# From linear logic to types for implicit computational complexity

Patrick Baillot

CNRS / ENS Lyon

Days in Logic 2018
Aveiro

January 25, 2018

# Introduction

- Complexity classes are defined by:
  - a computational model, e.g. TM
  - a constraint on resources, e.g. time, space or size
- This does not say much about **how** to compute within a certain complexity class.
  Complexity classes are defined "from the outside" ...

# Introduction: some questions

- **How** can we compute within a certain complexity class, for instance in FPTIME?
- Which bricks of computation can we use?
  data structures, primitive operations, control structures (e.g. loops) . . .
- . . . without the burden of managing explicit time annotations.

# Introduction: more questions . . .

- A related question: how and when can we compose and iterate functions of a given complexity class?
- Can we define a discipline for transparent and modular FPTIME programming?
- Can we give characterizations of complexity classes not relying on explicit resource bounds?

# Introduction: implicit computational complexity

- Logic and recursion theory can help addressing some of these questions !
- They have triggered
  Implicit computational complexity (ICC) :
  characterizing complexity classes by logics / languages
  without explicit bounds,
  but instead by restricting the constructions

# Introduction: implicit computational complexity

- Logic and recursion theory can help addressing some of these questions !
- They have triggered
  Implicit computational complexity (ICC) :
  characterizing complexity classes by logics / languages
  without explicit bounds,
  but instead by restricting the constructions

# Introduction: ICC systems

- ICC can be both foundations-oriented or certification-oriented
- ICC systems can often be expressed by
  (i) a programming language or calculus, (ii) a criterion on programs

# Various approaches to ICC

- recursion theory : safe recursion (Bellantoni-Cook) / ramified recursion (Leivant)
- **linear logic** (Girard) **this talk**
- types controlling sizes (non-size-increasing) (Hofmann)
- interpretation methods (Marion)
- ...

# The proofs-as-programs viewpoint

- our reference language here is $\lambda$-calculus
  untyped $\lambda$-calculus is Turing-complete
- type systems can guarantee termination
  ex: system F (polymorphic types)

# The proofs-as-programs viewpoint (2)

- proofs-as-programs correspondence

| | | |
|---|---|---|
| proof | = | type derivation |
| normalization | = | execution |
| | | |
| 2nd order intuitionistic logic | $\leftrightarrow$ | system F |

- some characteristics of $\lambda$-calculus:
  higher-order types
  no distinction between data / program

# Linear logic

- linear logic (LL):
  fine-grained decomposition of intuitionistic logic
  duplication is controlled with a specific connective !
  (exponential modality)
- some variants of linear logic with weak rules for ! have bounded complexity: light logics

# How to characterize complexity classes?

the computational engine
logical system

the specification
formula / type

variant of linear logic

formula / type

# How to characterize complexity classes?


the computational engine

logical system

variant of linear logic

the specification

formula / type

formula / type

# Outline of the course

1. $\lambda$-calculus and system F in a nutshell
2. elementary linear logic (ELL): elementary complexity
3. some finer characterizations in ELL
4. light linear logic (LLL): Ptime complexity
5. other linear logic variants
6. conclusion

# Notations for complexity classes

We denote

- $FDTIME(f(n))$ : functions on binary words computable by a Turing machine in time $O(f(n))$
- $FPTIME = \cup_k FDTIME(n^k)$, feasible functions
- $FEXPTIME = \cup_k FDTIME(2^{n^k})$
- *Elementary*: functions computable in time $2^n_k$, for some $k$, where

$$\begin{array}{rcl} 2^x_0 & = & x \\ 2^x_{k+1} & = & 2^{2^x_k} \end{array}$$

# $\lambda$-calculus

- $\lambda$-terms:

$$t, u ::= x \mid \lambda x.t \mid t\ u$$

notations: $\lambda x_1 x_2.t$ for $\lambda x_1.\lambda x_2.t$
$(t\ u\ v)$ for $((t\ u)\ v)$
substitution: $t[u/x]$

- $\beta$-reduction:
$\xrightarrow{1}$ relation obtained by context-closure of:

$$((\lambda x.t)u) \xrightarrow{1} t[u/x]$$

$\rightarrow$ reflexive and transitive closure of $\xrightarrow{1}$.

# Typed $\lambda$-terms

system F types:

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

simple types: without $\forall$

simply typed terms, in Church-style:

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

# Proofs-as-programs correspondence (Curry-Howard)

| | | |
|---|---|---|
| **2nd-order intuitionistic logic proof** | $\Rightarrow$ | **typed term** |
| formula | | type |
| proof of $A_1, \ldots, A_n \vdash B$ | | $M^B$, with free variables $x_i : A_i$, $1 \leq i \leq n$ |
| normalization of proof (cut elimination) | | $\beta$-reduction of term |

# Data types in F

Booleans:

$B^F \quad = \quad \forall\alpha.\alpha \to \alpha \to \alpha$

$\underline{true} = \lambda x.\lambda y.x \qquad \underline{false} = \lambda x.\lambda y.y$

Church unary integers:

$N^F \quad = \quad \forall\alpha.(\alpha \to \alpha) \to (\alpha \to \alpha)$

example

$\underline{2} \quad = \quad \lambda f^{\alpha\to\alpha}.\lambda x^{\alpha}.(f\ (f\ x)) : N^F$

Church binary words:

$W^F \quad = \quad \forall\alpha.(\alpha \to \alpha) \to (\alpha \to \alpha) \to (\alpha \to \alpha)$

example

$\underline{<1,1,0>} \quad = \quad \lambda s_0^{\alpha\to\alpha}.\lambda s_1^{\alpha\to\alpha}.\lambda x^{\alpha}.(s_1\ (s_1\ (s_0\ x))) : W^F$

# Examples of terms (1)

addition

$$
\begin{array}{lcl}
add & = & \lambda nmfx.(n\ f)\ (m\ f\ x) \\
 & : & N \rightarrow N \rightarrow N
\end{array}
$$

multiplication

$$
\begin{array}{lcl}
mult & = & \lambda nmf.(n\ (m\ f)) \\
 & : & N \rightarrow N \rightarrow N
\end{array}
$$

squaring

$$
\begin{array}{lcl}
square & = & \lambda nf.(n\ (n\ f)) \\
 & : & N \rightarrow N \rightarrow N
\end{array}
$$

# Iteration

For each inductive data type an associated iteration principle.
For instance, for $N = \forall\alpha.(\alpha \to \alpha) \to (\alpha \to \alpha)$, we can define an iterator *iter*:

$$iter = \lambda fxn.\,(n\ f\ x)\ : (A \to A) \to A \to N \to A, \quad \text{for any } A$$

then
$(iter\ t\ u\ \underline{n}) \to (t\ (t \ldots (t\ u) \ldots)$ $\quad$ ($n$ times)

