ARTICLE IN PRESS

ELSEVIER

Available online at www.sciencedirect.com

SCIENCE DIRECT®

Theoretical Computer Science ▮▮▮ (▮▮▮▮) ▮▮▮–▮▮▮

Theoretical Computer Science

www.elsevier.com/locate/tcs

Type inference for light affine logic via constraints on words☆

Patrick Baillot

Laboratoire d'Informatique de Paris-Nord (UMR 7030), CNRS/Université Paris-Nord, Institut Galilée, 99 av. J.-B. Clément, 93430 Villetaneuse, France

Received 15 October 2003; received in revised form 19 August 2004; accepted 30 August 2004
Communicated by P.-L. Curien

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 decision procedure.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Implicit computational complexity; Linear logic; Light affine logic; Lambda-calculus; Type inference; Polynomial time complexity; Constraints

1. Introduction

Functional languages have been advocated as languages amenable to reasoning on programs and specifications. Although 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

☆ Work partly supported by Action Spécifique CNRS *Méthodes formelles pour la Mobilité* (2002–03).
E-mail address: pb@lipn.univ-paris13.fr (P. Baillot).

0304-3975/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2004.08.014

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 computational complexity* for defining languages and calculi in which all programmable functions have a given complexity (e.g. [6,14–16,19]). Recall that the original goal of Implicit computational complexity is to give machine-independent characterizations of complexity classes, where the control over resources is not managed by explicit measures (e.g. clock) but implicit in the program constructs or the definitions of the calculus considered. Some of the languages mentioned are based on restrictions on the use of structural recursion, others on proof-theoretical methods or on type systems.

Here we are interested in *Light Affine Logic* (LAL) [1,2], a variant of Linear Logic with a polynomial time cut-elimination procedure (it was obtained as a simplification of Light *Linear* Logic [14]) and which characterizes the class P in the proofs-as-programs paradigm. Light logic has been studied under various aspects: as a logical system [2,14], as a variant of lambda-calculus via the Curry–Howard isomorphism [23] and semantically [5,17,21,24]; some extensions like *light set theory* [14,26] or a non-deterministic variant [20] have also been investigated. However the system is quite delicate to handle and therefore we think it is important to determine how much of the programming task in this setting could be automated.

In particular LAL can be used as a type system for ordinary lambda-calculus, ensuring the property that if a program is well-typed then it is PTIME. In this way type search provides a way to statically guarantee a time upper bound on a program. The type derivation can then be seen as a *certificate* that the program can be executed within the bound on any input. Note that this is a strong property that could not be checked simply by pragmatically executing the program, because what is given is a bound relative to a (possibly) infinite set of input values. Note also that even though we are using lambda-calculus as source language, the polynomial bound is *not* ensured on ordinary lambda-calculus reduction, that is to say β-reduction, but on the compilation of the lambda-calculus program into a *proof-net* (see [2]) and its execution by proof-net normalization.

Actually here we focus on type inference for propositional (quantifier free) LAL which is not very expressive. We consider it as an important first step though because this is the core of LAL and polymorphism brings difficulties of its own for type inference (recall type inference for system F is undecidable, [27]). We do not know whether type inference for second-order LAL is decidable. An alternative to polymorphism could be to extend the lambda-calculus language with functions on basic types and iterator constants (in the style of the languages in [6,16]). This is left for future work, together with the investigation of the complexity of type inference.

Related work. We already considered the problem of LAL 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.

Coppola and Martini studied in [8] type inference in Elementary Affine Logic (EAL), a system corresponding to Kalmar elementary complexity (see also [10]), for which they showed decidability of type inference. Their algorithm was based on the idea of first proposing a simple type derivation for the term and then interpolating this derivation with modality

rules in order to find a suitable EAL derivation (in the line of the works on linear decorations as [11]). The approach we follow here is closer to that proposed by Coppola and Ronchi della Rocca in [9], where they introduce a notion of principal typing and give another type inference algorithm for EAL: propose a pattern of type-derivation with free parameters and express its correctness by a system of constraints (linear equations over integers in the case of EAL).

A preliminary version of the present work appeared as [4].

Outline. After recalling the principles of LAL in Section 3 we give the natural LAL type assignment system for lambda-calculus (Section 4), 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 6), where a degree of freedom 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 7 we establish how to solve the constraints to decide the previous problem. This amounts to solve systems of inequations on words.

Acknowledgement. We wish to thank Roberto Amadio for suggesting the use of subtyping for LAL typing, and François Pottier and Kazushige Terui for useful discussions. We are also indebted to the anonymous referees for many accurate comments and suggestions.

2. Preliminaries

We give in this short section a few preliminary definitions and notations.

Lambda-calculus terms are defined by: $t ::= x \mid \lambda x.t \mid (t\ t)$. We denote the set of free variables of a term t by $FV(t)$.

We denote by $t\{u_1/x_1, \ldots, u_n/x_n\}$ (or simply $t\{u_i/x_i\}$ if there is no ambiguity) the simultaneous substitution of terms u_i for variables x_i ($1 \leqslant i \leqslant n$) in term t (with the usual discipline for avoiding variable capture). In the case where $u_i = u$ for $1 \leqslant i \leqslant n$ we denote it as $t\{u/x_1, \ldots, x_n\}$.

We denote by $\rightarrow$ the one-step β-reduction relation on terms, defined as the contextual closure of the relation $\stackrel{0}{\rightarrow}$ given by: $((\lambda x.t)\ u) \stackrel{0}{\rightarrow} t\{u/x\}$. Then $\stackrel{*}{\rightarrow}$ denotes the reflexive and transitive closure of $\rightarrow$.

We denote by $\stackrel{\eta *}{\rightarrow}$ the η-reduction relation on terms, defined as the contextual, reflexive and transitive closure of the relation $\stackrel{\eta}{\rightarrow}$ given by: $\lambda x.(t\ x) \stackrel{\eta}{\rightarrow} t$ if $x \notin FV(t)$.

Typing judgements in various systems will be denoted by $\Gamma \vdash t : A$, where Γ is a set of type declarations $\Gamma = \{x_1 : A_1, \ldots, x_n : A_n\}$ and the x_is are distinct. Then $\Gamma(y)$ is defined iff $y = x_i$ for $1 \leqslant i \leqslant n$, and then $\Gamma(y) = A_i$. Moreover $\Gamma \backslash \{y\}$ will stand for Γ if $y \neq x_i$ ($1 \leqslant i \leqslant n$) and $\Gamma \backslash \{x_i : A_i\}$ if $y = x_i$.

We will consider words over a finite alphabet, with concatenation of word s with word s' denoted as: ss'. The empty word is written ε. The length of a word s is denoted as $|s|$.

We will denote lists as $\langle a_1, \ldots, a_n \rangle$ and the empty list as ε. Adding an element a at the beginning or end of a list l will be respectively written as $a :: l$ and $l :: a$. Appending list l' to list l will be written as $l :: l'$.

3. 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 §.

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 (Figs. 1 and 2) then in the resulting term b, level i will only depend on the levels j for $j \leqslant i$ of t and a (see [5] for a semantical interpretation of this property).

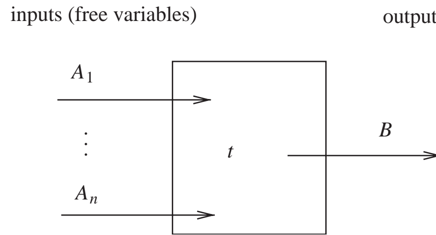

Fig. 1. Representation of a typed term.

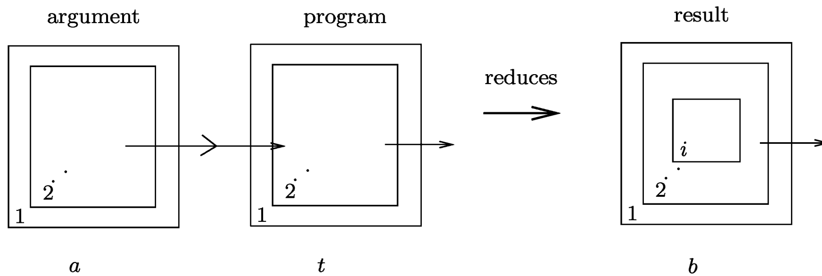

Fig. 2. Stratification.

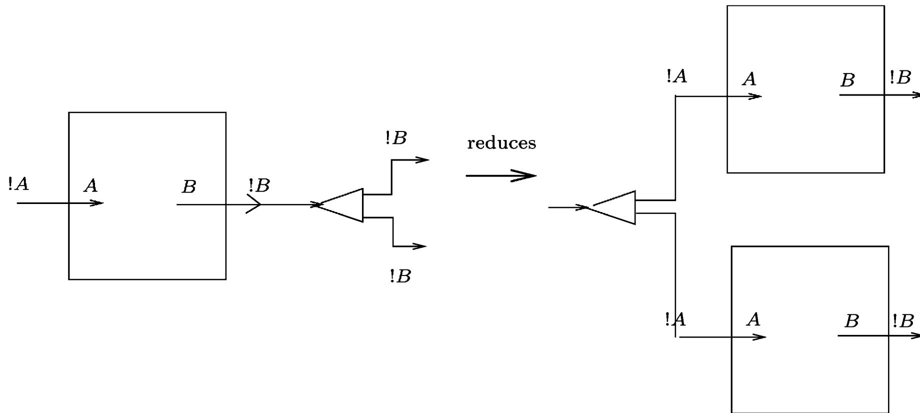

Fig. 3. ! allows being duplicated.

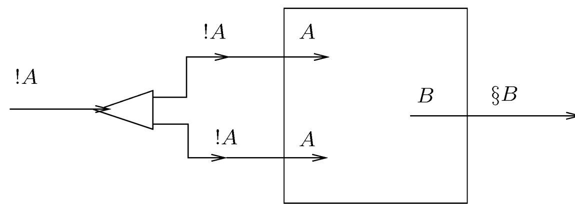

Fig. 4. § enables identification of variables.

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 (Fig. 3). The ! modality therefore has two roles: switching level and allowing duplication.

Another system is based on stratification and the ! modality, Elementary Affine Logic; it guarantees 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 § is introduced for. Applying § 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 (Fig. 4).

4. Typing in LAL

4.1. Type system

We want to type lambda-terms in LAL. LAL types are given by the following grammar (over a denumerable set of propositional variables):

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

We stick here to the implicational fragment of LAL (without $\otimes$) for simplicity, but considering the case with $\otimes$ would not add much difficulty.The ! (*bang*) and § (*neutral*) connectives are called *exponentials*.

We use a natural deduction presentation of the type-assignment system in the lines of [7,8] (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* [2,14] to the study of reduction, but it is easier to understand for typing. The rules are given in Fig. 5.

Conditions with Fig. 5:

(1) The a_is belong to $\{!, \S\}$ and satisfy: if $n \geqslant 2$ 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. §-promotion). Note that condition (1) includes the restriction described in Section 3: 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 when 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

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

Observe that this rule acts both on the type of the term and on those of the free variables, adding one modality to each. In the case of §-promotion, if $n \geqslant 2$ 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.

We call *depth* of a derivation $\mathcal{D}$ the maximal number of r.h.s. premises of (*prom*) rules in branches of $\mathcal{D}$.

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

$$\frac{\Gamma_1 \vdash t_1 : A \multimap B \quad \Gamma_2 \vdash t_2 : A}{\Gamma_1, \Gamma_2 \vdash (t_1 t_2) : B}\ \text{(appl)} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \multimap B}\ \text{(abst)}$$

$$\frac{\Gamma_1 \vdash t_1 : a_1 A_1 \ \cdots\ \Gamma_n \vdash t_n : a_n A_n \quad x_1 : A_1, \ldots, x_n : A_n \vdash t : B}{\Gamma_1, \ldots, \Gamma_n \vdash t\{t_1/x_1, \ldots, t_n/x_n\} : a_0 B}\ \text{(1)(prom)}$$

$$\frac{\Gamma \vdash t' : !A \quad x_1 : !A, \ldots, x_n : !A, \Delta \vdash t : B}{\Gamma, \Delta \vdash t\{t'/x_1, \ldots, x_n\} : B}\ \text{(contr)}$$

Fig. 5. Type assignment LAL.

Recall that proof-nets are a graph syntax for proofs in Linear Logic [13] and related systems (we will not give here the definition of proof-nets). To any LAL derivation $\mathcal{D}$ an LAL proof-net $R_{\mathcal{D}}$ [2] can be associated. The rule (*prom*) corresponds to introduction of a box (either !-box or §-box) in the proof-net. Intuitively the derivation of the r.h.s. premise of the rule is enclosed in the box. The *depth* of a proof-net R is the maximal number of nested boxes in R; thus the depth of $\mathcal{D}$ is by definition the same as the depth of $R_{\mathcal{D}}$.

