

# Type inference for polynomial time complexity via constraints on words

Patrick Baillot  
CNRS - Laboratoire d'Informatique de Paris-Nord  
Université Paris-Nord, Villetaneuse, France.  
pb@lipn.univ-paris13.fr

## Abstract

Light Affine Logic (LAL) is a system due to Girard and Asperti capturing the complexity class P in a proof-theoretical approach based on linear logic. LAL provides a typing for lambda-calculus which guarantees that a well-typed program is executable in polynomial-time on any input. We prove that the LAL type inference problem for lambda-calculus is decidable (for propositional LAL). To establish this result we reformulate the type-assignment system into an equivalent one which makes use of subtyping and is more flexible. We then use a reduction to a satisfiability problem for a system of inequations on words over a binary alphabet, for which we provide a solving algorithm.

Functional languages have been advocated as languages amenable to reasoning on programs and specifications. However though a lot of work has been done on techniques for checking *qualitative* issues such as the fact that a program meets its specification, there seems to have been less success on *quantitative* ones such as how to structurally ensure that a program fits a certain time or space complexity bound. Maybe this means that some conceptual tools are still needed on the foundational side, lambda-calculus, logic and rewriting systems, for handling quantitative aspects.

Since the last decade quite a lot of progress was done in the field of *implicit complexity* aiming at defining languages and calculi in which all programmable functions have a given complexity ([14, 11, 13, 4, 12]). Some of them are based on restrictions on the use of structural recursion, others on proof-theoretical methods or on type systems.

Here we are interested in the type-theoretic approach for control of complexity. More specifically we consider *light affine logic* (LAL) ([1]), a variant of linear logic (LL) which can be used as a type system for lambda-calculus: it offers the property that all well-typed programs are PTIME (it was obtained as a simplification of light linear logic, [11]). We stick to lambda-calculus in a first step to establish the basic results expected on a sound theoretical ground, but with the idea of enriching it afterwards into a more realistic functional language.

Light affine logic has been mainly studied from the logical point of view (but for a few works as [17], [18]) and the question of decidability of type-inference for lambda-calculus terms was still open. Our main contribution is to turn it into a (potentially) usable type system by establishing decidability of typability for the propositional fragment (without polymorphism). To do so we propose a new type-assignment system which is equivalent to the former one but takes advantage of subtyping to allow more flexibility. We address the problem of type-inference by applying the well-studied approach of typing by constraints generation and solving (see for instance [15]). These constraints we need to handle are inequations on words over a binary alphabet. We isolate an important property of the systems generated in our situation which makes it possible to give a resolution algorithm.

In this paper we focus on establishing decidability of type-inference, which is already non trivial. We leave for future work the investigation of the complexity of this problem and the search for practically efficient methods.

**Related work.** We already considered the problem of typability in [3] but in a restricted setting: the term had to be in normal form and a type was fixed for the argument. With these conditions we had to deal with Presburger arithmetic constraints.

The approach we follow here is close to that proposed in [7] for type inference in Elementary Affine Logic (EAL): propose a pattern of type-derivation with free parameters and express its correctness by a system of constraints (linear inequations over integers in the case of EAL). This paper also defines a notion of principal typing in EAL. Decidability of type inference for lambda-terms in EAL had been established in [6].

**Outline.** After recalling the principles of LAL in section 1 we give the natural LAL type assignment system for lambda-calculus (section 2), define the subtyping relation and propose our reformulation of type-assignment with subtyping. Words appear as modalities in types allowing for the control of duplication. We then consider abstract derivations and abstract terms (section 4), where a degree of liberty is left for the modalities by leaving some free word parameters. An abstract derivation can be instantiated into a plain derivation provided some constraints on parameters (words) are satisfied (derivation instantiation problem). We show how typability can be reduced to the derivation instantiation problem for some derivations in canonical form. In section 5 we establish how to solve the constraints to decide the previous problem. This amounts to solve systems of inequations on words.

**Acknowledgement.** Thanks to Roberto Amadio for suggesting the use of subtyping for LAL typing, and to François Pottier and Kazushige Terui for useful discussions.

**Notations.** Length of a word  $s$ :  $|s|$ . Concatenation of word  $s$  with word  $s'$ :  $ss'$ . Syntactic substitution in a lambda-term:  $t\{u/x\}$ . We denote by  $\rightarrow^\eta$  the reflexive and transitive  $\eta$ -reduction relation on lambda-terms.

## 1 Introduction to Light affine logic

We start with an informal introduction to the principles of Light affine logic (LAL). Throughout this paper, by Light affine logic we mean in fact Intuitionistic light affine logic.

LAL controls the complexity of reduction of a term (or proof) by enforcing a strict discipline on the duplication of subterms. It relies on two key features:

1. *stratification*,
2. two modalities (called *exponentials*):  $!$  and  $\S$ .

Point 1 means that a typed term is organized into strata or levels. This organization is static: if a subterm is initially at level  $i$ , its reducts will remain so during execution. Moreover if a term  $t$  is fed with an argument  $a$  (figures 1, 2) then in the resulting term  $b$ , level  $i$  will only depend on the levels  $j$  for  $j \leq i$  of  $t$  and  $a$  (see [2] for a semantical interpretation of this property).

How do we change level in a term? This is done with the modalities: applying a  $!$  to a typed term  $t : B$  at level  $i$  we get a term at level  $i + 1$ ; this term of type  $!B$  can then be duplicated during reduction (figure 3). The  $!$  modality therefore has 2 roles: switching level and allowing duplication.

Another system is based on stratification and the  $!$  modality, Elementary Affine Logic; it guarantees Kalmar elementary complexity for the terms. LAL needs to be more strict to cut down the complexity to polynomial time. Hence it requires that for applying  $!$  to a term  $t$  (thus making  $t$  duplicable) the term should have *at most one free variable*. This is a way of preventing chains of duplications leading to exponentially long sequences of reductions.

However one has to switch levels also for terms with more than one free variable. This is what the new modality  $\S$  is introduced for. Applying  $\S$  to a typed term  $t : B$  at level  $i$  we get a term  $t' : \S B$  at level  $i + 1$ , but this new term is *not* duplicable. Still, one advantage of  $t' : \S B$  is that it allows identification of free variables (with same types) and in this way enables the duplication of other terms (figure 4).

## 2 Typing in LAL

### 2.1 Type system

We want to type lambda-terms in light affine logic. Recall lambda-calculus terms are defined by:

$$t ::= x \mid \lambda x t \mid (t)t.$$

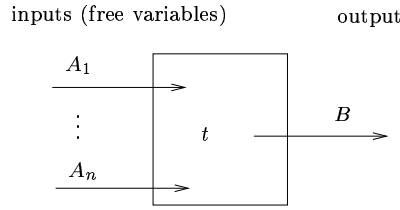


Figure 1: Representation of a typed term.

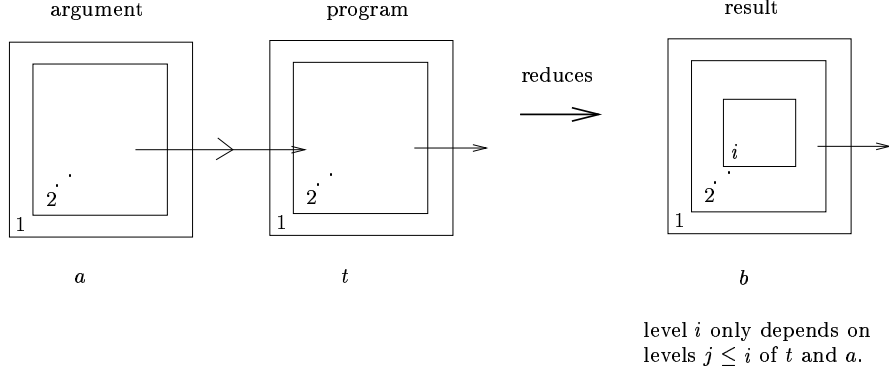


Figure 2: Stratification.

