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

Vincent Atassi*, Patrick Baillot*, and Kazushige Terui**

[1] LIPN, Univ. Paris 13 / CNRS , France, atassi@lipn.univ-paris13.fr
[2] LIPN, Univ. Paris 13 / CNRS, France, pb@lipn.univ-paris13.fr
[3] National Institute of Informatics, Japan, terui@nii.ac.jp

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 criterion. This way the user could stick to a simple programming language and the compiler would check whether the program satisfies the criterion, 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

* Partially supported by projects CRISS (ACI), GEOCAL (ACI), NO-CoST (ANR).
** Partially supported by Grant-in-Aid for Scientific Research, MEXT, Japan.

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 ([Ter04a]), 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 ([Wel99]) 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 (*e.g.* [Ama05], [HJ03]). Elementary linear logic (EAL, [Gir98,DJ03]) in particular is another variant of Linear logic which characterizes Kalmar elementary time and has applications to optimal reduction. Type inference for propositional EAL (without second-order) 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.

Due to space constraints some proofs are omitted in this paper, but can be found in [ABT06].

Ackowledgements. The authors wish to thank Laurent Régnier for useful discussions related to the topic of this paper.

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):

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

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

It is well known that there is no sensible resource bound (i.e. time/space) on the execution of system F terms in general. On the other hand, we are practically interested in those terms which can be executed in polynomial time. Since the class $\mathcal{P}$ of such terms is not recursively enumerable (as can be easily shown by reduction of the complement of Hilbert's 10th problem), we are naturally led to the study of sufficiently large subclasses of $\mathcal{P}$. The system $DLAL$ gives such a class in a purely type-theoretic way.

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 with 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 (§ 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.

An example of concrete program typable in $DLAL$ is given in Section 6.

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

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

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:

$$d(\alpha) = 0, \qquad d(A \multimap B) = max(d(A), d(B)), \qquad d(\forall\alpha.B) = d(B),$$
$$d(\S A) = d(A) + 1, \quad d(A \Rightarrow B) = max(d(A) + 1, d(B)).$$

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]:

$$\frac{}{;x^{A^-}:A\vdash x^{A^-}:A}\ (\mathrm{Id})$$

$$\frac{\Gamma;x^{A^-}:A,\Delta\vdash M:B}{\Gamma;\Delta\vdash\lambda x^{A^-}.M:A\multimap B}\ (\multimap \mathrm{i}) \qquad \frac{\Gamma_1;\Delta_1\vdash M:A\multimap B \quad \Gamma_2;\Delta_2\vdash N:A}{\Gamma_1,\Gamma_2;\Delta_1,\Delta_2\vdash (M)N:B}\ (\multimap \mathrm{e})$$

$$\frac{x^{A^-}:A,\Gamma;\Delta\vdash M:B}{\Gamma;\Delta\vdash\lambda x^{A^-}.M:A\Rightarrow B}\ (\Rightarrow \mathrm{i}) \qquad \frac{\Gamma;\Delta\vdash M:A\Rightarrow B \quad ;z:C\vdash N:A}{\Gamma,z:C;\Delta\vdash (M)N:B}\ (\Rightarrow \mathrm{e})\ (*)$$

$$\frac{\Gamma_1;\Delta_1\vdash M:A}{\Gamma_1,\Gamma_2;\Delta_1,\Delta_2\vdash M:A}\ (\mathrm{Weak}) \qquad \frac{x_1:A,x_2:A,\Gamma;\Delta\vdash M:B}{x:A,\Gamma;\Delta\vdash M[x/x_1,x/x_2]:B}\ (\mathrm{Cntr})$$

$$\frac{;\Gamma,\Delta\vdash M:A}{\Gamma;\S\Delta\vdash M:\S A}\ (\S \mathrm{i}) \qquad \frac{\Gamma_1;\Delta_1\vdash N:\S A \quad \Gamma_2;x:\S A,\Delta_2\vdash M:B}{\Gamma_1,\Gamma_2;\Delta_1,\Delta_2\vdash M[N/x]:B}\ (\S \mathrm{e})$$

$$\frac{\Gamma;\Delta\vdash M:A}{\Gamma;\Delta\vdash \Lambda\alpha.M:\forall\alpha.A}\ (\forall \mathrm{i})\ (**) \qquad \frac{\Gamma;\Delta\vdash M:\forall\alpha.A}{\Gamma;\Delta\vdash (M)B^-:A[B/\alpha]}\ (\forall \mathrm{e})$$

(*) $z:C$ can be absent.
(**) α does not occur free in Γ,Δ.