We have:

Lemma 1 (*Substitution*). *If the judgments* $\Gamma \vdash u : A$ *and* $x : A, \Delta \vdash t : B$ *are LAL-derivable then so is the judgment* $\Gamma, \Delta \vdash t\{u/x\} : B$.

Proof. By induction on the derivation of $x : A, \Delta \vdash t : B$. □

Lemma 2. *If* $\Gamma \vdash t : A$ *is derivable and* t_0 *is the normal form of* t *for* β*-reduction, then* $\Gamma \vdash t_0 : A$ *is derivable.*

Proof (Sketch)**.** Consider a proof-net R corresponding to the derivation of $\Gamma \vdash t : A$ and let R_0 be its normal form for cut-elimination. Then an LAL derivation can be retrieved from R_0 and the corresponding term t_0 is obtained by β-reduction from t. Finally, as R_0 is normal the term t_0 is in normal form with respect to β-reduction. □

Note that Lemma 2 states a weaker property than subject-reduction. Subject-reduction itself, as pointed out by Terui [25] is not satisfied by LAL with respect to β-reduction. This is essentially due to the fact that the (*contr*) rule allows the use of sharing in type derivations. Here is an example showing that typing is not preserved by $\rightarrow$ (one step of β-reduction):

consider the term $t = [(a\ (x\ (I\ y)))\ (x\ (I\ y)]$, with $I = \lambda z.z$.

We have $t \rightarrow t_1$, where $t_1 = [(a\ (x\ y))\ (x\ (I\ y)]$.

The judgement $x : A \multimap !B, y : A, a : !B \multimap !B \multimap C \vdash_{LAL} t : C$ is derivable, but the same judgement is not valid for t_1. This is because for typing t one can use a sharing of the two subterms $(x\ (I\ y))$, which is not possible for t_1. Note that LAL subject-reduction is however satisfied by light affine lambda calculus [23], where sharing is explicit.

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

Lemma 3. *If* $\vdash_{LAL} t : A$ *then* $\vdash_{IL} t : [A]$.

The main property of LAL-typed terms is the following one, which is a consequence of the results of [1,14]:

Proposition 4. *If* $\vdash_{LAL} t : A \multimap B$ *is obtained by an LAL derivation* $\mathcal{D}_1$*, then there exists a polynomial* P *such that*:
for any derivation $\mathcal{D}_2$ *of a judgement* $\vdash_{LAL} u : A$*, if we denote by* $\mathcal{D}$ *the derivation* $\vdash (t\ u) : B$ *obtained from* $\mathcal{D}_1$ *and* $\mathcal{D}_2$ *and by* R *the corresponding proof-net, then* R *can be normalized in* $P(|u|)$ *steps, where* $|u|$ *denotes the size of* u.

Moreover the β-normal form of the lambda-term $(t\ u)$ can be extracted from the normal form R_0 of R.

Remark 1.

(1) It follows from this Proposition that if A and B are data types then t denotes a polynomial time function, because the polynomial step reduction of the proof-net R can be done in polynomial time.
(2) The degree of the polynomial bounding the number of steps of the reduction of R only depends on the depth of R. As here we are considering the quantifier-free fragment of LAL this depth is already given by the type $A \multimap B$.
(3) Instead of proof-nets one can also use light affine λ-calculus [23] with the same bound on the number of reduction steps.

With second-order quantifiers there is also a completeness result (see [2,22]).

It is important to note that the statement of Proposition 4 refers to the number of normalization steps of proof-nets and not to the β-reduction of the lambda-term $(t\ u)$ itself. As pointed out in [2], LAL-typed lambda-terms can have exponentially long β-reduction sequences. Here is an example (adapted from [2]):

Consider the term: $t = \lambda x.\lambda y.[[p_1\ (x\ y)](x\ y)]$ where $p_1 = \lambda a.\lambda b.a$. It can be typed as: $\vdash t : !(!A \multimap !A) \multimap !(!A \multimap !A)$.

Now consider the family of terms: $U_n = (t \ldots (t\ (t\ I)) \ldots)$ with $n \geqslant 0$ applications of t, and where $I = \lambda x.x$. Let $T_n = (U_n\ I)$.

Taking $A = \alpha \multimap \alpha$, the term T_n can be typed as $\vdash T_n : !!(\alpha \multimap \alpha)$ and the derivation has depth 2 (so this depth is independent of n). The term T_n reduces to I and we define the reduction sequence s_n by
s_0 is given by: $(I\ I) \ \rightarrow\ I$
and s_{n+1} is defined inductively by

$$\begin{array}{lll} T_{n+1} & \rightarrow & [(\lambda y.[p_1\ (U_n\ y)(U_n\ y)])\ I] \\ & \rightarrow & (p_1\ T_n\ T_n) \\ & \stackrel{s_n}{\rightarrow} & (p_1\ I\ T_n) \quad \text{(reduction of l.h.s. argument)} \\ & \stackrel{s_n}{\rightarrow} & (p_1\ I\ I) \quad \text{(reduction of r.h.s. argument)} \\ & \rightarrow & I. \end{array}$$

Thus, denoting by $|s_n|$ the number of steps of the sequence s_n we get $|s_{n+1}| \geqslant 2|s_n|$, and so $|s_n| \geqslant 2^n$. The proof-net corresponding to the type derivation of T_n however normalizes in a number of steps polynomial in $|T_n|$, hence polynomial in n.

The proof-nets can be seen as an intermediate language into which LAL-typed lambda-terms are compiled in order to be executed efficiently. Note that the proof-net of Proposition 4 is actually obtained from the type derivation of the term. Thus the LAL type derivation does not only ensure that the program *can* be executed with a certain bound but also provides the necessary information to actually compile the term and perform the execution.

4.2. Modalities and subtyping

When typing lambda-terms we have to apply to certain types several ! / §. 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 it the type $!\S A$.

Observe that if t can be typed with $!A$ then it can also be typed with $\S A$; it is sufficient to replace in the derivation a !-promotion by a §-promotion. Similarly a term with type $!!\S A$ can also be attributed the type $!\S\S A$ 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 the order $\preccurlyeq$ on $\mathcal{L}$ as the least reflexive relation on $\mathcal{L}$ satisfying:

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

Note that this relation is transitive by definition and that $s \preccurlyeq s'$ implies $|s| = |s'|$. For $a \in \{!, \S\}$ we write a^k for $a \ldots a$ 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_0, s_1 in $\mathcal{L}$ such that $s_0 \preccurlyeq s_1$:

$$\frac{x_1 : A_1 \vdash t : B}{x_1 : s_0 A_1 \vdash t : s_1 B} \qquad \frac{\vdash t : B}{\vdash t : s_1 B} .$$

For any value of n and s_i in $\mathcal{L}$ $(1 \leqslant i \leqslant n)$ of length $k \geqslant 1$ we can derive

$$\frac{x_1 : A_1, \ldots, x_n : A_n \vdash t : B}{x_1 : s_1 A_1, \ldots, x_n : s_n A_n \vdash t : \S^k B} .$$

We call these derived rules *multiple promotions* and denote them by (*mprom*).

We adopt the convention of identifying the types T and εT, where ε is the empty word. We will consider variables for words, for which we distinguish two classes: (i) *bicolored* variables, denoted as $u, v, w \ldots$ are valued in $\mathcal{L}$,
(ii) *monocolored* variables, denoted as $p, q, r \ldots$ 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 and transitive relation on types given by

$$\begin{array}{rcll} u\alpha & \leqslant & u'\alpha & \text{iff} \quad u \preccurlyeq u', \\ u(A_1 \multimap A_2) & \leqslant & u'(A'_1 \multimap A'_2) & \text{iff} \quad u \preccurlyeq u',\ A'_1 \leqslant A_1 \text{ and } A_2 \leqslant A'_2. \end{array}$$

In fact we have:

Lemma 5. *If $A_1 \leqslant A_2$ then there exists a term t such that $x : A_1 \vdash_{LAL} t : A_2$ and $t \stackrel{\eta *}{\to} x$, so $\lambda x.t \stackrel{\eta *}{\to} \lambda x.x$.*

The idea of this lemma is that if $A_1 \leqslant A_2$ then $A_1 \vdash A_2$ is provable in LAL, not by a mere axiom in general but by an expansion of axiom where only promotion steps are modified, which gives as lambda-term an η-expansion of identity.

Proof (Lemma 5)**.** The proof is by induction over A_1. If $A_1 = u_1 A'_1$ and $A_2 = u_2 A'_2$ with $u_1 \preccurlyeq u_2$ and $A'_1 \leqslant A'_2$, then by induction hypothesis: $x : A'_1 \vdash_{LAL} t : A'_2$ with $t \stackrel{\eta *}{\to} x$. By applying the (*mprom*) rule we described before we get: $x : A_1 \vdash_{LAL} t : A_2$.

ARTICLE IN PRESS

$$\frac{}{x : A_1 \vdash x : A_2}\ A_1 \leqslant A_2\ \text{(var)} \qquad \frac{\Gamma \vdash t : B}{\Gamma, x : A \vdash t : B}\ \text{(weak)}$$

$$\frac{\Gamma_1 \vdash t_1 : A_1 \multimap B \quad \Gamma_2 \vdash t_2 : A_2}{\Gamma_1, \Gamma_2 \vdash (t_1 t_2) : B}\ A_2 \leqslant A_1\ \text{(appl)} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \multimap B}\ \text{(abst)}$$

$$\frac{\Gamma_1 \vdash t_1 : a_1 A'_1 \ \cdots\ \Gamma_n \vdash t_n : a_n A'_n \quad x_1 : A_1, \ldots, x_n : A_n \vdash t : B}{\Gamma_1, \ldots, \Gamma_n \vdash t\{t_i/x_i\} : a_0 B}\ a_i A'_i \leqslant a_0 A_i\ \text{(prom)(1)}$$

$$\frac{\Gamma \vdash t' : !A \quad x_1 : A_1, \ldots, x_n : A_n, \Delta \vdash t : B}{\Gamma, \Delta \vdash t\{t'/x_1, \ldots, x_n\} : B}\ !A \leqslant A_i\ \text{(contr)}$$

Fig. 6. Type assignment LALs.

Consider the case $A_1 = B_1 \multimap B_2$, $A_2 = B'_1 \multimap B'_2$. We have: $B_2 \leqslant B'_2$, $B'_1 \leqslant B_1$. By i.h. we get $y : B'_1 \vdash_{LAL} t_1 : B_1$ and $x : B_2 \vdash_{LAL} t_2 : B'_2$ with $t_1 \overset{\eta *}{\to} y$, $t_2 \overset{\eta *}{\to} x$. So we have $z : B_1 \multimap B_2, y : B'_1 \vdash_{LAL} (z\ t_1) : B_2$. By Lemma 1 we thus get that $z : B_1 \multimap B_2, y : B'_1 \vdash_{LAL} t_2\{(z\ t_1)/x\} : B'_2$, and applying rule (*abst*) we have $z : B_1 \multimap B_2 \vdash_{LAL} t : B'_1 \multimap B'_2$, for $t = \lambda y.t_2\{(z\ t_1)/x\}$.
Finally observe that by the hypothesis on t_1 and t_2 we have:
$\lambda y.t_2\{(z\ t_1)/x\} \overset{\eta *}{\to} \lambda y.(z\ t_1) \overset{\eta *}{\to} \lambda y.(z\ y) \overset{\eta *}{\to} z$.
Therefore the induction hypothesis is valid for $A_1 \multimap A_2$, which ends the proof. □

Lemma 5 suggests considering $\leqslant$ as a subtyping relation. Now we can reformulate our type-assignment system using this relation: this is the system LALs (LAL with subtyping) defined in Fig. 6.

Conditions with Fig. 6:
(1) if $n \geqslant 2$ then $a_0 = \S$.
Observe that LAL rules can be seen as particular cases of LALs rules. We have:

Proposition 6. *If $\Gamma \vdash_{LAL} t : A$ then $\Gamma \vdash_{LALs} t : A$.*
Conversely, if $\Gamma \vdash_{LALs} t : A$ then there exists a term t' such that $t' \overset{\eta *}{\to} t$ and $\Gamma \vdash_{LAL} t' : A$.