LAL types are given by the following grammar:

$$T := \alpha | T \multimap T | !T | \S T.$$

The  $!$  (*bang*) and  $\S$  (*paragraph*) connectives are called *exponentials*.

We use a natural deduction presentation of the type-assignment system in the lines of [6, 5] (it can also be presented in a sequent calculus style as in [1, 3]). This formulation is not as well adapted as that of *proof-nets* ([1]) to the study of reduction, but it is easier to understand for typing. The rules are given on fig.5.

Conditions with figure 5:

(1) The  $a_i$ s belong to  $\{!, \S\}$  and satisfy: if  $n \geq 2$  then  $a_0 = \S$  and if  $a_0 = !$  and  $n = 1$  then  $a_1 = !$ .

The rule (*prom.*) is called promotion. If  $a_0 = !$  (resp.  $a_0 = \S$ ) we say it is a  $!$ -promotion (resp.  $\S$ -promotion). Note that condition (1) includes the restriction described in section 1: one can apply a  $!$ -promotion only if the term has at most one free variable.

The rule (*prom.*) is important as it is the only one to change the level : the level of term  $t$  increases by 1; this is displayed on the type by adding the exponential  $a_0$ .

A particular case of application of (*prom.*) is that where the  $n$  left premises  $\Gamma_i \vdash t_i : a_i A_i$  are of the form  $x_i : a_i A_i \vdash x_i : a_i A_i$ ; in that case we can simply write the application of the rule as:

$$\frac{x_1 : A_1, \dots, x_n : A_n \vdash t : B}{x_1 : a_1 A_1, \dots, x_n : a_n A_n \vdash t : a_0 B} \text{ (prom.)}$$

Observe that (*prom.*) acts both on the type of the term and on those of the free variables, adding one modality to each. In the case of  $\S$ -promotion, if  $n \geq 2$  and say  $a_1 A_1 = a_2 A_2 = !A$  for example we can then apply a contraction on  $x_1$  and  $x_2$ .

As expected we have:

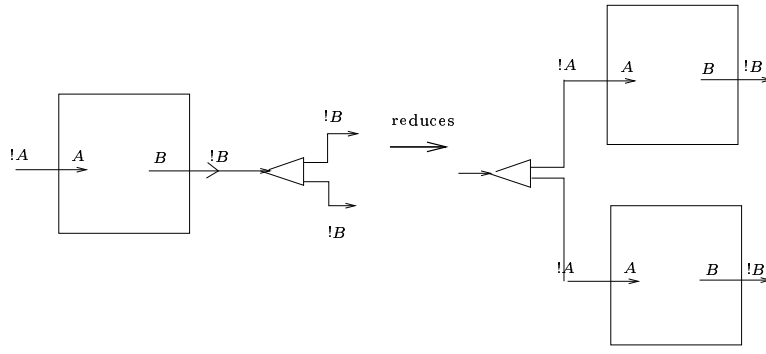


Figure 3: ! allows being duplicated.

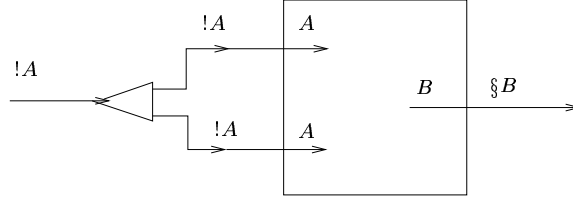


Figure 4: § enables identification of variables.

**Lemma 1** *LAL satisfies the subject reduction property and the following rule is derivable:*

$$\frac{\Gamma \vdash t' : A \quad x : A, \Delta \vdash t : B}{\Gamma, \Delta \vdash t\{t'/x\} : B} \text{ (subst.)}$$

LAL can be seen as a refinement of simple types. Indeed if we denote by *IL* (intuitionistic logic) the system of simple types, there is a forgetful map  $[\cdot] : LAL \rightarrow IL$ , obtained by erasing exponentials and replacing  $\multimap$  with  $\rightarrow$ . At the level of derivations we have:

**Lemma 2** *If  $\vdash_{LAL} t : A$  then  $\vdash_{IL} t : [A]$ . Moreover in that case there exists  $A'$  in LAL such that  $\vdash_{LAL} t : A'$  and  $[A']$  is the principal type of  $t$  in *IL*.*

The main property of LAL-typed terms is the following one, which is a consequence of the results of [11, 1]:

**Proposition 3** *If  $\vdash_{LAL} t : A \multimap B$  then there exists a polynomial  $P$  such that:*

*for any  $u$  such that  $\vdash_{LAL} u : A$ , the term  $(t)u$  can be evaluated in  $P(|u|)$  steps, where  $|u|$  denotes the size of  $u$ .*

$\frac{}{x : A \vdash x : A} \text{ (var.)}$	$\frac{\Gamma \vdash t : B}{\Gamma, x : A \vdash t : B} \text{ (weak.)}$
$\frac{\Gamma_1 \vdash t_1 : A \multimap B \quad \Gamma_2 \vdash t_2 : A}{\Gamma_1, \Gamma_2 \vdash (t_1 t_2) : B} \text{ (appl.)}$	$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x. t : A \multimap B} \text{ (abst.)}$
$\frac{\Gamma_1 \vdash t_1 : a_1 A_1 \cdots \Gamma_n \vdash t_n : a_n A_n \quad x_1 : A_1, \dots, x_n : A_n \vdash t : B}{\Gamma_1, \dots, \Gamma_n \vdash t\{t_i/x_i\} : a_0 B} \text{ (1)(prom.)}$	
$\frac{\Gamma \vdash t' : !A \quad x_1 : !A, \dots, x_n : !A, \Delta \vdash t : B}{\Gamma, \Delta \vdash t\{t'/x_1, \dots, x_n\} : B} \text{ (contr.)}$	

Figure 5: Type assignement LAL.

With second-order quantifiers there is also a completeness result (see [16]). Here we stick to the propositional fragment however, though it is not as expressive, because polymorphism brings problems of its own for type inference (recall type inference for system F is undecidable, [19]).

In appendix A we give some examples of typed terms acting on lists.

## 2.2 Modalities and subtyping

When typing lambda-terms we have to apply to certain types several  $!$  /  $\S$ . For instance we might want to identify two variables  $x_1, x_2$  in a subterm  $t : A$ , which leads us to give type  $\S A$  to  $t$ , and then make  $t$  duplicable, which requires giving the type  $! \S A$ .

Observe that if  $t$  can be typed with  $!A$  then it can be typed with  $\S A$ ; it is sufficient to replace in the derivation a  $!$ -promotion by a  $\S$ -promotion. Similarly a term with type  $!! \S A$  can also be attributed the type  $! \S A$  or  $\S ! \S A$  for example. More generally, to study typability it is useful to be able to state which types can be replaced by which ones. For that we will define a partial order on words over  $\{!, \S\}$ .

We consider  $\mathcal{L} = \{!, \S\}^*$  and call its elements *modalities*. We define an order  $\preceq$  on  $\mathcal{L}$  by:

$$\begin{array}{lcl} ! & \preceq & \S \\ au & \preceq & a'u' \quad \text{with } a, a' \in \{!, \S\} \text{ iff} \\ & & (a \preceq a' \text{ and } u \preceq u') \\ s & \preceq & \epsilon \quad \text{iff } s = \epsilon. \end{array}$$

Note that  $s \preceq s'$  implies  $|s| = |s'|$ .

For  $a \in \{!, \S\}$  we write  $a^k$  for  $a \dots a$  with  $k$  repetitions.