**examples:**
$double : N \to N$
$exp = \lambda n.(iter\ double\ \underline{1}\ n)\ : N \to N$
$tower = \lambda n.(iter\ exp\ \underline{1}\ n)\ : N \to N$

## Examples of terms (2)

concatenation

$$\begin{array}{lcl} \textit{conc} & = & \lambda u^W.\lambda v^W.\lambda s_0.\lambda s_1.\lambda x.(u\ s_0\ s_1)\ (v\ s_0\ s_1\ x) \\ & : & W \to W \to W \end{array}$$

length

$$\begin{array}{lcl} \textit{length} & = & \lambda u^W.\lambda f^{\alpha\to\alpha}.(u\ f\ f)^{\alpha\to\alpha} \\ & : & W \to N \end{array}$$

repeated concatenation

$$\begin{array}{lcl} \textit{rep} & = & \lambda n^N.\lambda v^W.[\textit{iter}\ (\textit{conc}\ v)^{W\to W}\ \underline{\textit{nil}}^W\ n^N] \\ & = & \lambda n^N.\lambda v^W.[n\ (\textit{conc}\ v)\ \underline{\textit{nil}}]^W \\ & : & N \to W \to W \end{array}$$

# System F and termination

**Theorem (Girard)**

If a term is well typed in $F$, then it is strongly normalizable.

Thus a type derivation can be seen as a termination witness.
In particular, a term $t : W \rightarrow W$ represents a function on words which terminates on all inputs.

Can we refine this system in order to guarantee feasible termination, that is to say in polynomial time?

# Linear logic

- Linear logic (LL) arises from the decomposition

$$A \Rightarrow B \equiv\ !A \multimap B$$

- the ! modality accounts for duplication (contraction)
- ! satisfies the following principles:

$$!A \multimap\ !A \otimes\ !A \qquad \frac{A \vdash B}{!A \vdash\ !B} \qquad !A \multimap A$$

$$!A \otimes\ !B \multimap\ !(A \otimes B) \qquad !A \multimap\ !!A$$

# Elementary linear logic (ELL) [Girard95]

- Language of formulas:

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

  Denote $!^k A$ for $k$ occurrences of $!$.

- The system is designed in such a way that the following principles are **not** provable

$$!A \multimap A, \quad !A \multimap !!A$$

- Defined to characterize elementary time complexity, that is to say in time bounded by $2^n_k$, for arbitrary $k$.

# Elementary linear logic rules

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

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \multimap B}\ (\multimap \mathrm{i}) \qquad \frac{\Gamma_1 \vdash t : A \multimap B \quad \Gamma_2 \vdash u : A}{\Gamma_1, \Gamma_2 \vdash (t\ u) : B}\ (\multimap \mathrm{e})$$

$$\frac{x_1 :!A, x_2 :!A, \Gamma \vdash t : B}{x :!A, \Gamma \vdash t[x/x_1, x/x_2] : B}\ (\mathrm{Cntr}) \qquad \frac{\Gamma \vdash t : A}{\Gamma, x : B \vdash t : A}\ (\mathrm{Weak})$$

$$\frac{x_1 : B_1, \ldots, x_n : B_n \vdash t : A}{x_1 :!B_1, \ldots, x_n :!B_n \vdash t :!A}\ (!\ \mathrm{i}) \qquad \frac{\Gamma_1 \vdash u :!A \quad \Gamma_2, x :!A \vdash t : B}{\Gamma_1, \Gamma_2 \vdash t[u/x] : B}\ (!\ \mathrm{e})$$

# Elementary linear logic rules (2/2)

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t : \forall \alpha . A} \ (\forall \text{ i}) \ (*) \qquad \frac{\Gamma \vdash t : \forall \alpha . A}{\Gamma \vdash t : A[B/\alpha]} \ (\forall \text{ e})$$

where $(*) : \alpha \notin \Gamma$.

## Elementary linear logic rules: remarks

- This is actually elementary **affine** logic (EAL), because of the unrestricted weakening (not only on $!A$ formulas).
- However throughout this talk we will say linear instead of affine, so ELL will mean EAL ...
- These rules are natural deduction style rules. There is also a sequent calculus presentation of ELL.

# Forgetful map from ELL to F

Consider $(.)^{-} : ELL \rightarrow F$ defined by:

$$(!A)^{-} = A^{-}, \quad (A \multimap B)^{-} = A^{-} \rightarrow B^{-}, \quad (\forall \alpha.A)^{-} = \forall \alpha.A^{-}, \quad \alpha^{-} = \alpha.$$

## Proposition

If $\Gamma \vdash_{ELL} t : A$ then $t$ is typable in F with type $A^{-}$.

If $A^{-} = T$, say $A$ is a decoration of $T$ in ELL.

# Data types in ELL

- Church unary integers

| system F: | ELL: |
|---|---|
| $N^F$ | $N^{ELL}$ |
| $\forall\alpha.(\alpha \to \alpha) \to (\alpha \to \alpha)$ | $\forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha)$ |

Example: integer 2, in F:

$$\underline{2} = \lambda f^{(\alpha\to\alpha)}.\lambda x^{\alpha}.(f\ (f\ x))\ .$$

- Church binary words

| system F: | ELL: |
|---|---|
| $W^F$ | $W^{ELL}$ |
| $\forall\alpha.(\alpha \to \alpha) \to (\alpha \to \alpha) \to (\alpha \to \alpha)$ | $\forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha)$ |

Example: $w = \langle 1, 0, 0\rangle$, in F:

$$\underline{w} = \lambda s_0^{(\alpha\to\alpha)}.\lambda s_1^{(\alpha\to\alpha)}.\lambda x^{\alpha}.(s_1\ (s_0\ (s_0\ x)))\ .$$

# Representation of functions

- a term $t$ of type $!^k N \multimap !^l N$, for some $k$, $l$, represents a function over unary integers
- $!^k W \multimap !^l W$, for some $k$, $l$: function over binary words

# Examples of ELL terms (1)

- some examples of terms

addition

$$
\begin{array}{lll}
add & = & \lambda nmfx.(n\ f)\ (m\ f\ x) \\
 & : & N \multimap N \multimap N
\end{array}
$$

multiplication

$$
\begin{array}{lll}
mult & = & \lambda nmf.(n\ (m\ f)) \\
 & : & N \multimap N \multimap N
\end{array}
$$

squaring

$$
\begin{array}{lll}
square & = & \lambda nf.(n\ (n\ f)) \\
 & : & !N \multimap !N
\end{array}
$$

# Iteration in ELL

recall the iterator *iter*:

$$iter = \lambda f x n.\ (n\ f\ x)\ : !(A \multimap A) \multimap !A \multimap N \multimap !A$$

with $(iter\ t\ u\ \underline{n}) \to (t\ (t\ \ldots (t\ u) \ldots))$ $\quad$ ($n$ times)

**examples:**

$double : N \multimap N$

$exp = (iter\ double\ \underline{1}) : N \multimap !N$

remark: *exp* cannot be iterated; $tower = (iter\ \exp\ \underline{1})$ non ELL typable.

