Verification of Ptime reducibility for system F terms via Dual Light Affine Logic.

Vincent Atassi, Patrick Baillot, Kazushige Terui

Verification of Ptime reducibility for system F terms via Dual Light Affine Logic

Vincent Atassi*
LIPN, Univ. Paris-Nord, France
atassi@lipn.univ-paris13.fr

Patrick Baillot*
LIPN, Univ. Paris-Nord, France
pb@lipn.univ-paris13.fr

Kazushige Terui
NII, Tokyo, Japan
terui@nii.ac.jp

March 27, 2006

Abstract

In a previous work we introduced Dual Light Affine Logic (DLAL) ([BT04]) as a variant of Light Linear Logic suitable for guaranteeing complexity properties on lambda-calculus terms: all typable terms can be evaluated in polynomial time and all Ptime functions can be represented. In the present work we address the problem of typing lambda-terms in second-order DLAL. For that we give a procedure which, starting with a term typed in system F, finds all possible ways to decorate it into a DLAL typed term. We show that our procedure can be run in time polynomial in the size of the original Church typed system F term.

1 Introduction

Several works have studied programming languages with intrinsic computational complexity properties. This line of research, Implicit computational complexity (ICC), is motivated both by the perspective of automated complexity analysis, and by foundational goals, in particular to give natural characterizations of complexity classes, like Ptime or Pspace. Different calculi have been used for this purpose coming from primitive recursion, lambda-calculus, rewriting systems (*e.g.* [BC92, MM00, LM93])... A convenient way to see these systems is in general to describe them as a subset of programs of a larger language satisfying certain criteria: for instance primitive recursive programs satisfying safe/ramified recursion conditions, rewriting systems admitting a termination ordering and quasi interpretation, etc...

Inference. To use such ICC systems for programming purpose it is natural to wish to automatize the verification of the criteria. This way the user could stick to a simple

*Work partially supported by projects CRISS (ACI), GEOCAL (ACI), NO-CoST (ANR)

programming language and the compiler would check whether the program satisfies the criteria, in which case a complexity property would be guaranteed.

In general this decision procedure involves finding a certain *witness*, like a type, a proof or a termination ordering. Depending on the system this witness might be useful to provide more precise information, like an actual bound on the running time, or a suitable strategy to evaluate the program. It might be used as a certificate guaranteeing a particular quantitative property of the program.

Light linear logic. In the present work we consider the approach of Light linear logic (LLL) ([Gir98]), a variant of Linear logic which characterizes polynomial time computation, within the proofs-as-programs correspondence. It includes higher-order and polymorphism, and can be extended to a naive set theory ([Ter04]), in which the provably total functions correspond to the class of polynomial time functions.

The original formulation of LLL by Girard was quite complicated, but a first simplification was given by Asperti with Light Affine Logic (LAL) ([AR02]). Both systems have two modalities (one more than Linear logic) to control duplication. There is a forgetful map to system F terms (polymorphic types) obtained by erasing some information (modalities) in types; if an LAL typed term t is mapped to an F-typed term M we also say that t is a *decoration* of M.

So an LAL program can be understood as a system F program, together with a typing guarantee that it can be evaluated in polynomial time. As system F is a reference system for the study of polymorphically typed functional languages and has been extensively studied, this seems to offer a solid basis to LAL.

However LAL itself is still difficult to handle and following the previous idea for the application of ICC methods, we would prefer to use plain lambda-calculus as a front-end language, without having to worry about the handling of modalities, and instead to delegate the LAL typing part to a type inference engine. The study of this approach was started in [Bai02]. For it to be fully manageable however several conditions should be fulfilled:

1. a suitable way to execute the lambda-terms with the expected complexity bound,
2. an efficient type inference,
3. a typed language which is expressive enough so that a reasonable range of programs is accepted.

The language LAL presents some drawback for the first point, because the LAL typed terms need to be evaluated with a specific graph syntax, *proof-nets*, in order to satisfy the polynomial bound, and plain beta reduction can lead to exponential blow-up. In a previous work ([BT04]) we addressed this issue by defining a subsystem of LAL, called Dual Light Affine Logic (DLAL). It is defined with both linear and non-linear function types. It is complete for Ptime just as LAL and its main advantage is that it is also Ptime sound w.r.t. beta reduction: a DLAL term admits a bound on the length of all its beta reduction sequences. Hence DLAL stands as a reasonable substitute for plain LAL for typing issues.

Concerning point 2, as type inference for system F is undecidable we don't try to give a full-fledged type inference algorithm from untyped terms. Instead, to separate

the polymorphic part issue from the proper DLAL part one, we assume the initial program is already typed in F. Either the system F typing work is left to the user, or one could use a partial algorithm for system F typing for this preliminary phase.

So the contribution of the present work is to define an efficient algorithm to decide if a system F term can be decorated in a DLAL typed term. This was actually one of the original motivations for defining DLAL. We show here that decoration can be performed in polynomial time. This is obtained by taking advantage of intuitions coming from proof-nets, but it is presented in a standard form with a first phase consisting in generating constraints expressing typability and a second phase for constraints solving. One difficulty is that the initial presentation of the constraints involves disjunctions of linear constraints, for which there is no obvious Ptime bound. Hence we provide a specific resolution strategy.

The complete algorithm is already implemented in ML, in a way that follows closely the specification given in the article. It is modular and usable with any linear constraints solver. The code is commented, and available for public download (Section 6). With this program one might thus write terms in system F and verify if they are Ptime and obtain a time upper bound. It should in particular be useful to study further properties of DLAL and to experiment with reasonable size programs.

The point 3 stressed previously about expressivity of the system remains an issue which should be explored further. Indeed the DLAL typing discipline will in particular rule out some nested iterations which might in fact be harmless for Ptime complexity. This is related to the line of work on the study of intensional aspects of Implicit computational complexity ([MM00, Hof03]).

However it might be possible to consider some combination of DLAL with other systems which could allow for more flexibility, and we think a better understanding of DLAL, and in particular of its type inference, is a necessary step in that direction.

Related work. Inference problems have been studied for several ICC systems ([Ama05, HJ03]). Elementary linear logic (EAL) in particular is another variant of Linear logic which characterizes Kalmar elementary time and has applications to optimal reduction. Type inference in the propositional fragment of this system has been studied in [CM01, CRdR03, CDLRdR05] and [BT05] which gives a polynomial time procedure. Type inference for LAL was also investigated, in [Bai02, Bai04]. To our knowledge the present algorithm is however the first one for dealing with polymorphic types in a EAL-related system, and also the first one to infer light types in polynomial time.

Notations. Given a lambda-term t, $FV(t)$ will be the set of its free variables. The prefix relation on words will be denoted by $\leq$.

2 From system F to $DLAL$

The language $\mathcal{L}_F$ of system F types is given by:

$$T, U ::= \alpha \mid T \to U \mid \forall \alpha.T$$

We assume that a countable set of term variables $x^T, y^T, z^T, \ldots$ is given for each type T. The terms of system F are built as follows (here we write M^T to indicate that

the term M has type T):

$$x^T \quad (\lambda x^T.M^U)^{T\to U} \quad ((M^{T\to U})N^T)^U$$

$$(\Lambda\alpha.M^U)^{\forall\alpha.U} \quad ((M^{\forall\alpha.U})T)^{U[T/\alpha]}$$

with the proviso that when building a term $\Lambda\alpha.M^U$, α may not occur freely in the types of free term variables of M (the *eigenvariable condition*).

It is well known that there is no sensible resource bound (i.e. time/space) on the execution of system F terms in general. To impose some bounds, a more refined type system is required. $DLAL$ serves well as such a type system.

The language $\mathcal{L}_{DLAL}$ of DLAL types is given by:

$$A, B ::= \alpha \mid A \multimap B \mid A \Rightarrow B \mid \S A \mid \forall\alpha.A$$

We note $\S^0 A = A$ and $\S^{k+1}A = \S\S^k A$. The erasure map $(.)^-$ from $\mathcal{L}_{DLAL}$ to $\mathcal{L}_F$ is defined by:

$$(\S A)^- = A^-, \quad (A \multimap B)^- = (A \Rightarrow B)^- = A^- \to B^-,$$

and $(.)^-$ commutes to the other connectives. We say $A \in \mathcal{L}_{DLAL}$ is a *decoration* of $T \in \mathcal{L}_F$ if $A^- = T$.