By applying repetitively the (*prom.*) rule (in the case  $n = 0$  or  $n = 1$ ) one can derive the following rule, for  $s_1, s_0$  in  $\mathcal{L}$  such that  $s_0 \preceq s_1$ :

$$\frac{x_1 : A_1 \vdash t : B}{x_1 : s_1 A_1 \vdash t : s_0 B} \text{ (prom.')}$$

For any value of  $n$  and  $s_i$  in  $\mathcal{L}$  ( $1 \leq i \leq n$ ) of length  $k \geq 0$  we can derive:

$$\frac{x_1 : A_1, \dots, x_n : A_n \vdash t : B}{x_1 : s_1 A_1, \dots, x_n : s_n A_n \vdash t : \S^k B} \text{ (prom.'')}$$

We call (*prom.'*) and (*prom.ii*) multiple promotions.

We will consider variables for words; we distinguish two classes of variables:

- *bicolored* variables, denoted as  $u, v \dots$  are valued in  $\mathcal{L}$ ,
- *monocolored* variables, denoted as  $p, q, r \dots$  are valued in  $\{\S\}^*$ .

Let these classes be denoted respectively as  $\mathcal{V}_b$  and  $\mathcal{V}_m$ , and  $\mathcal{V} = \mathcal{V}_b \cup \mathcal{V}_m$ . Of course a monocolored word is equivalently given by its length.

We consider the reflexive relation on types given by :

$$\begin{array}{lcl} u\alpha & \leq & u'\alpha \quad \text{iff } u \preceq u' \\ (A_1 \multimap A_2) & \leq & (A'_1 \multimap A'_2) \quad \text{iff} \\ & & A'_1 \leq A_1 \quad \text{and } A_2 \leq A'_2 \\ u(A_1 \multimap A_2) & \leq & u'(A'_1 \multimap A'_2) \quad \text{iff} \\ u \preceq u' \quad \text{and} & (A_1 \multimap A_2) & \leq (A'_1 \multimap A'_2) \end{array}$$

In fact we have:

**Lemma 4** *If  $A_1 \leq A_2$  then there exists a term  $t$  such that  $x : A_1 \vdash_{LAL} t : A_2$  and  $t \rightarrow^\eta x$ , so  $\lambda x.t \rightarrow^\eta \lambda x.x$ .*

$\frac{}{x : A_1 \vdash x : A_2} A_1 \leq A_2 \text{ (var.)}$	$\frac{\Gamma \vdash t : B}{\Gamma, x : A \vdash t : B} \text{ (weak.)}$
$\frac{\Gamma_1 \vdash t_1 : A_1 \multimap B \quad \Gamma_2 \vdash t_2 : A_2}{\Gamma_1, \Gamma_2 \vdash (t_1 t_2) : B} A_2 \leq A_1 \text{ (appl.)}$	$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x. t : A \multimap B} \text{ (abst.)}$
$\frac{\Gamma_1 \vdash t_1 : a_1 A'_1 \cdots \Gamma_n \vdash t_n : a_n A'_n \quad x_1 : A_1, \dots, x_n : A_n \vdash t : B}{\Gamma_1, \dots, \Gamma_n \vdash t\{t_i/x_i\} : a_0 B} a_i A'_i \leq a_0 A_i \text{ (prom.)(1)}$	
$\frac{\Gamma \vdash t' : !A \quad x_1 : A_1, \dots, x_n : A_n, \Delta \vdash t : B}{\Gamma, \Delta \vdash t\{t'/x_1, \dots, x_n\} : B} !A \leq A_i \text{ (contr.)}$	

Figure 6: Type assignment LALs.

**Proof :** The proof is by induction over  $A_1$ . If  $A_1 = u_1 A'_1$  and  $A_2 = u_2 A'_2$  with  $u_1 \preceq u_2$  and  $A'_1 \leq A'_2$ , then by i.h.:  $x : A'_1 \vdash_{LAL} t : A'_2$  with  $t \rightarrow^\eta x$ . By applying the (*prom.*) we described before we get:  $x : A_1 \vdash_{LAL} t : A_2$ .

Consider the case  $A_1 = B_1 \multimap B_2$ ,  $A_2 = B'_1 \multimap B'_2$ . We have:  $B_2 \preceq B'_2$ ,  $B'_1 \preceq B_1$ . By i.h. we get  $y : B'_1 \vdash_{LAL} t_1 : B_1$ ,  $x : B_2 \vdash_{LAL} t_2 : B'_2$  with  $t_1 \rightarrow^\eta y$ ,  $t_2 \rightarrow^\eta x$ . We can then easily derive the following judgement:  $z : B_1 \multimap B_2 \vdash_{LAL} \lambda y. (\lambda x. t_2)(z)t_1 : B'_1 \multimap B'_2$ . As subject reduction is satisfied we deduce that for  $t = \lambda y. t_2\{(z)t_1/x\}$  we have:  $z : B_1 \multimap B_2 \vdash_{LAL} t : B'_1 \multimap B'_2$ . Moreover:  $\lambda y. t_2\{(z)t_1/x\} \rightarrow^\eta \lambda y. (z)t_1 \rightarrow^\eta \lambda y. (z)y \rightarrow^\eta z$ .  $\square$

This suggests considering  $\leq$  as a subtyping relation. Now we can reformulate our type-assignment system using this relation: see the system LALs (LAL with subtyping) on fig.6.

Conditions with figure 6:

(1) if  $n \geq 1$  then  $a_0 = \S$ ,

Observe that LAL rules can be seen as particular cases of LALs rules. We have:

**Proposition 5** *If  $\Gamma \vdash_{LAL} t : A$  then  $\Gamma \vdash_{LALs} t : A$ .*

*Conversely, if  $\Gamma \vdash_{LALs} t : A$  then there exists  $t'$  such that  $t' \rightarrow^\eta t$  and  $\Gamma \vdash_{LAL} t' : A$ .*

The proof is given in appendix B.

### 3 Constraints

Before going on with typing, let us define the constraints we will need to consider. An *inequation on words*  $I$  is a constraint of the following form:

$$a_1 \dots a_k \preceq a_{k+1} \dots a_l \quad (I)$$

where the  $a_i$ s are constants or word variables:  $a_i \in \mathcal{V} \cup \mathcal{L}$ .

We denote by  $s_1, s_2 \dots$  words over  $\mathcal{V} \cup \mathcal{L}$ , so an inequation is of the form  $s_1 \preceq s_2$ . An *inequation system*  $\mathcal{S}$  is a finite conjunction of inequations:

$$\mathcal{S} = I_1 \wedge \dots \wedge I_N.$$

Given  $I$  (resp.  $\mathcal{S}$ ),  $Par(I)$  (resp.  $Par(\mathcal{S})$ ) is the set of word variables (or *parameters*) occurring in  $I$  (resp.  $\mathcal{S}$ ).

An *instantiation*  $\phi$  of  $\mathcal{S}$  is a map  $\phi : Par(\mathcal{S}) \rightarrow \mathcal{L}$  such that for any  $p$  in  $Par(\mathcal{S}) \cap \mathcal{V}_m$ ,  $\phi(p) \in \{\S\}^*$  ( $\phi$  is *compatible with colors*).

We also denote by  $\phi$  the extension to  $(Par(\mathcal{S}) \cup \mathcal{L})^*$  given by:  $\phi(a) = a$  if  $a \in \mathcal{L}$  and  $\phi(a_1 \dots a_k) = \phi(a_1) \dots \phi(a_k)$ .

An instantiation  $\phi$  of  $\mathcal{S} = I_1 \wedge \dots \wedge I_N$  is a *solution* of  $\mathcal{S}$  if for any  $1 \leq j \leq N$  the inequation  $I_j$  holds when each variable  $a$  is replaced by  $\phi(a)$ .