$coercion = (iter\ succ\ \underline{0}) : N \multimap !N$ : an identity, but changes the type

# Iteration in ELL

recall the iterator *iter*:

$$iter = \lambda f x n.\ (n\ f\ x)\ : !(A \multimap A) \multimap !A \multimap N \multimap !A$$

with $(iter\ t\ u\ \underline{n}) \rightarrow (t\ (t\ \ldots (t\ u) \ldots))$ $\quad (n$ times$)$

**examples:**

$double : N \multimap N$

$exp = (iter\ double\ \underline{1}) : N \multimap !N$

remark: $exp$ cannot be iterated; $tower = (iter\, \mathrm{exp}\ \underline{1})$ non ELL typable.

$coercion = (iter\ succ\ \underline{0}) : N \multimap !N$ : an identity, but changes the type

# Iteration in ELL

recall the iterator *iter*:

$$iter = \lambda f x n.\, (n\ f\ x)\ :\ !(A \multimap A) \multimap\ !A \multimap N \multimap\ !A$$

with $(iter\ t\ u\ \underline{n}) \to (t\ (t\ \ldots (t\ u) \ldots))$ $\quad$ ($n$ times)

**examples:**

$double : N \multimap N$

$exp = (iter\ double\ \underline{1}) : N \multimap\ !N$

remark: *exp* cannot be iterated; $tower = (iter\,\mathrm{exp}\ \underline{1})$ non ELL typable.

$coercion = (iter\ succ\ \underline{0}) : N \multimap\ !N$ : an identity, but changes the type

# Examples of ELL terms (2)

concatenation

$$\begin{array}{lcl} \mathit{conc} & = & \lambda u.\lambda v.\lambda s_0.\lambda s_1.\lambda x.(u\ s_0\ s_1)\ (v\ s_0\ s_1\ x) \\ & : & W \multimap W \multimap W \end{array}$$

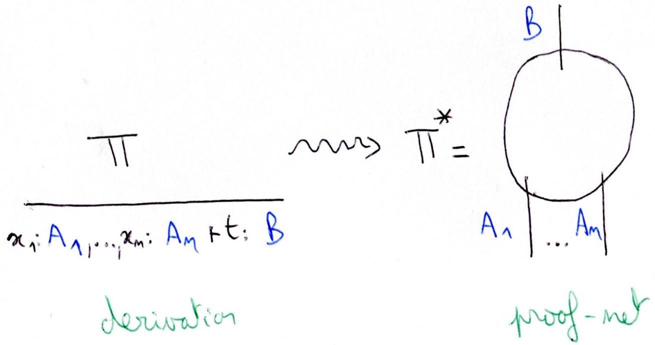
length

$$\begin{array}{lcl} \mathit{length} & = & \lambda u.\lambda f.(u\ f\ f) \\ & : & W \multimap N \end{array}$$

repeated concatenation

$$\begin{array}{lcl} \mathit{rep} & = & \lambda n.\lambda v.[\mathit{iter}\ (\mathit{conc}\ v)\ \underline{\mathit{nil}}\ n] \\ & = & \lambda n.\lambda v.[n\ (\mathit{conc}\ v)\ \underline{\mathit{nil}}] \\ & : & N \multimap !W \multimap !W \end{array}$$

# From derivations to proof-nets



# Elementary linear logic rules, again

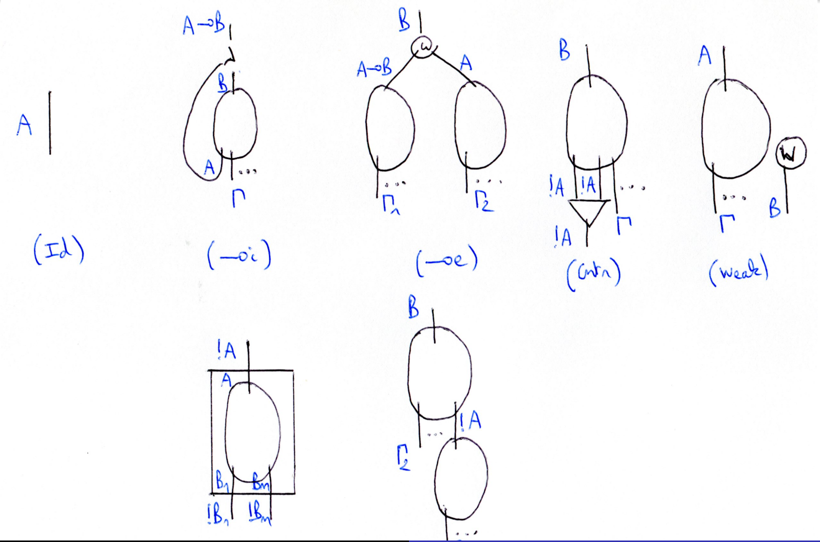
$$\frac{}{x : A \vdash x : A}\ (\mathrm{Id})$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \multimap B}\ (\multimap\ \mathrm{i}) \qquad \frac{\Gamma_1 \vdash t : A \multimap B \quad \Gamma_2 \vdash u : A}{\Gamma_1, \Gamma_2 \vdash (t\ u) : B}\ (\multimap\ \mathrm{e})$$

$$\frac{x_1 :!A, x_2 :!A, \Gamma \vdash t : B}{x :!A, \Gamma \vdash t[x/x_1, x/x_2] : B}\ (\mathrm{Cntr}) \qquad \frac{\Gamma \vdash t : A}{\Gamma, x : B \vdash t : A}\ (\mathrm{Weak})$$

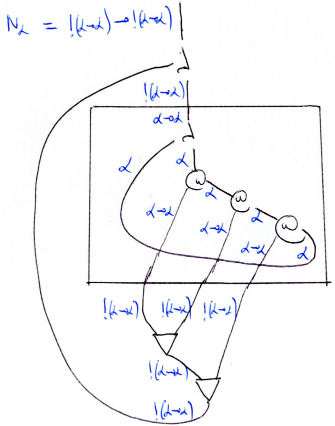
$$\frac{x_1 : B_1, \ldots, x_n : B_n \vdash t : A}{x_1 :!B_1, \ldots, x_n :!B_n \vdash t :!A}\ (!\ \mathrm{i}) \qquad \frac{\Gamma_1 \vdash u :!A \quad \Gamma_2, x :!A \vdash t : B}{\Gamma_1, \Gamma_2 \vdash t[u/x] : B}\ (!\ \mathrm{e})$$