A *declaration* is a pair of the form $x^T : B$ with $B^- = T$. It is often written as $x : B$ for simplicity. A *judgement* is of the form $\Gamma; \Delta \vdash M : A$, where M is a system F term, $A \in \mathcal{L}_{DLAL}$ and Γ and Δ are disjoint sets of declarations. When Δ consists of $x_1 : A_1, \ldots, x_n : A_n$, $\S\Delta$ denotes $x_1 : \S A_1, \ldots, x_n : \S A_n$. The type assignment rules are given on Figure 1. Here, we assume that the substitution $M[N/x]$ used in ($\S$ e) is *capture-free*. Namely, no free type variable α occurring in N is bound in $M[N/x]$. We write $\Gamma; \Delta \vdash_{DLAL} M : A$ if the judgement $\Gamma; \Delta \vdash M : A$ is derivable.

Recall that binary words, in $\{0,1\}^*$, can be given the following type in F:

$$W_F = \forall\alpha.(\alpha \to \alpha) \to (\alpha \to \alpha) \to (\alpha \to \alpha)$$

A corresponding type in DLAL, containing the same terms, is given by:

$$W_{DLAL} = \forall\alpha.(\alpha \multimap \alpha) \Rightarrow (\alpha \multimap \alpha) \Rightarrow \S(\alpha \multimap \alpha)$$

The *depth* $d(A)$ of a $DLAL$ type A is defined by:

$$\begin{array}{rcll} d(\alpha) & = & 0, & d(\forall\alpha.B) = d(B), \\ d(A \multimap B) & = & max(d(A), d(B)), & d(\S A) = d(A) + 1, \\ d(A \Rightarrow B) & = & max(d(A) + 1, d(B)). & \end{array}$$

A type A is said to be Π_1 if it does not contain a negative occurrence of $\forall$; like for instance W_{DLAL}.

The fundamental properties of $DLAL$ are the following [BT04]:

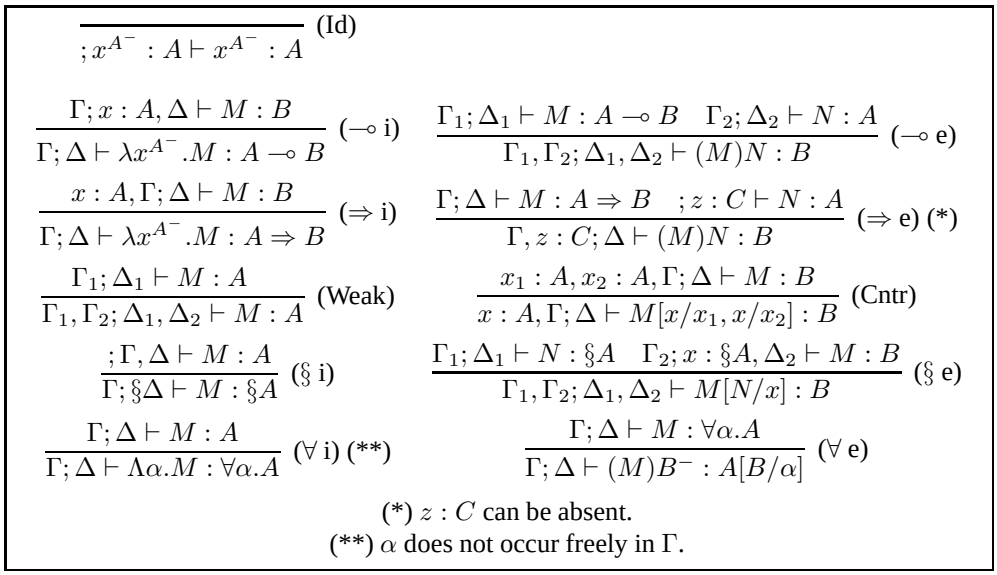

Figure 1: Typing system F terms in $DLAL$

Theorem 1

1. *Let M be a closed term of system F that has a Π_1 type A in $DLAL$. Then M can be normalized in $O(|M|^{2^d})$ steps by β-reduction, where $d = d(A)$ and $|M|$ is the structural size of M.*
2. *Every Ptime function $f : \{0,1\}^* \longrightarrow \{0,1\}^*$ can be represented by a closed term M of type $W_{DLAL} \multimap \S^d W_{DLAL}$ for some $d \geq 0$.*

Notice that the result 1 holds neither for Light linear logic nor Light affine logic. Although they are logics of polynomial time, they require some special proof syntax such as proof nets [Gir98, AR02] or light affine lambda calculus [Ter01] to guarantee polynomial time bounds.

The result 1 implies that if we ignore the embedded types occurring in M, the normal form of M can be computed in polynomial time, when the depth is fixed. It moreover shows that a term M of type $W_{DLAL} \multimap \S^d W_{DLAL}$ is Ptime, because then for any Church word $\underline{w}$ we have that $(M)\ \underline{w}$ has type $\S^d W_{DLAL}$, and can thus be evaluated in time $O(|\underline{w}|^{2^{d+1}})$.

The result 2 on the other hand guarantees that $DLAL$ has sufficient expressive power, at least enough to (extensionally) represent all polynomial time functions.

Now, let $M^{W_F \to W_F}$ be a system F typed term and suppose that we know that it has a $DLAL$ type $W_{DLAL} \multimap \S^d W_{DLAL}$ for some $d \geq 0$. Then, by the consequence of the above theorem, we know that the term M is Ptime. Thus by assigning $DLAL$ types to a given system F term, one can *statically verify* a polynomial time bound for its execution.

As a first step to elaborate this idea to use $DLAL$ for resource verification of system F terms, we address the following:

Problem 1 ($DLAL$ typing) *Given a closed term M^T of system F, determine if there is a decoration A of M such that $\vdash_{DLAL} M : A$.*

(Here the closedness assumption is only for readability.)

In the sequel, we show that there is a polynomial time algorithm for solving the $DLAL$ typing problem.

This should be contrasted with the fact that the set of system F terms representing Ptime functions is not recursively enumerable (this can be easily proved by reduction of Hilbert's 10th problem).

Hence even though $DLAL$ does not capture all Ptime terms, the general problem is undecidable and this type system gives a partial but efficiently realizable verification method.

3 Characterizing $DLAL$ typability

3.1 Pseudo-terms

To address the $DLAL$ typing problem, it is convenient to introduce an intermediary syntax which is more informative than system F terms (but not more informative than $DLAL$ derivations themselves).

First we decompose $A \Rightarrow B$ into $!A \multimap B$. The language $\mathcal{L}_{DLAL\star}$ of $DLAL\star$ types is given by:

$$\begin{array}{rcl} A & ::= & \alpha \mid D \multimap A \mid \forall\alpha.A \mid \S A \\ D & ::= & A \mid !A \end{array}$$

There is a natural map $(.)^\star$ from $\mathcal{L}_{DLAL}$ to $\mathcal{L}_{DLAL\star}$ such that $(A \Rightarrow B)^\star = !A^\star \multimap B^\star$ and commutes with the other operations. The erasure map $(.)^-$ from $\mathcal{L}_{DLAL\star}$ to $\mathcal{L}_F$ can be defined as before. A $DLAL\star$ type is called a *bang type* if it is of the form $!A$, and otherwise called a *linear type*. In the sequel, A, B, C stand for linear types, and D, E for either bang or linear types.

We assume there is a countable set of term variables $x^D, y^D, z^D, \ldots$ for each $D \in DLAL\star$. The *pseudo-terms* are defined by the following grammar:

$$t, u ::= x^D \mid \lambda x^D.t \mid (t)u \mid \Lambda\alpha.t \mid (t)A \mid \S t \mid \bar{\S} t,$$

where A is a linear type and D is an arbitrary one. The idea is that $\S$ corresponds to the main door of a $\S$-box (or a !-box) in *proof-nets* ([Gir87, AR02]) while $\bar{\S}$ corresponds to auxiliary doors. But note that there is no information in the pseudo-terms to link occurrences of $\S$ and $\bar{\S}$ corresponding to the same box, nor distinction between $\S$-boxes and !-boxes.

There is a natural erasure map from pseudo-terms to system F terms, which we will also denote by $(.)^-$, consisting in removing all occurrences of $\S$, replacing x^D with x^{D^-} and $(t)A$ with $(t)A^-$. When $t^- = M$, t is called a *decoration* of M.