**Example 1** *Consider  $\mathcal{S}$  given by:*

$$\begin{aligned} u_1 \S u_2 &\preceq p_1 u_3! \\ u_5 u_4 &\preceq u_2! \end{aligned}$$

*The instantiation  $\phi$  given by:  $\phi(p_1) = \S$ ,  $\phi(u_1) = \phi(u_3) = \epsilon$ ,  $\phi(u_2) = \phi(u_4) = \phi(u_5) = !$ , is a solution of  $\mathcal{S}$ .*

$$\begin{aligned}
\mathcal{T}(T_1 \leq T_2) &= \text{false if } [T_1] \neq [T_2] \\
\mathcal{T}(s\alpha \leq s'\alpha) &= (s \preceq s') \\
\mathcal{T}(A_1 \multimap A_2 \leq A'_1 \multimap A'_2) &= \mathcal{T}(A'_1 \leq A_1) \wedge \mathcal{T}(A_2 \leq A'_2) \\
\mathcal{T}(s(A_1 \multimap A_2) \leq s'(A'_1 \multimap A'_2)) &= (s \preceq s') \wedge \mathcal{T}(A'_1 \leq A_1) \wedge \mathcal{T}(A_2 \leq A'_2).
\end{aligned}$$

Figure 7: Map  $\mathcal{T}$

**Remark 6** Note that equations over words (on the binary alphabet) with concatenation (as considered in [10]) can be seen as a special case: an equation  $s_1 = s_2$  can be encoded as  $(s_1 \preceq s_2) \wedge (s_2 \preceq s_1)$ .

## 4 Abstract typing

Finding a LALs type derivation for a term  $t$  brings up two difficulties:

- finding the *skeleton* of the derivation, in particular where to do the contractions and the (multiple) promotions;
- working out how many modalities to apply at each multiple promotion and choose between ! and § for each.

To address the second point we will use types with variables instead of modalities (called *abstract types*) and then try to find suitable modalities to instantiate the variables.

As to the first point we will show that if we consider abstract types, then we can enumerate all possible derivation skeletons for a term (section 4.2). After that we will be ready to describe our type inference method.

### 4.1 Abstract types

Let us call *abstract types* types built with word variables:

$$T := \alpha | T \multimap T | aT,$$

where  $a$  belongs to  $\mathcal{V} \cup \mathcal{L}$ .

As with LAL there is a forgetful map from abstract types to simple types, which we denote again as  $[\cdot]$ . Denote by  $Par(T)$  the set of word variables appearing in  $T$ .

Given an abstract type  $A$  we denote by  $\widehat{A}$  the abstract type obtained by removing the external modalities and variables of  $A$ :  $\widehat{A}$  is defined inductively by:

$$\begin{aligned}
\widehat{A} &= A \quad \text{if } A = \alpha \text{ or } A_1 \multimap A_2; \\
\widehat{(aA)} &= \widehat{A}.
\end{aligned}$$

Given an *instantiation*  $\phi : Par(T) \rightarrow \mathcal{L}$  compatible with colors, we define  $\phi(T)$  as the LAL type obtained by replacing in  $T$  word variables by their image.

**Inequations on abstract types.** Given 2 abstract types  $T_1$  and  $T_2$ , a solution of the inequation  $T_1 \leq T_2$  is an instantiation  $\phi : Par(T_1) \cup Par(T_2) \rightarrow \mathcal{L}$  such that  $\phi(T_1) \leq \phi(T_2)$ .

We define a map  $\mathcal{T}$  from inequations on abstract types to systems of inequations on words (it follows directly from the definition of the subtyping relation): see figure 7.

**Lemma 7** A map  $\phi$  is a solution of the set of inequations on abstract types  $\{A_1 \leq A_2, \dots, A_{2k+1} \leq A_{2k+2}\}$  iff it is a solution of the system  $\bigwedge_{i=0}^k \mathcal{T}(A_{2i+1} \leq A_{2i+2})$ .

An *abstract type derivation* (a.t.d.)  $\mathcal{D}$  is a derivation of judgements with abstract types, built from the rules in figure 8.

- Conditions: in (prom.): if  $n \geq 2$  then  $v \in \mathcal{V}_m$  else  $v \in \mathcal{V}_b$ .  
in (contr.):  $u \in \mathcal{V}_b$  and *fresh* (not appearing in the rest of the derivation).

$$\begin{array}{c}
\text{(abst.), (appl.), (var), (weak.) as in fig.6.} \\
\frac{\Gamma_1 \vdash t_1 : A'_1 \cdots \Gamma_n \vdash t_n : A'_n \quad x_1 : A_1, \dots, x_n : A_n \vdash t : B}{\Gamma_1, \dots, \Gamma_n \vdash t\{t_i/x_i\} : vB} \quad A'_i \leq vA_i(\text{prom.}) \\
\frac{\Gamma \vdash t' : A \quad x_1 : A_1, \dots, x_n : A_n, \Delta \vdash t : B}{\Gamma, \Delta \vdash t\{t'/x_1, \dots, x_n\} : B} \quad A \leq !u\hat{A}, !u\hat{A} \leq A_i \text{ (contr.)}
\end{array}$$

Figure 8: Abstract typing: *LALa*.

$$\begin{array}{c}
\frac{}{x : A \vdash x : A} \text{ (var.)} \qquad \frac{\Gamma \vdash t : B}{\Gamma, x : A \vdash t : B} \text{ (weak.)} \\
\frac{\Gamma_1 \vdash t_1 : A \rightarrow B \quad \Gamma_2 \vdash t_2 : A}{\Gamma_1, \Gamma_2 \vdash (t_1 t_2) : B} \text{ (appl.)} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \rightarrow B} \text{ (abst.)} \\
\frac{\Gamma_1 \vdash t_1 : A_1 \cdots \Gamma_n \vdash t_n : A_n \quad x_1 : A_1, \dots, x_n : A_n \vdash t : B}{\Gamma_1, \dots, \Gamma_n \vdash t\{t_i/x_i\} : B} \text{ (1)(prom.)} \\
\frac{\Gamma \vdash t' : A \quad x_1 : A, \dots, x_n : A, \Delta \vdash t : B}{\Gamma, \Delta \vdash t\{t'/x_1, \dots, x_n\} : B} \text{ (contr.) } n \geq 2
\end{array}$$

Figure 9: Type assignment ILS.

The inequalities associated to the rules are not seen as conditions for the application of the rule as before, but as constraints which are added to the derivation.

In fact all we want to impose for (*contr.*) is that  $A$  is of the form  $!A'$ , for some  $A'$ , and that  $A \leq A_i$  for  $1 \leq i \leq n$ . This is equivalent to the solvability of  $A \leq !u\hat{A} \leq A_i$  with a fresh  $u$ .

The set of word variables occurring in  $\mathcal{D}$  is denoted by  $Par(\mathcal{D})$  and given an instantiation  $\phi$  compatible with colors we define  $\phi(\mathcal{D})$  as expected. An instantiation  $\phi$  is a *solution* of  $\mathcal{D}$  if  $\phi(\mathcal{D})$  corresponds to a valid LALs type derivation.

One can ask whether given an a.t.d. we can decide if it has a solution; call it the *a.t.d. instantiation problem*. We will address this problem for a restricted class of a.t.d. that we will define in the next section (canonical derivations).

## 4.2 Simple type derivation with sharing

From now on we consider LALs derivations with multiple promotions. If  $\mathcal{D}$  is a LALs derivation, we denote by  $[D]$  the tree of judgements obtained by replacing each LAL formula  $A$  by  $[A]$ . Then  $[D]$  is a simple type derivation that we call *skeleton* of  $\mathcal{D}$ . We want to give a direct description of skeletons.

Given a term  $t$  let us denote by  $FV(t)$  the free variables *occurring* in  $t$ . We consider the typing rules for simple types of figure 9, with the conditions:

- in rule (*contr.*) we require that  $n \geq 2$  and that  $x_1, x_2$  belong to  $FV(t)$ ,
- in (*prom.*), all  $x_i$  should belong to  $FV(t)$ .

We call this set of rules ILS (intuitionistic logic with sharing).