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 $A_1 \leqslant A$, we apply Lemma 5 and get a derivation of $x : A_1 \vdash_{LAL} t' : A$ with $t' \overset{\eta *}{\to} x$.
- 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 $t'_i \overset{\eta *}{\to} t_i$ for $i = 1, 2$. By Lemma 5 as $!A \leqslant A_i$ for $i = 1, 2$ we get terms w_i with $w_i \overset{\eta *}{\to} x_i$ and such that

$x_i' : !A \vdash_{LAL} w_i : A_i$ holds. By using these two judgments, the judgment $x_1 : A_1, x_2 : A_2, \Delta \vdash t_2 : B$ and Lemma 1 (two times) we get
$x_1' : !A, x_2' : !A, \Delta \vdash_{LAL} t_2'\{w_1/x_1, w_2/x_2\} : B$.
Finally applying rule (*contr*) to this last judgment and $\Gamma \vdash t_1 : !A$ we obtain: $\Gamma, \Delta \vdash_{LAL} t'' : B$, for $t'' = (t_2'\{w_1/x_1, w_2/x_2\})\{t_1/x_1', x_2'\}$.
As $t_2' \overset{\eta *}{\to} t_2$, $w_i \overset{\eta *}{\to} x_i'$ we have
$t_2'\{w_1/x_1, w_2/x_2\} \overset{\eta *}{\to} t_2\{x_1'/x_1, x_2'/x_2,\}$so $t'' \overset{\eta *}{\to} t_2\{t_1/x_1, x_2\}$.
- the case of a (prom) rule is handled in a similar way to that of (contr). □

This proposition is only stated to relate formally LALs to LAL. What is important is that t' is extensionally equivalent to t. In practice one could execute directly LALs typed terms with the same complexity bound as LAL typed terms by adapting in a straightforward way the light lambda-calculus introduced by Terui in [23] (basically it would require allowing substituting a §-typed variable by a !-typed term, which does not alter the polynomial bound).

Our main motivation for considering LALs instead of LAL is to make type inference easier. However note that even before considering inference, as a type system LALs is more flexible than LAL:
(1) typing is more versatile: a typed term can be applied to more arguments,
(2) the contraction rule is more general: identified variables do not need to have the same type.

By (1) we mean that, for example: in LAL a term $t : (!A \multimap \S B) \multimap C$ cannot be directly applied to an argument $u : \S A \multimap !B$; this needs first *re*typing u or t, for instance retyping u with type $!A \multimap \S B$ (thus losing some information on u). In LALs the application can be done with the actual types. Therefore LALs typing allows for a more general usage of typed terms.

5. 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:

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

where the a_is 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 $s_1 \leqslant s_2$. 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* ϕ of $\mathcal{S}$ is a map $\phi : Par(\mathcal{S}) \to \mathcal{L}$ such that for any p in $Par(\mathcal{S}) \cap \mathcal{V}_m$, $\phi(p) \in \{\S\}^*$ (ϕ is *compatible with colors*). We also denote by ϕ 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 ϕ of $\mathcal{S} = I_1 \wedge \dots \wedge I_N$ is a *solution* of $\mathcal{S}$ if for any $1 \leqslant j \leqslant N$ the inequation I_j holds when each variable a is replaced by $\phi(a)$.

Example 1. Consider $\mathcal{S}$ given by

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

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

Remark 2. Note that equations over words (on the binary alphabet) with concatenation (as considered e.g. in [12]) can be seen as a special case: an equation $s_1 = s_2$ can be encoded as $(s_1 \preccurlyeq s_2) \wedge (s_2 \preccurlyeq s_1)$.

6. Abstract typing

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

- finding the general form 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 we can define a notion of *canonical term construction* and that each term has a finite number of such constructions (Section 6.2). After that we will be ready to describe our type inference method.

6.1. Abstract types

Let us call *abstract types* types built with word variables: $T := \alpha \mid T \multimap T \mid aT$, where a belongs to $\mathcal{V} \cup \mathcal{L}$. Remember that we identify εT and T.

As with LAL there is a forgetful map from abstract types to simple types, which we denote again as [.]. 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 word 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* $\phi : Par(T) \to \mathcal{L}$ compatible with colors, we define $\phi(T)$ as the LAL type obtained by replacing in T word variables by their image:

$$\begin{aligned} \phi(\alpha) &= \alpha, \\ \phi(T_1 \multimap T_2) &= \phi(T_1) \multimap \phi(T_2), \\ \phi(aT) &= \phi(a)\phi(T). \end{aligned}$$

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

$$\begin{aligned} \mathcal{T}(T_1 \leqslant T_2) &= \mathit{false} \quad \text{if } [T_1] \neq [T_2] \\ \mathcal{T}(s\alpha \leqslant s'\alpha)) &= (s \preccurlyeq s') \\ \mathcal{T}(s(A_1 \multimap A_2) \leqslant s'(A_1' \multimap A_2')) &= (s \preccurlyeq s') \wedge \mathcal{T}(A_1' \leqslant A_1) \wedge \mathcal{T}(A_2 \leqslant A_2'). \end{aligned}$$

Fig. 7. Map $\mathcal{T}$.

(var), (weak), (abst) as in Fig.6.

$$\frac{\Gamma_1 \vdash t_1 : s(A_1 \multimap B) \quad \Gamma_2 \vdash t_2 : A_2}{\Gamma_1, \Gamma_2 \vdash (t_1 t_2) : B} \; s = \varepsilon,\, A_2 \leqslant A_1 \text{ (appl)}$$

$$\frac{\Gamma_1 \vdash t_1 : A_1' \; \cdots \; \Gamma_n \vdash t_n : A_n' \quad x_1 : A_1, \ldots, x_n : A_n \vdash t : B}{\Gamma_1, \ldots, \Gamma_n \vdash t\{t_i/x_i\} : vB} \; A_i' \leqslant vA_i \text{(prom) (2)}$$

$$\frac{\Gamma \vdash t' : A \quad x_1 : A_1, \ldots, x_n : A_n, \Delta \vdash t : B}{\Gamma, \Delta \vdash t\{t'/x_1, \ldots, x_n\} : B} \; A \leqslant !u\widehat{A},\, !u\widehat{A} \leqslant A_i \text{ (contr)}$$

Fig. 8. Abstract typing: LALa.

Fig. 7 defines 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). For a technical reason we will need to consider that $\mathcal{T}$ can also be applied to inequations on words (acting as identity): $\mathcal{T}(s_1 \preccurlyeq s_2) = (s_1 \preccurlyeq s_2)$.

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

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

Conditions:

in (prom): (2) if $n \geqslant 2$ then $v \in \mathcal{V}_m$ else $v \in \mathcal{V}_b$.

in (contr): $u \in \mathcal{V}_b$ and is *fresh* (does not appear in the rest of the derivation).

In fact all we want to impose for (*contr*) is that A is of the form $!A'$, for some A', and that $A \leqslant A_i$ for $1 \leqslant i \leqslant n$. This is equivalent to the solvability of $A \leqslant !u\widehat{A} \leqslant A_i$ with a fresh u. Condition (2) on (*prom*) is analogous to the condition (1) we had for promotion in LAL and LALs.

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. Note that for (appl) we use a constraint directly expressed on words: $s = \varepsilon$, which can be equivalently written as $s \preccurlyeq \varepsilon$.

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

Problem 1 (A.t.d. instantiation problem)**.** *Given an abstract type derivation $\mathcal{D}$, does it have a solution ϕ?*

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

$$\frac{\Gamma_1 \vdash t_1 : A \to B \quad \Gamma_2 \vdash t_2 : A}{\Gamma_1, \Gamma_2 \vdash (t_1 t_2) : B}\ (\text{appl}) \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\ (\text{abst})$$

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

$$\frac{\Gamma \vdash t' : A \quad x_1 : A, \ldots, x_n : A, \Delta \vdash t : B}{\Gamma, \Delta \vdash t\{t'/x_1, \ldots, x_n\} : B}\ (\text{contr})\ n \geqslant 2$$

Fig. 9. Type assignment ILS.

$$\frac{}{x \vdash x}\ (\text{var}) \qquad \frac{\Gamma \vdash t}{\Gamma, x \vdash t}\ (\text{weak})$$

$$\frac{\Gamma_1 \vdash t_1 \quad \Gamma_2 \vdash t_2}{\Gamma_1, \Gamma_2 \vdash (t_1 t_2)}\ (\text{appl}) \qquad \frac{\Gamma, x \vdash t}{\Gamma \vdash \lambda x.t}\ (\text{abst})$$

$$\frac{\Gamma_1 \vdash t_1 \ \cdots \ \Gamma_n \vdash t_n \quad x_1, \ldots, x_n \vdash t}{\Gamma_1, \ldots, \Gamma_n \vdash t\{t_i/x_i\}}\ (\text{prom})$$

$$\frac{\Gamma \vdash t' \quad x_1, \ldots, x_n, \Delta \vdash t}{\Gamma, \Delta \vdash t\{t'/x_1, \ldots, x_n\}}\ (\text{contr})\ n \geqslant 2$$

Fig. 10. Rules for term constructions.

We will address this problem for a restricted class of abstract type derivations that we will define in the next section (canonical derivations).

6.2. From term constructions to abstract derivations

From now on we consider LALs derivations with multiple promotions. If $\mathcal{D}$ is an LALs derivation, we denote by $[D]$ the tree of judgements obtained by replacing each LAL formula A by $[A]$. Then $[\mathcal{D}]$ is a simple type derivation that we call *simple type skeleton* of $\mathcal{D}$. We want to give a direct description of simple type 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 Fig. 9, with the conditions:

(1) in rule (*contr*) we require that $n \geqslant 2$ and that $x_1, \ldots, x_n$ belong to $FV(t)$,
(2) in (*prom*), all x_i should belong to $FV(t)$.

We call this set of rules ILS (intuitionistic logic with sharing). Derivations in this system will allow us to handle simple type skeletons of (some) LALs derivations.

We might even want to keep less information from LALs derivations and erase types altogether. For that we consider trees of judgements of the form $\Gamma \vdash t$, where Γ is a set of variables and t is a term built from the rules in Fig. 10 (adapted from ILS rules) and with conditions (1) and (2). We call such a tree a *term construction*. Note that to any ILS or LALs derivation $\mathcal{D}$ we can associate a term construction by erasing all type annotations; we will denote it as $\mathcal{T}(\mathcal{D})$.

Let us fix some vocabulary for LALs derivations, ILS derivations and term constructions. We say an application of the (*prom*) rule is *basic* if all t_i are variables and all Γ_i have only one variable. We say an occurrence of $(weak)$ rule $\mathcal{D}$ is *terminal* if it is the last rule of $\mathcal{D}$ or if it is followed only by a sequence of $(weak)$ rules.

Definition 1. A term construction is canonical if a basic (*prom*) is never r.h.s. premise of another (*prom*) and each $(weak)$ rule is either terminal or followed by an (*abst*) rule on the weakened variable. An ILS (resp. LALs) derivation is canonical if the associated term construction is canonical.

We have:

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

Proof (Sketch)**.** Consider an LALs derivation $\mathcal{D}$ of $\Gamma \vdash t : A$. If $\mathcal{D}$ contains an occurrence of $(weak)$ which is not terminal and not followed by an (*abst*) rule on the weakened formula, then we can commute this rule top-down with other rules until it meets either:
(i) a (*prom*) rule,
(ii) a (*contr*) rule on the weakened formula
or (iii) an (*abst*) rule on the weakened formula or a terminal $(weak)$ rule.

In case (i) we remove the premise $\Gamma_i \vdash t_i : A_i$ of (*prom*) corresponding to the weakened formula A_i and add instead $(weak)$ rules on Γ_i after (*prom*). In case (ii) we remove the $(weak)$ rule, which decreases the arity *n* of the (*contr*) rule. Repeating this procedure we eventually get an LALs derivation $\mathcal{D}_1$ such that any $(weak)$ rule is terminal or followed by an (*abst*) rule on the weakened formula.

The derivation $\mathcal{D}_1$ can still contain (*contr*) rules of arity $n = 1$. For such an occurrence $(contr)_0$ one can perform commutations of rules until the r.h.s. premise of $(contr)_0$ is a (var) rule; in this last case we remove both $(contr)_0$ and the (var) rule. This way we obtain an LALs derivation $\mathcal{D}_2$ satisfying the same conditions as $\mathcal{D}_1$ and such that all its (*contr*) rules have arity $n \geqslant 2$. It follows that $\mathcal{D}_2$ satisfies conditions (1) and (2) from above and $[\mathcal{D}_2]$ is an ILS derivation.

Finally we turn $\mathcal{D}_2$ into an LALs derivation $\mathcal{D}_3$ such that $[\mathcal{D}_3]$ is a canonical ILS derivation by replacing if necessary some consecutive promotions by one multiple promotion. □

Proposition 9. *There is an algorithm that given a term t gives all canonical term constructions of $FV(t) \vdash t$; there is a finite number of such canonical derivations.*

A proof of this proposition is given in Appendix A.