For our purpose, it is sufficient to consider the class of *regular* pseudo-terms, given by:

$$t ::= x^D \mid \lambda x^D.t \mid (t)u \mid \Lambda\alpha.t \mid (t)A \mid \S^m t,$$

where m is an arbitrary value in $\mathbb{Z}$ and $\S^m t$ denotes $\underbrace{\S \cdots \S}_{m \text{ times}}\ t$ if if $m \geq 0$, and $\underbrace{\bar{\S} \cdots \bar{\S}}_{-m \text{ times}}\ t$ otherwise. In other words, a pseudo-term is regular if and only if it does not contain any subterm of the form $\S\bar{\S}u$ or $\bar{\S}\S u$.

3.2 Local typing condition

We now try to assign types to pseudo-terms in a locally compatible way. A delicate point in $DLAL$ is that it is sometimes natural to associate *two* types to one variable x. For instance, we have $x : A; \vdash_{DLAL} x : \S A$ in $DLAL$, and this can be read as $x : !A \vdash x : \S A$ in terms of $DLAL\star$ types. We thus distinguish between the *input types*, which are inherent to variables, and the *output types*, which are inductively assigned to all pseudo-terms. The condition (i) below is concerned with the output types. In the sequel, D° denotes $\S A$ if D is of the form $!A$, and otherwise denotes D itself.

A pseudo-term t satisfies the *local typing condition* if the following holds:

(i) one can inductively assign a *linear* type (called *the output type*) to each subterm of t in the following way (here the notation t_A indicates that t has the output type A):

$$(x^D)_{D^\circ} \qquad (\S t_A)_{\S A} \qquad (\bar{\S} t_{\S A})_A \qquad (\lambda x^D.t_B)_{D \multimap B}$$
$$((t_{D \multimap B})u_{D^\circ})_B \qquad (\Lambda\alpha.t_A)_{\forall\alpha.A} \qquad ((t_{\forall\alpha.A})B)_{A[B/\alpha]}$$

(ii) when a variable x occurs more than once in t, it is typed as $x^{!A}$,

(iii) t satisfies the eigenvariable condition.

We also say that t is *locally typed*.

Notice that when D is a bang type, there is a type mismatch between D and D° in the case of application. For instance, $(t_{!A \multimap B})u_{\S A}$ satisfies (i) whenever t and u do. This mismatch will be settled by the bang condition below. Observe also that the local typing rules are syntax-directed.

3.3 Boxing conditions

We now recall definitions and results from [BT05] giving some necessary conditions for a pseudo-term to be typable (in [BT05] these conditions are used for Elementary Affine Logic typing). We consider words over the language $\mathcal{L} = \{\S, \bar{\S}\}^\star$. If t is a pseudo-term and u is an occurrence of subterm in t, let $\mathsf{doors}(t, u)$ be the word

inductively defined as follows:

$$\begin{array}{lcl}
\text{if } t = u\text{:} & & \mathsf{doors}(t,u) = \epsilon, \\
\text{else:} & & \\
\mathsf{doors}(\S t, u) & = & \S :: (\mathsf{doors}(t,u)) \\
\mathsf{doors}(\bar{\S} t, u) & = & \bar{\S} :: (\mathsf{doors}(t,u)) \\
\mathsf{doors}(\lambda y^D.t_1, u) & = & \mathsf{doors}(\Lambda\alpha.t_1, u) \\
 & = & \mathsf{doors}((t_1)A, u) = \mathsf{doors}(t_1, u) \\
\mathsf{doors}((t_1)t_2, u) & = & \mathsf{doors}(t_i, u) \text{ where } t_i \text{ is the} \\
 & & \text{subterm containing } u.
\end{array}$$

That is to say, $\mathsf{doors}(t,u)$ collects the modal symbols $\S$, $\bar{\S}$ occurring on the path from the root to the node u in the term tree of t. We define a map: $s : \mathcal{L} \to \mathbb{Z}$ by:

$$s(\epsilon) = 0, \qquad s(\S :: l) = 1 + s(l), \qquad s(\bar{\S} :: l) = -1 + s(l).$$

A word $l \in \mathcal{L}$ is *weakly well-bracketed* if $\forall l' \leq l, s(l') \geq 0$, and is *well-bracketed* if this condition holds and moreover $s(l) = 0$.

Bracketing condition. Let t be a pseudo-term. We say that t satisfies the *bracketing condition* if:

(i) for any occurrence of free variable x in t, $\mathsf{doors}(t,x)$ is well-bracketed;

moreover for any occurrence of an abstraction subterm $\lambda x.v$ of t,

(ii) $\mathsf{doors}(t, \lambda x.v)$ is weakly well-bracketed, and

(iii) for any occurrence of x in v, $\mathsf{doors}(v,x)$ is well-bracketed.

This condition is sufficient to rule out the canonical morphisms for dereliction and digging, which are not valid in $DLAL$ (nor in EAL):

$$(\lambda x^{\S A}.\bar{\S} x)_{\S A \multimap A} \qquad (\lambda x^{\S A}.\S x)_{\S A \multimap \S\S A}$$

Since $\mathsf{doors}(\bar{\S} x, x) = \bar{\S}$ and $\mathsf{doors}(\S x, x) = \S$, they do not satisfy the bracketing condition (iii).

Bang condition. A subterm u is called a *bang subterm* of t if it occurs as $(t'_{!A \multimap B})u_{\S A}$ in t. We say that a locally typed pseudo-term t satisfies the *bang condition* if for any bang subterm u of t,

(i) u contains at most one free variable $x^{!C}$, having a bang type $!C$.

(ii) for any subterm v of u such that $v \neq u$ and $v \neq x$, $s(\mathsf{doors}(u,v)) \geq 1$.

This condition is sufficient to rule out the canonical morphisms for monoidalness $!A \otimes !B \multimap !(A \otimes B)$ and $\S A \multimap !A$ which are not valid in LAL (the following terms and types are slightly more complicated since $\mathcal{L}_{DLAL\star}$ does not explicitly contain a type of the form $A \multimap !B$):

$$\lambda x^{!(A \multimap B)}.\lambda y^{!B \multimap C}.\lambda z^{!A}.(y)\S((\bar{\S} x)\bar{\S} z)$$

$$\lambda x^{\S A}.\lambda y^{!A\multimap B}.(y)\S(\bar{\S}x)$$

In the first pseudo-term, the bang subterm $\S((\bar{\S}x)\bar{\S}z)$ contains more than one free variables. In the second pseudo-term, the bang subterm $\S(\bar{\S}x)$ contains a free variable typed by a linear type. Hence they both violate the bang condition (i).

Λ-Scope condition. The previous conditions, bracketing and bang, would be enough to deal with boxes in the propositional fragment of $DLAL$. For handling second-order quantification though, we need a further condition to take into account the sequentiality enforced by the quantifiers. For instance consider the following two formulas (the second one is known as *Barcan's formula*):

$$\S\forall\alpha.A \multimap \forall\alpha.\S A \quad (1)$$

$$\forall\alpha.\S A \multimap \S\forall\alpha.A \quad (2)$$

Assuming α occurs freely in A, formula (1) is provable while (2) is not. Observe that we can build the following pseudo-terms which are locally typed and have respectively type (1) and (2):

$$\begin{aligned} t_1 &= \lambda x^{\S\forall\alpha.A}.\Lambda\alpha.\S((\bar{\S}x)\alpha) \\ t_2 &= \lambda x^{\forall\alpha.\S A}.\S\Lambda\alpha.\bar{\S}((x)\alpha) \end{aligned}$$

Both pseudo-terms satisfy the previous conditions, but t_2 does not correspond to a $DLAL$ derivation.

Let u be a locally typed pseudo-term. We say that u *depends on* α if the type of u contains a free variable α. We say that a locally typed pseudo-term t satisfies the Λ-*scope condition* if: for any subterm $\Lambda\alpha.u$ of t and for any subterm v of u that depends on α, $\mathsf{doors}(u, v)$ is weakly well-bracketed.

Coming back to our example: t_1 satisfies the Λ-scope condition, but t_2 does not, because $(x)\alpha$ depends on α and nevertheless $\mathsf{doors}(\bar{\S}((x)\alpha), (x)\alpha) = \bar{\S}$ is not weakly well-bracketed.