# ELL Proof-Nets



# ELL proof-net : example

Proof-net $R_3$ representing Church integer $\underline{3}$:

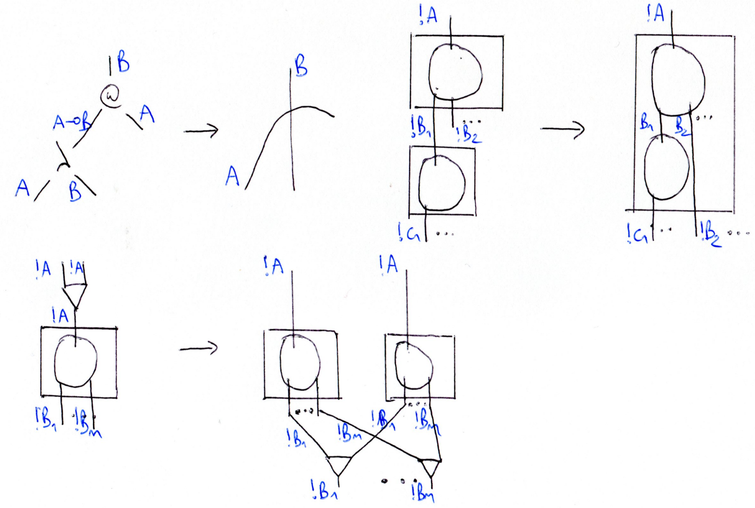


# ELL proof-net: depth

- Depth of an edge $e$ in a proof-net $R$: number of boxes it is contained in.
- Depth $d(R)$ of proof-net $R$: maximal depth of its edges.
- Example:
  The previous proof-net $R_3$ has depth 1.
  Any proof-net $R_n$ representing $n$ has depth 1.

# ELL proof-net reduction : cut elimination

# ELL proof-net reduction : cut elimination



# Methodology

- write programs with ELL typed $\lambda$-terms
- evaluate them by:
  compiling them into proof-nets, and then performing proof-net reduction
- beware:
  - proof-net reduction does not exactly match $\beta$-reduction
  - ELL does not satisfy subject reduction

  but that's all right for our present goal ...

# ELL proof-net reduction properties

- Recall: depth of an edge $e$ in a proof-net $R$ = number of boxes it is contained in.
- We have

**Proposition (Stratification)**

The depth of an edge does not change during reduction.

Consequence: the depth $d$ of a proof-net does not increase during reduction.

- **Level-by-level reduction strategy**:
  $R$ proof-net of depth $d$
  perform reduction successively at depth $0, 1 \ldots, d$.

# Level-by-level reduction of ELL proof-nets

- let $R$ be an ELL proof-net of depth $d$
  $|R|_i$ = number of nodes at depth $i$ = size at depth $i$
  $|R|$ = total size
  round $i$: reduction at depth $i$
  there are $d+1$ rounds for the reduction of $R$
- **what happens during round $i$?**
  - $|R|_i$ decreases at each step
    thus there are at most $|R|_i$ steps (size bounds time)
  - but $|R|_{i+1}$ can increase at each step, in fact it can double
  - hence round $i$ can cause an exponential size increase
- on the whole we have a $2_d^{|R|}$ size increase
- this yields a $O(2_d^{|R|})$ bound on the number of steps

# ELL complexity results

**Theorem (Proof-net complexity)**

If $R$ is an ELL proof-net of depth $d$, then it can be reduced to its normal form in $O(2_d^{|R|})$ steps.

**Theorem (Representable functions)**

The functions representable by a term of type $N \multimap !^k N$, where $k \geq 0$ , are exactly the elementary time functions.

# Proof of the representability theorem

- $\subseteq$ (soundness):
  if $t : N \multimap !^k N$ for some $k$, then $t$ represents an elementary function $f$.

  **proof**: compute $(t \underline{n})$ by proof-net reduction.
- $\supseteq$ (completeness):
  if $f : \mathbb{N} \to \mathbb{N}$ is an elementary function, then there exists $k$ and $t : N \multimap !^k N$ such that $t$ represents $f$.

  **proof**: simulation of $O(2^n_i)$-time bounded Turing machine, for any $i$.

# From linear logic to types for implicit computational complexity (Part 2)

Patrick Baillot

CNRS / ENS Lyon

Days in Logic 2018
Aveiro

January 26, 2018

# Introduction: recap

- Implicit computational complexity (ICC) :
  characterizing complexity classes by logics / languages
  without explicit bounds,
  but instead by restricting the constructions
- we are considering here the proofs-as-programs approach for ICC ...
- ...illustrating the use of linear logic and its weak variants.

# Elementary linear logic (ELL) [Girard95]

Language of formulas:

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

We denote $!^k A$ for $k$ occurrences of $!$.

# Elementary linear logic rules

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

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \multimap B}\ (\multimap \mathrm{i}) \qquad \frac{\Gamma_1 \vdash t : A \multimap B \quad \Gamma_2 \vdash u : A}{\Gamma_1, \Gamma_2 \vdash (t\ u) : B}\ (\multimap \mathrm{e})$$

$$\frac{x_1 :!A, x_2 :!A, \Gamma \vdash t : B}{x :!A, \Gamma \vdash t[x/x_1, x/x_2] : B}\ (\mathrm{Cntr}) \qquad \frac{\Gamma \vdash t : A}{\Gamma, x : B \vdash t : A}\ (\mathrm{Weak})$$

$$\frac{x_1 : B_1, \ldots, x_n : B_n \vdash t : A}{x_1 :!B_1, \ldots, x_n :!B_n \vdash t :!A}\ (!\ \mathrm{i}) \qquad \frac{\Gamma_1 \vdash u :!A \quad \Gamma_2, x :!A \vdash t : B}{\Gamma_1, \Gamma_2 \vdash t[u/x] : B}\ (!\ \mathrm{e})$$

# Elementary linear logic rules (2/2)

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t : \forall \alpha . A} \ (\forall \text{ i}) \ (*) \qquad \frac{\Gamma \vdash t : \forall \alpha . A}{\Gamma \vdash t : A[B/\alpha]} \ (\forall \text{ e})$$

where $(*) : \alpha \notin FV(\Gamma)$.

