  in  $\Theta'\{x_1 \rightarrow \Sigma_1, \dots, x_n \rightarrow \Sigma_n\}$ . When the algorithm synthesizes  $M : \sigma$ , all unification constraints in  $A$  regarding  $\Sigma_1 \dots \Sigma_n$  disappear, and we obtain  $A'$  (up to renaming). In order to prove that  $A'$  is  $A$  minus the set of unification constraints produced by  $\Sigma_1 \dots \Sigma_n$ , it is sufficient to inspect the definitions of  $\mathcal{P}$  and  $\mathcal{U}$ . As the solution space has increased, it is not possible that  $A'$  has no solution.  $\square$

**Corollary 1.** *Let  $M$  be a simply typeable  $\lambda$ -term and let  $\sigma$  be its principal type schema. For any basis  $\Gamma$  and EAL formula  $C: \Gamma \vdash_{\text{EAL}} M : C$  iff  $\mathcal{S}(M : \sigma) = \langle \Theta, B, A \rangle$ ,  $A$  admits an integral solution  $X$  and there exists a substitution  $S$  from type variables to EAL-types such that  $S(X(B)) \subseteq \Gamma$  and  $S(X(\Theta)) = C$ .*

The corollary gives a weak notion of principal type for EAL. Any EAL type of a term arises as an instance of a solution of the constraints obtained for its simple principal type schema. The result, however, does not say anything on the structure of these !-decorated instances. The study of a general notion of principal schema for EAL is the subject of [CR03]. On the other hand, the corollary is enough to establish the decidability of type inference.

**Theorem 6.** *It is decidable whether, given a type-free  $\lambda$ -term  $M$ , there exist an EAL formula  $C$  and a basis  $\Gamma$  such that  $\Gamma \vdash_{\text{EAL}} M : C$ .*

## 4 Conclusions

We have presented an algorithm for assigning EAL types to type-free, pure  $\lambda$ -terms, obtained as the (technically non trivial) elaboration of the idea of “box decoration” of a simple type derivation. The complexity of the proposed algorithm is exponential in the size of the lambda term. The bound is essentially due to the rule (31) for the product union of sets of slices of critical points. Clearly the exponential bound is also an upper bound for the solution of the set of constraints. The algorithm is shown to be complete with respect to the notion of EAL types introduced in Definition 1. If we change the constraints collected by the algorithm, the same technique can be used to obtain linear



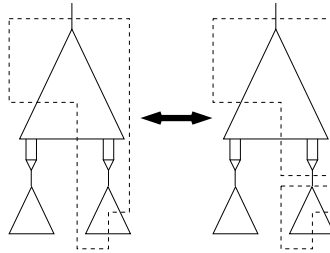


Figure 22: Box fusion for arbitrary contractions.

$\Gamma \vdash_{LL} B$  with inverted order of exponential rules (the proof is similar for the other cases).

The use of linear constraints allows now the use of linear programming techniques to obtain decorations with specific properties. By minimizing the objective function  $\sum_i n_i$ , we obtain decorations using a minimal number of boxes. Or, we may minimize only the use of  $\epsilon$  and  $\delta$  rules, if we minimize  $\sum_j (d_j + b_j) + \sum_k e_k$ . In the language of optimal reduction, these are decorations introducing a minimal number of brackets and croissants, and are thus the natural candidates to be used as initial translations for those  $\lambda$ -terms which does not have an EAL type.

## 4.2 Arbitrary contractions

Instead of using Definition 1, we may would like an algorithm complete with respect to the notion given directly by Figure 1, that is, allowing arbitrary contractions (and not only variable contractions) in the sharing graphs. Proceed as follows. Given a generic sharing graph, first decompose it into several subgraphs with the property that no fan faces a subgraph; then read them back, obtaining a set of lambda-terms. For example, the graph of Figure 4 of Section 1 can be decomposed in  $\lambda z.\lambda x.\lambda w.(k k w)$  and  $(x z)$ . After the decomposition, call the type synthesis algorithm separately on every subterm, calculate the suitable unification constraints with  $\mathcal{U}$ , collect all the constraints in a single system, and solve it.