3.4 Correctness of the conditions

Proposition 2 *If M is a system F term such that the following judgement holds in DLAL:*

$$(*) \quad x_1 : A_1, \ldots, x_m : A_m; y_1 : B_1, \ldots, y_n : B_n \vdash M : C,$$

then there is a decoration t of M with type $C^\star$ and with free variables $x_1^{!A_1^\star}, \ldots, x_m^{!A_m^\star}$, $y_1^{B_1^\star}, \ldots, y_n^{B_n^\star}$ which is regular and satisfies the local typing, bracketing, bang and Λ-scope conditions.

See the Appendix for the proof.

We want now to examine the converse property. First observe that whenever pseudo-terms $\lambda x^D.t$, $(t)u$, $\Lambda\alpha.t$, $(t)A$ satisfy the local typing, bracketing, bang and Λ-scope conditions, so do the immediate subterms t and u. The case of $\S t$ is handled by the following key lemma (already used for $EAL^\star$ in [BT05]):

Lemma 3 (Boxing) *If $\S(t_A)$ is a pseudo-term which satisfies the local typing, bracketing, bang and Λ-scope conditions, then there exist v_A, $(u_1)_{\S B_1}, \ldots, (u_n)_{\S B_n}$, unique (up to renaming of v's free variables) such that:*

1. *$FV(v) = \{x_1^{B_1}, \ldots, x_n^{B_n}\}$ and each x_i occurs exactly once in v,*
2. *$\S t = \S v[\bar{\S} u_1/x_1, \ldots, \bar{\S} u_n/x_n]$ (substitution is assumed to be capture-free),*
3. *$v, u_1, \ldots, u_n$ satisfy the same conditions.*

Proof. Similar to the proof of Lemma 5 in [BT05]. See the Appendix.

Thanks to the previous lemma, we can now prove:

Theorem 4 *Let M be a system F term. Then $x_1 : A_1, \ldots, x_m : A_m; y_1 : B_1, \ldots, y_n : B_n \vdash M : C$ is derivable in $DLAL$ if and only if there is a decoration t of M with type $C^\star$ and with free variables $x_1^{!A_1^\star}, \ldots, x_m^{!A_m^\star}$, $y_1^{B_1^\star}, \ldots, y_n^{B_n^\star}$ which is regular and satisfies the local typing, bracketing, bang and Λ-scope conditions.*

See Appendix A for the proof. As a consequence, our $DLAL$ typing problem boils down to:

Problem 2 (decoration) *Given a system F term M, determine if there exists a decoration t of M which is regular and satisfies the local typing, bracketing, bang and Λ-scope conditions.*

4 Parameterization and constraints

4.1 Parameterized terms and instantiations

To solve the decoration problem (Problem 2), one needs to explore the infinite set of decorations. This can be effectively done by introducing an abstract kind of types and terms with symbolic parameters, and expressing the conditions for such abstract terms to be materialized by boolean and integer constraints over those parameters (like in the related type inference algorithms for EAL or LAL mentioned in the introduction).

We use two sorts of parameter: *integer parameters* $\mathbf{n}, \mathbf{m}, \ldots$ meant to range over $\mathbb{Z}$, and *boolean parameters* $\mathbf{b_1}, \mathbf{b_2}, \ldots$ meant to range over $\{0, 1\}$. We also use *linear combinations of integer parameters* $\mathbf{c} = \mathbf{n_1} + \cdots + \mathbf{n_k}$, where $k \geq 0$ and each $\mathbf{n_i}$ is an integer parameter. In case $k = 0$, it is written as $\mathbf{0}$.

The set of *parameterized types* (*p-types* for short) is defined by:

$$\begin{array}{rcl} F & ::= & \alpha \mid D \multimap A \mid \forall \alpha.A \\ A & ::= & \S^{\mathbf{c}} F \\ D & ::= & \S^{\mathbf{b},\mathbf{c}} F \end{array}$$

where $\mathbf{b}$ is a boolean parameter and $\mathbf{c}$ is a linear combination of integer parameters. In the sequel, A, B, C stand for *linear p-types* of the form $\S^{\mathbf{c}} F$, and D for *bang p-types* of the form $\S^{\mathbf{b},\mathbf{c}} F$, and E for arbitrary p-types. When $D = \S^{\mathbf{b},\mathbf{c}} F$, D° denotes the linear

p-type $\S^{\mathbf{c}}F$. We assume that there is a countable set of variables $x^D, y^D, \ldots$ for each bang p-type D. The *parameterized pseudo-terms* (*p-terms* for short) are defined by the following grammar:

$$t \quad ::= \quad x^D \mid \lambda x^D.t \mid (t)u \mid \Lambda\alpha.t \mid (t)A \mid \S^{\mathbf{m}}t.$$

We denote by $par^{bool}(t)$ the set of boolean parameters of t, and by $par^{int}(t)$ the set of integer parameters of t.

An *instantiation* $\phi = (\phi^b, \phi^i)$ for a p-term t is given by two maps $\phi^b : par^{bool}(t) \to \{0,1\}$ and $\phi^i : par^{int}(t) \to \mathbb{Z}$. The map ϕ^i can be naturally extended to linear combinations $\mathbf{c} = \mathbf{n_1} + \cdots + \mathbf{n_k}$ by $\phi^i(\mathbf{c}) = \phi^i(\mathbf{n_1}) + \cdots + \phi^i(\mathbf{n_k})$. An instantiation ϕ is said to be *admissible* for a p-type E if for any linear combination $\mathbf{c}$ occurring in E, we have $\phi^i(\mathbf{c}) \geq 0$, and moreover whenever $\S^{\mathbf{b},\mathbf{c}}F$ occurs in E, $\phi^b(\mathbf{b}) = 1$ implies $\phi^i(\mathbf{c}) \geq 1$. When ϕ is admissible for E, a type $\phi(E)$ of $DLAL\star$ is obtained by replacing each $\S^{\mathbf{c}}F$ and $\S^{\mathbf{b},\mathbf{c}}F$ with $\phi^b(\mathbf{b}) = 0$ by $\S^{\phi^i(\mathbf{c})}\phi(F)$, and $\S^{\mathbf{b},\mathbf{c}}F$ with $\phi^b(\mathbf{b}) = 1$ by $!\S^{\phi^i(\mathbf{c})-1}\phi(F)$.

So informally speaking, in $\S^{\mathbf{b},\mathbf{c}}F$ the $\mathbf{c}$ stands for the number of modalities ahead of the type, while the boolean $\mathbf{b}$ serves to determine whether the first modality, if any, is $\S$ or $!$.

An instantiation ϕ for a p-term t is said to be *admissible* for t if it is admissible for all p-types occurring in t. When ϕ is admissible for t, a regular pseudo-term $\phi(t)$ can be obtained by replacing each $\S^{\mathbf{m}}u$ with $\S^{\phi^i(\mathbf{m})}u$, each x^D with $x^{\phi(D)}$, and each $(t)A$ with $(t)\phi(A)$.

As for pseudo-terms there is an erasure map $(.)^-$ from p-terms to system F terms consisting in forgetting modalities and parameters.

A *linear free decoration* (*bang free decoration*, resp.) of a system F type T is a linear p-type (bang p-type, resp.) E such that (i) $E^- = T$, (ii) each linear combination $\mathbf{c}$ occurring in E consists of a single integer parameter $\mathbf{m}$, and (iii) the parameters occurring in E are mutually distinct. Two free decorations $\overline{T}_1$ and $\overline{T}_2$ are said to be *distinct* if the set of parameters occurring in $\overline{T}_1$ is disjoint from the set of parameters in $\overline{T}_2$.

The *free decoration* $\overline{M}$ of a system F term M (which is unique up to renaming of parameters) is obtained as follows: first, to each type T of a variable x^T used in M, we associate a bang free decoration $\overline{T}$, and to each type U occurring as $(N)U$ in T, we associate a linear free decoration $\overline{U}$ with the following proviso:

(i) one and the same $\overline{T}$ is associated to all occurrences of the same variable x^T;

(ii) otherwise mutually distinct free decorations $\overline{T}_1, \ldots, \overline{T}_n$ are associated to different occurrences of T.