# Data types in ELL

- Church unary integers

| system F: | ELL: |
|---|---|
| $N^F$ | $N^{ELL}$ |
| $\forall\alpha.(\alpha \to \alpha) \to (\alpha \to \alpha)$ | $\forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha)$ |

Example: integer 2, in F:

$$\underline{2} = \lambda f^{(\alpha\to\alpha)}.\lambda x^{\alpha}.(f\ (f\ x))\ .$$

- Church binary words

| system F: | ELL: |
|---|---|
| $W^F$ | $W^{ELL}$ |
| $\forall\alpha.(\alpha \to \alpha) \to (\alpha \to \alpha) \to (\alpha \to \alpha)$ | $\forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha)$ |

Example: $w = \langle 1, 0, 0\rangle$, in F:

$$\underline{w} = \lambda s_0^{(\alpha\to\alpha)}.\lambda s_1^{(\alpha\to\alpha)}.\lambda x^{\alpha}.(s_1\ (s_0\ (s_0\ x)))\ .$$

# ELL complexity results

**Theorem (Proof-net complexity)**

If $R$ is an ELL proof-net of depth $d$, then it can be reduced to its normal form in $O(2_d^{|R|})$ steps.

**Theorem (Representable functions)**

The functions representable by a term of type $N \multimap !^k N$, where $k \geq 0$ , are exactly the elementary time functions.

# Characterization of complexity classes


the computational engine

logical system

the specification

formula / type

Elementary linear logic (ELL)

$\{N \multimap !^k N\}_{k \geq 0} \equiv Elementary$

# Characterization of complexity classes

the computational engine
logical system

the specification
formula / type

Elementary linear logic (ELL)

$\{N \multimap !^k N\}_{k \geq 0} \equiv \mathit{Elementary}$

# ELL: towards finer characterizations

- We have seen a characterization of the *Elementary* class (elementary complexity) in ELL
- But can we get more fined-grained characterizations? Characterize smaller complexity classes ?

# The system $ELL_\mu$

- we can extend ELL by adding a new construction $\mu\alpha.A$ for formula fixpoints, with the following rules:

$$\frac{\Gamma \vdash t : A[\mu\alpha.A/\alpha]}{\Gamma \vdash t : \mu\alpha.A}\ (\mu f) \qquad \frac{\Gamma \vdash t : \mu\alpha.A}{\Gamma \vdash t : A[\mu\alpha.A/\alpha]}\ (\mu u)$$

  We call $ELL_\mu$ this system.

- **the previous results on *ELL* also hold for $ELL_\mu$** (same bound on cut-elimination).

# Refine the complexity bounds

- By the previous analysis we know that a term $t : !W \multimap !^2 B$ can be evaluated in $O(2^n_2)$, so it is in 2-EXPTIME ...
- but actually it is in ... PTIME

## New characterization in $ELL_\mu$

## New characterization in $ELL_\mu$

**Theorem**

We consider the system $ELL_\mu$.
The functions representable by proofs of $!W \multimap !^2 B$ are exactly the class PTIME, of polynomial time predicates.

# Key Lemma for the soundness proof

For proving complexity soundness we use a more precise bound than before:

**Lemma (Size bound)**

Let $R$ be a proof-net with:

- only exponential cuts at depth 0,
- $k$ cuts at depth 0.

Let $R'$ be the proof-net obtained by reducing $R$ at depth 0. Then we have:

$$|R'|_1 \leq |R|_0^k . |R|_1 .$$

# Fixpoints and Scott integers

in $ELL_\mu$ we can define new data types, eg Scott integers:

$$N_S = \mu\alpha.\forall\beta.(\alpha \multimap \beta) \multimap \beta \multimap \beta$$

in $\lambda$-calculus notation:

$$\begin{array}{lcl} \underline{0} & = & \lambda s.\lambda x.x \\ \underline{n+1} & = & \lambda s.\lambda x.(s\ \underline{n}) \end{array}$$

They allows for constant time predecessor and zero-test, but . . . no iterator.

Similarly one defines $W_S$ for Scott binary words.
We get:

$$case \quad : \quad \forall\alpha.(W_S \multimap \alpha) \multimap (W_S \multimap \alpha) \multimap \alpha \multimap (W_S \multimap \alpha)$$

# Proof of completeness for $!W \multimap !!B$ and PTIME

- any polynomial can be represented with a proof of $!N \multimap !N$. we have $length : W \multimap N$.
- **using type fixpoints** we can define a type $Config_S$ for TM configurations, based on Scott words, with:
  - proofs

$$\begin{aligned} init : &\quad W \vdash Config_S \\ accept? : &\quad Config_S \vdash B \end{aligned}$$

  - for any TM $\mathcal{M}$ a proof

$$step : Config_S \vdash Config_S$$

then, by iterating *step* $q(|w|)$ times on input $(init(w))$ we get:

$$!W \vdash !^2 Config_S.$$

composing with *accept?* we get: $!W \vdash !^2 B$.

# Why do we need type fixpoints?

Without type fixpoints we can define using second-order a type *Config* based on Church integers (following [Asperti-Roversi2002]). We get the same types for *length*, *init*, *step*, and we also obtain by iteration:

$$!W \vdash !^2 Config.$$

However the problem is then that:

$$accept? : Config \vdash !B$$

So this gives:

$$!W \vdash !^3 B,$$

which is not the type needed ...

# General characterization theorem

**Theorem**

We consider the system $ELL_\mu$.

- The functions representable by proofs of $!W \multimap !^2 B$ are exactly the class PTIME;
- The functions representable by proofs of $!W \multimap !^{k+2} B$ are exactly the class $k$-EXPTIME ($k \geq 1$).

where $k\text{-EXPTIME} = \cup_{i \in \mathbb{N}} DTIME(2_k^{n^i})$

Note that we do not use fixpoints in the types above ... but they are used in the proofs.

# What about function classes ?

**Theorem**

We consider the system $ELL_\mu$. The functions representable by proofs of $!W \multimap !^2 W_S$ are exactly the class FPTIME;

recall:

$W$: type of Church binary words

$W_S$: type of Scott binary words

However this characterization is not so satisfactory because of the I/O distinct data-types : these programs cannot be composed!

# An alternative view on function classes

[B.-DeBenedetti-RonchiDellaRocca2017]

Let us define a new data-type:

$$\mathbb{W}_k =_{def} !^k N \otimes !^{k+1} W_S$$

**Theorem**

We consider the system $ELL_\mu$.
For $k \geq 0$, the functions representable by proofs of $\mathbb{W}_1 \multimap \mathbb{W}_{k+1}$ are exactly the class $k$-FEXPTIME.

For FPTIME we have the type $\mathbb{W}_1 \multimap \mathbb{W}_1$, and now these programs can be composed !

