An introduction to light logics, or Implicit complexity by taming the duplication

Patrick Baillot

CNRS / ENS Lyon

Shonan meeting on ICC and applications

November 6, 2013

Introduction

- Implicit computational complexity (ICC) :
  characterizing complexity classes by programming languages / calculi without explicit bounds,
  but instead by restricting the constructions
- either theory-oriented or certification-oriented
- often conveniently formulated by:
  (i) a general programming language, (ii) a criterion on programs

Various approaches to ICC

- ramified recursion (Leivant, Leivant-Marion) / safe recursion (Bellantoni-Cook)
- variants of linear logic (light logics) **this talk**
- interpretation methods
- ...

ICC vs. complexity analysis

specificities of ICC w.r.t. automatic complexity analysis:

- complexity certificate (e.g. type)
- modular

but

- only rough complexity bounds
- less general analysis (specific programming discipline)

The proofs-as-programs viewpoint

- our reference language here is λ-calculus
  untyped λ-calculus is Turing-complete
- type systems can guarantee termination
  ex: system F (polymorphic types)
- proofs-as-programs correspondence

proof	$=$	type derivation
normalization	$=$	execution
intuitionistic logic	$\leftrightarrow$	system F

- some characteristics of λ-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)
- variants of linear logic with different rules for ! have bounded complexity: light logics
  these logics (or subsystems) can be used as type systems for λ-calculus
  thus:
  (i) general language= λ-calculus, (ii) criterion= typability

Outline of the talk

1. a recap on λ-calculus and system F
2. elementary linear logic (ELL): elementary complexity
3. light linear logic (LLL): Ptime complexity
4. other linear logic variants
5. conclusion

λ-calculus

- λ-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: $t[u/x]$

- β-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 λ-terms

system F types:

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

simple types: without ∀

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-programs correspondence (Curry-Howard)

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

Some types and data types

Polymorphic identity:

$$\lambda x^{\alpha}.x \quad : \quad \forall\alpha.(\alpha \to \alpha)$$

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

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

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

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

Examples of terms

concatenation

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

length

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

repeated concatenation

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

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 $t : W \to W$ 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

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

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

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

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

Forgetful map from ELL to F

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

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

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
- some examples of terms

$$
\begin{array}{lcl}
\text{addition} & & \\
add & = & \lambda nmfx.(n\ f)\ (m\ f\ x) \\
 & : & N \multimap N \multimap N \\
 & & \\
\text{multiplication} & & \\
mult & = & \lambda nmf.(n\ (m\ f)) \\
 & : & N \multimap N \multimap N \\
\text{squaring} & & \\
square & = & \lambda nf.(n\ (n\ f)) \\
 & : & !N \multimap !N
\end{array}
$$

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:

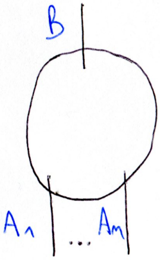
$double : N \multimap N$

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

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

From derivations to proof-nets

$$\frac{\Pi}{x_1 : A_1, \dots, x_n : A_n \vdash t : B} \rightsquigarrow \Pi^* =$$



derivation

proof-net

Elementary linear logic rules, again

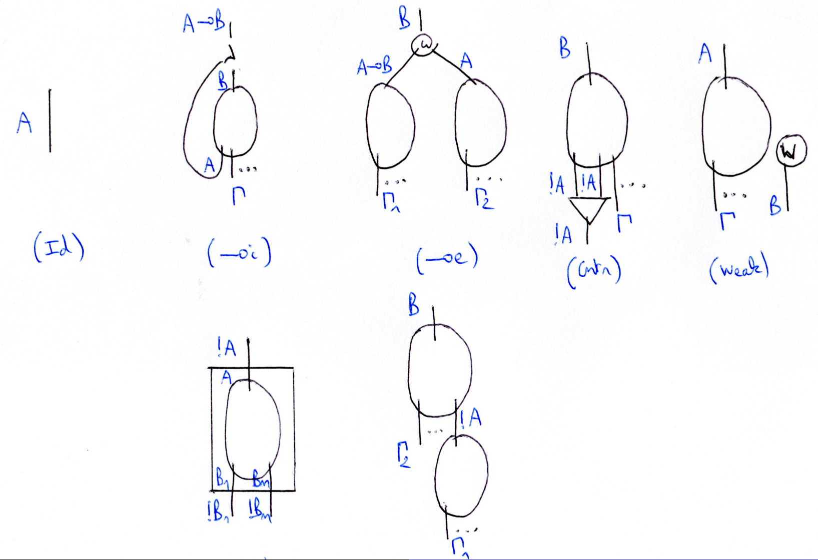
$$\frac{}{x : A \vdash x : A}\ (\mathrm{Id})$$

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

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

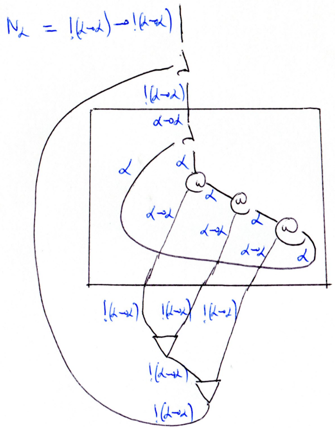
$$\frac{x_1 : B_1, \ldots, x_n : B_n \vdash t : A}{x_1 :!B_1, \ldots, x_n :!B_n \vdash t :!A}\ (!\ \mathrm{i}) \qquad \frac{\Gamma_1 \vdash u :!A \quad \Gamma_2, x :!A \vdash t : B}{\Gamma_1, \Gamma_2 \vdash t[u/x] : B}\ (!\ \mathrm{e})$$