$\overline{M}$ is now defined by induction on the construction of M:

$$\begin{array}{rclcrcl}
\overline{x^T} & = & \S^{\mathbf{m}}x^{\overline{T}} & & & & \\
\overline{\lambda x^T.M} & = & \S^{\mathbf{m}}\lambda x^{\overline{T}}.\overline{M} & & \overline{(M)N} & = & \S^{\mathbf{m}}((\overline{M})\overline{N}) \\
\overline{\Lambda\alpha.M} & = & \S^{\mathbf{m}}\Lambda\alpha.\overline{M} & & \overline{(M)T} & = & \S^{\mathbf{m}}((\overline{M})\overline{T})
\end{array}$$

where all newly introduced parameters $\mathbf{m}$ are chosen to be fresh. The key property of free decorations is the following:

Lemma 5 *Let M be a system F term and t be a regular pseudo-term. Then t is a decoration of M if and only if there is an admissible instantiation ϕ for $\overline{M}$ such that $\phi(\overline{M}) = t$.*

Hence our decoration problem boils down to:

Problem 3 (instantiation) *Given a system F term M, determine if there exists an admissible instantiation ϕ for $\overline{M}$ such that $\phi(\overline{M})$ satisfies the local typing, bracketing, bang and Λ-scope conditions.*

For that we will need to be able to state the conditions of Theorem 4 on p-terms; they will yield some constraints on parameters. We will speak of *linear inequations*, meaning in fact both linear equations and linear inequations.

4.2 Unification constraints

To express the unifiability of two p-types E_1 and E_2, we define a set $\mathcal{U}(E_1, E_2)$ of constraints by

$$\begin{array}{rcl}
\mathcal{U}(\alpha, \alpha) & = & \emptyset, \\
\mathcal{U}(D_1 \multimap A_1, D_2 \multimap A_2) & = & \mathcal{U}(D_1, D_2) \cup \mathcal{U}(A_1, A_2), \\
\mathcal{U}(\forall\alpha.A_1, \forall\alpha.A_2) & = & \mathcal{U}(A_1, A_2), \\
\mathcal{U}(\S^{\mathbf{c_1}} F_1, \S^{\mathbf{c_2}} F_2) & = & \{\mathbf{c_1} = \mathbf{c_2}\} \cup \mathcal{U}(F_1, F_2), \\
\mathcal{U}(\S^{\mathbf{b_1},\mathbf{c_1}} F_1, \S^{\mathbf{b_2},\mathbf{c_2}} F_2) & = & \{\mathbf{b_1} = \mathbf{b_2}, \mathbf{c_1} = \mathbf{c_2}\} \cup \mathcal{U}(F_1, F_2),
\end{array}$$

and undefined otherwise. It is straightforward to observe:

Lemma 6 *Let E_1, E_2 be two p-types such that $\mathcal{U}(E_1, E_2)$ is defined, and ϕ be an admissible instantiation for E_1 and E_2. Then $\phi(E_1) = \phi(E_2)$ if and only if ϕ is a solution of $\mathcal{U}(E_1, E_2)$.*

4.3 Local typing constraints

For any p-type E, $\mathcal{M}(E)$ denotes the set $\{\mathbf{c} \geq \mathbf{0} : \mathbf{c} \text{ occurs in } E\} \cup \{\mathbf{b} = \mathbf{1} \Rightarrow \mathbf{c} \geq \mathbf{1} : \S^{\mathbf{b},\mathbf{c}} F \text{ occurs in } E\}$. Then ϕ is admissible for E if and only if ϕ is a solution of $\mathcal{M}(E)$.

When A is a linear p-type $\S^{\mathbf{c}} F$, $B[A/\alpha]$ denotes a p-type obtained by replacing each $\S^{\mathbf{c}'}\alpha$ in B with $\S^{\mathbf{c}'+\mathbf{c}} F$ and each $\S^{\mathbf{b},\mathbf{c}'}\alpha$ with $\S^{\mathbf{b},\mathbf{c}'+\mathbf{c}} F$.

Now consider the free decoration $\overline{M}$ of a system F typed term M. We assign to each subterm t of $\overline{M}$ a *linear* p-type B (indicated as t_B) and a set $\mathcal{M}(t)$ of constraints as on Figure 2. Notice that any linear p-type is of the form $\S^{\mathbf{c}} F$. Moreover, since t comes from a system F typed term, we know that F is an implication when t occurs as $(t_{\S^{\mathbf{c}} F})u$, and F is a quantification when t occurs as $(t_{\S^{\mathbf{c}} F})A$. The unification $\mathcal{U}(D^\circ, A)$ used in $\mathcal{M}((t)u)$ is always defined, and finally, $\overline{M}$ satisfies the eigenvariable condition.

Let $\mathsf{Ltype}(\overline{M})$ be the set $\mathcal{M}(\overline{M}) \cup \{\mathbf{b} = \mathbf{1} : x^{\S^{\mathbf{b},\mathbf{c}} F} \text{ occurs more than once in } \overline{M}\}$.

$$
\begin{array}{lrcl}
(x^D)_{D^\circ} & \mathcal{M}(x) & = & \mathcal{M}(D) \\
(\S^{\mathbf{m}} t_{\S^{\mathbf{c}} F})_{\S^{\mathbf{m}+\mathbf{c}} F} & \mathcal{M}(\S^{\mathbf{m}} t) & = & \{\mathbf{m}+\mathbf{c} \geq 0\} \cup \mathcal{M}(t) \\
(\lambda x^D.t_A)_{\S^{\mathbf{0}}(D \multimap A)} & \mathcal{M}(\lambda x^D.t) & = & \mathcal{M}(D) \cup \mathcal{M}(t) \\
((t_{\S^{\mathbf{c}}(D \multimap B)})u_A)_B & \mathcal{M}((t)u) & = & \{\mathbf{c}=\mathbf{0}\} \cup \mathcal{U}(D^\circ, A) \cup \mathcal{M}(t) \cup \mathcal{M}(u) \\
(\Lambda\alpha.t_A)_{\S^{\mathbf{0}}\forall\alpha.A} & \mathcal{M}(\Lambda\alpha.t) & = & \mathcal{M}(t) \\
((t_{\S^{\mathbf{c}}\forall\alpha.B})A)_{B[A/\alpha]} & \mathcal{M}((t)A) & = & \{\mathbf{c}=\mathbf{0}\} \cup \mathcal{M}(A) \cup \mathcal{M}(t)
\end{array}
$$

Figure 2: $\mathcal{M}(t)$ constraints.

4.4 Boxing constraints

In this section we need to recall some definitions from [BT05]. We consider the words over integer parameters $\mathbf{m}$, $\mathbf{n}$ ..., whose set we denote by $\mathcal{L}_p$.

Let t be a p-term and u an occurrence of subterm of t. We define, as for pseudo-terms, the word $\mathsf{doors}(t,u)$ in $\mathcal{L}_p$ as follows:

$$
\begin{array}{lcl}
\text{if } t = u\text{:} & & \mathsf{doors}(t,u) = \epsilon, \\
\text{else:} & & \\
\mathsf{doors}(\S^{\mathbf{m}} t, u) & = & \mathbf{m} :: (\mathsf{doors}(t,u)) \\
\mathsf{doors}(\lambda y^D.t_1, u) & = & \mathsf{doors}(\Lambda\alpha.t_1, u) \\
 & = & \mathsf{doors}((t_1)A, u) = \mathsf{doors}(t_1, u) \\
\mathsf{doors}((t_1)t_2, u) & = & \mathsf{doors}(t_i, u) \text{ when } t_i \text{ is the} \\
 & & \text{subterm containing } u.
\end{array}
$$

The sum $s(l)$ of an element l of $\mathcal{L}_p$ is a linear combination of integer parameters defined by:

$$s(\epsilon) = \mathbf{0}, \quad s(\mathbf{m} :: l) = \mathbf{m} + s(l).$$

For each list $l \in \mathcal{L}_p$, define $\mathsf{wbracket}(l) = \{s(l') \geq \mathbf{0} \mid l' \leq l\}$ and $\mathsf{bracket}(l) = \mathsf{wbracket}(l) \cup \{s(l) = \mathbf{0}\}$.

Given a system F term M, we define the following sets of constraints:
Bracketing constraints. $\mathsf{Bracket}(\overline{M})$ is the union of the following sets:

(i) $\mathsf{bracket}(\mathsf{doors}(\overline{M}, x))$ for each free variable x in $\overline{M}$,

and for each occurrence of an abstraction subterm $\lambda x.v$ of $\overline{M}$,

(ii) $\mathsf{wbracket}(\mathsf{doors}(\overline{M}, \lambda x.v))$,

(iii) $\mathsf{bracket}(\mathsf{doors}(v, x))$ for each occurrence of x in v.