This procedure computes all possible decorations, except those boxes that surround more than one subterm. However, the proof of the superimposing lemma allows to conclude that there is a decoration with a box around more than one subterm if and only if there exists a decoration with boxes only around a single subterm, with the same type (see Figure 22 for a graphical intuition).

## Acknowledgments

We are happy to thank Harry Mairson, for extended comments and criticism on previous versions of the paper; and Simona Ronchi della Rocca, for the many discussions, suggestions, and comments.

## References

- [ACM00] Andrea Asperti, Paolo Coppola, and Simone Martini. (Optimal) duplication is not elementary recursive. In *Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POLP-00)*, pages 96–107, N.Y., January 19–21 2000. ACM Press.
- [AG98] 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.
- [Asp98] Andrea Asperti. Light affine logic. In IEEE Computer Society, editor, *Proc. of Symposium on Logic in Computer Science*, pages 300–308, Indianapolis, Indiana, 1998.
- [Bai02] Patrick Baillot. Checking Polynomial Time Complexity with Types. In *Proc. of 2<sup>nd</sup> IFIP International Conference on Theoretical Computer Science (TCS 2002)*, volume 223 of *IFIP Conference Proceedings*, pages 370–382, Montréal, Québec, Canada, aug 2002.
- [Bai03] Patrick Baillot. Type inference for polynomial time complexity via constraints on words. Technical report, Laboratoire d’Informatique de Paris Nord, Université Paris 13, Institut Galilée, 2003.
- [BBdPH93] Nick Benton, Gavin Bierman, Valeria de Paiva, and Martin Hyland. A term calculus for intuitionistic linear logic. In M. Benzen and J.F. Groote, editors, *Typed Lambda Calculus and Applications. Int. Conference on Typed Lambda Calculus and Applications, TLCA ’93*, volume 664 of *Lecture Notes in Computer Science*, pages 75–90, March 1993.
- [CM01] 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.
- [CR03] P. Coppola and S. Ronchi Della Rocca. Principal Typing in Elementary Affine Logic. In *Proc. of Typed Lambda Calculi and Applications, 6th International Conference, TLCA 2003*, 2003. To appear.
- [DJ03] Vincent Danos and Jean-Baptiste Joinet. Linear logic and elementary time. *Information and Computation*, 183(1):123–137, 2003.
- [DJS95] 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.

- [Gir98] Jean-Yves Girard. Light linear logic. *Information and Computation*, 204(2):143–175, 1998.
- [Kat90] Vinod K. Kathail. *Optimal Interpreters for Lambda-calculus Based Functional Programming Languages*. PhD thesis, MIT, May 1990.
- [Lam90] 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.
- [Lév80] 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.
- [Mai92] Harry G. Mairson. A simple proof of a theorem of Statman. *Theoretical Computer Science*, 103(2):387–394, September 1992.
- [Pra65] Dag Prawitz. *Natural Deduction*. Almqvist & Wiksell, 1965.
- [Rov92] L. Roversi. A compiler from Curry-typed  $\lambda$ -terms to linear- $\lambda$ -terms. In *Theoretical Computer Science: Proceedings of the Fourth Italian Conference*, pages 330 – 344, L'Aquila (Italy), October 1992. World Scientific.
- [Rov98] Luca Roversi. A Polymorphic Language which is Typable and Polystep. In *Proceedings of the Asian Computing Science Conference (ASIAN'98)*, volume 1538 of *Lecture Notes in Computer Science*, pages 43 – 60, Manila (The Philippines), December 1998. Springer Verlag.
- [Sch94] Harold Schellinx. *The Noble Art of Linear Decorating*. PhD thesis, Institute for Logic, Language and Computation, University of Amsterdam, 1994.

## A Appendix

We have already observed that the simply typed lambda term

$$(\lambda n.(n \lambda y.(n \lambda z.y)) \lambda x.(x (x y))) : o$$

is not typeable in EAL. If one knows optimal reduction [AG98], this can be seen in a simple way, writing the term as a sharing graph and reducing it in the abstract algorithm by matching fans by labels (see Figure 23 where the redexes fired at every step are indicated by a dashed oval). The sharing graph in normal form is a *cycle*, that is a sharing graph which does not correspond to any  $\lambda$ -term

(least to say to  $y$ , which is the normal form of the given term). This means that the oracle *is* needed for the reduction of this term, and hence it cannot have a type in EAL.