Definition 2. We say a term construction $\mathcal{T}_1$ of $x_1, \dots, x_n \vdash t$ admits ILS type judgement $x_1 : A_1, \dots, x_n : A_n \vdash t : B$ if there is an ILS derivation $\mathcal{D}$ of this judgement such that $\mathcal{T}(\mathcal{D}) = \mathcal{T}_1$.

$$\frac{\vdash I \quad y, z_1, z_2 \vdash (y\ z_1\ z_2)}{y \vdash t} \text{ (contr)}$$

Fig. 11. Example for Remark 3.

Proposition 10. *The term constructions admit the principal typing property (for ILS types): let $\mathcal{T}_1$ be a term construction, then it admits an ILS type judgement $x_1 : A_1, \ldots, x_n : A_n \vdash t : B$ (called its principal type) such that any $\Delta \vdash t : C$ is an ILS type judgement of $\mathcal{T}_1$ iff it is of the form $x_1 : \sigma A_1, \ldots, x_n : \sigma A_n \vdash t : \sigma B$, where σ is a substitution on type variables.*

Remark 3. Note that the principal type of a term construction $\mathcal{T}_1$ of $\Gamma \vdash t$ is not necessarily the principal type of t, because of sharing ((*contr*) rule). For instance consider the term $t = (y\ I\ I)$. It has principal type $y : (\alpha_1 \to \alpha_1) \to (\alpha_2 \to \alpha_2) \multimap \beta \vdash t : \beta$. However if $\mathcal{T}$ denotes the term construction of t ending with the rule of Fig. 11 (and the rest of the term construction done in the natural way), then $\mathcal{T}$ has principal type $y : (\alpha_1 \to \alpha_1) \to (\alpha_1 \to \alpha_1) \to \beta \vdash t : \beta$.

Finally we have:

Proposition 11. *Assume there is an LALs derivation $\mathcal{D}$ of $\Gamma \vdash t : A$ with associated term construction $\mathcal{T}_1 = \mathcal{T}(\mathcal{D})$. If $\mathcal{T}_1$ admits ILS principal type $\Delta \vdash t : B$ then there exists an LALs derivation $\mathcal{D}'$ with conclusion $\Gamma' \vdash t : A'$ such that $[\Gamma'] = \Delta$, $[A'] = B$ and $\mathcal{T}(D') = \mathcal{T}_1$.*

A result of this kind was proved in [9] for elementary affine logic (EAL) type derivations. However in the system EAL considered in this paper sharing was not allowed and we cannot here adapt directly this result to our purpose. Therefore we give a self-standing proof of Proposition 11 in Appendix B, using some techniques from [9].

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 α then $fd(A) = \{u\alpha,\ u \in \mathcal{V}_b\}$,
- if $A = A_1 \to A_2$ we take: $fd(A) = \{u(\overline{A_1} \multimap \overline{A_2}),\quad \text{s.t. } \overline{A_i} \in fd(A_i), Par(\overline{A_1}) \cap Par(\overline{A_2}) = \emptyset, u \notin Par(\overline{A_i})$ for $i = 1, 2\}$.

The idea of interpolating modality rules into an intuitionistic derivation in order to find the EAL derivations of the corresponding term was the original idea of [8]. Here, given a canonical ILS derivation $\mathcal{D}$ we decorate it into an 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 an LALa (var) rule of conclusion $x : \overline{A_1} \vdash x : \overline{A_2}$;
- 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 $\mathcal{D}_1, \ldots, \mathcal{D}_n, \mathcal{D}_0$ by a (*prom*) rule, we assume as before the $\overline{\mathcal{D}_i}$ have disjoint parameters, take v not occurring in the $\overline{\mathcal{D}_i}$ and belonging to $\mathcal{V}_m$ if $n \geqslant 2$, $\mathcal{V}_b$ if $n \leqslant 1$, and define $\overline{\mathcal{D}}$ by a (*prom*) rule on the $\overline{\mathcal{D}_i}$ with parameter v.

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

Lemma 12. *The judgment $\Gamma \vdash_{LALs} t : A$ is derivable iff there exists a canonical abstract derivation $\mathcal{D}$ with a solution ϕ such that $\phi(\mathcal{D})$ is a derivation of $\Gamma \vdash_{LALs} t : A$.*

6.3. From decorations to constraints

Now, given a term t with $FV(t) = \{x_1, \ldots, x_n\}$ our method for deciding of its typability in LALs is the following one:

- using Proposition 9 enumerate the canonical term constructions of $x_1, \ldots, x_n \vdash t$;
- using Proposition 10 determine the principal types of these canonical term constructions and the corresponding canonical ILS derivations $\mathcal{D}_1, \ldots, \mathcal{D}_n$;
- using the decoration procedure enumerate the corresponding canonical abstract derivations $\mathcal{D}'_1, \ldots, \mathcal{D}'_n$;
- for each such canonical abstract derivation $\mathcal{D}'_i$ search if it has a solution ϕ, in which case we get an LALs type derivation $\phi(\mathcal{D}'_i)$ for t.

By Lemma 12 we know that a solution to this procedure will yield a suitable LALs type derivation for the term t. Conversely if the term t is LALs typable then by Proposition 11 and Lemma 12 we know that the procedure will provide a canonical abstract derivation $\mathcal{D}'_i$ which has a solution ϕ. 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 $\mathcal{C}ons(\mathcal{D})$ in the following inductive way (keeping the notations of Fig. 8):

$$
\begin{aligned}
\mathcal{C}ons(\mathcal{D}) &= \{A_1 \leqslant A_2\} \text{ if } \mathcal{D} = (var),\\
\mathcal{C}ons(\mathcal{D}) &= \mathcal{C}ons(\mathcal{D}_1), \quad \text{ if } \mathcal{D} \text{ is obtained from } \mathcal{D}_1 \text{ by } (abst) \text{ or } (weak),\\
\mathcal{C}ons(\mathcal{D}) &= \mathcal{C}ons(\mathcal{D}_1) \cup \mathcal{C}ons(\mathcal{D}_2) \cup \{s \leqslant \varepsilon, A_2 \leqslant A_1\}\\
&\quad \text{if } \mathcal{D} \text{ is obtained from } \mathcal{D}_1, \mathcal{D}_2 \text{ by } (appl),\\
\mathcal{C}ons(\mathcal{D}) &= \cup_{i=0}^{n} \mathcal{C}ons(\mathcal{D}_i) \cup \{A'_i \leqslant v A_i, 1 \leqslant i \leqslant n\},\\
&\quad \text{if } \mathcal{D} \text{ is obtained from } \mathcal{D}_1, \ldots \mathcal{D}_n, \mathcal{D}_0 \text{ by } (prom),\\
\mathcal{C}ons(\mathcal{D}) &= \mathcal{C}ons(\mathcal{D}_1) \cup \mathcal{C}ons(\mathcal{D}_2) \cup \{A \leqslant !u\widehat{A}, !u\widehat{A} \leqslant A_i, 1 \leqslant i \leqslant n\}\\
&\quad \text{if } \mathcal{D} \text{ is obtained from } \mathcal{D}_1, \mathcal{D}_2 \text{ by } (contr).
\end{aligned}
$$

Given an abstract derivation $\mathcal{D}$, we know by lemma 7 that a map ϕ is a solution of $\mathcal{D}$ iff ϕ is a solution of the system of inequations $\mathcal{T}(\mathcal{C}ons(\mathcal{D}))$. Say a system of inequations $\mathcal{S}$ is a *canonical abstract derivation system* (c.a.d. system) if there exists a canonical abstract derivation $\mathcal{D}$ such that $\mathcal{S} = \mathcal{T}(\mathcal{C}ons(\mathcal{D}))$.

7. Solving the constraints

7.1. 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 [12]). But we know of no general result which would apply to the systems of inequations we are considering. However we can here take advantage of a strong property of the systems we are interested in, *stratification*.

Definition 3. Let $\mathcal{S}$ be a system of inequations and ϕ be a solution. Say ϕ is a *stratified solution* of $\mathcal{S}$ if there exists a *depth* function d defined on $Par(\mathcal{S})$ and the (occurrences of) constants in $\mathcal{S}$, with values in $\mathbb{N}$, and such that:
(1) for $s \leqslant s'$ in $\mathcal{S}$ with $s = u_1 s_1$, $s' = u_2 s_2$ we have $d(u_1) = d(u_2)$,
(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 $ind(s)$ (*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).

Considering a stratified solution ϕ, d and an inequation (I) of the system, if $(I) = s_1 \leqslant s_2$ we will denote $d(I) = d(s_1)$ (so also $d(s_2)$) and $ind(I) = ind(s_1)$ (also $ind(s_2)$).

Example 2. Consider the system $\mathcal{S}$ given in Fig. 12. It does not have any stratified solution. Indeed, assume there was one ϕ with depth d. Inequation I_1 implies that $|\phi(u_2)| = 1$. Inequation I_2 tells that $\phi(u_3) \neq \varepsilon$ (because $\S \leqslant !$ does not hold) and so $|\phi(u_3)| \geqslant 1$. By I_2 we have $d(u_2) < d(u_3)$, and by I_3: $d(u_3) \geqslant d(u_2)$, hence a contradiction.

Proposition 13. *If $\mathcal{S}$ is a canonical abstract derivation system, then it is stratified.*

The notion of depth defined here coincides with the notion of depth considered in proof-nets and mentioned in Section 4.

Proof (Proposition 13)**.** We define two new functions $k(.,.)$ and $l(.,.)$. If w is an occurrence of parameter in an abstract type A then $k(w, A)$ is the list of parameters in the scope of which w is, in A. For instance
if $A = u_1 u_2 (p_1 u_3 \alpha \multimap u_4 w p_2 \beta)$ then $k(w, A) = \langle u_1, u_2, u_4 \rangle$.
We consider a canonical abstract derivation $\mathcal{D}$. Let R be an occurrence of rule in $\mathcal{D}$. We denote by $l(R, \mathcal{D})$ the list of parameters associated to promotion rules with right premise

$$\begin{cases} \S & \leqslant u_2 & I_1 \\ u_1 \S & \leqslant u_2 ! u_3 & I_2 \\ u_3 u_2 & \leqslant ! u_4 & I_3 \end{cases}$$

Fig. 12.

below R in the derivation tree $\mathcal{D}$ (excluding R itself): if R is the last rule of $\mathcal{D}$ then $l(R, \mathcal{D}) = \varepsilon$, otherwise assuming R occurs in derivation $\mathcal{D}_0$, then

- if $\mathcal{D}$ is obtained from $\mathcal{D}_0$ by a (prom) with parameter v and $\mathcal{D}_0$ is right premise of the rule then $l(R, \mathcal{D}) = v :: l(R, \mathcal{D}_0)$;
- if $\mathcal{D}$ is obtained from $\mathcal{D}_0$ by a (prom) and $\mathcal{D}_0$ is not right premise, or if $\mathcal{D}$ is obtained from $\mathcal{D}_0$ by another rule then $l(R, \mathcal{D}) = l(R, \mathcal{D}_0)$.

Finally we define for a parameter v occurring in $\mathcal{D}$ a list $l(v, \mathcal{D})$:

- if v is introduced by a promotion rule R then: $l(v, \mathcal{D}) = l(R, \mathcal{D})$,
- if v is introduced by a contraction rule R then: $l(v, \mathcal{D}) = l(R, \mathcal{D}) :: !$,
- if v is introduced by a (var) rule R of conclusion $x : A_1 \vdash x : A_2$ and v occurs in A_i then: $l(v, \mathcal{D}) = l(R, \mathcal{D}) :: k(v, A_i)$.

Note that this definition makes sense because $\mathcal{D}$ is a canonical abstract derivation and so each parameter is introduced by at most one rule (and in the case of (var) appears in only one of the two formulas A_1, A_2). □

To simplify the notation we will write $l(R)$ (resp. $l(v)$) for $l(R, \mathcal{D})$ (resp. $l(v, \mathcal{D})$) when there is no ambiguity.

Now, assume we have a solution ϕ of $\mathcal{D}$. We want to prove that this solution is stratified. For that we define a function $d(.)$ on parameters in the following way:

$d(v) = \sum_{i=1}^{n} |\phi(v_i)|$ if $l(v) = \langle v_n, \ldots, v_1 \rangle$,
$d(v) = 0$ if $l(v)$ is the empty list.

We need to show that ϕ and $d(.)$ satisfy conditions (1) and (2) of definition 3. For that we use an intermediary lemma, whose proof is given in appendix C:

Lemma 14. *If parameter u_1 (resp. u_2) occurs in B_1 (resp. B_2) and constraint $B_1 \leqslant B_2$ is associated to rule R, then*:
$l(u_1) = l(R) :: k(u_1, B_1)$,
$l(u_2) = l(R) :: k(u_2, B_2)$.