Bang constraints. A subterm u_A that occurs as $(t_{\S^{\mathbf{c}'}(\S^{\mathbf{b},\mathbf{c}} F \multimap B)})u_A$ in $\overline{M}$ is called a *bang subterm* of $\overline{M}$ with the *critical parameter* $\mathbf{b}$. Now $\mathsf{Bang}(\overline{M})$ is the union of the following sets: for each bang subterm u of $\overline{M}$ with a critical parameter $\mathbf{b}$,

(i) $\{\mathbf{b} = \mathbf{0}\}$ if u contains strictly more than one occurrence of free variable, and $\{\mathbf{b} = \mathbf{1} \Rightarrow \mathbf{b}' = \mathbf{1}\}$ if u contains exactly one occurrence of free variable $x^{\S^{\mathbf{b}',\mathbf{c}'} F'}$.

(ii) $\{\mathbf{b} = 1 \Rightarrow s(\mathsf{doors}(u, v)) \geq \mathbf{1} \ : \ v$ is a subterm of u such that $v \neq u$ and $v \neq x\}$.

Λ-**Scope constraints**. $\mathsf{Scope}(\overline{M})$ is the union of the following sets:

- $\mathsf{wbracket}(\mathsf{doors}(u, v))$ for each subterm $\Lambda\alpha.u$ of $\overline{M}$ and for each subterm v of u that depends on α.

We denote $\mathsf{Const}(\overline{M}) = \mathsf{Ltype}(\overline{M}) \cup \mathsf{Bracket}(\overline{M}) \cup \mathsf{Bang}(\overline{M}) \cup \mathsf{Scope}(\overline{M})$. We then have:

Theorem 7 *Let M be a system F term and ϕ be an instantiation for $\overline{M}$. Then: ϕ is admissible for M and $\phi(\overline{M})$ satisfies the local typing, bracketing, bang and Λ-scope conditions if and only if ϕ is a solution of $\mathsf{Const}(\overline{M})$.*

Moreover, the number of (in)equations in $\mathsf{Const}(\overline{M})$ is quadratic in the size of M.

5 Solving the constraints

From a proof-net point of view, naively one might expect that finding a DLAL decoration could be decomposed into first finding a suitable EAL decoration (that is to say a box structure) and then determining which boxes should be ! ones. This however cannot be turned into a valid algorithm because there can be an infinite number of EAL decorations in the first place.

Our method will thus proceed in the opposite way: first solve the boolean constraints, which corresponds to determine which !-boxes are necessary, and then complete the decoration by finding a suitable box structure.

5.1 Solving boolean constraints

We divide $\mathsf{Const}(\overline{M})$ into three disjoint sets $\mathsf{Const}^b(\overline{M})$, $\mathsf{Const}^i(\overline{M})$ and $\mathsf{Const}^m(\overline{M})$:

- A *boolean constraint* $\mathbf{s} \in \mathsf{Const}^b(\overline{M})$ consists of only boolean parameters. $\mathbf{s}$ is of one of the following forms:
  $\mathbf{b_1} = \mathbf{b_2}$ (in $\mathsf{Ltype}(\overline{M})$)
  $\mathbf{b} = \mathbf{1}$ (in $\mathsf{Ltype}(\overline{M})$)
  $\mathbf{b} = \mathbf{0}$ (in $\mathsf{Bang}(\overline{M})$)
  $\mathbf{b} = \mathbf{1} \Rightarrow \mathbf{b'} = \mathbf{1}$ (in $\mathsf{Bang}(\overline{M})$)

- A *linear constraint* $\mathbf{s} \in \mathsf{Const}^i(\overline{M})$ deals with integer parameters only. A linear constraint $\mathbf{s}$ is of one of the following forms:
  $\mathbf{c_1} = \mathbf{c_2}$ (in $\mathsf{Ltype}(\overline{M})$)
  $\mathbf{c} \geq \mathbf{0}$ (in $\mathsf{Ltype}(\overline{M})$, $\mathsf{Bracket}(\overline{M})$, $\mathsf{Scope}(\overline{M})$)
  $\mathbf{c} = \mathbf{0}$ (in $\mathsf{Ltype}(\overline{M})$ and $\mathsf{Bracket}(\overline{M})$)

- A *mixed constraint* $\mathbf{s} \in \mathsf{Const}^m(\overline{M})$ contains a boolean parameter and a linear combination and is of the following form:
  $\mathbf{b} = \mathbf{1} \Rightarrow \mathbf{c} \geq \mathbf{1}$ (in $\mathsf{Ltype}(\overline{M})$ and $\mathsf{Bang}(\overline{M})$)

We consider the set of instantiations on boolean parameters and the extensional order $\leq$ on these maps: $\psi^b \leq \phi^b$ if for any $\mathbf{b}$, $\psi^b(\mathbf{b}) \leq \phi^b(\mathbf{b})$.

Lemma 8 *$\mathsf{Const}^b(\overline{M})$ has a solution if and only if it has a minimal solution ψ^b. The latter can be computed in time polynomial in the number of boolean constraints in $\mathsf{Const}^b(\overline{M})$.*

Proof. Assuming that $\mathsf{Const}^b(\overline{M})$ has a solution, we can compute the minimal one by a standard resolution procedure. See Appendix A. ■

5.2 Solving integer constraints

When ϕ^b is a boolean instantiation, $\phi^b\mathsf{Const}^m(\overline{M})$ denotes the set of linear constraints defined as follows: for any constraint of the form $\mathbf{b} = \mathbf{1} \Rightarrow \mathbf{c} \geq \mathbf{1}$ in $\mathsf{Const}^m(\overline{M})$, $\mathbf{c} \geq \mathbf{1}$ belongs to $\phi^b\mathsf{Const}^m(\overline{M})$ if and only if $\phi^b(\mathbf{b}) = 1$. It is then clear that (*) (ϕ^b, ϕ^i) is a solution of $\mathsf{Const}(\overline{M})$ if and only if ϕ^b is a solution of $\mathsf{Const}^b(\overline{M})$ and ϕ^i is a solution of $\phi^b\mathsf{Const}^m(\overline{M}) \cup \mathsf{Const}^i(\overline{M})$.

Proposition 9 *$\mathsf{Const}(\overline{M})$ admits a solution if and only if it has a solution $\psi = (\psi^b, \psi^i)$ such that ψ^b is the minimal solution of $\mathsf{Const}^b(\overline{M})$.*

Proof. Suppose that $\mathsf{Const}(\overline{M})$ admits a solution (ϕ^b, ϕ^i). Then by the previous lemma, there is a minimal solution ψ^b of $\mathsf{Const}^b(\overline{M})$. Since $\psi^b \leq \phi^b$, we have $\psi^b\mathsf{Const}^m(\overline{M}) \subseteq \phi^b\mathsf{Const}^m(\overline{M})$. Since ϕ^i is a solution of $\phi^b\mathsf{Const}^m(\overline{M}) \cup \mathsf{Const}^i(\overline{M})$ by (*) above, it is also a solution of $\psi^b\mathsf{Const}^m(\overline{M}) \cup \mathsf{Const}^i(\overline{M})$. This means that (ψ^b, ϕ^i) is a solution of $\mathsf{Const}(\overline{M})$. ■

Coming back to the proof-net intuition, Proposition 9 means that given a syntactic tree of term there is a most general (minimal) way to place ! boxes (and accordingly ! subtypes in types), that is to say: if there is a DLAL decoration for this tree then there is one with precisely this minimal distribution of ! boxes.

Now notice that $\psi^b\mathsf{Const}^m(\overline{M}) \cup \mathsf{Const}^i(\overline{M})$ is a linear inequation system, for which a polynomial time procedure for searching a rational solution is known.

