

From Proof-Nets to Linear Logic Type Systems for Polynomial Time Computing

Patrick Baillot *

LIPN, CNRS & Université Paris 13,
99 av. J-B. Clément, 93430 Villetaneuse, France
patrick.baillot@lipn.univ-paris13.fr

Abstract. In this presentation we give an overview of Dual Light Affine Logic (*DLAL*), a polymorphic type system for lambda calculus ensuring that typable lambda terms are executable in polynomial time. We stress the importance of proof-nets from Light linear logic for the design of this type system and for a result establishing that typable lambda-terms can be evaluated efficiently with optimal reduction. We also discuss the issue of *DLAL* type inference, which can be performed in polynomial time for system F terms. These results have been obtained in collaborations with Terui [1], Atassi and Terui [2], and Coppola and Dal Lago [3].

Implicit Computational Complexity (ICC) is concerned with the study of computation with bounded time or memory. It has emerged from early works such as those of Leivant [4], Bellantoni and Cook [5], Jones [6], Leivant and Marion [7]. Lambda calculus and functional programming play a key role in this field. A particular interest is attached to *feasible computing*, by which we mean computing in polynomial time in the size of the input (PTIME).

Instead of seeing execution time simply as a result of observation, the driving motivation here is to shed some light on the nature of feasible computing, by unveiling some invariants or some programming methodologies which can in a modular way ensure that the resulting programs remain in the feasible class. Challenging goals in this area are to obtain manageable programming languages for feasible computing or to delineate some constructive proof systems in which extracted programs would be certified to be of polynomial time complexity.

An important issue is that of *intensional expressivity*: even if all polynomial time *functions* are representable, not all ICC systems have the same expressive power when it comes to implement concrete *algorithms*.

Linear logic. Here we focus our attention on the *linear logic* (*LL*) approach to ICC, which is based on the *proofs-as-programs* correspondence. By giving a logical status to the operation of duplication linear logic provides a fine-grained way to study and control the dynamics of evaluation. Indeed various choices of rules for the modalities (*exponential* connectives), regulating duplication, result in variants of linear logic with different bounds on proof normalization (cut elimination).

* Partially supported by project NO-CoST (ANR, JC05_43380).

This approach lead in particular to *Bounded (BLL)* [8], *Soft (SLL)* [9] and *Light linear logics (LLL)* [10] (or its variant *Light Affine Logic LAL* [11]), which correspond to PTIME computation. The language used is that of *proof-nets*, a graph syntax for proofs. Using the Curry-Howard correspondence the system *LAL* for instance gives a calculus for polynomial time computing in the ICC approach (see *e.g.* [12]).

Types. We will describe a type system for lambda calculus obtained from Light linear logic, called *Dual Light Affine Logic (DLAL)*. If a term acting say, on binary lists, is well typed in *DLAL*, then it runs in polynomial time. The advantage here with respect to *LAL* is that the source language, lambda calculus, is a standard one and the discipline for ensuring polynomial time bounds is managed by the type system. A nice aspect also w.r.t. other type-based ICC systems such as *e.g.* [13] is that the lambda calculus does not contain constants and recursor, but instead the data types and the corresponding iteration schemes are definable, as in system F. Indeed, *DLAL*, as other type systems from Light logics can be seen as a refinement of system F types.

Proof-nets and boxes. Essentially a *DLAL* type derivation corresponds to an *LAL* proof-net. The main extra information with respect to the underlying system F term lies in the presence of *boxes*, corresponding to the use of modalities. Boxes are a standard notion in the proof-net technology. They are usually needed to perform proof-net normalization. We emphasize here a double aspect of boxes in Light logics:

- from a methodological point of view: boxes are a key feature in Light logics (and thus in the design of Light type systems) because they allow to enforce some invariants which guarantee the complexity bound;
- from an operational point of view: boxes can somehow be forgotten for evaluation of (typable) terms; this can be achieved either by using the *DLAL* type system and β reduction, or by using *optimal reduction*, that is to say graph rewriting.

We will illustrate this double aspect of boxes and the interplay between lambda terms and proof-nets (Fig 1) by discussing the *DLAL* type assignment system, optimal reduction of typable terms and finally *DLAL* type inference.