# Characterization of complexity classes

the computational engine
logical system

the specification
formula / type

Elementary linear logic $ELL_\mu$

$!W \multimap !^{k+2}B \equiv k\text{-EXPTIME}$

$\mathbb{W}_1 \multimap \mathbb{W}_{k+1} \equiv k\text{-FEXPTIME}$

# Characterization of complexity classes



the computational engine
logical system

the specification
formula / type

Elementary linear logic $ELL_{\mu}$

$!W \multimap !^{k+2}B \equiv k\text{-EXPTIME}$

$\mathbb{W}_1 \multimap \mathbb{W}_{k+1} \equiv k\text{-FEXPTIME}$

# Characterization of complexity classes


the computational engine
logical system

the specification
formula / type

Elementary linear logic $ELL_\mu$

$!W \multimap !^{k+2}B \equiv k\text{-EXPTIME}$

$\mathbb{W}_1 \multimap \mathbb{W}_{k+1} \equiv k\text{-FEXPTIME}$

## Comparison with previous works

- Jones 2001:
  read-only functional programs with arguments of order $\leq k$
  $\equiv$ k-EXP
- Leivant 2002:
  second-order intuitionistic logic with comprehension restricted to order $\leq k$ formulas
  $\equiv$ k-EXP

in these settings: restriction of a particular operation inside the proof or program

by contrast in $EAL_\mu$ the condition is only on the conclusion (type) of the proof.

# Questioning the robustness of ELL

- Can we enrich the language by adding new primitives, and keep the properties?
- We already saw that for type fixpoints
- What about adding an FPTIME primitive $F$, with type $F : W \multimap W$?

# Extending ELL [B.-Ghyselen2018]

**Proposition**

Consider an extension of ELL with a finite number of FPTIME primitives $F_i$ of type $W \multimap W$. Then the functions in $!W \multimap !B$ (resp. $!W \multimap !^2 B$) are in FPTIME (resp. 2-FEXPTIME).

# Improve the expressivity of ELL?

- Denote $nA \multimap B$ for $A \multimap A \multimap \cdots \multimap A \multimap B$, with $n$ occurrences of $A$.
- There are only few functions of type $nW \multimap W$ IN ELL.
- We would like a generic way of adding new primitives of type $nW \multimap W$ to the language.

# Linear sized types

Consider the language $s\ell T$ given by:

- terms: $\lambda$-terms + constructors + iterators
- types:

$$
\begin{array}{llll}
\text{indexes} & I, J & := & a \mid n \in \mathbb{N}^* \mid I + J \mid I \cdot J \\
\text{types} & D, D' & := & N^I \mid W^I \mid D \multimap D' \mid D \otimes D'
\end{array}
$$

- typing rules

# Linear sized types (2)

- Examples of s$\ell$T terms:

$$\begin{array}{lcl} \lambda x.s_0(s_1(x)) & : & W^a \multimap W^{a+2} \\ add = \lambda x.itern(\lambda y.succ(y), x) & : & N^I \multimap N^J \multimap N^{I+J} \end{array}$$

- We have:

**Proposition**

The functions representable by s$\ell$T terms are exactly the class FPTIME.

# Enriched ELL

Consider the language of $\mathrm{ELL}^{+}$ (enriched ELL) defined by:

- $\mathsf{s}\ell\mathsf{T}$ typing rules,
- the rules

$$\frac{\vdash t : W^{a_1}, \ldots, W^{a_n} \multimap W^{l}}{\vdash t : W, \ldots, W \multimap W}$$

- ELL typing rules.

This is a kind of 2-layers language.

# Enriched ELL [B.-Ghyselen2018]

**Theorem**

In ELL$^+$ we have:

- The functions representable by terms of type $!W \multimap !B$ are exactly PTIME.
- For $k \geq 0$ the functions representable by terms of type $!W \multimap !^{k+1}B$ are exactly $2k$-EXPTIME.

# Some terms in ELL$^+$

In ELL$^+$ we can write terms for:

- $SAT : N \multimap W \multimap !B$,
  where a CNF formula is given by the number of distinct variables and the encoding as a word.
- $QBF_k : kN \multimap B \multimap W \multimap !B$,
  for testing satisfiability of quantified boolean formulas with $k$ alternations of quantifiers.
- $SUBSET\_SUM : W \multimap W \multimap !B$,
  where the first word represents an integer and the second one a set of integers.

# Some references


- J.-Y. Girard. Light linear logic. Information and Computation, 143(2):175– 204, 1998.
- V. Danos, J.-B. Joinet. Linear logic and elementary time. Inf. Comput. 183(1): 123-137, 2003.
- P. B. On the expressivity of elementary linear logic: Characterizing ptime and an exponential time hierarchy. Information and Computation, 241:3 -31, 2015.
- P. B., E. De Benedetti, S. Ronchi Della Rocca. Characterizing Polynomial and Exponential Complexity Classes in Elementary Lambda-calculus. To appear in Information and Computation, 2017 (conference version in IFIP TCS 2014).
- P. B., A. Ghyselen. Combining Linear Logic and Size Types for Implicit Complexity. Tech Report HAL, 2018.

# From linear logic to types for implicit computational complexity (Part 3)

Patrick Baillot

CNRS / ENS Lyon

Days in Logic 2018
Aveiro

January 27, 2018

# Introduction: recap

- Implicit computational complexity (ICC) :
  characterizing complexity classes by logics / languages
  without explicit bounds,
  but instead by restricting the constructions
- We are considering here the proofs-as-programs approach for ICC, with linear logic.
- In the 2 first lectures we investigated Elementary linear logic (ELL).

# Characterization of complexity classes


the computational engine
logical system

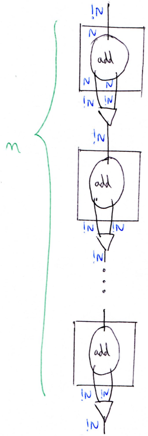
the specification
formula / type

Elementary linear logic $ELL_\mu$

$!W \multimap !^{k+2}B \equiv k\text{-EXPTIME}$

$\mathbb{W}_1 \multimap \mathbb{W}_{k+1} \equiv k\text{-FEXPTIME}$

# Taming the exponential blow-up in ELL?



# Taming the exponential blow-up in ELL?

## Light linear logic (LLL) [Girard95]

- Language of formulas:

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

  intuition: $\S$ a new modality for non-duplicable boxes

- The following principles are still **not** provable

$$!A \multimap A, \quad !A \multimap !!A$$

# Light linear logic rules

- rules (Id), ($\multimap$ i), ($\multimap$ e), (Cntr), (Weak): as in ELL.
- new rules (! i), (! e), (§ i), (§ e):

$$\frac{x : B \vdash t : A}{x :!B \vdash t :!A}\ (!\ \mathrm{i}) \qquad \frac{\Gamma_1 \vdash u :!A \quad \Gamma_2, x :!A \vdash t : B}{\Gamma_1, \Gamma_2 \vdash t[u/x] : B}\ (!\ \mathrm{e})$$

$$\frac{\Gamma, \Delta \vdash t : A}{!\Gamma, \S\Delta \vdash t : \S A}\ (\S\ \mathrm{i}) \qquad \frac{\Gamma_1 \vdash u : \S A \quad \Gamma_2, x : \S A \vdash t : B}{\Gamma_1, \Gamma_2 \vdash t[u/x] : B}\ (\S\ \mathrm{e})$$

at most one free variable in the premise judgement of (! i) rule.