We can give a formal proof, by calling the type inference algorithm on such a term. The following is a trace of the execution, where each box delimits the call and return of a single type inference rule:

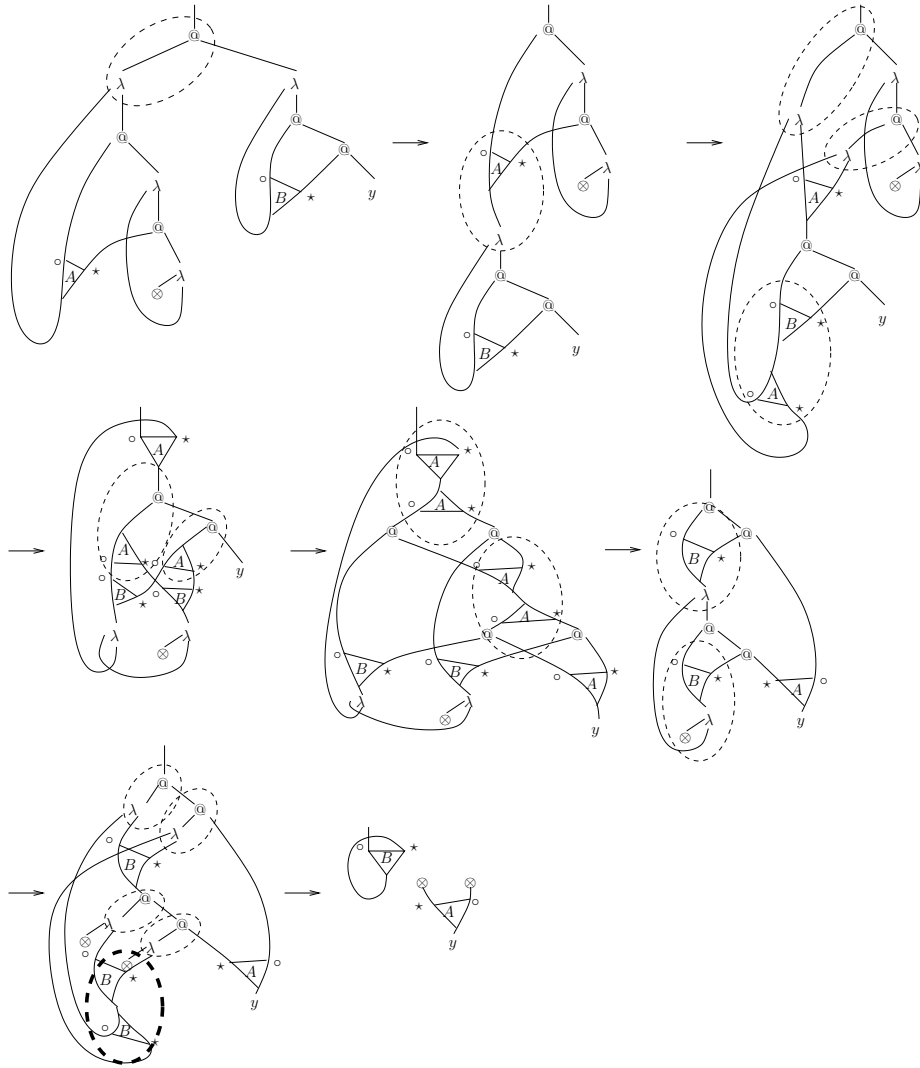


Figure 23: Incorrect reduction of  $(\lambda n.(n \lambda y.(n \lambda z.y)) \lambda x.(x (x y)))$ .

$$\mathcal{S}(\lambda n.(n \lambda y.(n \lambda z.y)) \lambda x.(x (x y))) : o$$

$$\mathcal{S}(\lambda n.(n \lambda y.(n \lambda z.y)) : ((o \rightarrow o) \rightarrow o) \rightarrow o)$$

$$\mathcal{S}((n \lambda y.(n \lambda z.y)) : o)$$

$$\mathcal{S}(n : (o \rightarrow o) \rightarrow o)$$

$$\mathcal{P}((o \rightarrow o) \rightarrow o) = p_1(p_2(p_3 \multimap p_4) \multimap p_5)$$