**Definition 1** *Say an application of the (*prom.*) rule is simple if all  $t_i$  are variables. An ILS derivation is canonical if a simple (*prom.*) is never immediately followed by another (*prom.*) and (*weak.*) rules are applied as late as possible in the derivation.*

We have:

**Lemma 8** *If  $\Gamma \vdash_{LALs} t : A$  then there exists a LALs derivation  $\mathcal{D}$  of this judgement such that  $[D]$  is a canonical ILS derivation.*

**Proposition 9** *There is an algorithm that given a simply typed term  $\Gamma \vdash_{IL} t : A$  gives all canonical ILS derivations of this judgement; there is a finite number of such canonical derivations.*

A proof of this proposition is given in appendix C.

Given a simple type  $A$  we define its set  $fd(A)$  of *free decorations in LALa* in the following way:

- if  $A$  is an atomic type  $\alpha$  then  $fd(A) = \{u\alpha, u \in \mathcal{V}_b\}$ ;
- if  $A = A_1 \rightarrow A_2$  we take  $fd(A) = \{u(\overline{A_1} \multimap \overline{A_2}), \text{ s.t. } \overline{A_i} \in fd(A_i), Par(fd(A_1)) \cap Par(fd(A_2)) = \emptyset, u \notin Par(fd(A_i)) \text{ for } i = 1, 2\}$

Given a canonical ILS derivation  $\mathcal{D}$  we decorate it into a LALa derivation  $\overline{\mathcal{D}}$  by attributing to each occurrence of (*prom.*) a fresh parameter. More formally we define  $\overline{\mathcal{D}}$  by induction on  $\mathcal{D}$ :

- if  $\mathcal{D}$  is simply a (*var*) rule on  $A$ , we take  $\overline{A_1}, \overline{A_2}$  in  $fd(A)$  with disjoint parameters and  $\overline{\mathcal{D}}$  is a LALa (*var*) rule with  $\overline{A_1}, \overline{A_2}$ ;
- if  $\mathcal{D}$  is obtained by (*weak.*) on  $\mathcal{D}_1$  with  $A$ , we take  $\overline{A}$  in  $fd(A)$  with parameters disjoint from those of  $\overline{\mathcal{D}}_1$ , and  $\overline{\mathcal{D}}$  is obtained by (*weak.*) on  $\overline{\mathcal{D}}_1$  with  $\overline{A}$ ;
- if  $\mathcal{D}$  is obtained by (*abst.*) on  $\mathcal{D}_1$ , define  $\overline{\mathcal{D}}$  by (*abst.*) on  $\overline{\mathcal{D}}_1$ ;
- if  $\mathcal{D}$  is obtained from  $\mathcal{D}_1, \mathcal{D}_2$  by a (*appl.*) rule (resp. (*contr.*)) we can assume  $\overline{\mathcal{D}}_1, \overline{\mathcal{D}}_2$  have disjoint sets of parameters (otherwise rename their parameters) and  $\overline{\mathcal{D}}$  is obtained by (*appl.*) (resp. (*contr.*)) on  $\overline{\mathcal{D}}_1, \overline{\mathcal{D}}_2$ ;
- if  $\mathcal{D}$  is obtained from  $\mathcal{D}_1, \dots, \mathcal{D}_n, \mathcal{D}_0$  by a (*prom.*) rule, we assume as before the  $\overline{\mathcal{D}}_i$  have disjoint parameters, take  $u$  not occurring in the  $\overline{\mathcal{D}}_i$  and belonging to  $\mathcal{V}_m$  if  $n \geq 2$ ,  $\mathcal{V}_b$  if  $n \leq 1$ , and define  $\overline{\mathcal{D}}$  by a (*prom.*) rule on the  $\overline{\mathcal{D}}_i$  with parameter  $u$ .

We call *canonical abstract derivation* (c.a.d.) a LALa derivation obtained in this way from a canonical ILS derivation.

### 4.3 Towards LAL type inference

Now, given a (closed) term  $t$  our method for deciding of its typability in LALs is the following one:

- compute the principal type  $A$  of  $t$  in IL,
- using Prop. 9 and the decoration list the c.a.d. of conclusion  $\vdash_{LALa} u : \overline{A}$ ,
- for each such c.a.d.  $\mathcal{D}$  search if it has a solution  $\phi$ , in which case we get a LALs type derivation  $\phi(\mathcal{D})$  for  $t$ .

This method is valid because of lemma 2. So what remains to be done to prove the decidability of typability is to establish that finding a solution of a c.a.d. is decidable.

We associate to an abstract derivation  $\mathcal{D}$  a set of typing constraints  $Cons(\mathcal{D})$  in the following inductive way (keeping the notations of fig.8):

$$\begin{aligned}
Cons(\mathcal{D}) &= \{A_1 \leq A_2\} \text{ if } \mathcal{D} = (var), \\
Cons(\mathcal{D}) &= Cons(\mathcal{D}_1) \cup Cons(\mathcal{D}_2) \cup \{A_2 \leq A_1\}, \\
&\quad \text{if } \mathcal{D} \text{ is obtained from } \mathcal{D}_1, \mathcal{D}_2 \text{ by (appl)}, \\
Cons(\mathcal{D}) &= \cup_{i=0}^n Cons(\mathcal{D}_i) \cup \{A'_i \leq vA_i, 1 \leq i \leq n\}, \\
&\quad \text{if } \mathcal{D} \text{ is obtained from } \mathcal{D}_1, \dots, \mathcal{D}_n, \mathcal{D}_0 \\
&\quad \text{by (prom)}, \\
Cons(\mathcal{D}) &= Cons(\mathcal{D}_1) \cup Cons(\mathcal{D}_2) \cup \\
&\quad \{A \leq !u\widehat{A}, !u\widehat{A} \leq A_i, 1 \leq i \leq n\}, \\
&\quad \text{if } \mathcal{D} \text{ is obtained from } \mathcal{D}_1, \mathcal{D}_2 \text{ by (contr)}.
\end{aligned}$$

**Proposition 10** *Let  $\mathcal{D}$  be an abstract derivation: a map  $\phi$  is a solution of  $\mathcal{D}$  iff  $\phi$  is a solution of the system of inequations  $\mathcal{T}(Cons(\mathcal{D}))$ .*

Say a system of inequations  $\mathcal{S}$  is a *c.a.d. system* if there exists a c.a.d.  $\mathcal{D}$  such that  $\mathcal{S} = \mathcal{T}(Cons(\mathcal{D}))$ .

## 5 Solving the constraints

### 5.1 Stratification

Now we want to solve c.a.d. systems of inequations. Note that if we had *equations* instead of inequations we could apply Makanin's theorem which shows decidability of such systems (see for instance [10]). But we know of no general result which would apply to the systems of inequations we are considering. However we can here exploit a strong property of the systems we are interested in, *stratification*.

**Definition 2** *Let  $\mathcal{S}$  be a system of inequations and  $\phi$  be a solution. Say  $\phi$  is a stratified solution of  $\mathcal{S}$  if there exists a depth function  $d : \text{Par}(\mathcal{S}) \rightarrow \mathbb{N}$  such that:*

- for  $s \preceq s'$  in  $\mathcal{S}$  with  $s = u_1 s_1$ ,  $s' = u_2 s_2$  we have  $d(u_1) = d(u_2)$ ,
- for  $s = u_1 \dots u_n$  a member of inequation we have:  $d(u_{i+1}) = d(u_i) + |\phi(u_i)|$ .  
If  $s = u_1 \dots u_n$  we also define  $d(s)$  as  $d(u_1)$  and  $\text{ind}(s)$  (internal depth of  $s$ ) as  $d(u_n) + |\phi(u_n)|$ .

We say a system  $\mathcal{S}$  is stratified if all its solutions are stratified (so in particular if it has no solution).

Stratification prevents in some sense circularity in the search for solutions. It allows solving the system proceeding level by level.

