Linear Logic, Types and Implicit Computational Complexity

Patrick Baillot

LIPN, CNRS Université Paris 13
11 mars 2008

soutenance d’habilitation à diriger les recherches en informatique
Université Paris 13
Introduction

- Proofs are used to certify various properties of programs: termination, correctness...

- Semantics and logic aim at providing a structured understanding of computing, amenable to formal reasoning, however in practice time and memory usage (resources) play a crucial role.

Can proofs and semantics be refined so as to take into account time/space properties?
→ revisit logical and semantical foundations in this perspective
Computational complexity

- computational complexity: analysing time/memory usage of programs, functions. Initially based on (unstructured) computational models like Turing machines, boolean circuits...

- feasible computing: computing in polynomial time (Ptime) w.r.t. the size of the input.
Implicit computational complexity (ICC)

- Define characterizations of complexity classes (like Ptime) based on language restrictions (types, restrictions on control structures...) and not on external measure conditions.

- Goals:
  1. study complexity classes (*functions*),
  2. analyse programs (static analysis).

- various approaches (since the 90s)
  - **ramified/safe recursion**: primitive recursive definitions, with disciplines for nesting (Leivant, Bellantoni-Cook);
  - **linear logic** (Girard, Lafont)
  - **functional programming** (Jones);
  - **term rewriting**: *quasi-interpretations* (Bonfante-Marion-Moyen);
  - **non-size-increasing linear types** (Hofmann)
Two research lines in ICC

- **geometrical viewpoint**: linear logic, nets, geometry of interaction...

- **algorithmic viewpoint**:
  1. validate more common algorithms,
  2. decide effectively if a program is validated,
  quasi-interpretations for TRS, *non-size-increasing* types . . .

Point 1. deals with *intensional expressivity*: even if all Ptime *functions* are represented by an ICC system, it does not validate all Ptime algorithms (e.g. Quicksort).
Linear logic

- framework considered here: *Curry-Howard correspondence*
  
  proofs = programs
  
  formulas \( (A \rightarrow B) \) = types
  
  proof normalization = program execution
  
  in particular:
  
  intuitionistic logic \( \leftrightarrow \) simply typed \( \lambda \)-calculus
  
  (basis of functional languages, like CAML)

- *linear logic* (LL): fine grained decomposition of intuitionistic logic. LL gives a logical status to duplication, with specific connectives (exponentials \( !, ? \)).
several variants of LL corresponding to different complexity classes: *light logics* [GSS92,Girard98,Lafont04]

present work:
study of light logics under various aspects:
- denotational semantics
- computational properties
- types: usage for $\lambda$-calculus

It has been carried out as part of several projects:
GEOCAL (ACI), CRISS (ACI), NOCoST (ANR).
Overview

- Background on linear logic and light logics

- Contributions:
  1. Denotational semantics
  2. Curry-Howard correspondence for light logics
  3. Light logics as types: first typing methods
  4. Design of a light type system and efficient inference (zoom)
  5. Optimal reduction of $\lambda$-calculus

- Research perspectives

- Conclusion
Background: Linear logic (LL)

intuitionistic logic    linear logic

\[ A \rightarrow B \quad \quad !A \rightarrow B \]

Formulas of intuitionistic linear logic:

\[ A, B ::= \alpha \mid A \rightarrow B \mid A \otimes B \mid !A \mid \forall \alpha. A \]

Contraction and weakening rules:

\[
\begin{align*}
\frac{!A, !A, \Gamma \vdash B}{!A, \Gamma \vdash B} & \quad \text{cont} \\
\frac{\Gamma \vdash B}{!A, \Gamma \vdash B} & \quad \text{weak}
\end{align*}
\]
Light linear logic LLL (Girard 95)

The system LLL is obtained by:

- restricting the rule of the LL connective !:

\[
\frac{\Gamma \vdash A}{!\Gamma \vdash !A} \quad \text{if } |\Gamma| \leq 1
\]

ELL (Elementary linear logic): same rule but no condition on $|\Gamma|$.