Let us show that ϕ, d satisfy condition (1). Let $s \preccurlyeq s'$ be in $\mathcal{S} = \mathcal{T}(\mathcal{C}ons(\mathcal{D}))$ with $s = u_1 s_1$, $s' = u_2 s_2$: there exists a rule R of $\mathcal{D}$ with constraint $B_1 \leqslant B_2$ such that: $s \preccurlyeq s' \in \mathcal{T}(B_1 \leqslant B_2)$. Then either $u_1 \in B_1$ and $u_2 \in B_2$, or $u_1 \in B_2$ and $u_2 \in B_1$. Let us assume for instance we are in the first case (the second one is similar).

By Lemma 14 we have

$$l(u_1) = l(R) :: k(u_1, B_1), \tag{1}$$
$$l(u_2) = l(R) :: k(u_2, B_2). \tag{2}$$

Let us denote $k(u_1, B_1) = \langle v_n, \ldots, v_1 \rangle, k(u_2, B_2) = \langle w_m, \ldots, w_1 \rangle$. As ϕ is a solution of $\mathcal{D}$ we have $\phi(B_1) \leqslant \phi(B_2)$, which in particular implies that: $\sum_{i=1}^{n} |\phi(v_i)| = \sum_{j=1}^{m} |\phi(w_j)|$.

So from equalities (1) and (2) and the definition of d we get $d(u_1) = d(u_2)$; condition (1) of Definition 3 is thus satisfied.

Let us examine condition (2) of definition 3. Let $s = u_1 \ldots u_n$ be a member of inequation of $\mathcal{S}$. Then s appears in a formula A of a constraint associated to a rule R; so u_i and u_{i+1}

both appear in A, and by lemma 14 we have
$l(u_{i+1}) = l(R) :: k(u_{i+1}, A)$,
$l(u_i) \quad = l(R) :: k(u_i, A)$.

As the sequence $u_1 \dots u_n$ occurs in A we have
$k(u_{i+1}, A) = k(u_i, A) :: u_i$,
so $l(u_{i+1}) \quad = l(u_i) :: u_i$.
Hence by definition of d: $d(u_{i+1}) = d(u_i) + |\phi(u_i)|$. Therefore the solution ϕ is stratified. □

Now we have:

Theorem 4. *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 5. *The existence of a solution for a c.a.d. system is decidable.*

With Section 6.3 we then get:

Corollary 6. *The derivation instantiation problem (Problem 1) for canonical abstract derivations is decidable.*

and from that our main result follows:

Theorem 7. *Given a lambda-term t with $FV(t) = \{x_1, \dots, x_n\}$, one can decide whether there exists an LALs derivation of conclusion $x_1 : A_1, \dots, x_n : A_n \vdash_{LALs} t : A$.*

Let us come back to the proof of theorem 4. 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 15. *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 ϕ is a solution, then define ψ 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 $! \preccurlyeq \S$ the map ψ is also a solution of $\mathcal{S}$.

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

7.2. Informal description of the algorithm

We now give an algorithm to decide whether a system has a stratified solution. When applied to a stratified system the algorithm will thus allow to find a solution or determine that there is none.

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 (measure 0), 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 $a_1 \dots a_n \leqslant s_1 !_0 s_2 \quad (I)$ of $\mathcal{S}$. We can assume the a_is are characters (! or §) or word variables. After instantiation by a solution, the two words on each side of I should have same length, and as $! \leqslant \S$ 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

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

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 ε. 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 $!_0$.

Now, a naive non-deterministic algorithm could proceed by repeating the following task: choose a ! on r.h.s.; eliminate it and eliminate its residuals. When reaching $\mathcal{S}$ with $mes(\mathcal{S}) = 0$ solve it and track back a solution to the original system if there is one. However this procedure does not terminate in general. If we consider the tree of all its non-deterministic runs (with systems as nodes, and a branching for each choice of inequation splitting) it has infinite branches.

Our algorithm will refine this procedure by pruning some branches of the search tree, thus keeping only finite branches. The key feature is that we only look for stratified solutions, so at some points we already know that no stratified solution will be found and we can give up the search.

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 into *steps* consisting in eliminating a r.h.s. ! (the way we just sketched) and creating residuals. Basically, the trick is that a member of inequation cannot get twice a residual of the same !. There will be a possibility of interrupting a step (hence stopping the current execution without giving a solution) if we get into a configuration with no stratified solution. In such a case the algorithm should be run again.

If $\mathcal{S}$ does not have any stratified solution then all executions end with an interruption or a $\mathcal{S}'$ with $mes(\mathcal{S}') = 0$ and no stratified solution. If $\mathcal{S}$ has stratified solutions, then at least one of them is reached by an execution.

7.3. The algorithm

We will handle the following data:

- $\mathcal{R}$: 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. $\overline{\mathcal{S}}$ and $\underline{\mathcal{S}}$ are disjoint subsets of $\mathcal{S}$ such that $\mathcal{S} = \overline{\mathcal{S}} \cup \underline{\mathcal{S}}$;
- 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 of $\overline{\mathcal{S}}$ with residuals of $!_0$; $\underline{\mathcal{S}}$ is the subset of inequations of $\mathcal{S}$ that cannot contain residuals of $!_0$.

Notation: we denote by $\mathcal{S}\langle s \to u\rangle$ the result of the substitution in a system $\mathcal{S}$ of inequations of all occurrences of a variable u by the word s.

The algorithm is then given by:

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

  ROUND:
  - $\overline{\mathcal{S}} := \mathcal{S}$; $\underline{\mathcal{S}} := \emptyset$; $P := \varepsilon$ (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:

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

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

    STEP:
    pop $(I_l, !_0)$ from P: $\quad I_l : \quad u_1 \dots u_n \leqslant s_1 !_0 s_2$
    guess u_j (bicolored variable or !) such that $!_0$ "belongs" to u_j;
    $\overline{\mathcal{S}} := (\overline{\mathcal{S}} \backslash \{I_l\})\langle u_{j1}!u_{j2} \to u_j\rangle$
    $\underline{\mathcal{S}} := (\underline{\mathcal{S}} \cup \{u_1 \dots u_{j1} \leqslant s_1 \ (I_{l1})\} \cup \{u_{j2} \dots u_n \leqslant s_2 \ (I_{l2})\}$
    if $\underline{\mathcal{S}}$ contains an occurrence of u_j: STOP.
    $\mathcal{S} := \overline{\mathcal{S}} \cup \underline{\mathcal{S}}$
    $\mathcal{R} := \mathcal{R} \cup \{u_j = u_{j1}!u_{j2}\}$
    if u_j is a variable, push on P the inequations of $\overline{\mathcal{S}}$ in which a u_j has been replaced in the r.h.s. (i.e. containing residuals in the r.h.s.)
    end of STEP.

  end of ROUND.
- When we have $mes(\mathcal{S}) = 0$ we compute the existence of a (monocolored) solution, and if there is one, using $\mathcal{R}$ we track back a solution of the original system $\mathcal{S}_0$.

We give an example of run in Appendix D.

7.4. Properties of the algorithm

7.4.1. Termination

Each STEP trivially terminates since it contains no loop. Each ROUND does also terminate because STEP decreases $|\overline{\mathcal{S}}|$ by 1. Now let us observe that each ROUND decreases $mes(\mathcal{S})$ by 1. Indeed a ROUND selects a $!_0$ in a r.h.s. of $\mathcal{S}$ and removes it. During a ROUND, the only ! that can be added in r.h.s. members of $\mathcal{S}$ are residuals of the $!_0$. The residuals of $!_0$ in $\overline{\mathcal{S}}$ are stored in P and removed; if a residual is to appear in $\underline{\mathcal{S}}$ the algorithm stops (line 5 of STEP). So each ROUND does decrease $mes(\mathcal{S})$ by 1, hence the non-deterministic algorithm terminates.

Remark 8. Note that the crucial argument for termination is the fact that STEP decrements $|\overline{\mathcal{S}}|$, which comes from line 3 of STEP:
$\overline{\mathcal{S}} := (\overline{\mathcal{S}}\backslash\{I_l\})\langle u_{j1}!u_{j2} \to u_j\rangle$.

This means that the algorithm will not try to remove residuals from the inequations (I_{l1}) and (I_{l2}) coming from (I_l). In fact if $\underline{\mathcal{S}}$ contains an occurrence of u_j in an r.h.s., then the algorithm stops (line 5 of STEP): the reason for that is that in this case the remaining reachable solutions of the system are not stratified (so we prune the corresponding subtree of the tree of possible executions). Indeed the algorithm is not complete with respect to all solutions, but only with respect to stratified solutions.

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 ith 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 $|\mathcal{S}_{i+1}| \leqslant 2|\mathcal{S}_i|$. In conclusion the length of any run is bounded by $2^{mes(\mathcal{S})} \cdot |\mathcal{S}|$.

7.4.2. Correctness

It is rather easy to check 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 ψ 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 ϕ 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 $\mathcal{R}$.

7.4.3. Completeness

Let us now examine the completeness issue, which is more delicate. Assume $\mathcal{S}_0$ has a stratified solution ϕ with depth d and let us show that there is a run of the algorithm leading to this solution. We describe one possible execution of the non-deterministic algorithm, using the knowledge of ϕ, d. We denote by d the depth function at any moment of the execution (its domain is extended to the variables introduced during the execution).

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 $s_1 \leqslant s_2$ can contain a residual of $!_0$ only if it satisfies $d(s_1) \leqslant d_0 < ind(s_1)$. The execution of the ROUND has the following invariant:
for any inequation $s_1 \leqslant s_2$ of $\underline{\mathcal{S}}$ we have: $d(s_1) > d_0$ or $ind(s_1) \leqslant d_0$. (*)

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

$$u_1 \dots u_{j1} \leqslant s_1 \quad (I_{l1}),$$
$$u_{j2} \dots u_n \leqslant s_2 \quad (I_{l2}).$$

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

$$\phi'(v) = \phi(v) \text{ for } v \neq u_j,$$
$$\phi'(u_{j1}) = t_1 \text{ prefix of } \phi(u_j) \text{ of length } d_0 - d(u_j) - 1,$$
$$\phi'(u_{j2}) = t_2 \text{ suffix of } \phi(u_j) \text{ of length } |\phi(u_j)| - |\phi'(u_{j1})| - 1.$$

We also define in the same way d' with $d'(u_{j1}) = d(u_j)$, $d'(u_{j2}) = d_0 + 1$. Then ϕ', 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: $ind'(I_{l1}) = d_0$, $d'(I_{l2}) = d'(u_{j2}) = d_0 + 1$. So execution of line 4 of STEP preserves the invariant (*). Indeed I_{l1} and I_{l2} cannot contain any further residual of $!_0$, which is why we don't include them in $\overline{\mathcal{S}'}$.

This execution will therefore terminate with a system $\mathcal{S}'$ with $mes(\mathcal{S}') = 0$ (without raising STOP). The system $\mathcal{S}'$ has a solution from which we can get a stratified solution to the initial system.

7.4.4. 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.

8. Conclusion and future work

The study of Linear logic proof-theory made possible the introduction of systems capturing complexity classes such as Light linear logic or more recently Soft linear logic [18] for polynomial time. We wanted here to make the point that this domain can be interfaced with typing techniques, for instance by taking advantage of type-theory tools such as subtyping or constraints solving. We followed the approach of using light logic as a non-standard type system used on ordinary lambda-calculus to verify a complexity property, namely that the programs can be run with a polynomial time bound (using proof-nets or light affine lambda-calculus as intermediate language). The first step in this direction was to establish decidability of type-inference, which we did for the quantifier-free fragment. For that we considered constraints on words; we showed that the systems arising in our setting satisfied an important regularity property linked to stratification and gave a decision procedure for these systems.

Several questions arise at this point. Can this approach be partially extended to the polymorphic setting, for instance if we start from a system F-typed term rather that from

an untyped term? The practicability of type inference and its modularity should also be investigated. We considered as source language here standard λ-calculus for the sake of generality, but as in the procedure one has to first choose a suitable sharing of subterms (Section 6.2) it might be more reasonable in practice to start with an intermediate language with explicit sharing possibly as in [9], or a generalization of the one in [23].

Appendix A. Proof of Proposition 9

Proof. We want to establish decidability and do not try here to give an efficient algorithm.

We want to construct, proceeding bottom-up, all possible canonical term constructions for $\Gamma \vdash t$. 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 judgements, with lexicographic order:

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

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

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 $n \geqslant 2$ and $x_1, \ldots, x_n \in FV(t)$, if t' is not a variable then $|t\{t'/x_1, \ldots, x_n\}| > |t|$ and $|t\{t'/x_1, \ldots, x_n\}| > |t'|$, and so the measure decreases. If t' is a variable then $|t\{t'/x_1, \ldots, x_n\}| = |t|$, but $rep(t\{t'/x_1, \ldots, x_n\}) > rep(t)$.