**Example 2** *Consider the system  $\mathcal{S}$  given by:*

$$\begin{array}{rclcl} \S & \preceq & u_2 & I_1 \\ u_1 \S & \preceq & u_2 ! u_3 & I_2 \\ u_3 u_2 & \preceq & ! u_4 & I_3 \end{array}$$

*It does not have any stratified solution. Indeed, assume there was one  $\phi$  with depth  $d$ . Inequation  $I_1$  implies that  $|\phi(u_2)| \geq 1$ . Inequation  $I_2$  tells that  $\phi(u_3) \neq \epsilon$  (because  $\S \preceq !$  does not hold) and so  $|\phi(u_3)| \geq 1$ . Then by  $I_2$  we have  $d(u_2) < d(u_3)$ , and by  $I_3$ :  $d(u_3) < d(u_2)$ , hence a contradiction.*

**Proposition 11** *If  $\mathcal{S}$  is an c.a.d. system, then it is stratified.*

It simply follows from the definition of the rules and from the fact that in a canonical abstract derivation, a parameter is used in only one rule. In fact, the depth has a concrete meaning when we represent light terms as *proof-nets* ([1]), a dag representation: in this syntax promotion corresponds to a box construction, and the depth of a subterm is the number of boxes it is contained in. It is the notion of level mentioned in section 1.

Now we have:

**Theorem 12** *Given a system  $\mathcal{S}$ , the existence of a stratified solution is decidable.*

This theorem will be proved in the rest of this section. From these two results we can then deduce:

**Theorem 13** *The existence of a solution for a c.a.d. system is decidable.*

With prop. 10 we then get:

**Corollary 14** *The derivation instantiation problem for canonical abstract derivations is decidable.*

and from that our main result follows:

**Theorem 15** *Given a closed lambda-term  $t$ , one can decide whether there exists a LAL type  $A$  such that  $\vdash_{LALs} t : A$ .*

Let us come back to the proof of theorem 12. We will consider two characteristics of systems of inequations:

- the *measure* of a system  $mes(\mathcal{S})$  is the number of !s in right members of inequations of  $\mathcal{S}$  (similarly for the number  $mes(\mathcal{I})$  of !s in the right member of inequation  $I$ ),
- the *size* of a system  $|\mathcal{S}|$  is the number of inequations of  $\mathcal{S}$ .

Let us first point out an easy case: when the system does not have any ! in right members:

**Proposition 16** *If  $mes(\mathcal{S}) = 0$  then one can decide if there exists a solution.*

**Proof :** The key is that one can look for a monocolored solution, that is to say with words in  $\{\S\}^*$ . Indeed assume  $\phi$  is a solution, then define  $\psi$  by: for any  $u$ ,  $\psi(u) = \S^k$  where  $k = |\phi(u)|$ . Then as there are no ! in r.h.s. members of  $\mathcal{S}$ , and as !  $\preceq$   $\S$  the map  $\psi$  is also a solution of  $\mathcal{S}$ .

Now, a monocolored solution  $\psi$  is completely defined by the lengths  $|\psi(u)|$ , so to find whether there is one it is sufficient to solve the system of linear inequalities (over integers) obtained by replacing the word parameters by length parameters.  $\square$

## 5.2 Informal description of the algorithm

We now give an algorithm to decide whether a system has a stratified solution. In fact we give a non-deterministic algorithm and we will then justify how to transform it into a deterministic one.

The idea of our algorithm is to non-deterministically reduce the solving of  $\mathcal{S}$  to the solving of a system with no ! in right members, which is a problem we saw was decidable. To do so we want to progressively eliminate the occurrences of ! in right members of inequations.

Take an inequation  $a_1 \dots a_n \preceq s_1 !_0 s_2$  ( $I$ ) of  $\mathcal{S}$ . We can assume the  $a_i$ s are characters (! or  $\S$ ) or word variables. The two words on both sides of  $I$  should have same length, and as !  $\preceq$   $\S$  if  $k$  is the position of the character  $!_0$  on the r.h.s., the character in position  $k$  on l.h.s. should be !. If we can guess which  $a_j$  contains this character we can replace  $I$  with:

$$\begin{aligned} a_j &= a_{j1} ! a_{j2} \\ a_1 \dots a_{j-1} a_{j1} &\preceq s_1 \\ a_{j2} a_{j+1} \dots a_n &\preceq s_2 \end{aligned}$$

First observe that this guess can be successful only if  $a_j$  is a bicolored variable (a  $u_j$ ) or a ! character. In the last case  $a_{j1}$  and  $a_{j2}$  are taken to be  $\epsilon$ . So to simplify (without avoiding the difficulty) we can assume the  $a_j$ 's are all bicolored variables.

The real problem is that  $a_j$  might appear in other inequations, possibly in r.h.s. members and that replacing  $a_j$  with  $a_{j1} ! a_{j2}$  we have introduced new !s in r.h.s. members. Let us call these !s and those that will appear when we try to eliminate them in the same way, *residuals* of  $!_0$ .

The algorithm will proceed by *rounds*, each round consisting in eliminating one ! on r.h.s. of inequation and all its residuals in r.h.s. members. At the end of a round the measure  $mes(\mathcal{S})$  will have decreased by 1.

A round will be divided in *steps* consisting in eliminating a r.h.s. ! (the way we just sketched) and creating residuals.

## 5.3 The algorithm

We will handle the following datas:

- $\mathcal{R}$ : set of equations. Initially:  $\mathcal{R} = \emptyset$ .
- $\mathcal{S}$  system of inequations handled as a set. Initially  $\mathcal{S}$  is the system  $\mathcal{S}_0$  to be solved.  $\overline{\mathcal{S}}$  is a subset of  $\mathcal{S}$ .
- stack  $P$  of inequations with one marked occurrence of ! in their right member (we denote them as pairs  $(I, !_0)$  where  $!_0$  is an occurrence of !). The elements of  $P$  belong to  $\overline{\mathcal{S}}$ .

During the whole run:  $\mathcal{S}$  is the current state of the system;  $\mathcal{R}$  keeps track of the variables we have deleted and how to retrieve their values from the current variables.

During a round:  $\overline{\mathcal{S}}$  is the subset of inequations that *might* contain residuals of the current  $!_0$ ;  $P$  contains the inequations with residuals of  $!_0$ .

The algorithm is then given by:

- repeat the ROUND until getting a system  $\mathcal{S}'$  with  $mes(\mathcal{S}') = 0$ .

**ROUND:**

- $\overline{\mathcal{S}} := \mathcal{S}; \quad P := \epsilon$  (empty stack);
- take in  $\mathcal{S}$  an inequation  $I_l$  with  $mes(I_l) > 0$  and  $!_0$  in the r.h.s. member of  $I_l$ :

$$I_l : \quad u_1 \dots u_n \preceq s_1 !_0 s_2$$

- push  $(I_l, !_0)$  on  $P$ .
- repeat the following procedure until  $P = \epsilon$ :

**STEP:**

pop  $(I_l, !_0)$  from  $P$ :

$$I_l : \quad u_1 \dots u_n \preceq s_1 !_0 s_2$$

guess  $u_j$  (monocolored variable or  $!$ ) such that  $!_0$  "belongs" to  $u_j$ ;

replace  $u_j$  by  $u_{j1}!u_{j2}$  in  $\mathcal{S}$  and  $\overline{\mathcal{S}}$  ( $u_{j1} = u_{j2} = \epsilon$  if  $u_j$  was  $!$ ):

$$\begin{aligned} \overline{\mathcal{S}} &:= (\overline{\mathcal{S}} \setminus \{I_l\}) < u_{j1}!u_{j2} \rightarrow u_j > \\ \mathcal{S} &:= (\mathcal{S} \setminus \{I_l\}) \cup \{u_1 \dots u_{j1} \preceq s_1 (I_{l1})\} \cup \\ &\quad \{u_{j2} \dots u_n \preceq s_2 (I_{l2})\} \\ \mathcal{S} &:= \mathcal{S} < u_{j1}!u_{j2} \rightarrow u_j > \\ \mathcal{R} &:= \mathcal{R} \cup \{u_j = u_{j1}!u_{j2}\} \end{aligned}$$

if  $u_j$  is a variable add to  $P$  the inequations of  $\overline{\mathcal{S}}$  in which a  $u_j$  has been replaced in the r.h.s. (ie containing residuals in the r.h.s.)

end of STEP.

end of ROUND.