- adding a new connective $\&$:

\[
\frac{\Gamma \vdash A}{\&\Gamma \vdash \& A}
\]

and $!A \rightarrow \& A$,

but $\& A$ does not allow for contraction.
# The family of light logics

<table>
<thead>
<tr>
<th>system</th>
<th>BLL</th>
<th>ELL</th>
<th>LLL</th>
<th>SLL</th>
</tr>
</thead>
<tbody>
<tr>
<td>complexity class characterized</td>
<td>Ptime</td>
<td>Elementary</td>
<td>Ptime</td>
<td>Ptime</td>
</tr>
<tr>
<td>reference</td>
<td>[GSS92]</td>
<td>[Gir98]</td>
<td>[Gir98]</td>
<td>[Laf04]</td>
</tr>
<tr>
<td>affine variants</td>
<td>EAL</td>
<td>LAL</td>
<td>SAL</td>
<td></td>
</tr>
</tbody>
</table>
LLL: main properties

- proofs can be represented by proof-nets: graphs with boxes (introduction of modalities $!, \otimes$).
  depth $d(R)$ of a proof-net $R$: maximal nesting of boxes.

**Theorem 1** [Gir98] Let $R$ be a proof-net of depth $d$. It can be normalized in $O((d + 1).|R|^{2d+1})$ steps.

$W$ type of binary words.

Proofs of conclusion $W \vdash \otimes^k W$, with $k \in \mathbb{N}$ then represent polynomial time functions.

Conversely, any polynomial time function can be represented in LLL. This is obtained by simulating Ptime Turing machines in the logic.
1. Denotational semantics

ELL, LLL have been defined from syntactic considerations. We gave categorical semantics which are valid models resp. of ELL and LLL, but not of ordinary LL.

Light logics proofs contain both (i) an algorithmic part and (ii) information about the quantitative behaviour of the algorithm (modality decoration).

Ex: (in LLL)

\[(N \multimap \#N) \multimap \#^2N\]

\(\rightarrow\) 2 possible ways to study them

1. directly by the Curry-Howard isomorphism:
   proof = program

2. as type systems for \(\lambda\)-calculus:
   proof = type derivation

(1) is suitable for analysing the dynamics;

(2) is convenient for distinguishing the programming part from the complexity certification one.
2. Curry-Howard correspondence for light logics

In this approach one programs with proofs, or with specific term calculi (with constructs for each connective).
We contributed to this approach with:

- a term calculus for SLL (soft $\lambda$-calculus),
  P. Baillot, V. Mogbil. Soft $\lambda$-calculus, a language for polynomial time computation, FOSSACS’04.

- study of the extension of LLL (affine variant LAL) with type fixpoints, and expressivity of its various fragments, w.r.t. uniform encodings of functions.
3. Light logics as type systems

Static typing for guaranteeing dynamic properties:

Language Property of well-typed program:
CAML absence of error at runtime
system F termination
light logic (Ptime) complexity bound

A type derivation for a λ-calculus term in a light logic can then be seen as a complexity certificate.
Typing in LAL

LAL as type system for $\lambda$-calculus:

\[
t \text{typed term} \quad \rightarrow \quad R \text{proof-net} \quad \rightarrow \quad \text{normalization of } R
\]

If a term is typable, then it represents a Ptime program.

*type inference problem:* given a term $t$, is it typable in LAL?

method: typing by decoration. start with a simple type derivation, and decorate it with $!, x$ modalities to obtain a suitable LAL derivation

non-trivial problem: there is no *a priori* bound on the number of modalities needed.
Typing by decoration in LAL

[Bai02]: typing algorithm based on parameterization and constraints solving.

Ex: simple type parameterized LAL type

\[ A \rightarrow B \quad !^{n_1 \otimes m_1} ( !^{n_2 \otimes m_2} A \rightarrow !^{n_3 \otimes m_3} B ) \]

domain of parameters: \( \mathbb{N} \)

Typability is reduced to a subclass of Presburger arithmetic constraints.

However, the algorithm has several limitations:

- a. input \( \lambda \)-term in \((\beta)\) normal form
- b. does not deal with polymorphism
- c. no complexity bound for the algorithm
Generalizing typing by decoration

- to overcome the previous limitation (a) ($\lambda$-term in normal form):
  decoration with word parameters:
  
  Ex: simple type parameterized LAL type
  
  \[ A \rightarrow B \quad w_1(w_2 A \rightarrow w_3 B) \]
  
  domain \{!, \#\}*

  express typability by words constraints.
  type inference in (propositional) LAL is then shown to be decidable.