Fig. 1. Typing system F terms in $DLAL$

Theorem 1.

1. *For every function* $f:\{0,1\}^*\longrightarrow\{0,1\}^*$ *in* DTIME$[n^k]$*, there exists a closed term* M *of type* $W_{DLAL}\multimap\S^dW_{DLAL}$ *with* $d=O(\log k)$ *representing* f*.*
2. *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*. Moreover, the size of any intermediary term occuring in normalization is also bounded by* $O(|M|^{2^d})$*.*

Although $DLAL$ does not capture all Ptime *algorithms* $\mathcal{P}$, the result 1 guarantees that $DLAL$ is at least expressive enough to represent all Ptime *functions*. In fact, $DLAL$ is as expressive as LAL even at the level of algorithms, because there exists a generic translation from LAL to $DLAL$ given by:

$$(!A)^o=\forall\alpha.((A^o\Rightarrow\alpha)\multimap\alpha), \qquad (.)^o \text{ commutes with other connectives than } !.$$

See [Ter04b] for details.

The result 2 on the other hand implies that if we ignore the embedded types occurring in M, the normal form of M can be computed in polynomial time (*by ordinary* β*-reduction*; that is the difference from LAL).

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 $W_{DLAL}\multimap\S^dW_{DLAL}$ for some $d\geq 0$. Then, by the consequence of the above theorem, we know that the term M is Ptime. In fact, given a binary word $w\in\{0,1\}^*$, consider its Church coding $\underline{w}$ of type W_{DLAL}. Then we have that $(M)\underline{w}$ has type $\S^dW_{DLAL}$, and can thus be evaluated in $O(|\underline{w}|^{2^{d+1}})$ steps. Thus by assigning a $DLAL$ type to a given system F term, one can *statically verify* a polynomial time bound for its execution.

In order to use $DLAL$ for resource verification of system F terms, we address the following problem:

Problem 2 ($DLAL$ typing). Given a closed term M^T of system F, determine if there is a decoration A of T 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.

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:

$$A ::= \alpha \mid D \multimap A \mid \forall\alpha.A \mid \S A\ , \quad D ::= A \mid !A\ .$$

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 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 $D \in \mathcal{L}_{DLAL\star}$. 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 $\S, \bar{\S}$, 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:

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

where m is an arbitrary value in $\mathbb{Z}$ and $\S^m u$ is $\S \cdots \S u$ (resp. $\bar{\S} \cdots \bar{\S} u$) with m (resp. $-m$) occurrences of $\S$ (resp. $\bar{\S}$) if $m \geq 0$ (resp. $m < 0$). So 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 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 $\mathcal{L} = \{\S, \bar{\S}\}^\star$ and $\leq$ the prefix ordering. 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. If $t = u$, define $\mathsf{doors}(t, u) = \epsilon$. Otherwise:

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

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 $s(l) = 0$: think of $\S$ and $\bar{\S}$ resp. as opening and closing brackets.

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 $A \multimap !B$):

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

In the first pseudo-term, the bang subterm $\S((\bar{\S}x)\bar{\S}z)$ contains more than one free variable. 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*):

$$(1)\ \S\forall\alpha.A \multimap \forall\alpha.\S A , \qquad (2)\ \forall\alpha.\S A \multimap \S\forall\alpha.A .$$

Assuming α occurs free 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):

$$t_1 = \lambda x^{\S\forall\alpha.A}.\Lambda\alpha.\S((\bar{\S}x)\alpha) , \qquad t_2 = \lambda x^{\forall\alpha.\S A}.\S\Lambda\alpha.\bar{\S}((x)\alpha) .$$

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.

So far we have introduced four conditions on pseudo-terms: local typing, bracketing, bang and Λ-scope. Let us call a regular pseudo-term satisfying these conditions *well-structured*. It turns out that the well-structured pseudo-terms exactly correspond to the $DLAL$ typing derivations.