- When we have  $mes(\mathcal{S}) = 0$  we compute a (monocolored) solution, and then using  $\mathcal{R}$  we track back a solution of the original system  $\mathcal{S}_0$ .

## 5.4 Analysis of the algorithm

**Termination.** It is easy to see that the non-deterministic algorithm terminates because: each round decreases the measure by 1; each round does terminate since a step decreases  $|\overline{\mathcal{S}}|$  by 1.

We can give an explicit bound. The number of rounds is bounded by  $mes(\mathcal{S})$ . If we denote by  $\mathcal{S}_i$  the system at the beginning of the  $i$ th round, the number of steps of round  $i$  is bounded by  $|\overline{\mathcal{S}}_i| = |\mathcal{S}_i|$ . At each step the size of the system increases by 1. So  $|\mathcal{S}_{i+1}| \leq 2|\mathcal{S}_i|$ . In conclusion the length of any run is bounded by  $2^{mes(\mathcal{S})} \cdot |\mathcal{S}|$ .

**Correctness.** It is also easy to establish correctness: consider two consecutive states of the system denoted as  $\mathcal{S}_i$  and  $\mathcal{S}_{i+1}$ . Remember that  $\mathcal{S}_{i+1}$  is obtained from  $\mathcal{S}_i$  by splitting an inequation  $I_l$  in two. Assume we have a solution  $\psi$  of  $\mathcal{S}_{i+1}$ , then keeping the notations used before we define  $\phi(u_j) := \psi(u_{j1})!\psi(u_{j2})$  and  $\phi(v) := \psi(v)$  for the other variables. It is clear that  $\phi$  is then a solution of  $\mathcal{S}_i$ . So if we have a solution of the final system, it can be lifted back to a solution of the initial system  $\mathcal{S}_0$  using the equalities in  $\mathcal{R}$ .

**Completeness.** Let us now examine the completeness issue, which is more delicate. Assume  $\mathcal{S}_0$  has a stratified solution  $\phi$  with depth  $d$  and let us show that there is a run of the algorithm leading to this solution.

During one round we try to eliminate a r.h.s.  $!_0$  and its residuals. The important point is that this round proceeds at fixed depth, that is to say that the residual  $!s$  have the same depth  $d_0$  as  $!_0$ . An inequation  $s_1 \preceq s_2$  can contain a residual of  $!_0$  only if it satisfies  $d(s_1) \leq d_0 < ind(s_1)$ .

Consider one state  $\mathcal{S}$  of the system with stratified solution  $\phi, d$ . We consider the inequation  $u_1 \dots u_n \preceq s_1 !_0 s_2$  ( $I_l$ ) from the top of the stack. Let  $j$  be such that  $d(u_j) \leq d_0 < d(u_{j+1})$  (or  $d(u_n) \leq d_0$  and  $j = n$ ). We choose  $u_j$  in this step and ( $I_l$ ) is replaced by:

$$\begin{aligned} u_1 \dots u_{j1} &\preceq s_1 & (I_{l1}) \\ u_{j2} \dots u_n &\preceq s_2 & (I_{l2}) \end{aligned}$$

Call  $\mathcal{S}'$  the new system. Define  $\phi'$  on  $Par(\mathcal{S}')$  by:

$$\begin{aligned}\phi'(v) &= \phi(v) \text{ for } v \neq u, \\ \phi'(u_{j_1}) &= t_1 \text{ prefix of } \phi(u) \text{ of length } d_0 - d(u) - 1, \\ \phi'(u_{j_2}) &= t_2 \text{ suffix of } \phi(u) \text{ of length } |\phi(u)| - |\phi'(u_{j_1})| - 1.\end{aligned}$$

We also define in the same way  $d'$  with

$$\begin{aligned}d'(u_{j_1}) &= d(u_j), \\ d'(u_{j_2}) &= d_0 + 1.\end{aligned}$$

Then  $\phi', d'$  is a stratified solution of  $\mathcal{S}'$ .

So if  $\mathcal{S}$  has a stratified solution then  $\mathcal{S}'$  has a stratified solution. Moreover for  $\mathcal{S}'$  we have:

$$\begin{aligned}ind'(I_{11}) &= d_0, \\ d'(I_{12}) &= d'(u_{j_2}) = d_0 + 1\end{aligned}$$

So  $I_{11}$  and  $I_{12}$  can not contain any further residual of  $!_0$ , hence we can exclude them from  $\overline{\mathcal{S}'}$ , the subset of  $\mathcal{S}$  in which we search for residuals of  $!_0$ . This is the action that makes  $|\mathcal{S}'|$  decrease.

## 5.5 A deterministic algorithm

Observe that at each step the non-deterministic choice is between a finite number of possibilities (the characters and word variables on the l.h.s. of the inequation currently examined). If we represent the runs of the non-deterministic algorithm as a tree we have finite branchings and all branches have finite length. Therefore a brute-force algorithm can deterministically completely explore the tree and solve the system. Using our bound for the length of the branches we can also provide a bound for the arities of the choices:  $w_0 + 2^{mes(\mathcal{S})} \cdot |\mathcal{S}|$ , where  $w_0$  is the number of word variables in the initial system.

## References

- [1] A. Asperti. Light affine logic. In *Proceedings LICS'98*. IEEE Computer Society, 1998.
- [2] P. Baillot. Stratified coherent spaces: a denotational semantics for light linear logic. Tech. report 0025, University of Edinburgh, 2000. to appear in *Theoretical Computer Science*, special issue on ICC'00.
- [3] P. Baillot. Checking polynomial time complexity with types. In *Second International IFIP Conference on Theoretical Computer Science*, Montreal, august 2002. Kluwer Academic Press.
- [4] S. Bellantoni, K.-H. Niggl, and H. Schwichtenberg. Higher type recursion, ramification and polynomial time. *Annals of Pure and Applied Logic*, 104(1-3), 2000.
- [5] P. Benton, G. Bierman, V. de Paiva, and J. Hyland. A term calculus for intuitionistic linear logic. In *Proceedings TLCA'93*, volume 664 of *LNCS*. Springer Verlag, 1993.
- [6] P. Coppola and S. Martini. Typing lambda-terms in elementary logic with linear constraints. In *Proceedings TLCA'01*, volume 2044 of *LNCS*, 2001.
- [7] P. Coppola and S. Ronchi della Rocca. Principal typing in Elementary Affine Logic. Workshop on Linear Logic (Copenhagen), July 2002.
- [8] V. Danos and J.-B. Joinet. Linear logic and elementary time. First Workshop on Implicit Computational Complexity (ICC'99), 1999.
- [9] V. Danos, J.-B. Joinet, and H. Schellinx. On the linear decoration of intuitionistic derivations. *Archive for Mathematical Logic*, 33(6), 1994.
- [10] V. Diekert. Makanin's algorithm. In *Algebraic Combinatorics on Words (Lothaire)*, chapter 12. Cambridge University Press, 2001.
- [11] J.-Y. Girard. Light linear logic. *Information and Computation*, 143:175–204, 1998.
- [12] M. Hofmann. Linear types and non-size-increasing polynomial time computation. In "*Proceedings of the 14<sup>th</sup> Symposium on Logic in Computer Science*". IEEE Computer Society, 1999.
- [13] M. Hofmann. Safe recursion with higher types and BCK-algebra. *Annals of Pure and Applied Logic*, 104(1-3), 2000.