Lemma 10 *$\psi^b\mathsf{Const}^m(\overline{M}) \cup \mathsf{Const}^i(\overline{M})$ has a solution in $\mathbb{Q}$ if and only if it has a solution in $\mathbb{Z}$.*

Proof. Clearly the set of solutions is closed under multiplication by a positive integer. ■

Theorem 11 *Let M be a System F term. Then one can decide in time polynomial in the number of constraints in $\mathsf{Const}(\overline{M})$ whether $\mathsf{Const}(\overline{M})$ admits a solution.*

Proof. First apply the procedure described in the proof of Lemma 8 to decide if there is a minimal solution ψ^b of $\mathsf{Const}^b(\overline{M})$. If it exists, apply the polynomial time procedure to decide if $\psi^b\mathsf{Const}^m(\overline{M}) \cup \mathsf{Const}^i(\overline{M})$ admits a solution in $\mathbb{Q}$. If it does, then we also have an integer solution. Otherwise, $\mathsf{Const}(\overline{M})$ is not solvable. ■

By combining Theorem 4, Lemma 5, Theorems 7 and 11, we obtain our main theorem:

Theorem 12 *Given a system F term M^T, it is decidable in time polynomial in the size of M whether there is a decoration A of T such that $\vdash_{DLAL} M : A$.*

6 Implementation

6.1 Overview

We designed an implementation of the type inference algorithm. The program is written in functional Caml and is quite concise (less than 1500 lines). A running program not only shows the actual feasibility of our method, but also is a great facility for building examples, and thus might allow for a finer study of the algorithm.

Data types as well as functions closely follow the previous description of the algorithm: writing the program in such a way tends to minimise the number of bugs, and speaks up for the robustness of the whole proof development.

The program consists of several successive parts:

1. Parsing phase: turns the input text into a concrete syntax tree. The input is an F typing judgement, in a syntax *à la* Church with type annotations at the binders. It is changed into the de Bruijn notation, and parameterized with fresh parameters. Finally, the abstract tree is decorated with parameterized types at each node.

2. Constraints generation: performs explorations on the tree and generates the boolean, linear and mixed constraints.

3. Boolean constraints resolution: gives the minimal solution of the boolean constraints, or answers negatively if the set admits no solution.

4. Constraints printing: builds the final set of linear constraints.

We use the simplex algorithm to solve the linear constraints. It runs in $O(2^n)$, which comes in contrast with the previous result of polynomial time solving, but has proven to be the best in practice (with a careful choice of the objective function).

6.2 An example of execution

As an example, let us consider the reversing function **rev** on binary words, applied to **1010**. **rev** can be defined by a single higher-order iteration, and thus represented by the following system F term:

$$\begin{array}{l}\lambda l^{W}.\Lambda\beta.\lambda so^{\beta\to\beta}.\lambda si^{\beta\to\beta}.(l\ (\beta\to\beta))\\ \quad \lambda a^{\beta\to\beta}.\lambda x^{\beta}.(a)(so)x\\ \quad \lambda a^{\beta\to\beta}.\lambda x^{\beta}.(a)(si)x\ (\Lambda\alpha.\lambda z^{\alpha}.z)\beta\end{array}$$

We apply it to :

$$\Lambda\alpha.\lambda so^{\alpha\to\alpha}.\lambda si^{\alpha\to\alpha}.\lambda x^{\alpha}.(si)(so)(si)(so)x,$$

representing the word **1010**. Since **rev** involves higher-order functionals and polymorphism, it is not so straightforward to tell, just by looking at the term structure, whether it works in polynomial time or not.

Given **rev(1010)** as input (coded by ASCII characters), our program produces 177 (in)equations on 79 variables. After constraint solving, we obtain the result, that can be read as:

$$
\begin{array}{l}
(\lambda l^{W}.\Lambda\beta.\lambda so^{!(\beta\multimap\beta)}.\lambda si^{!(\beta\multimap\beta)}.\\
\quad \bar{\S}(\bar{\S}((l\ (\beta\multimap\beta))\\
\quad \bar{\S}\lambda a^{\beta\multimap\beta}.\lambda x^{\beta}.(a)(\bar{\S}so)x\\
\quad \bar{\S}\lambda a^{\beta\multimap\beta}.\lambda x^{\beta}.(a)(\bar{\S}si)x)\\
\quad (\Lambda\alpha.\lambda z^{\alpha}.z)\beta)\\
\Lambda\alpha.\lambda so^{!\alpha\to\alpha}.\lambda si^{\alpha\to\alpha}.\bar{\S}\lambda x^{\alpha}.(\bar{\S}si)(\bar{\S}so)(\bar{\S}si)(\bar{\S}so)x
\end{array}
$$

It corresponds to the natural depth-1 typing of this term, with conclusion type $W_{DLAL}\multimap W_{DLAL}$. The solution ensures polynomial time termination, and in fact its depth guarantees normalization in a quadratic number of β-reduction steps.

Further examples, as well as the program itself, will be available at

`http://www-lipn.univ-paris13.fr/~atassi/`

References

[Ama05] R. Amadio. Synthesis of max-plus quasi-interpretations. *Fundamenta Informaticae*, 65:29–60, 2005.

[AR02] A. Asperti and L. Roversi. Intuitionistic light affine logic. *ACM Transactions on Computational Logic*, 3(1):1–39, 2002.

[Bai02] P. Baillot. Checking polynomial time complexity with types. In *Proceedings of IFIP TCS'02*, Montreal, 2002. Kluwer Academic Press.

[Bai04] P. Baillot. Type inference for light affine logic via constraints on words. *Theoretical Computer Science*, 328(3):289–323, 2004.

[BC92] S. Bellantoni and S. Cook. New recursion-theoretic characterization of the polytime functions. *Computational Complexity*, 2:97–110, 1992.

[BT04] P. Baillot and K. Terui. Light types for polynomial time computation in lambda-calculus. In *Proceedings of LICS'04*. IEEE Computer Press, 2004.

[BT05] P. Baillot and K. Terui. A feasible algorithm for typing in elementary affine logic. tlca 2005: 55-70. In *Proceedings of TLCA'05*, volume 3461 of *LNCS*, pages 55–70. Springer, 2005.

[CDLRdR05] P. Coppola, U. Dal Lago, and S. Ronchi della Rocca. Elementary affine logic and the call-by-value lambda calculus. In *Proceedings of TLCA'05*, volume 3461 of *LNCS*, pages 131–145. Springer, 2005.

[CM01] P. Coppola and S. Martini. Typing lambda-terms in elementary logic with linear constraints. In *Proceedings TLCA'01*, volume 2044 of *LNCS*, 2001.

[CRdR03] P. Coppola and S. Ronchi della Rocca. Principal typing in Elementary Affine Logic. In *Proceedings TLCA'03*, LNCS, 2003.

[Gir87] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[Gir98] J.-Y. Girard. Light linear logic. *Information and Computation*, 143:175–204, 1998.

[HJ03] M. Hofmann and S. Jost. Static prediction of heap space usage for first-order functional programs. In *Proc. ACM POPL'03*, 2003.

[Hof03] M. Hofmann. Linear types and non-size-increasing polynomial time computation. *Information and Computation*, 183(1):57–85, 2003.

[LM93] D. Leivant and J.-Y. Marion. Lambda-calculus characterisations of polytime. *Fundamenta Informaticae*, 19:167–184, 1993.

[MM00] J.-Y. Marion and J.-Y. Moyen. Efficient first order functional program interpreter with time bound certifications. In *Proceedings of LPAR 2000*, volume 1955 of *LNCS*, pages 25–42. Springer, 2000.

[Ter01] K. Terui. Light Affine Lambda-calculus and polytime strong normalization. In *Proceedings LICS'01*. IEEE Computer Society, 2001. Full version available at http://research.nii.ac.jp/∼ terui.

[Ter04] K. Terui. Light affine set theory: a naive set theory of polynomial time. *Studia Logica*, 77:9–40, 2004.

APPENDIX

A Proofs

Proof of Proposition 2.
Proof. First, one can build a (possibly non-regular) decoration M^+ of M satisfying the four conditions by induction on the derivation. Depending on the last typing rule used (see Figure 1), M^+ takes one of the following forms:

(Id)	$x^{A^\star}$	(§ e)	$M^+[N^+/x]$
($\multimap$ i)	$\lambda x^{A^\star}.M^+$	($\multimap$ e)	$(M^+)N^+$
($\Rightarrow$ i)	$\lambda x^{!A^\star}.M^+$	($\Rightarrow$ e)	$(M^+)\S N^+[\bar{\S} z^{!C^\star}/z]$
(Weak)	M^+	(Cntr)	$M^+[x/x_1, x/x_2]$
($\forall$ i)	$\Lambda\alpha.M^+$	($\forall$ e)	$(M^+)B^\star$
(§ i)	$\S M^+[\bar{\S} x_i^{!A_i^\star}/x_i, \bar{\S} y_j^{\S B_j^\star}/y_j]$		

where M^+ in (§ i) has free variables $x_1^{A_1}, \ldots, x_m^{A_m}, y_1^{B_1}, \ldots, y_n^{B_n}$.