$$= \langle p_1(p_2(p_3 \multimap p_4) \multimap p_5), \{n : p_1(p_2(p_3 \multimap p_4) \multimap p_5)\}, \emptyset, \emptyset \rangle$$

$$\mathcal{S}(\lambda y.(n \lambda z.y) : o \rightarrow o)$$

$$\mathcal{S}(n \lambda z.y : o)$$

$$\mathcal{S}(n : (o \rightarrow o) \rightarrow o)$$

$$\mathcal{P}((o \rightarrow o) \rightarrow o) = p_6(p_7(p_8 \multimap p_9) \multimap p_{10})$$

$$= \langle p_6(p_7(p_8 \multimap p_9) \multimap p_{10}), \{n : p_6(p_7(p_8 \multimap p_9) \multimap p_{10})\}, \emptyset, \emptyset \rangle$$

$$\mathcal{S}(\lambda z.y : o \rightarrow o)$$

$$\mathcal{S}(y : o)$$

$$\mathcal{P}(o) = p_{11}$$

$$= \langle p_{11}, \{y : p_{11}\}, \emptyset, \emptyset \rangle$$

$$\mathbb{B}(y, \{y : p_{11}\}, p_{11}, \emptyset, \emptyset) = \langle \{y : p_{11}\}, p_{11}, \emptyset \rangle$$

$$\mathcal{P}(\alpha) = p_{12}$$

$$= \langle p_{12} \multimap p_{11}, \{y : p_{11}\}, \emptyset, \emptyset \rangle$$

$$\mathbb{B}(\lambda z.y, \{y : p_{11}\}, p_{12} \multimap p_{11}, \emptyset, \emptyset)$$

$$\mathcal{B}(\{y : p_{11}\}, p_{12} \multimap p_{11}, \emptyset, \emptyset) = \langle \{y : p_{11}\}, p_{12} \multimap p_{11}, \emptyset \rangle$$

$$= \langle \{y : b_1 + p_{11}\}, b_1(p_{12} \multimap p_{11}), \emptyset \rangle$$

$$\mathcal{U}(p_7(p_8 \multimap p_9), b_1(p_{12} \multimap p_{11})) = \begin{cases} p_7 = b_1 \\ p_8 = p_{12} \\ p_9 = p_{11} \end{cases}$$

$$= \left\langle p_{10}, \{n : p_6(p_7(p_8 \multimap p_9) \multimap p_{10}), y : b_1 + p_{11}\}, \begin{cases} p_7 = b_1 \\ p_8 = p_{12} \\ p_9 = p_{11} \\ p_6 = 0 \end{cases}, \emptyset \right\rangle$$

$$= \langle p_{10}, \{n : b_1(p_8 \multimap p_9) \multimap p_{10}, y : b_1 + p_9\}, \emptyset, \emptyset \rangle$$

$$\mathbb{B}((n \lambda z.y), \{n : b_1(p_8 \multimap p_9) \multimap p_{10}, y : b_1 + p_9\}, p_{10}, \emptyset, \emptyset)$$

$$= \langle \{n : b_2(b_1(p_8 \multimap p_9) \multimap p_{10}), y : b_2 + b_1 + p_9\}, b_2 + p_{10}, \emptyset \rangle$$

$$\mathcal{C}(b_2 + b_1 + p_9) = \emptyset$$

$$= \langle b_2 + b_1 + p_9 \multimap b_2 + p_{10}, \{n : b_2(b_1(p_8 \multimap p_9) \multimap p_{10})\}, \emptyset, \emptyset \rangle$$

$$\mathbb{B}(\lambda y.(n \lambda z.y), \{n : b_2(b_1(p_8 \multimap p_9) \multimap p_{10})\}, b_2 + b_1 + p_9 \multimap b_2 + p_{10}, \emptyset, \emptyset)$$

$$= \langle \{n : b_3 + b_2(b_1(p_8 \multimap p_9) \multimap p_{10})\}, b_3(b_2 + b_1 + p_9 \multimap b_2 + p_{10}), \emptyset \rangle$$

$$\mathcal{U}(p_2(p_3 \multimap p_4), b_3(b_2 + b_1 + p_9 \multimap b_2 + p_{10})) = \begin{cases} p_2 = b_3 \\ p_3 = b_2 + b_1 + p_9 \\ p_4 = b_2 + p_{10} \end{cases}$$