1 Type System *DLAL* and Proof-Nets

Dual Light Affine Logic (DLAL) is a type system derived from Light Affine Logic [1]. Its type language is defined by:

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

where \multimap (resp. \Rightarrow) is a linear (resp. non-linear) arrow connective. An integer called *depth* is attached to each derivation. The main property of *DLAL* is:

Theorem 1. *If a lambda term t is typable in *DLAL* with a type derivation of depth d , then any β reduction sequence of t has length bounded by $O(|t|^{2^d})$.*

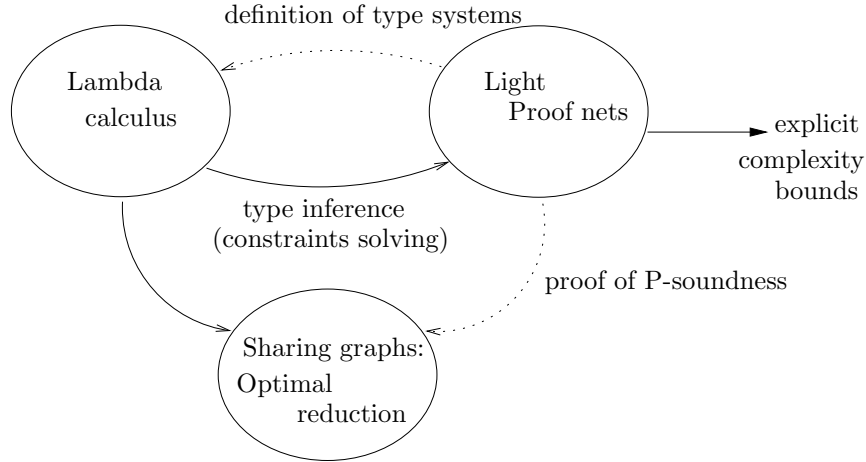


Fig. 1. Lambda terms and Light proof-nets.

$DLAL$ can be translated in LAL using $(A \Rightarrow B)^* = !A^* \multimap B^*$. Consequently any $DLAL$ type derivation corresponds to an LAL proof-net. These proof-nets have two kinds of boxes: $!$ -boxes, which are duplicable, and \S -boxes which are not. The depth of a $DLAL$ derivation corresponds to the maximal nesting of boxes in the corresponding proof-net.

All polynomial time *functions* can be represented in $DLAL$. However its intensional expressivity as that of other Light systems is actually quite weak (some simple PTIME system F terms are non typable). On the other hand testing if a term is typable can be done efficiently, as we will see in Section 3.

2 Evaluating without Boxes: Optimal Reduction

Boxes are an important information in proof-nets and are needed for their normalization (see *e.g.* [14]). However it turns out that $DLAL$ typable lambda terms can be evaluated with a local graph-rewriting procedure (without the boxes): Lamping's abstract algorithm for optimal reduction. The advantage of this abstract algorithm with respect to Lamping's general algorithm [15, 16] is that no bookkeeping is needed for managing the indices, which makes it particularly simple.

The fact that Lamping's algorithm is correct for LAL or EAL (Elementary Affine Logic) typed terms was actually a main motivation for the study of these systems for quite a while [17]. However a recent result [3] is that Lamping's abstract algorithm applied to $DLAL$ or LAL typable terms is of similar complexity as proof-net reduction, that is to say polynomial in the size, if the depth is fixed. This has been achieved by using as tools proof-nets and *context semantics* ([18]).

3 Recovering Boxes : Type Inference by Linear Programming

We stressed that typable lambda terms can be evaluated efficiently *without* the typing information. So, why would we need types anyway then ? Actually the typing can be important: (i) to get an explicit complexity bound for the reduction (Theorem 1); (ii) for modularity: to be able to combine terms together and stay in the typable class.

The specificity of *DLAL* type inference is the inference of modalities, so we consider as input a system F typed lambda term. Concretely type inference in *DLAL* for this term can then be performed by decorating the syntax tree of the term with boxes and the types with modalities, that is to say by *constructing a proof-net*. An algorithm searching for such decorations by using constraints generation and solving has been given in [2] (Fig. 2), after works on related systems, *e.g.* in [19–21].

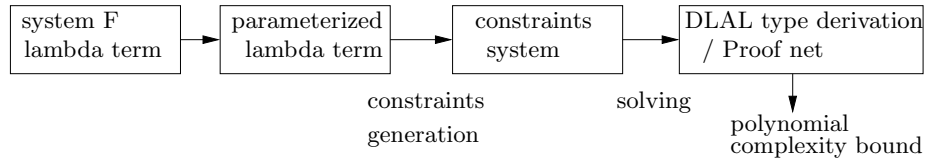


Fig. 2. *DLAL* type inference.

The algorithm relies on two ingredients:

- analysing where non-linear application (and hence !-box) is needed: this is expressed by *boolean constraints*;
- searching for a suitable distribution of boxes (! or § boxes): the key point here is to assign integer parameters for the number of (box) *doors* on each edge, and to search for instantiations of these parameters for which valid boxes can be reconstructed.