- [14] D. Leivant and J.-Y. Marion. Lambda-calculus characterisations of polytime. *Fundamenta Informaticae*, 19:167–184, 1993.
- [15] J. C. Mitchell. *Foundations for Programming Languages*. MIT Press, 1996.
- [16] L. Roversi. A P-time completeness proof for light logics. In *Proceedings CSL'99*, volume 1683 of *LNCS*. Springer-Verlag, 1999.
- [17] L. Roversi. Light affine logic as a programming language: a first contribution. *International Journal of Foundations of Computer Science*, 11(1), 2000.
- [18] K. Terui. Light Affine Lambda-calculus and polytime strong normalization. In *Proceedings LICS'01*. IEEE Computer Society, 2001.
- [19] J. B. Wells. Typability and type checking in system F are equivalent and undecidable. *Ann. Pure Appl. Logic*, 98(1-3), 1999.

# APPENDIX

## A Examples

We give some examples of typed terms representing elementary programs on lists.  
In system F lists of elements of type  $A$  can be given the following type:

$$\forall \alpha. (A \rightarrow \alpha \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha)$$

In LAL2 (polymorphic LAL) this type can be adapted to:

$$\forall \alpha. !(A \multimap \alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha)$$

Without polymorphism we will consider the family of types:

$$\mathcal{L}(A)_B = !(A \multimap B \multimap B) \multimap \S(B \multimap B)$$

We denote by  $\mathcal{L}(A)$  the type  $\mathcal{L}(A)_\alpha$ .

The list  $\langle a_1, \dots, a_n \rangle$  is represented by the term:  $\lambda c. \lambda e. (c)a_1 (c)a_2 \dots (c)a_n e$ .

The concatenation function can be represented by:

$$conc = \lambda l. \lambda l'. \lambda c. \lambda e. (l)c(l')ce$$

It can be given the type:  $\mathcal{L}(A) \multimap \mathcal{L}(A) \multimap \mathcal{L}(A)$

From this function we can define a function that concatenates a list  $l$  to itself:

$$sconc = \lambda l. \lambda c. \lambda e. (l)c(l)ce$$

We can check that:  $\vdash sconc : !\mathcal{L}(A) \multimap \S\mathcal{L}(A)$  holds.

Consider the function *double* defined by:  $double(\epsilon) = \epsilon$ ,  $double(a :: l) = a :: a :: double(l)$ . It is represented by the term:

$$d = \lambda l. \lambda c. \lambda e. ((l)\lambda y. \lambda z. (c)y(c)yz)e$$

We have:  $\vdash d : \mathcal{L}(!A) \multimap \mathcal{L}(!A)$  (the elements of the list must be duplicable, hence the type  $!A$ ).

## B Proof of Proposition 5

**Proof :** As LAL rules are particular instances of LALs rules, it is straightforward that  $\Gamma \vdash_{LAL} t : A$  implies  $\Gamma \vdash_{LALs} t : A$ .

For the other property we proceed by induction over derivations of  $\Gamma \vdash_{LALs} t : A$ .

- if the derivation is only an application of (var.) rule,  $t = x$ ,  $\Gamma = x : A_1$  with  $A_1 \leq A$ , we apply lemma 4 and get a derivation of  $x : A_1 \vdash_{LAL} t' : A$  with  $t' \rightarrow^n x$ .
- the cases of rules (weak.), (appl.), (abst.) do not raise any problem.
- if the derivation is obtained by a (contr.) rule from two LALs derivations  $\mathcal{D}_1$  and  $\mathcal{D}_2$  respectively of  $\Gamma \vdash t_1 : !A$  and  $x_1 : A_1, x_2 : A_2, \Delta \vdash t_2 : B$  (we assume  $n = 2$  for simplicity); the conclusion is

$$\Gamma, \Delta \vdash t_2 \{t_1/x_1, x_2\}.$$

By i.h. we get two LAL derivations  $\mathcal{D}'_1$  and  $\mathcal{D}'_2$  with terms  $t'_1, t'_2$  and  $t'_i \rightarrow^n t_i$  for  $i = 1, 2$ .

By lemma 4 as  $!A \leq A_i$  for  $i = 1, 2$  we get terms  $w_i$  with  $w_i \rightarrow^n x_i$  and using (subst.) we can derive in LAL:

$$\frac{\Gamma \vdash t_1 : !A \quad \frac{x'_1 : !A \vdash w_1 : A_1 \quad \frac{x'_2 : !A \vdash w_2 : A_2 \quad x_1 : A_1, x_2 : A_2, \Delta \vdash t'_2 : B}{(subst.)} \quad x_1 : A_1, x'_2 : !A, \Delta \vdash t'_2 \{w_2/x_2\} : B}{(subst.)}}{x'_1 : !A, x'_2 : !A, \Delta \vdash t'_2 \{w_1/x_1, w_2/x_2\} : B} \quad (contr.)}{\Gamma, \Delta \vdash t'' : B}$$

with  $t'' = (t'_2 \{w_1/x_1, w_2/x_2\}) \{t_1/x'_1, x'_2\}$ .

As  $t'_2 \rightarrow^n t_2$ ,  $w_i \rightarrow^n x'_i$  we have  $t'_2 \{w_1/x_1, w_2/x_2\} \rightarrow^n t_2 \{x'_1/x_1, x'_2/x_2\}$ , so  $t'' \rightarrow^n t_2 \{t_1/x_1, x_2\}$ .

- the case of a (*prom.*) rule is handled in a similar way to that of (*contr.*).

□

## C Proof of Proposition 9

**Proof :** We want to establish decidability and do not try here to give a good algorithm.

We want to construct, proceeding bottom-up, all possible canonical ILS derivations for  $\Gamma \vdash t : A$ . To show that this procedure terminates we provide a bound on the height of the branches of the derivation trees; it is then enough to observe that we can bound the arity of each rule and the search-space for the derivations will be delimited.

We consider the size function on lambda-terms given by:

$$|x| = 1, \quad |(t)u| = |t| + |u|, \quad |\lambda x.t| = |t| + 1.$$

Let  $n(x, t)$  denote the number of (free) occurrences of variable  $x$  in term  $t$ . We consider another function taking into account the number of repetitions of free variables:

$$rep(t) = \sum_{x \in FV(t)} (n(x, t) - 1).$$

We consider the following measure on typing judgements, with lexicographic order:

$$mes(\Gamma \vdash t : B) = (|t|, rep(t), \#\Gamma),$$

where  $\#\Gamma$  denotes the length of context  $\Gamma$ .

Now let us examine the various rules (applied bottom-up) and whether they make this measure decrease. Rules (*appl.*) and (*abst.*) make the size of the term decrease, so the measure of the judgements too. The (*weak.*) rules leaves  $|t|$ ,  $rep(t)$  unchanged but the length of the context decreases.

Consider the (*contr.*) rule. As we required that  $n \geq 2$  and  $x_1, x_2 \in FV(t)$ , if  $t'$  is not a variable then  $|t \{t'/x_1, \dots, x_n\}| > |t|$  and  $|t \{t'/x_1, \dots, x_n\}| > |t'|$ , and so the measure decreases. If  $t'$  is a variable then  $|t \{t'/x_1, \dots, x_n\}| = |t|$ , but  $rep(t \{t'/x_1, \dots, x_n\}) > rep(t)$ .

Let us examine the (*prom.*) rule. If it is not simple, that is to say one of the  $t_i$  is not a variable, then by the condition that  $x_i \in FV(t)$  we get:  $|t \{t_i/x_i\}| > |t|$ . A simple (*prom.*) however leaves the measure unchanged.

So simple (*prom.*) is the only instance of rule that leaves the measure unchanged. But as we required that in a canonical derivation there is no two consecutive applications of (*prom.*) the height of a branch is bounded by  $2mes(\Gamma \vdash t : B)$ , where  $\Gamma \vdash t : B$  is the initial judgement. □