It is easily verified that M^+ has a suitable type and satisfies the four conditions; let us just remark:

- The bang condition for $(M^+)\S N^+[\bar{\S} z^{!C^\star}/z]$ in ($\Rightarrow$ e) follows by the bracketing condition for N^+, which holds by the induction hypothesis, while the Λ-scope condition follows by the eigenvariable condition for N^+. Similarly for the case of (§ i).
- $M^+[N^+/x]$ in (§ e) satisfies the Λ-scope condition since substitution is capture-free, and satisfies the bang condition since x has a linear type and thus cannot appear inside a bang subterm of M^+.

Finally, the required regular pseudo-term t is obtained from M^+ by applying inside t the following rewrite rules as many times as possible:

$$\bar{\S}\S u \longrightarrow u, \qquad \S\bar{\S} u \longrightarrow u.$$

It is clear that the four conditions are preserved by these reductions. ■

Proof of Lemma 3.
Proof. Given $\S t$, assign an index to each occurrence of § and $\bar{\S}$ in $\S t$ to distinguish occurrences (we assume that the outermost § have index 0). One can then find closing brackets $\bar{\S}_1, \ldots, \bar{\S}_n$ that match the opening bracket $\S_0$ in $\S_0 t$. Replace each $(\bar{\S}_i u_i)_{B_i}$ with a fresh and distinct free variable $x_i^{B_i}$ ($1 \leq j \leq n$), and let $\S v$ be the resulting pseudo-term. This way one can obtain $v, u_1, \ldots, u_n$, such that condition 2 holds.

As to point 1., we claim that v does not contain a free variable other than $x_1, \ldots, x_n$. If there is any, say y, then it is also a free variable of t, thus the bracketing condition for $\S_0 t$ implies that $\mathsf{doors}(\S_0 t, y)$ is well-bracketed, and thus there is a closing bracket that matches $\S_0$ in the path from $\S_0 t$ to y. That means that y belongs to one of $u_1, \ldots, u_n$, not to v. Hence condition 1 holds.

We now need to check point 3. The bracketing condition for v, $u_1, \ldots, u_n$ can be shown as in [BT05]. The Λ-scope condition is easy to verify.

As to the local typing condition, the only nontrivial point is that v satisfies the eigenvariable condition. Suppose that the type B_i of x_i contains a bound variable α of v. Then $\S_0 t$ contains a subterm of the form $\Lambda\alpha.v'[\bar{\S}_i u_i/x_i]$ and u_i depends on α. However, $\mathsf{doors}(v'', u_i)$ with $v'' = v'[\bar{\S}_i u_i/x_i]$ cannot be weakly well-bracketed because $\bar{\S}_i$ should match the outermost opening bracket $\S_0$. This contradicts the Λ-scope condition for $\S_0 t$.

To show the bang condition for v (it is clear for $u_1, \ldots, u_n$), suppose that v contains a bang subterm v'. We claim that v' does not contain variables $x_1, \ldots, x_n$. If it contains any, say x_i, then $\S_0 t$ contains $v'[\bar{\S}_i u_i/x_i]$ and the bang condition for $\S_0 t$ implies that $s(\mathsf{doors}(v'', \bar{\S} u_i)) \geq 1$ with $v'' = v'[\bar{\S}_i u_i/x_i]$. On the other hand, we clearly have $s(\mathsf{doors}(\S_0 t, v'')) \geq 1$ because v'' contains the closing bracket $\bar{\S}_i$ that matches $\S_0$. As a consequence, we have $s(\mathsf{doors}(\S_0 t, \bar{\S}_i u_i)) \geq 2$. This means that $\bar{\S}_i$ does not match $\S_0$, a contradiction. As a consequence, v' does not contain $x_1, \ldots, x_n$. So v' occurs in $\S_0 t$, and therefore satisfies the bang condition. ■

Proof of Theorem 4. The 'only if' direction has already been given by Proposition 2. The other direction is proved by induction on the size of pseudo-term t.

When t is a variable $(x^D)_{D^\circ}$, the claim can be established by (Id) and (§ i). Note that t cannot be of the form $\bar{\S} u$ due to the bracketing condition.

When t is one of $\lambda x^D.u$, $(u)v$ (with v not a bang subterm), $\Lambda\alpha.u$, $(u)A$, the subterms u and v also satisfy all conditions. Hence we can use the induction hypothesis to show that t^- is typable in $DLAL$. When t is $\S u$, apply Lemma 3 and argue similarly by using rules (§ e).

When t is $(u_{!A \multimap B})v_{\S A}$, i.e., with v a bang subterm, we have $\Gamma; \Delta \vdash u^- : A \Rightarrow B$ with suitable Γ and Δ by the induction hypothesis.

If v is a variable, then it must be of the form $x^{!A}$ by the bang condition (i). Hence by applying ($\Rightarrow$ e) to $\Gamma; \Delta \vdash u^- : A \Rightarrow B$ and $; x : A \vdash x : A$, we obtain $\Gamma, x : A; \Delta \vdash (u^-)x : B$ as required.

If v is not a variable, then it must be of the form $\S v_0$ due to the bang condition (ii) and contain at most one free variable. Let us suppose that it contains $y^{!C}$. Now, the bracketing condition implies $s(\mathsf{doors}(\S v_0, y)) = 0$ while the bang condition implies $s(\mathsf{doors}(\S v_0, v')) \geq 1$ for any subterm v' of v_0 other than y. Therefore, combined with Lemma 3, it follows that v is actually of the form $\S v_1[\bar{\S} y/x]$, where v_1 contains a variable x^C and satisfies all the conditions. By induction hypothesis, we have $; x : C \vdash v_1^- : A$, and hence $; y : C \vdash v_1^-[y/x] : A$ by renaming. Therefore, we obtain $\Gamma, y : C; \Delta \vdash (u^-)v^- : B$ by ($\Rightarrow$ e). ■

Proof of Lemma 8. Let $\mathcal{B} := \mathsf{Const}^b(\overline{M})$. Apply repeatedly the following steps until reaching a fixpoint:

- if $\mathbf{b_1} = \mathbf{b_2} \in \mathcal{B}$ and $\mathbf{b_1} = \mathbf{0} \in \mathcal{B}$ (resp. $\mathbf{b_1} = \mathbf{1} \in \mathcal{B}$), then let $\mathcal{B} := \mathcal{B} \cup \{\mathbf{b_2} = \mathbf{0}\}$ (resp. $\mathcal{B} := \mathcal{B} \cup \{\mathbf{b_2} = \mathbf{1}\}$);
- if $(\mathbf{b} = \mathbf{1} \Rightarrow \mathbf{b}' = \mathbf{1}) \in \mathcal{B}$ and $\mathbf{b} = \mathbf{1} \in \mathcal{B}$, then let $\mathcal{B} := \mathcal{B} \cup \{\mathbf{b}' = \mathbf{1}\}$.

It is obvious that this can be done in a polynomial number of steps and that the resulting system $\mathcal{B}$ is equivalent to $\mathsf{Const}^b(\overline{M})$.

Now, if $\mathcal{B}$ contains a pair of equations: $\mathbf{b} = 0, \mathbf{b} = 1$, then it is inconsistent. Otherwise define the boolean instantiation ψ^b such that $\psi^b(\mathbf{b}) := 1$ if $\mathbf{b} = 1 \in \mathcal{B}$ and $\psi^b(\mathbf{b}) := 0$ otherwise:

It is clear that ψ^b is a solution of $\mathcal{B}$. In particular, observe that any constraint of the form $(\mathbf{b} = 1 \Rightarrow \mathbf{b}' = 1)$ in $\mathcal{B}$ is satisfied by ψ^b. Moreover any solution ϕ^b of $\mathcal{B}$ satisfies: $\psi^b \leq \phi^b$. Therefore if $\mathsf{Const}^b(\overline{M})$ has a solution then it has a minimal one. ■