In this way a set of constraints on boolean (\mathbf{b}_i) and integer (\mathbf{n}_i) parameters is associated to a term, expressing its typability. It contains:

- boolean constraints, *e.g.* $\mathbf{b}_1 = \mathbf{b}_2, \mathbf{b}_1 = \mathbf{0}$
- linear constraints, *e.g.* $\sum_i \mathbf{n}_i \leq 0$
- mixed boolean/linear constraints, *e.g.* $\mathbf{b}_1 = 1 \Rightarrow \sum_i \mathbf{n}_i \leq 0$.

A resolution method for solving the constraints system is given by the following two-step procedure:

1. boolean phase: search for the *minimal* solution to the boolean constraints. This corresponds to doing a linearity analysis (determine which applications are linear and which ones are non-linear).
2. linear programming phase: once the constraints system is instantiated with the boolean solution, we get a linear constraints system, that can be solved with linear programming methods. This corresponds to finding a concrete distribution of boxes satisfying all the conditions.

This resolution procedure is correct and complete and it can be performed in polynomial time w.r.t. the size of the original system F term. Any solution of the constraints system gives a valid *DLAL* type derivation.

References

1. Baillot, P., Terui, K.: Light types for polynomial time computation in lambda-calculus. In: Proceedings of LICS'04, IEEE Computer Society (2004) 266–275
2. Atassi, V., Baillot, P., Terui, K.: Verification of Ptime reducibility for system F terms via Dual Light Affine Logic. In: Proceedings of CSL'06. Number 4207 in LNCS, Springer (2006) 150–166
3. Baillot, P., Coppola, P., Dal Lago, U.: Light logics and optimal reduction: Completeness and complexity. In: Proceedings of LICS'07, IEEE Computer Society (2007) to appear.
4. Leivant, D.: Predicative recurrence and computational complexity I: word recurrence and poly-time. In: Feasible Mathematics II. Birkhauser (1994) 320–343
5. Bellantoni, S., Cook, S.: New recursion-theoretic characterization of the polytime functions. *Computational Complexity* **2** (1992) 97–110
6. Jones, N.: LOGSPACE and PTIME characterized by programming languages. *Theoretical Computer Science* **228**(1-2) (1999) 151–174
7. Leivant, D., Marion, J.Y.: Lambda-calculus characterisations of polytime. *Fundamenta Informaticae* **19** (1993) 167–184
8. Girard, J.Y., Scedrov, A., Scott, P.: Bounded linear logic: A modular approach to polynomial time computability. *Theoretical Computer Science* **97** (1992) 1–66
9. Lafont, Y.: Soft linear logic and polynomial time. *Theoretical Computer Science* **318**(1–2) (2004) 163–180
10. Girard, J.Y.: Light linear logic. *Information and Computation* **143** (1998) 175–204
11. Asperti, A., Roversi, L.: Intuitionistic light affine logic. *ACM Transactions on Computational Logic* **3**(1) (2002) 1–39
12. Terui, K.: Light affine lambda calculus and polynomial time strong normalization. *Archive for Mathematical Logic* **46**(3-4) (2007) 253–280
13. Hofmann, M.: Linear types and non-size-increasing polynomial time computation. *Information and Computation* **183**(1) (2003) 57–85
14. Mazza, D.: Linear logic and polynomial time. *Mathematical Structures in Computer Science* **16**(6) (2006) 947–988
15. Lamping, J.: An algorithm for optimal lambda calculus reduction. In: Proceedings of POPL'90, ACM (1990) 16–30
16. Gonthier, G., Abadi, M., Lévy, J.J.: The geometry of optimal lambda reduction. In: Proceedings of POPL'92, ACM (1992) 15–26
17. Asperti, A.: Light affine logic. In: Proceedings of LICS'98, IEEE Computer Society (1998)
18. Dal Lago, U.: Context semantics, linear logic and computational complexity. In: Proceedings of LICS'06, IEEE Computer Society (2006) 169–178
19. Coppola, P., Martini, S.: Optimizing optimal reduction: A type inference algorithm for elementary affine logic. *ACM Trans. Comput. Log.* **7**(2) (2006) 219–260
20. Coppola, P., Ronchi Della Rocca, S.: Principal typing for lambda calculus in elementary affine logic. *Fundam. Inform.* **65**(1-2) (2005) 87–112
21. Coppola, P., Dal Lago, U., Ronchi della Rocca, S.: Elementary affine logic and the call-by-value lambda calculus. In: Proceedings of TLCA'05. Volume 3461 of LNCS., Springer (2005) 131–145