# Light linear logic principles

- The following formulas are provable:

$$!A \multimap \S A \qquad \S A \otimes \S B \multimap \S(A \otimes B)$$

- The following one is **not** provable in LLL, though it is in ELL:

$$!A \otimes !B \multimap !(A \otimes B)$$

# Forgetful map from LLL to ELL

Consider $(.)^e : LLL \to ELL$ defined by:

$$(\S A)^e = !A^e, \quad (!A)^e = !A^e$$

and other connectives unchanged.

**Proposition**

If $\Gamma \vdash_{LLL} t : A$ then $\Gamma^e \vdash_{ELL} t : A^e$.

# Data types in LLL

- Church unary integers

| system F: | LLL: |
|---|---|
| $N^F$ | $N^{LLL}$ |
| $\forall\alpha.(\alpha \to \alpha) \to (\alpha \to \alpha)$ | $\forall\alpha.!(\alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha)$ |

Example: integer 2, in F:

$$\underline{2} = \lambda f^{(\alpha\to\alpha)}.\lambda x^{\alpha}.(f\ (f\ x))\ .$$

- Church binary words

| system F: | LLL: |
|---|---|
| $W^F$ | $W^{LLL}$ |
| $\forall\alpha.(\alpha \to \alpha) \to (\alpha \to \alpha) \to (\alpha \to \alpha)$ | $\forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha)$ |

Example: $w = \langle 1, 0, 0\rangle$, in F:

$$\underline{w} = \lambda s_0^{(\alpha\to\alpha)}.\lambda s_1^{(\alpha\to\alpha)}.\lambda x^{\alpha}.(s_1\ (s_0\ (s_0\ x)))\ .$$

# Representation of functions

- a term $t$ of type $!^k N \multimap \S^l N$, for some $k$, $l$, represents a function over unary integers
  $!^k W \multimap \S^l W$: function over binary words.
- some examples of terms

addition

$$add = \lambda nmfx.(n\ f)\ (m\ f\ x) \quad : \quad N \multimap N \multimap N$$

double

$$double = \lambda nfx.(n\ f)\ (n\ f\ x) \quad : \quad !N \multimap \S N$$

concatenation

$$conc \quad : \quad W \multimap W \multimap W$$

# Iteration in LLL

we can type the iterator *iter*:

$$iter = \lambda fxn.\,(n\ f\ x)\ :\ !(A \multimap A) \multimap {\color{red}!A} \multimap N \multimap {\color{red}\S A}$$

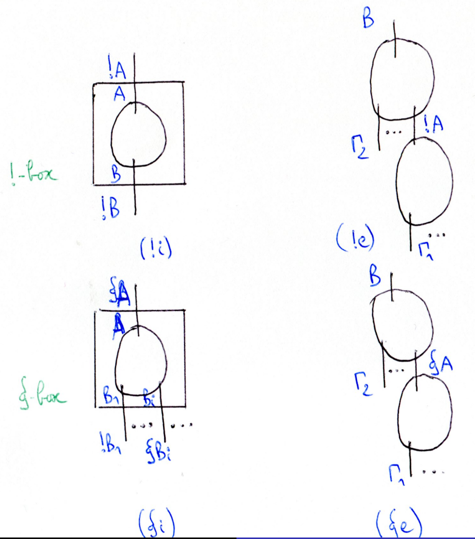
**examples:**

$(add\underline{3}) : N \multimap N$ can be iterated

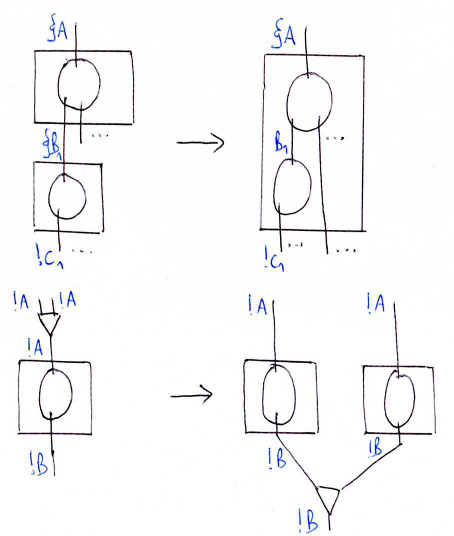
$double :!N \multimap \S N$ cannot be iterated

thus some exponentially growing terms are not typable

# LLL proof-nets



# LLL proof-net reduction



# Level-by-level reduction of LLL proof-nets

- as in ELL we use a level-by-level strategy
- let $R$ be an LLL proof-net of depth $d$
  round $i$: reduction at depth $i$
  there are $d+1$ rounds for the reduction of $R$
- **what happens during round $i$?**
  - $|R|_i$ decreases at each step
    thus there are at most $|R|_i$ steps (size bounds time)
  - yet $|R|_{i+1}$ can increase:
    during round $i$ we can have a quadratic increase:

$$|R'|_{i+1} \leq |R|_{i+1}^2$$

- this repeats $d$ times, so on the whole we have a $|R|^{2^d}$ size increase
- this yields a $O(|R|^{2^d})$ bound on the number of steps

# LLL complexity results

**Theorem (Proof-net complexity)**

If $R$ is an LLL proof-net of depth $d$, then it can be reduced to its normal form in $O(|R|^{2^d})$ steps.

Thus at fixed depth $d$ we have a polynomial bound.

**Theorem (Representable functions)**

The functions representable by a term of type $W \multimap \S^k W$, for $k \geq 0$, are exactly the functions of FP (polynomial time functions).

# Characterization of complexity classes

the computational engine
logical system

the specification
formula / type

Light linear logic *LLL*

$\{W \multimap \S^k W\}_{k\geq 0} \equiv \mathrm{FPTIME}$

# Light linear logic and typing

Is LLL a good type system for lambda calculus ...?

Actually there are two problems:

- it does not satisfy subject-reduction,
- it does not ensure polynomial time complexity for $\beta$-reduction ...