Let us examine the $(prom)$ rule. If it is not basic, 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 basic $(prom)$ however leaves the measure unchanged.

So basic $(prom)$ is the only instance of rule that leaves the measure unchanged. But it follows from the definition of canonical term construction that there are no two consecutive applications of basic $(prom)$. Hence the height of a branch is bounded by $2\ mes(\Gamma \vdash t : B)$, where $\Gamma \vdash t$ is the initial judgement. □

Appendix B. Proof of Proposition 11

The proof of Proposition 11 will require some intermediary definitions and results. The key result will be Proposition 21.

B.1. LALs inequations and type variables substitution

Let $\mathcal{I}$ denote a set of inequations $\{T_i \leqq U_i, 1 \leqslant i \leqslant n\}$ where the T_i and U_i are LAL types. We consider the problem of finding a substitution σ from type variables to LAL formulas (called *LAL substitution*) such that $\sigma T_i \leqslant \sigma U_i$ holds for any $1 \leqslant i \leqslant n$. Note that this problem is not the same as the one considered in Section 6.1 where we were searching substitutions of word variables solving a system of inequations (in that case type variables were unchanged).

Let $[\mathcal{I}]$ denote the set of IL equations $[\mathcal{I}] = \{[T_i] \equiv [U_i], 1 \leqslant i \leqslant n\}$. The problem of finding a substitution mapping variables to IL types (IL substitution) satisfying $[\mathcal{I}]$ is a unification problem and if it has a solution there is a *most* general unifier (m.g.u.).

If σ is an LAL substitution, $[\sigma]$ will denote the IL substitution defined by $[\sigma](\alpha) = [\sigma(\alpha)]$ for all type variables α. Note that:

Lemma 16. *If σ is a solution of $\mathcal{I}$ then $[\sigma]$ is a solution of $[\mathcal{I}]$.*

Now, to relate solutions of $[\mathcal{I}]$ to solutions of $\mathcal{I}$ we will consider a new operation on LAL types. Given an IL formula F we define a partial map $(.)|_F$ from LAL formulas to LAL formulas by

$$\begin{aligned} A|_{F_1 \to F_2} &= \begin{cases} u(A_1|_{F_1} \multimap A_2|_{F_2}) & \text{if } A = u(A_1 \multimap A_2), \\ \text{undefined} & \text{if } A = u\beta \ (\beta \text{ type variable}), \end{cases} \\ A|_\alpha &= u\alpha, \ \text{if } A = u(A_1 \multimap A_2) \text{ or } A = u\beta. \end{aligned}$$

The following lemmas can then be easily verified:

Lemma 17. *If $A|_F$ is defined then $[A|_F] = F$. Moreover for any A we have $A|_{[A]} = A$.*

Lemma 18. *If $A|_F$ is defined then for any u of $\{!, \S\}^\star$ we have: $(uA)|_F = u(A|_F)$.*

Lemma 19. *If A, B are LAL formulas, we have $A \leqslant B$ holds iff: $[A] = [B]$ and for all F of IL, $A|_F$ is defined iff $B|_F$ is defined, and $A|_F \leqslant B|_F$.*

Lemma 20. *If A is an LAL formula, F an IL formula, σ an IL substitution and $[A] = \sigma F$, then $A|_F$ is defined.*

We are now equipped to prove the following proposition:

Proposition 21. *Let $\mathcal{I}$ be a set of LAL inequations. If $\mathcal{I}$ admits a solution and τ denotes the m.g.u. of $[\mathcal{I}]$ then there exists a solution σ of $\mathcal{I}$ such that $[\sigma] = \tau$.*

Proof. Assume $\mathcal{I}$ has a solution σ and let us introduce a solution σ_0 such that: $[\sigma_0] = \tau$.

We define the LAL substitution σ_0 by: $\sigma_0\alpha = (\sigma\alpha)|_{\tau\alpha}$, for all type variables α.

The fact that $\sigma_0\alpha$ is well defined for all α follows from Lemma 20, the fact that τ is m.g.u. of $[\mathcal{I}]$ and that $[\sigma]$ is a solution of $[\mathcal{I}]$.

We then obtain:

Lemma 22. *For all formula A of LAL, we have*: $(\sigma A)|_{\tau A}$ *is well-defined and* $\sigma_0 A = (\sigma A)|_{\tau A}$.

Proof. By structural induction on A, using the definition of σ_0, of $(.)|_F$ and Lemma 18. □

Now, let $A \leqq B$ be an inequation of $\mathcal{I}$. As σ is a solution of $\mathcal{I}$ we have: $\sigma A \leqslant \sigma B$ (*).

As $[A] \equiv [B]$ is an equation of $[\mathcal{I}]$ and τ is a solution of $[\mathcal{I}]$ we have: $\tau[A] = \tau[B]$.

By Lemma 22 $(\sigma A)|_{\tau A}$ and $(\sigma B)|_{\tau B}$ are well-defined.

Then by Lemma 19 as $\tau[A] = \tau[B]$ from (*) we get: $(\sigma A)|_{\tau A} \leqslant (\sigma B)|_{\tau B}$. So finally by Lemma 22: $\sigma_0 A \leqslant \sigma_0 B$. Therefore σ_0 is a solution of $\mathcal{I}$. □

B.2. Relating LALs derivations to simple type derivations

In this section we will use a calculus and methods inspired from [9]. Indeed for proving Proposition 10 we need to use a term syntax for LALs proofs. Light affine lambda-calculus [23] is an efficient tool for this purpose, but as we do not need here to establish computational properties on these terms but wish to have a syntax close to the presentation of LALs we adopted we use a term calculus analogous to that of [8,9]) for Elementary affine Logic.

The set of LA-terms Λ^{LA} is defined by the grammar:

$$M ::= x \mid \lambda x.M \mid (M\ M) \mid\ \dagger M\ [M/x_1, \ldots, M/x_n] \mid [M]_{M=x_1,\ldots,x_n},$$

where † stands for ! or §.

The *erasure* of an LA-term M is a lambda-term M^- defined by

$$(\dagger M\ [M_1/x_1, \ldots, M_n/x_n])^- = M^-\{M_1^-/x_1, \ldots, M_n^-/x_n\},$$
$$([M]_{M_1=x_1,\ldots,x_n})^- = M^-\{M_1^-/x_1, \ldots, x_n\}$$

and $(.)^-$ commutes to the other constructions.

We say an LA-term M term is *valid* if any variable occurs at most once in M and if for any subterm of the form $!M\ [M/x_1, \ldots, M/x_n]$ we have $n \leqslant 1$.

The rules of LALs can be seen as typing rules for LA-terms, by adapting in the straightforward way the rules of Fig. 6: the only changes are on (*prom*) and (*contr*):

$$\frac{\Gamma_1 \vdash M_1 : a_1 A_1' \ \cdots\ \Gamma_n \vdash M_n : a_n A_n' \quad x_1 : A_1, \ldots, x_n : A_n \vdash M : B}{\Gamma_1, \ldots, \Gamma_n \vdash a_0 M\ [M_1/x_1, \ldots, M_n/x_n] : a_0 B}\ a_i A_i' \leqslant a_0 A_i \text{ (prom)},$$

$$\frac{\Gamma \vdash M_1 : !A \quad x_1 : A_1, \ldots, x_n : A_n, \Delta \vdash M_0 : B}{\Gamma, \Delta \vdash [M_0]_{M_1=x_1,\ldots,x_n} : B}\ !A \leqslant A_i \text{ (contr)}$$

with for (*prom*) the condition:

(1) if $n \geqslant 2$ then $a_0 = \S$.

Any derivation of a judgement $\Gamma \vdash_{LALs} M : A$ with M an LA-term obviously corresponds to an LALs proof. Note that actually the untyped term M gives all the structure of the proof (assuming a convention for $(weak)$ rules), but the types used in the (var) rules.

If M is LALs typable then it is a valid term. We want to give a method to compute for a valid LA-term M all its LALs type judgements, if any. For that we associate to M a type judgement scheme $T_L(M) = \langle \Gamma; \Delta \rangle$ and a set of inequations $\mathcal{I}(M)$ corresponding to the conditions expressing the validity of a derivation. The pair $T_L(M)$ and the set $\mathcal{I}(M)$ are defined by induction on M as below; note that they are defined up to renaming of type variables.

- if $M = x$:
  then $T_L(M) = \langle x : \alpha; \alpha \rangle$, $\mathcal{I}(x) = \emptyset$.
- if $M = \lambda x.M_1$ and $T_L(M_1) = \langle \Gamma; B \rangle$:
  then $T_L(M) = \langle \Delta; A \multimap B \rangle$ with: $A = \Gamma(x)$ and $\Delta = \Gamma \backslash \{x\}$ if $\Gamma(x)$ is defined; $A = \alpha$ (fresh variable) and $\Delta = \Gamma$ otherwise.
- if $M = (M_1\ M_2)$ and $T_L(M_i) = \langle \Gamma_i; A_i \rangle$ for $i = 1, 2$:
  then $T_L(M) = \langle \Gamma_1, \Gamma_2; \alpha \rangle$ (α fresh variable) and $\mathcal{I}(M) = \mathcal{I}(M_1) \cup \mathcal{I}(M_2) \cup \{A_1 \leqq (A_2 \multimap \alpha)\}$.
- if $M = \dagger M_0\ [M_1/x_1, \ldots, M_n/x_n]$ and $T_L(M_0) = \langle x_1 : A_1, \ldots, x_n : A_n; B \rangle$, and $T_L(M_i) = \langle \Gamma_i; A'_i \rangle$ for $1 \leqslant i \leqslant n$:
  then $T_L(M) = \langle \Gamma_1, \ldots, \Gamma_n; \dagger B \rangle$ and $\mathcal{I}(M) = \cup_{i=0}^{n} \mathcal{I}(M_i) \cup \{A'_i \leqq \dagger A_i, 1 \leqslant i \leqslant n\}$.
- if $M = [M_0]_{M_1 = x_1, \ldots, x_n}$ and $T_L(M_0) = \langle x_1 : A_1, \ldots, x_n : A_n, \Delta; B \rangle$, $T_L(M_1) = \langle \Gamma; A \rangle$:
  then $T_L(M) = \langle \Gamma, \Delta; B \rangle$ and $\mathcal{I}(M) = \cup_{i=0}^{1} \mathcal{I}(M_i) \cup \{A \leqq !\alpha, !\alpha \leqq A_i, 1 \leqslant i \leqslant n\}$ (where α is fresh).

In a similar way we consider the typing of valid LA-terms in IL. The judgements will be denoted as $\Gamma \vdash_{ILS} M : A$. The typing rules are the same as for typing ordinary lambda-terms in IL but for the two extra rules (*prom*) and (*contr*):

$$\frac{\Gamma_1 \vdash_{ILS} M_1 : A_1 \ \cdots \ \Gamma_n \vdash_{ILS} M_n : A_n \quad x_1 : A_1, \ldots, x_n : A_n \vdash_{ILS} M : B}{\Gamma_1, \ldots, \Gamma_n \vdash_{ILS} a_0 M\ [M_1/x_1, \ldots, M_n/x_n] : B} \ \text{(prom), (1)}$$

$$\frac{\Gamma \vdash_{ILS} M_1 : A \quad x_1 : A, \ldots, x_n : A, \Delta \vdash_{ILS} M_0 : B}{\Gamma, \Delta \vdash_{ILS} [M_0]_{M_1 = x_1, \ldots, x_n} : B} \ \text{(contr)}.$$

(1) if $n \geqslant 2$ then $a_0 = \S$.

As in the case of LALs, to a valid LA-term M we associate a type judgement scheme $T_I(M) = \langle \Gamma; \Delta \rangle$ and a set of equations $\mathcal{E}(M)$ defined by

- if $M = x$:
  then $T_I(M) = \langle x : \alpha; \alpha \rangle$, $\mathcal{E}(x) = \emptyset$.
- if $M = \lambda x.M_1$ and $T_I(M_1) = \langle \Gamma; B \rangle$:
  then $T_I(M) = \langle \Delta; A \multimap B \rangle$ with $A = \Gamma(x)$ and $\Delta = \Gamma \backslash \{x\}$ if $\Gamma(x)$ is defined; $A = \alpha$ (fresh variable) and $\Delta = \Gamma$ otherwise.
- if $M = (M_1\ M_2)$ and $T_I(M_i) = \langle \Gamma_i; A_i \rangle$ for $i = 1, 2$:
  then $T_I(M) = \langle \Gamma_1, \Gamma_2; \alpha \rangle$ (α fresh variable) and $\mathcal{E}(M) = \mathcal{E}(M_1) \cup \mathcal{E}(M_2) \cup \{A_1 \equiv (A_2 \multimap \alpha)\}$.