$$= \left\langle p_5, \left\{ \begin{array}{l} n : p_1(p_2(p_3 \multimap p_4) \multimap p_5), \\ n : b_3 + b_2(b_1(p_8 \multimap p_9) \multimap p_{10}) \end{array} \right\}, \begin{cases} p_2 = b_3 \\ p_3 = b_2 + b_1 + p_9 \\ p_4 = b_2 + p_{10} \\ p_1 = 0 \end{cases}, \emptyset \right\rangle$$

$$\begin{aligned}
&= \left\langle p_5, \left\{ \begin{array}{l} n : b_3(b_2 + b_1 + p_9 \multimap b_2 + p_{10}) \multimap p_5, \\ n : b_3 + b_2(b_1(p_8 \multimap p_9) \multimap p_{10}) \end{array} \right\}, \emptyset, \emptyset \right\rangle \\
&\mathbb{B} \left( (n \lambda y. (n \lambda z. y)), \left\{ \begin{array}{l} n : b_3(b_2 + b_1 + p_9 \multimap b_2 + p_{10}) \multimap p_5, \\ n : b_3 + b_2(b_1(p_8 \multimap p_9) \multimap p_{10}) \end{array} \right\}, p_5, \emptyset, \emptyset \right) \\
&= \left\langle \left\{ \begin{array}{l} n : b_4(b_3(b_2 + b_1 + p_9 \multimap b_2 + p_{10}) \multimap p_5), \\ n : b_4 + b_3 + b_2(b_1(p_8 \multimap p_9) \multimap p_{10}) \end{array} \right\}, b_4 + p_5, \emptyset \right\rangle \\
&\mathcal{C}(b_4(b_3(b_2 + b_1 + p_9 \multimap b_2 + p_{10}) \multimap p_5), b_4 + b_3 + b_2(b_1(p_8 \multimap p_9) \multimap p_{10})) \\
&= \left\{ \begin{array}{l} b_4 \geq 1 \\ b_4 = b_4 + b_3 + b_2 \\ b_3 = b_1 \\ b_2 + b_1 + p_9 = p_8 \\ b_2 + p_{10} = p_9 \\ p_5 = p_{10} \end{array} \right. = \left\{ \begin{array}{l} b_4 \geq 1 \\ b_3 = 0 \\ b_2 = 0 \\ b_1 = 0 \\ p_8 = p_5 \\ p_9 = p_5 \\ p_{10} = p_5 \end{array} \right. \\
&= \langle b_4((p_5 \multimap p_5) \multimap p_5) \multimap b_4 + p_5, \emptyset, \{b_4 \geq 1\}, \emptyset \rangle
\end{aligned}$$