Theorem 3. *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 well-structured.*

The 'only-if' direction can be shown by induction on the length of the derivation. To show the converse, we observe that whenever pseudo-terms $\lambda x^D.t$, $(t)u$, $\Lambda\alpha.t$, $(t)A$ are well-structured, so are 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 4 (Boxing). *If $\S(t_A)$ is a well-structured pseudo-term, then there exist pseudo-terms v_A, $(u_1)_{\S B_1}$, ..., $(u_n)_{\S B_n}$, 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. *$v, u_1, \ldots, u_n$ are well-structured.*

As a consequence of Theorem 3, our $DLAL$ typing problem boils down to:

Problem 5 (decoration). Given a system F term M, determine if there exists a decoration t of M which is well-structured.

4 Parameterization and constraints

4.1 Parameterized terms and instantiations

To solve the decoration problem (Problem 5), 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 parameters: *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:

$$F ::= \alpha \mid D \multimap A \mid \forall\alpha.A, \qquad A ::= \S^{\mathbf{c}} F, \qquad D ::= \S^{\mathbf{b},\mathbf{c}} F\,,$$

where $\mathbf{b}$ is a boolean parameter and $\mathbf{c}$ is a linear combination of integer parameters. 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 § or !. 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 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 $\S^{\mathbf{b},\mathbf{c}'+\mathbf{c}}F$. 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) $t, u \ldots$ are defined by the following grammars:

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

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 as follows:

$$\phi(\S^{\mathbf{c}}F) = \S^{\phi^i(\mathbf{c})}\phi(F), \qquad \begin{array}{ll}\phi(\S^{\mathbf{b},\mathbf{c}}F) = \S^{\phi^i(\mathbf{c})}\phi(F) & \text{if } \phi^b(\mathbf{b}) = 0,\\ \quad = !\S^{\phi^i(\mathbf{c})-1}\phi(F) & \text{otherwise,}\end{array}$$

and ϕ commutes with the other connectives. 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 *free linear decoration* (*free bang 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 free bang decoration $\overline{T}$, and to each type U occurring as $(N)U$ in M, we associate a free linear 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:

$$\overline{x^T} = \S^{\mathbf{m}}x^{\overline{T}}, \quad \overline{\lambda x^T.M} = \S^{\mathbf{m}}\lambda x^{\overline{T}}.\overline{M}, \quad \overline{(M)N} = \S^{\mathbf{m}}((\overline{M})\overline{N}),$$
$$\overline{\Lambda\alpha.M} = \S^{\mathbf{m}}\Lambda\alpha.\overline{M}, \quad \overline{(M)T} = \S^{\mathbf{m}}((\overline{M})\overline{T}),$$

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

Lemma 6. *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 7 (instantiation). Given a system F term M, determine if there exists an admissible instantiation ϕ for $\overline{M}$ such that $\phi(\overline{M})$ is well-structured.

For that we will need to be able to state the four conditions (local typing, bracketing, bang, and Λ-scope) 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 Local typing constraints

First of all, we need 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

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

and undefined otherwise. It is straightforward to observe:

Lemma 8. *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 $\mathcal{U}(E_1, E_2)$.*

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 $\mathcal{M}(E)$.

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 set $\mathcal{U}(D^\circ, A)$ used in $\mathcal{M}((t)u)$ is always defined, and finally, $\overline{M}$ satisfies the eigenvariable condition.

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

4.3 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 $\mathcal{L}_p$.

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

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

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. If $t = u$, define $\mathsf{doors}(t, u) = \epsilon$. Otherwise:

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

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 has strictly more than one occurrence of free variable, and
$\{\mathbf{b} = \mathbf{1} \Rightarrow \mathbf{b}' = \mathbf{1}\}$ if u has exactly one occurrence of free variable $x^{\S^{\mathbf{b}',\mathbf{c}'} F'}$.
(ii) $\{\mathbf{b} = \mathbf{1} \Rightarrow s(\mathsf{doors}(u, v)) \geq \mathbf{1} \ : \ v$ 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:
(i) $\mathsf{wbracket}(\mathsf{doors}(u, v))$ for each subterm $\Lambda\alpha.u$ of $\overline{M}$ and for each subterm v of u that depends on α.
We denote $\mathsf{Const}(\overline{M}) = \mathsf{Ltype}(\overline{M}) \cup \mathsf{Bracket}(\overline{M}) \cup \mathsf{Bang}(\overline{M}) \cup \mathsf{Scope}(\overline{M})$. Then:

Theorem 9. *Let M be a system F term and ϕ be an instantiation for $\overline{M}$. Then: ϕ is admissible for M and $\phi(\overline{M})$ is well-structured 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 split $\mathsf{Const}(\overline{M})$ into three disjoint sets $\mathsf{Const}^b(\overline{M})$, $\mathsf{Const}^i(\overline{M})$, $\mathsf{Const}^m(\overline{M})$:

- 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 $\mathsf{Ltype}(\overline{M})$), $\mathbf{c} = \mathbf{0}$ (in $\mathsf{Ltype}(\overline{M})$ and $\mathsf{Bracket}(\overline{M})$),
  $\mathbf{c} \geq \mathbf{0}$ (in $\mathsf{Ltype}(\overline{M})$, $\mathsf{Bracket}(\overline{M})$, $\mathsf{Scope}(\overline{M})$).
- 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 10. *$\mathsf{Const}^b(\overline{M})$ has a solution if and only if it has a minimal solution ψ^b. Moreover one can decide in time polynomial in the cardinality of $\mathsf{Const}^b(\overline{M})$ if there exists a solution, and in that case provide a minimal one.*

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 11. *$\mathsf{Const}(\overline{M})$ 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 11 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 12. *$\psi^b\mathsf{Const}^m(\overline{M}) \cup \mathsf{Const}^i(\overline{M})$ has a solution in $\mathbb{Q}$ if and only if it has a solution in $\mathbb{Z}$.*

Theorem 13. *Let M be a System F term. Then one can decide in time polynomial in the cardinality of $\mathsf{Const}(\overline{M})$ whether $\mathsf{Const}(\overline{M})$ admits a solution.*

Proof. First decide if there is a solution of $\mathsf{Const}^b(\overline{M})$, and if it exists, let ψ^b be the minimal one (Lemma 10). Then 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 (Lemma 12). Otherwise, $\mathsf{Const}(\overline{M})$ is not solvable.■

By combining Theorem 3, Lemma 6, Theorems 9 and 13, we finally get:

Theorem 14. *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

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).

Example of execution. Let us consider the reversing function on binary words. It can be defined by a single higher-order iteration, and thus represented by the following system F term, denoted **rev**:

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

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**, in order to force a meaningful typing. 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:

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

It corresponds to the natural depth-1 typing of the term **rev**, 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 and the program are available at:

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

7 Conclusion

We showed that typing of system F terms in $DLAL$ can be performed in a feasible way, by reducing typability to a constraints solving problem and designing a resolution algorithm. This demonstrates a practical advantage of $DLAL$ over LAL, while keeping the other important properties. Other typing features could still be automatically infered, like coercions (see [Ata05] for the case of EAL).

This work illustrates how Linear logic proof-net notions like boxes can give rise to techniques effectively usable in type inference, even with the strong boxing discipline of $DLAL$, which extends previous work on EAL. We expect that some of these techniques could be adapted to other variants of Linear logic, existing or to be defined in the future.

References

[ABT06] V. Atassi, P. Baillot, and K. Terui. Verification of Ptime reducibility for system F terms via Dual Light Affine Logic. Technical Report HAL ccsd-00021834, july 2006.

[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.

[Ata05] V. Atassi. Inférence de type en logique linéaire élémentaire. Master's thesis, Université Paris 13, 2005.

[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 LICS'04*. IEEE Computer Press, 2004.

[BT05] P. Baillot and K. Terui. A feasible algorithm for typing in elementary affine logic. 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.

[DJ03] V. Danos and J.-B. Joinet. Linear logic and elementary time. *Information and Computation*, 183(1):123–137, 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.

[Ter04a] K. Terui. Light affine set theory: a naive set theory of polynomial time. *Studia Logica*, 77:9–40, 2004.

[Ter04b] K. Terui. A translation of LAL into DLAL. Preprint, http://research.nii.ac.jp/∼ terui, 2004.

[Wel99] J. B. Wells. Typability and type checking in system F are equivalent and undecidable. *Ann. Pure Appl. Logic*, 98(1-3), 1999.