- if $M = \dagger M_0\ [M_1/x_1, \ldots, M_n/x_n]$ and $T_I(M_0) = \langle x_1 : A_1, \ldots, x_n : A_n; B\rangle$, and $T_I(M_i) = \langle \Gamma_i; A'_i\rangle$ for $1 \leqslant i \leqslant n$:
  then $T_I(M) = \langle \Gamma_1, \ldots, \Gamma_n; \dagger B\rangle$ and $\mathcal{E}(M) = \cup_{i=0}^{n} \mathcal{E}(M_i) \cup \{A'_i \equiv A_i, 1 \leqslant i \leqslant n\}$.
- if $M = [M_0]_{M_1 = x_1, \ldots, x_n}$ and $T_I(M_0) = \langle x_1 : A_1, \ldots, x_n : A_n, \Delta; B\rangle$, $T_I(M_1) = \langle \Gamma; A\rangle$:
  then $T_I(M) = \langle \Gamma, \Delta; B\rangle$ and $\mathcal{E}(M) = \cup_{i=0}^{1} \mathcal{E}(M_i) \cup \{A \equiv \alpha, \alpha \equiv A_i, 1 \leqslant i \leqslant n\}$ (where α is fresh).

As in Section 6.2 for LALs or ILS typing derivations, to an LA-term M we can associate a term construction $\mathcal{T}(M)$ in a natural way.

Finally we have:

Proposition 23. *Let M be a valid LA-term. The judgement $\Gamma \vdash_{LALs} M : A$ (resp. $\Gamma \vdash_{ILS} M : A$) is derivable iff there exists a solution σ of $\mathcal{I}(M)$ (resp. $\mathcal{E}(M)$) such that $\sigma\Delta \subseteq \Gamma$, $\sigma B = A$, where $\langle \Delta, B\rangle = T_L(M)$ (resp. $\langle \Delta, B\rangle = T_I(M)$).*

Proposition 24. *The valid terms of Λ_{LA} admit the principal type property in ILS and the principal type of M is given by $\langle \tau\Delta; \tau B\rangle$ where $\langle \Delta; B\rangle = T_I(M)$ and τ is the m.g.u. of $\mathcal{E}(M)$.*

Proof. It follows directly from the definitions of $\mathcal{E}(M)$ and $T_I(M)$ and the properties of the m.g.u. □

Lemma 25. *Assume M is a valid LA-term and $\Gamma \vdash_{ILS} M : A$ is its principal ILS type. Then if $t = M^-$ and $\mathcal{T}_1 = \mathcal{T}(M)$, the principal type of $\mathcal{T}_1$ is $\Gamma \vdash_{ILS} t : A$.*

Proposition 26. *Let M be a valid LA-term. If M is typable in LALs then there exists Γ, A in LAL such that $\Gamma \vdash_{LALs} M : A$ is derivable and $[\Gamma] \vdash_{ILS} M : [A]$ is the principal type of M in IL.*

Proof. One can prove by structural induction on valid LA-terms that for any M:
if $T_L(M) = \langle \Delta; B\rangle$ then $T_I(M) = \langle [\Delta]; [B]\rangle$ and $\mathcal{E}(M) = [\mathcal{I}(M)]$.

If M is LALs typable then it is ILS typable, thus by Proposition 23 both $\mathcal{I}(M)$ and $\mathcal{E}(M)$ have solutions. Then as $\mathcal{E}(M) = [\mathcal{I}(M)]$, by Proposition 21 if τ denotes the m.g.u. of $\mathcal{E}(M)$ there exists a solution σ of $\mathcal{I}(M)$ such that $[\sigma] = \tau$. Hence by Proposition 23 again, $\sigma\Delta \vdash_{LALs} M : \sigma B$ is derivable. Moreover we have: $T_I(M) = \langle [\Delta]; [B]\rangle$ so by Proposition 24, $T_I(M) = \langle \tau[\Delta]; \tau[B]\rangle$ is the ILS principal type of M. Finally $\tau[\Delta] = [\sigma\Delta]$ and $\tau[B] = [\sigma B]$. □

Now we can prove Proposition 11:

Proof (Proposition 11)**.** Let $\mathcal{D}$ be an LALs derivation of $\Gamma \vdash_{LALs} t : A$. This derivation gives an LALs type derivation for an LA-term M, with conclusion $\Gamma \vdash_{LALs} M : A$, and we have $M^- = t$. Let $\mathcal{T}_1 = \mathcal{T}(\mathcal{D})$; we have $\mathcal{T}_1 = \mathcal{T}(M)$. Now let $\Delta \vdash M : B$ be the principal ILS type of M, then by Proposition 26 there exists Γ', A' in LAL such that $\Gamma' \vdash_{LALs} M : A'$ is derivable and $[\Gamma'] = \Delta$ and $[A'] = B$. This implies that $\Gamma' \vdash_{LALs} t : A'$ has a derivation $\mathcal{D}'$,

with $\mathcal{T}(\mathcal{D}') = \mathcal{T}(M) = \mathcal{T}_1$. Moreover by Lemma 25, $\Delta \vdash t : B$ is the principal type of $\mathcal{T}_1$, which concludes the proof. □

Appendix C. Proof of Lemma 14

To prove Lemma 14 we first establish another lemma:

Lemma 27. *If parameter v occurs in A and rule R has a premise of the form $\Gamma \vdash t : A$ or $\Gamma, x : A \vdash t : B$, then*:

- *if $R = (prom)$ with associated parameter u and the judgement containing A is right premise of this rule then*: $l(v) = l(R) :: u :: k(v, A)$,
- *otherwise*: $l(v) = l(R) :: k(v, A)$.

Proof (Lemma 27)**.** Let R' be the rule introducing v and n be the number of rules between R' and R in the corresponding branch of the derivation tree ($n = 1$ if R immediately follows R'). Note that $n \geqslant 1$ because R does not introduce v (one of its premises contains A). We proceed by induction on n.

- If $n = 1$ then:

$$l(R') = \begin{cases} l(R) :: u & \text{if } R = (prom) \text{ and the conclusion of } R' \text{ is right premise of } R\ , \\ l(R) & \text{otherwise .} \end{cases}$$

  - if $R' = (prom)$ we have: $l(v) = l(R')$, $k(v, A) = \varepsilon$, so

$$l(R') = \begin{cases} l(R) :: u :: k(v, A) & \text{if } R = (prom) \text{ and the conclusion of } R' \text{ is right premise of } R, \\ l(R) :: k(v, A) & \text{otherwise.} \end{cases}$$

  - if $R' = (contr)$ then: $l(v) = l(R') :: !$, $k(v, A) = !$, so the property is satisfied.
  - if $R' = (var)$ then : $l(v) = l(R') :: k(v, A)$, so the property is valid.
- If $n \geqslant 2$ let R'' be the rule immediately preceding R with conclusion containing A. Then:

$$l(R') = \begin{cases} l(R) :: u & \text{if } R = (prom) \text{ and the conclusion of } R' \text{ is right premise of } R\ , \\ l(R) & \text{otherwise .} \end{cases}$$

Using the induction hypothesis on R'' and the fact that A is in the conclusion of R'' we get: $l(v) = l(R'') :: k(v, A)$, so the hypothesis is also satisfied by R. □

Proof (Lemma 14)**.** If $R = (var)$, $(appl)$ or $(contr)$ then the statement follows directly from applying lemma 27 to R, B_1 and B_2.

Otherwise if $R = (prom)$ then using the notations of Fig. 8 there are an A'_i and an A_j such that $B_1 = A'_i$ and $B_2 = vA_j$. By applying Lemma 27 respectively to R and A'_i and to

R and A_j we get
$l(u_1) = l(R) :: k(u_1, A'_i) = l(R) :: k(u_1, B_1)$,
$l(u_2) = l(R) :: v :: k(u_2, A_j) = l(R) :: k(u_1, B_2)$,
which ends the proof. □

Appendix D. Example of constraints resolution

We illustrate the use of the algorithm from 7.3 on a simple example. We consider the following lambda-term: $t = (P\ (P\ \lambda x.x))$, with $P = \lambda y.\lambda z.(y\ (y\ z))$. Note that the term P is the Church integer 2.

Here is one canonical term construction for P (among several possible):

$$\mathcal{T}_1 = \quad \frac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{z \vdash z \qquad y_2 \vdash y_2}{y_2, z \vdash (y_2\ z)}\,\text{(appl)} \qquad y_1 \vdash y_1}{y_1, y_2, z \vdash (y_1\ (y_2\ z))}\,\text{(appl)}}{y_1, y_2 \vdash \lambda z.(y_1\ (y_2\ z))}\,\text{(abst)}}{y_1, y_2 \vdash \lambda z.(y_1\ (y_2\ z))}\,\text{(prom)}}{y \vdash \lambda z.(y\ (y\ z))}\,\text{(contr)}}{y \vdash \lambda z.(y\ (y\ z))}\,\text{(prom)}}{\vdash P}\,\text{(abst)}$$

From that we give a canonical term construction $\mathcal{T}_2$ for t:

$$\frac{\mathcal{T}_1\ \dfrac{\vdash \cdots}{\vdash P} \qquad \dfrac{\dfrac{\mathcal{T}_1\ \dfrac{\vdash \cdots}{\vdash P} \qquad \dfrac{\dfrac{x \vdash x}{\vdash \lambda x.x}}{\vdash \lambda x.x}\,\text{(prom)}}{\vdash (P\ \lambda x.x)}\,\text{(appl)}}{\vdash (P\ \lambda x.x)}\,\text{(prom)}}{\vdash (P\ (P\ \lambda x.x))}\,\text{(appl)}$$

Some other examples of term constructions for t can be obtained for instance by applying as last rule:

$$\frac{\vdash P \qquad z_1, z_2 \vdash (z_1\ (z_2\ \lambda x.x))}{\vdash t}\,\text{(contr)}.$$

We will consider in the following the term constructions $\mathcal{T}_1$ and $\mathcal{T}_2$. They respectively have principal types $\vdash P : (\alpha \to \alpha) \to (\alpha \to \alpha)$ and $\vdash t : (\alpha \to \alpha)$. Let us denote $\beta = \alpha \multimap \alpha$.

ARTICLE IN PRESS

From the ILS derivation obtained from $\mathcal{T}_1$ with its principal type we define the following abstract derivation $\mathcal{D}_1$ for P:

$$\mathcal{D}_1 = \quad (1)\; \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{y_1 : \beta, y_2 : \beta, z : \alpha \vdash (y_1(y_2 z)) : \alpha}}{y_1 : \beta, y_2 : \beta \vdash \lambda z.(y_1(y_2 z)) : \beta}\ \text{(prom)}}{y_1 : q\beta, y_2 : q\beta \vdash \lambda z.(y_1(y_2 z)) : q\beta}\ \text{(contr)}}{y : !w\beta \vdash \lambda z.(y(yz)) : q\beta}\ \text{(prom)}}{\dfrac{y : u!w\beta \vdash \lambda z.(y(yz)) : uq\beta}{\vdash P : u!w\beta \multimap uq\beta}}$$

with constraints: (1) $\quad !w\beta \leqslant q\beta$.

The derivation $\mathcal{D}_1$ is actually not quite a canonical abstract derivation as we have not decorated the types in (var) rules (replacing for instance an occurrence of β by $u_1(u_2\alpha \multimap u_2\alpha)$). This is only to keep a small number of word variables for the good readability of the example.

We denote by $\mathcal{D}'_1$ the derivation $\mathcal{D}_1$ where parameters u, w and q have been renamed into u', w', q' respectively. Then the following derivation $\mathcal{D}$ is an abstract derivation for t:

$$(3)\; \dfrac{\mathcal{D}'_1\ \dfrac{\vdash \cdots}{\vdash P : u'!w'\beta \multimap u'q'\beta} \qquad \dfrac{(2)\; \dfrac{\mathcal{D}_1\ \dfrac{\vdash \cdots}{\vdash P : u!w\beta \multimap uq\beta} \qquad \dfrac{\dfrac{x : \alpha \vdash x : \alpha}{\vdash \lambda x.x : \beta}}{\vdash \lambda x.x : v\beta}\ \text{(prom)}}{\vdash (P\ \lambda x.x) : uq\beta}\ \text{(appl)}}{\vdash (P\ \lambda x.x) : v'uq\beta}\ \text{(prom)}}{\vdash (P\ (P\ \lambda x.x)) : u'q\beta}\ \text{(appl)}$$

with constraints: (2) $\quad v\beta \leqslant u!w\beta$, $\quad$ (3) $\quad v'uq\beta \leqslant u'!w'\beta$. Note that u, v, w, u', v' are bicolored variables whereas q, q' are monocolored variables. The set of type constraints $\mathit{Cons}(\mathcal{D})$ is given in Fig. 13 and the corresponding set $\mathcal{S}$ of word constraints in Fig. 14.