# Light linear logic and typing (2)

Example:

$$y :!(!A \multimap !A \multimap !A), z :!!A \quad \vdash_{LLL} \quad (\lambda x.yxx)^n z : \S !A$$

$t_n = (\lambda x.yxx)^n z$,
$t_n \xrightarrow{\beta} u_n$ with
$u_0 = z, \qquad u_n = y\ u_{n-1}\ u_{n-1}.$
we have: $|t_n| \sim c.n, \quad |u_n| \sim 2^n.$

hence: any beta-reduction of $t_n$ to $u_n$ costs exponential space on a Turing Machine !

even though: using proof-nets these reductions are done in polynomial time.
culprit : sharing allowed by !,
it entails that: for $\mathcal{D}$ type derivation for $t$, we might have $|t| \gg |\mathcal{D}|$.

# How to fix this problem?

# Type system DLAL

To overcome the problems with typing in LLL:
we restrict the use of ! to $!A \multimap B$.

The DLAL (Dual Light Affine Logic) type system:

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

Typing judgements of the form: $\Gamma; \Delta \vdash t : A$, where
$\Gamma$ contains duplicable variables,
$\Delta$ contains linear variables.

# DLAL: complexity bounds

DLAL satisfies the subject-reduction property.

**Theorem (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 $O((d+1)\cdot|t|^{2^{d+1}})$.

Remarks:

- one in fact shows a bound $O((d+1)\cdot|t|^{2^d})$ on the number of $\beta$-steps and then uses the fact that the cost of each step is here bounded;
- this bound holds for any reduction strategy;
- in particular, if $\vdash t : W \multimap \S^k W$ then $t$ is Ptime.

# DLAL: PTIME extensional completeness

**Theorem (Completeness)**

For any polynomial time function $f : \{0,1\}^* \to \{0,1\}^*$, there exists a term $t$ representing it and typable in DLAL with a type $W \multimap \S^k W$, for a certain integer $k \in \mathbb{N}$.

## Can we check DLAL typability?

DLAL type inference problem for system F terms:

| | |
|---|---|
| **input:** | system F term $t$ |
| **problem:** | does there exist a DLAL derivation for $t$ ? |

main issue:

- decorate the F derivation with ! / §
- for that, find out where to put boxes

. . . boils down to constructing a proof-net.

# Lambda term to proof-net: the difficulties

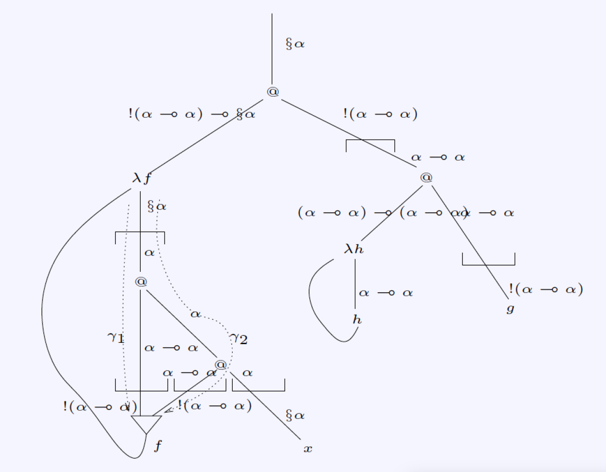
How can we find out the boxes needed ?

At first sight there are several difficulties

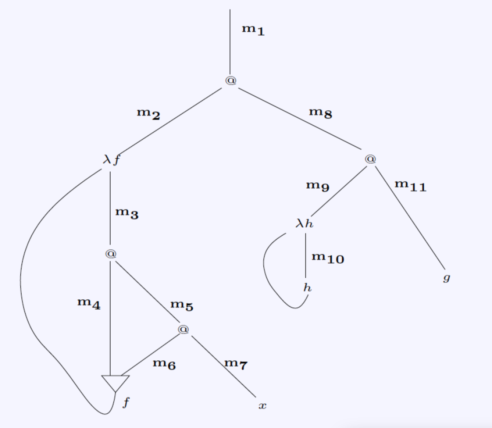
1. no a priori bound on the number of boxes needed
2. even for box positions there is an exponential number of possibilities
3. furthermore: distinguish between ! and § boxes

Idea: we search for doors instead of boxes.

# Example: term with doors



# Example: parameterized term




$\S^{\mathbf{m_1}}(\S^{\mathbf{b_2},\mathbf{m_2}}\alpha \rightarrow \S^{\mathbf{m_3}}\alpha)$, with boolean parameters $\mathbf{b_2}$ and integer parameters $\mathbf{m_1}$.

# Type inference

- We express typability by a set of constraints on parameters, expressing e.g. : boxes are well-formed, a !-box has at most one auxilliary door etc
- We get mixed boolean-linear constraints.
- We give a resolution procedure for deciding whether the constraints system is decidable, using linear programming.
- This resolution procedure is PTIME.

# DLAL type inference

Finally we get:

**Theorem**

The DLAL type inference problem for system F terms can be decided in PTIME.

## About the expressivity of LLL and DLAL

The completeness result is an extensional one,
but the intensional expressivity of LLL and DLAL is limited.

Indeed: rich features (higher-order, polymorphism) but "pessimistic" account of iteration ...

## Some references


- J.-Y. Girard. Light linear logic. Information and Computation, 143(2):175– 204, 1998.
- Andrea Asperti, Luca Roversi. Intuitionistic Light Affine Logic. ACM Trans. Comput. Log. 3(1): 137-175 (2002).
- P. B., Kazushige Terui. Light types for polynomial time computation in lambda calculus. Inf. Comput. 207(1): 41-62 (2009)
- Vincent Atassi, P. B., Kazushige Terui. Verification of Ptime Reducibility for system F Terms: Type Inference in Dual Light Affine Logic. Logical Methods in Computer Science 3(4) (2007)
- Ugo Dal Lago. Context semantics, linear logic, and computational complexity. ACM Trans. Comput. Log. 10(4): 25:1-25:32 (2009)

# A glimpse of the linear logics zoo

- for FPTIME
  - soft linear logic: [Lafont04]
    a simple system, but with more constrained programming
  - bounded linear logic: [GSS92]
    $!_{P(\vec{x})} A$ : more explicit, but more flexible
- for PSPACE
  - $STA_B$ [GMRdR08] : extends soft linear logic with a craftly typed conditional
- for LOGSPACE
  - $IntML$ [DLS10]: evaluation by computation by interaction
- for P/poly (non-uniform computation):
  - parsimonious $\lambda$-calculus [MazzaTerui15]

# Conclusions and perspectives

- linear logic can be used for implicit complexity
- with two ingredients:

  choice of the logic

  choice of the formulas/types 

- these systems lead to type systems for $\lambda$-calculus, ensuring complexity properties
- w.r.t. other ICC approaches:
  - handle higher-order computation
  - but limited intensional expressivity

relations with other ICC systems still to explore