ELL Proof-Nets

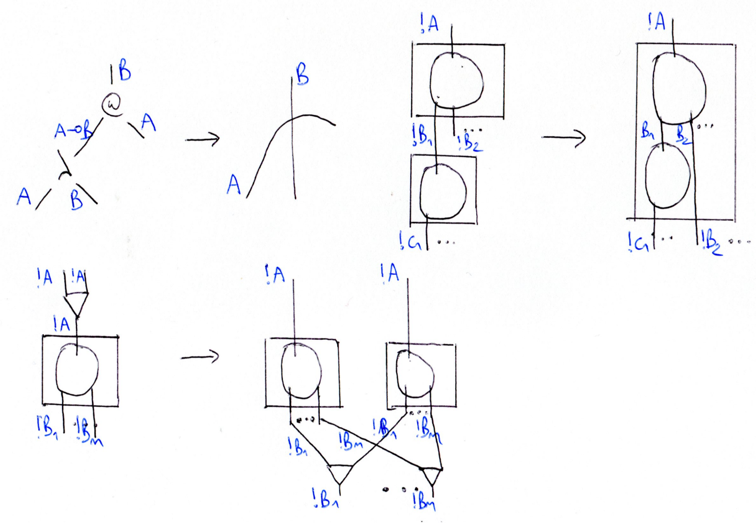


ELL proof-net : example

Church integer $\underline{3}$:



ELL proof-net reduction



Methodology

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

  but that's all right for our present goal ...
  More about that in tomorrow's talk, without proof-nets.

ELL proof-net reduction properties

- 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
  $|R|_i$ = 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.

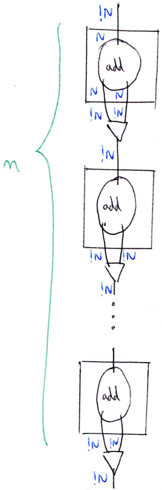
Proof of the representability theorem

- $\subseteq$ (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 $O(2_i^n)$-time bounded Turing machine, for any i.

Taming the exponential blow-up?



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):

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

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

where if $\Gamma = x_1 : B_1, \ldots, x_k : B_k$,
$\dagger\Gamma = x_1 : \dagger B_1, \ldots, x_k :: \dagger B_k$, for $\dagger =!, \S$.

Forgetful map from LLL to ELL

Consider $(.)^e : LLL \rightarrow ELL$ 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

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

double

$$\begin{array}{lcl} double & = & \lambda nfx.(n\ f)\ (n\ f\ x) \\ & : & !N \multimap \S N \end{array}$$

concatenation

$$\begin{array}{lcl} conc & : & W \multimap W \multimap W \end{array}$$

Iteration in LLL

we can type the iterator *iter*:

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

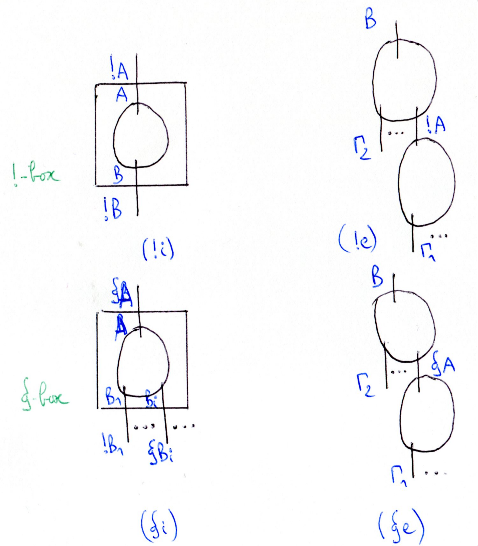
examples:

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

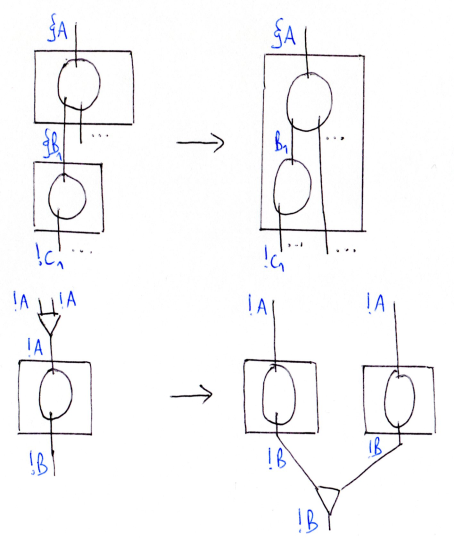
$double\ :!N \multimap \S N$ cannot be iterated

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).

Further comments about LLL

- **LLL and λ-calculus**:
  a proper type system for λ-calculus can be designed out of LLL, which ensures a strong polynomial time bound on β-reduction (and not only on proof-net reduction)
- **about expressivity**:
  the completeness result is an extensional one
  but the intensional expressivity of LLL is quite limited

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

A glimpse of a linear logics zoo

- for P
  - 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 EXPTIME and k-EXPTIME
  - ELL again: see tomorrow's talk
- for PSPACE
  - STA_B [GMRdR08] : extends soft linear logic with a craftly typed conditional
- for LOGSPACE
  - *IntML* [DLS10]: evaluation by computation by interaction

Conclusions and perspectives

- while ramified recursion is based on a stratification of data, ELL / LLL are based on a stratification of programs
- they yield type systems for λ-calculus
- w.r.t. other ICC approaches:
  - handle higher-order computation
  - but limited intensional expressivity

  relations with other ICC systems are still to explore
- light logics are languages for higher-order computation, but we only characterize first-order complexity classes . . . what about higher-order complexity?