We demonstrate the execution of the algorithm from section 7.3 on $\mathcal{S}$. We give the state of the system after line 3 of ROUND and at the end of each run of the STEP subprocedure (Figs. 15–17).

We have $\mathit{mes}(\mathcal{S}_5) = 0$. The corresponding problem on integers is given in Fig. 18. The set of solutions to $\mathcal{E}$ is given in Fig. 19.

$$\mathit{Cons}(\mathcal{D}) = \left\{ \begin{array}{ccc} !w\beta & \leqslant & q\beta \\ v\beta & \leqslant & u!w\beta \\ !w'\beta & \leqslant & q'\beta \\ v'uq\beta & \leqslant & u'!w'\beta \end{array} \right.$$

Fig. 13.

$$\mathcal{S} = \left\{ \begin{array}{cccc} !w & \preccurlyeq & q & I_1 \\ v & \preccurlyeq & u!w & I_2 \\ v'uq & \preccurlyeq & u'!w' & I_3 \\ !w' & \preccurlyeq & q' & I_4 \end{array} \right.$$

Fig. 14.

ARTICLE IN PRESS

ROUND 1

$$\overline{\mathcal{S}_1} = \begin{cases} !w \preccurlyeq q & I_1 \\ v \preccurlyeq u!w & I_2 \\ v'uq \preccurlyeq u'!_0 w' & I_3 \\ !w' \preccurlyeq q' & I_4 \end{cases}$$
$\underline{\mathcal{S}_1} = \emptyset$
$\mathcal{R}_1 = \emptyset$
$P_1 = \langle (I_3, !_0) \rangle$

ROUND 1, STEP 1

$$\overline{\mathcal{S}_2} = \begin{cases} !w \preccurlyeq q & I_1 \\ v \preccurlyeq u_1 !_1 u_2 !w & I_{21} \\ !w' \preccurlyeq q' & I_4 \end{cases}$$
$$\underline{\mathcal{S}_2} = \begin{cases} v'u_1 \preccurlyeq u' & I_{31} \\ u_2 q \preccurlyeq w' & I_{32} \end{cases}$$
$\mathcal{R}_2 = \{ u = u_1 !u_2$
$P_1 = \langle (I_{21}, !_1) \rangle$

Fig. 15.

ROUND 1, STEP 2

$$\overline{\mathcal{S}_3} = \begin{cases} !w \preccurlyeq q & I_1 \\ !w' \preccurlyeq q' & I_4 \end{cases}$$
$$\underline{\mathcal{S}_3} = \begin{cases} v'u_1 \preccurlyeq u' & I_{31} \\ u_2 q \preccurlyeq w' & I_{32} \\ v_1 \preccurlyeq u_1 & I_{211} \\ v_2 \preccurlyeq u_2 !_2 w & I_{212} \end{cases}$$
$$\mathcal{R}_3 = \begin{cases} u = u_1 !u_2 \\ v = v_1 !v_2 \end{cases}$$
$P_3 = \varepsilon$

ROUND 2

$$\overline{\mathcal{S}_4} = \begin{cases} !w \preccurlyeq q & I_1 \\ v_1 \preccurlyeq u_1 & I_{211} \\ v_2 \preccurlyeq u_2 !_2 w & I_{212} \\ v'u_1 \preccurlyeq u' & I_{31} \\ u_2 q \preccurlyeq w' & I_{32} \\ !w' \preccurlyeq q' & I_4 \end{cases}$$
$\underline{\mathcal{S}_4} = \emptyset$
$$\mathcal{R}_4 = \begin{cases} u = u_1 !u_2 \\ v = v_1 !v_2 \end{cases}$$
$P_4 = \langle (I_{212}, !_2) \rangle$

Fig. 16.

ROUND 2, STEP 1

$$\overline{\mathcal{S}_5} = \begin{cases} !w \preccurlyeq q & I_1 \\ v_1 \preccurlyeq u_1 & I_{211} \\ v'u_1 \preccurlyeq u' & I_{31} \\ u_2 q \preccurlyeq w' & I_{32} \\ !w' \preccurlyeq q' & I_4 \end{cases}$$
$$\underline{\mathcal{S}_5} = \begin{cases} v_{21} \preccurlyeq u_2 & I_{2121} \\ v_{22} \preccurlyeq w & I_{2122} \end{cases}$$
$$\mathcal{R}_5 = \begin{cases} u = u_1 !u_2 \\ v = v_1 !v_2 \\ v_2 = v_{21} !v_{22} \end{cases}$$
$P_5 = \varepsilon$

$$\mathcal{S}_5 = \begin{cases} !w \preccurlyeq q & I_1 \\ v_1 \preccurlyeq u_1 & I_{211} \\ v_{21} \preccurlyeq u_2 & I_{2121} \\ v_{22} \preccurlyeq w & I_{2122} \\ v'u_1 \preccurlyeq u' & I_{31} \\ u_2 q \preccurlyeq w' & I_{32} \\ !w' \preccurlyeq q' & I_4 \end{cases}$$

Fig. 17.

From that we get a set of solutions to $\mathcal{S}_5$, given in Fig. 20 (but note that it is not necessarily the complete set of solutions). Using $\mathcal{R}_5$ we conclude that $\mathcal{S}$ is solvable and has as subset of solutions the set given in Fig. 21, which gives the following possible types for t: $\S^{l'+l_1+k+1}\beta$, with $l', l_1, k \in \mathbb{N}$, so $\S^m \beta$ for any $m \in \mathbb{N}$.

An alternative way of executing the algorithm on $\mathcal{S}$ would have been to start with the ! of the r.h.s. of I_3 but choose as variable v' instead of u; or to start with the ! of the r.h.s. of I_2.

$$\mathcal{E} = \left\{ \begin{array}{lcl} |!w| & = & |q| \\ |v_1| & = & |u_1| \\ |v_{21}| & = & |u_2| \\ |v_{22}| & = & |w| \\ |v'| + |u_1| & = & |u'| \\ |u_2| + |q| & = & |w'| \\ |!w'| & = & |q'| \end{array} \right. \iff \left\{ \begin{array}{lcl} |w| + 1 & = & |q| \\ |v_1| & = & |u_1| \\ |v_{21}| & = & |u_2| \\ |v_{22}| & = & |w| \\ |v'| + |u_1| & = & |u'| \\ |u_2| + |q| & = & |w'| \\ |w'| + 1 & = & |q'| \end{array} \right.$$

Fig. 18.

$$\left\{ \begin{array}{lcl} k, l_1, l_2, l' & \in & \mathbb{N} \\ |w| & = & k \\ |q| & = & k+1 \\ |u_1| & = & |v_1| = l_1 \\ |u_2| & = & |v_{21}| = l_2 \\ |v_{22}| & = & k \\ |v'| & = & l' \\ |u'| & = & l' + l_1 \\ |w'| & = & l_2 + k + 1 \\ |q'| & = & l_2 + k + 2 \end{array} \right.$$

Fig. 19.

$$\left\{ \begin{array}{lcl} k, l_1, l_2, l' & \in & \mathbb{N} \\ w & = & v_{22} = \S^k \\ q & = & \S^{k+1} \\ u_1 & = & v_1 = \S^{l_1} \\ u_2 & = & v_{21} = \S^{l_2} \\ v' & = & \S^{l'} \\ u' & = & \S^{l'+l_1} \\ w' & = & \S^{l_2+k+1} \\ q' & = & \S^{l_2+k+2} \end{array} \right.$$

Fig. 20.

$$\left\{ \begin{array}{lcl} k, l_1, l_2, l' & \in & \mathbb{N} \\ w & = & \S^k \\ q & = & \S^{k+1} \\ u & = & \S^{l_1}!\S^{l_2} \\ v & = & \S^{l_1}!\S^{l_2}!\S^k \\ u' & = & \S^{l'+l_1} \\ v' & = & \S^{l'} \\ w' & = & \S^{l_2+k+1} \\ q' & = & \S^{l_2+k+2} \end{array} \right.$$

Fig. 21.

References

[1] A. Asperti, Light affine logic, in: Proceedings LICS'98, IEEE Computer Society, Silver Spring, MD, 1998, pp. 300–308.
[2] A. Asperti, L. Roversi, Intuitionistic light affine logic, ACM Trans. Comput. Logic 3 (1) (2002) 137–175.
[3] P. Baillot, Checking polynomial time complexity with types, in: Second International IFIP Conference on Theoretical Computer Science, Montreal, Kluwer Academic Press, Dordrecht, August 2002, pp. 370–382.
[4] P. Baillot, Type inference for polynomial time complexity via constraints on words, Preprint 02-03, LIPN, Université Paris XIII, January 2003.
[5] P. Baillot, Stratified coherence spaces: a denotational semantics for Light Linear Logic, Theoret. Comput. Sci. 318 (1–2) (2004) 29–55.
[6] S. Bellantoni, K.H. Niggl, H. Schwichtenberg, Higher type recursion, ramification and polynomial time, Ann. Pure Appl. Logic 104 (1–3) (2000) 17–30.

ARTICLE IN PRESS

[7] P.N. Benton, G.M. Bierman, V.C.V. de Paiva, J.M.E. Hyland, A term calculus for intuitionistic linear logic, in: Proceedings TLCA'93, Lecture Notes in Computer Science, Vol. 664, Springer, Berlin, 1993, pp. 75–90.
[8] P. Coppola, S. Martini, Typing lambda-terms in elementary logic with linear constraints, in: Proceedings TLCA'01, Lecture Notes in Computer Science, Vol. 2044, Springer, Berlin, 2001, pp. 76–90.
[9] P. Coppola, S. Ronchi della Rocca, Principal typing in elementary affine logic, in: Proceedings TLCA'03, Lecture Notes in Computer Science, Vol. 2701, Springer, Berlin, 2003, pp. 90–104.
[10] V. Danos, J.-B. Joinet, Linear logic and elementary time, Inform. Comput. 183 (1) (2003) 123–137.
[11] V. Danos, J.-B. Joinet, H. Schellinx, On the linear decoration of intuitionistic derivations, Arch. Math. Logic 33 (6) (1994).
[12] V. Diekert, Makanin's algorithm, in: Algebraic Combinatorics on Words (Lothaire), Cambridge University Press, Cambridge, 2001(Chapter 12).
[13] J.-Y. Girard, Linear logic, Theoret. Comput. Sci. 50 (1987) 1–102.
[14] J.-Y. Girard, Light linear logic, Inform. and Comput. 143 (1998) 175–204.
[15] M. Hofmann, Linear types and non-size-increasing polynomial time computation, in: Proceedings of the 14th Symposium on Logic in Computer Science, IEEE Computer Society, 1999, pp. 464–473.
[16] M. Hofmann, Safe recursion with higher types and BCK-algebra, Ann. Pure Appl. Logic 104 (1–3) (2000) 113–166.
[17] M.I. Kanovich, M. Okada, A. Scedrov, Phase semantics for light linear logic, Theoret. Comput. Sci. 294 (3) (2003) 525–549.
[18] Y. Lafont, Soft linear logic and polynomial time, Theoret. Comput. Sci. 318 (1–2) (2004) 163–180.
[19] D. Leivant, J.-Y. Marion, Lambda-calculus characterisations of polytime, Fundam. Inform. 19 (1993) 167–184.
[20] F. Maurel, Nondeterministic light logics and NP-time, in: Proceedings of TLCA'03, Lecture Notes in Computer Science, Vol. 2701, Springer, Berlin, 2003.
[21] A.S. Murawski, C.H.L. Ong, Discreet games, Light Affine Logic and PTIME computation, in: Proceedings CSL'00, Lecture Notes in Computer Science, Vol. 1862, Springer, Berlin, 2000, pp. 427–441.
[22] L. Roversi, A P-time completeness proof for light logics, in: Proceedings CSL'99, Lecture Notes in Computer Science, Vol. 1683, Springer, Berlin, 1999, pp. 469–483.
[23] K. Terui, Light Affine Lambda-calculus and polytime strong normalization, in: Proceedings LICS'01, IEEE Computer Society, Silver Spring, MD, 2001.
[24] K. Terui, Light logic and polynomial time computation, Ph.D. Thesis, Keio University, Tokyo, 2002.
[25] K. Terui, personal communication, 2003.
[26] K. Terui, Light affine set theory: a naive set theory of polynomial time, Stud. Logica 77 (2004) 9–40.
[27] J.B. Wells, Typability and type checking in system F are equivalent and undecidable, Ann. Pure Appl. Logic 98 (1–3) (1999) 111–156.