- how to overcome limitations (b) and (c) ?
  a possibility is to simplify the target type system
4. Design of a light type system and efficient inference

Two pitfalls of LAL as a type system for $\lambda$-calculus:

- it does not ensure subject-reduction,
- no polynomial bound on the number of $\beta$-reduction steps for typed terms (even if there is one on proof-net normalization).

One possible solution: restrict the type system, (essentially) by removing types of the form $A \rightarrow !B$. 

Type system DLAL

type language DLAL:

\[ A, B ::= \alpha | A \rightarrow B | A \Rightarrow B | \exists A | \forall \alpha. A \]

typing judgements of the form: \( \Gamma; \Delta \vdash t : A \), where \( \Gamma \) (resp. \( \Delta \)) contains non-linear (resp. linear) variables.

translation from DLAL to LAL: \( (A \Rightarrow B)^* = !A^* \rightarrow B^* \).

Key ingredient of the type system:
if \( (t^A \Rightarrow B u^A) \) is typable, then \( u \) should have at most one occurrence of free variable.

**Theorem 2 (Strong Ptime bound)**  If \( t \) is typable in DLAL with a derivation of depth \( d \), then any \( \beta \) reduction of \( t \) can be performed in time \( O((d + 1) \cdot |t|^{2d+1}) \).

P. Baillot, K. Terui. Light types for polynomial time computation in \( \lambda \)-calculus. LICS’04.

---

Linear Logic, Types and Implicit Computational Complexity – p.21/37
before addressing type inference in DLAL, a simpler problem is that of type inference in EAL, because this system has only one modality, ’!’.

P. Baillot, K. Terui. A feasible algorithm for typing in EAL. TLCA’05

key ingredient: *boxing criterion*

- instead of searching directly for boxes, search for *doors* (opening, closing),
- doors can be matched to form valid boxes *iff* some *bracketing conditions* are satisfied on certain paths.
Example

\[ M = (\lambda f. (f \ (f \ x)))\ ((\lambda h. h) \ g) \]

derivation of \( x : !\alpha, g : !(\alpha \rightarrow \alpha) \vdash M : !\alpha \)
the paths $\gamma_1$, $\gamma_2$ have to be well-bracketed
Example: Parameterized term

example of parameterized type: \( !^{m_1} (!^{m_2} \alpha \rightarrow !^{m_3} \alpha) \)

domain of parameters \( m_i : \mathbb{Z} \).

bracketing conditions:

\[ m_3 \geq 0, \quad m_3 + m_4 \geq 0, \]
\[ m_3 \geq 0, \quad m_3 + m_5 \geq 0 \]
\[ m_3 + m_5 + m_6 \geq 0, \]

etc.
typability of a term in EAL is thus reduced to a linear inequations system. these linear constraints system can be solved in Ptime (by relaxing over \( \mathbb{Q} \)).

it follows that:

**Theorem 3 ([BT05])** given a simply typed term \( t \) it can be decided in time polynomial in \( |t| \) whether \( t \) can be decorated in EAL.
we design a procedure for DLAL typing along the following lines:

- System F lambda term
- Parameterized lambda term
- Constraints system
- DLAL type derivation / Proof net
  
  - Decoration
  - Constraints generation
  - Solving
  - Term with polynomial bound
Type inference in DLAL

parameterized types are now of the following form:

\[ \gamma^{n_1} (\gamma^{n_2,b} A \multimap \gamma^{n_3} B) \]

where \( b \) is a boolean parameter.

idea:

if \( b = 0 \):

\[ \gamma^{n_1} (\gamma^{n_2} A \multimap \gamma^{n_3} B) \]

if \( b = 1 \):

\[ \gamma^{n_1} (\gamma^{n_2-1} A \Rightarrow \gamma^{n_3} B) \]

constraints obtained for expressing typability: mixed boolean/linear constraints

\[ e.g., \quad b_1 = b_2 \Rightarrow \sum_i n_i \geq 0, \]

\[ \ldots \]

these constraints can be solved in Ptime, thanks to a specific procedure.
polymorphism: it is handled with further bracketing conditions.

**Theorem 4**  Given a system F typed lambda term $t$, it can be decided in time polynomial in $|t|$ whether $t$ can be decorated in DLAL.

hence the previous limitations (a) (*normal form*), (b) (*polymorphism*), (c) (*complexity bound*) have been overcome.