$$\begin{aligned}
&\mathcal{S}(\lambda x. (x (x y)) : (o \rightarrow o) \rightarrow o) \\
&\mathcal{S}((x (x y)) : o) \\
&\mathcal{S}(x : o \rightarrow o) \\
&= \langle p_1(p_2 \multimap p_3), \{x : p_1(p_2 \multimap p_3)\}, \emptyset, \emptyset \rangle \\
&\mathcal{S}((x y) : o) \\
&\mathcal{S}(x : o \rightarrow o) \\
&= \langle p_4(p_6 \multimap p_7), \{x : p_4(p_6 \multimap p_7)\}, \emptyset, \emptyset \rangle \\
&\mathcal{S}(y : o) \\
&= \langle p_8, \{y : p_8\}, \emptyset, \emptyset \rangle \\
&\mathcal{U}(p_6, p_8) = \{p_6 = p_8\} \\
&= \langle p_7, \{x : p_6 \multimap p_7, y : p_6\}, \emptyset, \emptyset \rangle \\
&\mathbb{B}((x y), \{x : p_6 \multimap p_7, y : p_6\}, p_7, \emptyset, \emptyset) \\
&= \langle \{x : b_1(p_6 \multimap p_7), y : b_1 + p_6\}, b_1 + p_7, \emptyset \rangle \\
&\mathcal{U}(b_1 + p_7, p_2) = \{b_1 + p_7 - p_2 = 0\} \\
&sls = \left\{ \left( b_1 + p_7 - p_2 = 0, \left\{ \begin{array}{l} x : b_1(p_6 \multimap p_7), \\ y : b_1 + p_6 \end{array} \right\} \right) \right\} \\
&= \left\langle p_3, \left\{ \begin{array}{l} x : p_2 \multimap p_3, \\ x : b_1(p_6 \multimap p_7), \\ y : b_1 + p_6 \end{array} \right\}, \{b_1 + p_7 - p_2 = 0\}, sls \right\rangle \\
&\mathbb{B} \left( (x (x y)), \left\{ \begin{array}{l} x : p_2 \multimap p_3, \\ x : b_1(p_6 \multimap p_7), \\ y : b_1 + p_6 \end{array} \right\}, p_3, sls, \{b_1 + p_7 - p_2 = 0\} \right) \\
&\mathcal{B} \left( \left\{ \begin{array}{l} x : p_2 \multimap p_3, \\ x : b_1(p_6 \multimap p_7), \\ y : b_1 + p_6 \end{array} \right\}, p_3, sls, \{b_1 + p_7 - p_2 = 0\} \right) \\
&= \left\langle \left\{ \begin{array}{l} x : b_2(p_2 \multimap p_3), \\ x : b_1(p_6 \multimap p_7), \\ y : b_1 + p_6 \end{array} \right\}, b_2 + p_3, \{b_1 + p_7 - p_2 - b_2 = 0\} \right\rangle \\
&= \left\langle \left\{ \begin{array}{l} x : b_3 + b_2(p_2 \multimap p_3), \\ x : b_3 + b_1(p_6 \multimap p_7), \\ y : b_3 + b_1 + p_6 \end{array} \right\}, b_3 + b_2 + p_3, \{b_1 + p_7 - p_2 - b_2 = 0\} \right\rangle
\end{aligned}$$

$$\begin{aligned}
& \mathcal{C}(b_3 + b_2(p_2 \multimap p_3), b_3 + b_1(p_6 \multimap p_7)) = \begin{cases} b_3 + b_2 \geq 1 \\ b_3 + b_2 = b_3 + b_1 \\ p_2 = p_6 \\ p_3 = p_7 \end{cases} = \begin{cases} b_3 + b_2 \geq 1 \\ b_2 = b_1 \\ p_2 = p_6 \\ p_3 = p_7 \end{cases} \\
& = \left\langle \begin{cases} b_3 + b_2(p_2 \multimap p_3) \multimap b_3 + b_2 + p_3, \{y : b_3 + b_1 + p_6\}, \\ b_1 + p_7 - p_2 - b_2 = 0 \\ b_3 + b_2 \geq 1 \\ b_2 = b_1 \\ p_2 = p_6 \\ p_3 = p_7 \end{cases}, \emptyset \right\rangle \\
& = \langle b_3 + b_1(p_2 \multimap p_2) \multimap b_3 + b_1 + p_2, \{y : b_3 + b_1 + p_2\}, \{b_3 + b_1 \geq 1\}, \emptyset \rangle \\
& \mathbb{B}(\lambda x.(x(x\ y)), \{y : b_3 + b_1 + p_2\}, b_3 + b_1(p_2 \multimap p_2) \multimap b_3 + b_1 + p_2, \emptyset, \{b_3 + b_1 \geq 1\}) \\
& = \langle \{y : b_2 + b_3 + b_1 + p_2\}, b_2(b_3 + b_1(p_2 \multimap p_2) \multimap b_3 + b_1 + p_2), \{b_3 + b_1 \geq 1\} \rangle \\
& \mathcal{W}(b_4((p_5 \multimap p_5) \multimap p_5, b_2(b_3 + b_1(p_2 \multimap p_2) \multimap b_3 + b_1 + p_2))) \\
& = \begin{cases} b_4 = b_2 \\ p_5 = p_2 \\ p_5 = b_3 + b_1 + p_2 \end{cases} = \begin{cases} b_4 = b_2 \\ p_5 = p_2 \\ b_3 + b_1 = 0 \end{cases}
\end{aligned}$$

Notice that the last constraint  $b_3 + b_1 = 0$  is incompatible with the previous  $b_3 + b_1 \geq 1$  hence the set of solutions is empty.