# Summary of type inference results

<table>
<thead>
<tr>
<th>Type systems</th>
<th>LAL</th>
<th>LAL</th>
<th>EAL</th>
<th>DLAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity guaranteed by typing</td>
<td>Ptime</td>
<td>Ptime</td>
<td>Elementary</td>
<td>Ptime</td>
</tr>
<tr>
<td>Reference</td>
<td>[Bai02]</td>
<td>[Bai04b]</td>
<td>[BT05]</td>
<td>[ABT07]</td>
</tr>
<tr>
<td>Constraints for inference</td>
<td>Presburger arithmetic</td>
<td>word constraints</td>
<td>linear inequations</td>
<td>mixed boolean/linear constraints</td>
</tr>
<tr>
<td>Complexity of the inference algorithm</td>
<td>Not known (*)</td>
<td>Not known (*)</td>
<td>Ptime</td>
<td>Ptime</td>
</tr>
</tbody>
</table>

(*) at least exponential.
5. Optimal reduction of $\lambda$-calculus

Optimal reduction (Lévy, Lamping): evaluation method that does not duplicate any redex.

Abstract sharing graphs are built from the following nodes ($i \in \mathbb{N}$):

Translation from an LAL (resp. EAL) proof-net $R$ to an abstract sharing graph $T(R)$:

- give an index to each contraction node (for instance its depth),
- remove boxes.
Optimal reduction: Lamping’s abstract algorithm

The indexes on fans remain unchanged (no bookkeeping): this abstract algorithm is *not* correct for arbitrary terms, but is correct for EAL/LAL.
Bounding optimal reduction for LAL/EAL

Does optimal reduction satisfy the complexity bounds of light systems?

\[ \lambda \text{-terms} \]

\[ \vdash \frac{t \xrightarrow{\beta} t'}{R \xrightarrow{n} R'} \quad n \leq p_d(|R|), \text{ with } d = d(R). \]

\[ \text{sharing graphs} \quad T(R) \xrightarrow{m} G' \]

**Theorem 5**  If \( R \) is an LAL proof-net of depth \( d \), the number of steps \( m \) of the optimal reduction of the sharing graph \( T(R) \) is bounded by \( q_d(|R|) \), where \( q_d \) is a polynomial depending on \( d \).

The proof uses techniques from geometry of interaction.

Extend and simplify light logics.
Understand and improve intensional expressivity in ICC.
Adapt ICC approaches to other settings: concurrency.

- **Extension of light logics and applications:**
  - *Linear logic by levels:* with D. Mazza we defined a subclass of LL proof-nets thanks to a notion of *levels*, without $\Box$ boxes [BM07]. The system $L^4$ subsumes LLL and admits a Ptime bound.

- Goals:
  - Define a corresponding type system for $\lambda$-calculus. Type inference should be simplified (fewer constraints).
  - Can one use it for program extraction from proofs?
  - Define a light AF2 (in the style of light set theory)
  - Proof-search would be made easier in this setting (no $\Box$ boxes)
Abstract study of ICC criteria:

with Dal Lago and Moyen we studied properties of first-order functional programs satisfying the Ptime criterion based on quasi-interpretations [BDLM06]. for that we defined a notion of blind abstractions of programs.

Can one in this way give, for various ICC criteria, necessary conditions on the behaviour of programs validated by the criterion?

This could be a way to (i) compare together different ICC criteria, (ii) find ways to extend these criteria.
**Research perspectives**

- **ICC for concurrent systems:**
  could one define ICC criteria for process languages like the $\pi$-calculus?

goal: bounding the size of the system, or the number of steps before terminating.

  some work already done in the setting of synchronous languages (Amadio *et al*), based on quasi-interpretations

  one could try to adapt to the $\pi$-calculus the ingredients underlying light logics, possibly using intermediate languages like nets (differential nets or multiport IN), or higher-order process languages based on $\lambda$-calculus (HOPLA)
Conclusion

- we have recast the study of light logics as a study of \( \lambda \)-calculus,
- the *geometric* approach can lead to algorithmic methods (type inference),
  this approach provides also techniques for the study of evaluation (optimal reduction),
  this could be applied to further developments (proof-nets, GoI),
- could this approach be made more expressive intensionally (algorithms) ? *e.g.* by combining it with other ICC characterizations like NSI, quasi-interpretations.