Enhancing Elementary Affine Logic type inference with implicit cœrcions

Vincent Atassi — LIPN, Univ. Paris 13

TYPES Workshop, April 2006

Time bounded logics

- Origins: Subsystems of linear logic (Girard 98)
- Complexity properties are *Intrinsic* to the system (They don't rely on an analysis independent of type system)
- Complexity bounds are *implicit* (no bound is demanded to the user)
- It provides a *logical* characterisation of complexity classes

$\Rightarrow$ We use (polymorphic) multiplicative fragment as a type system for λ-calculus

$\Rightarrow$ Performs higher-order polymorphic langage complexity analysis

From Linear Logic (LL) to bounded time logics

$$\frac{}{A \vdash A}\,(\text{AX}) \qquad \frac{\Gamma \vdash A \qquad A, \Delta \vdash B}{\Gamma, \Delta \vdash B}\,(\text{CUT})$$

$$\frac{\Gamma \vdash A \qquad B, \Delta \vdash C}{\Gamma, A \multimap B \vdash C}\,(\multimap\text{-l}) \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B}\,(\multimap\text{-r})$$

Figure: Core MLL

$\Rightarrow$ Usual rules of intuitionnistic logic. Except that there is no contraction: cut-elimination is in a linear number of steps.
$\Rightarrow$ Types *linear* lambda-calculus

From Linear Logic (LL) to bounded time logics (2)

$$\frac{}{A \vdash A}\,(\text{AX}) \qquad \frac{\Gamma \vdash A \qquad A, \Delta \vdash B}{\Gamma, \Delta \vdash B}\,(\text{CUT})$$

$$\frac{\Gamma \vdash A \qquad B, \Delta \vdash C}{\Gamma A \multimap B \vdash C}\,(\multimap\text{-l}) \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B}\,(\multimap\text{-r})$$

$$\frac{A, \Gamma \vdash B}{!A, \Gamma \vdash B}\,\epsilon \qquad \frac{!!A, \Gamma \vdash B}{!A, \Gamma \vdash B}\,\delta \qquad \frac{\Gamma \vdash B}{!\Gamma \vdash !B}\,!$$

$$\frac{!A, !A, \Gamma \vdash B}{!A, \Gamma \vdash B}\,\text{CONTR} \qquad \frac{\Gamma \vdash B}{!A, \Gamma \vdash B}\,\text{WEAK}$$

Figure: MLL + exponentials

$\Rightarrow$ Contraction, controled by modality "!", allows to recover full expressive power of intuitionnistic logic: cut-elimination is non-elementary

$\Rightarrow$ Variants of the rules on modalities will yield intermediary complexity classes.

Restriction of modality introduction

- Full Linear Logic:
$$\frac{A, \Gamma \vdash B}{!A, \Gamma \vdash B}\,\epsilon \quad , \quad \frac{!!A, \Gamma \vdash B}{!A, \Gamma \vdash B}\,\delta \quad , \quad \frac{\Gamma \vdash B}{!\Gamma \vdash !B}\,!$$
- Elementary Linear Logic: only $\dfrac{\Gamma \vdash B}{!\Gamma \vdash !B}\,!$
- Light Linear Logic (polynomial time): $\dfrac{A \vdash B}{!A \vdash !B}\,!$,
$\dfrac{\vdash B}{\vdash !B}\,!$ ' and $\dfrac{\Gamma, \Delta \vdash B}{\S\Gamma, !\Delta \vdash !B}\,\S$ (additionnal modality)

$\Rightarrow$ Boxes formed this way have a stratification property which leads to time bounded normalization

$\Rightarrow$ this stratification also allows for the application of the abstract part of Lamping's algorithm for Optimal Reduction

Some results

- ELL (resp. LLL) proof-nets reduce in elementary (resp. polynomial) time
- One can encode a Turing machine in LLL if a polynomial bound is given
- Elementary functions can be encoded in ELL
- It is undecidable whether a λ-term will reduce in polynomial time
- It is undecidable wether an F typed λ-term will reduce in polynomial time

Related works

There is already an abundant litterature in this field:

- Base works:
  - Girard, Inf. & Comput. 98: ELL and LLL
  - Asperti, LICS 98; Asperti-Roversi 02: Affine variants
- Type inference:
  - Coppola-Martini, 2001: first type inference for EAL
  - Baillot, TCS 04: type inference for propositionnal LAL
  - Baillot-Terui, TLCA 05: *efficient* type inference for EAL
  - Submitted: Atassi-Baillot-Terui 06: *efficient* type inference for a polymorphic LAL variant (system F type decoration)

Outline of the talk

We will focus on type inference for Elementary Affine Logic, a variant of ELL.
Complexity analysis performed isn't that useful, but the system is simpler and presented method scales to a polynomial logic.

- Presentation of EAL type system
- Type inference algorithm
- Extension of type system with subtyping to handle cœrcions

EAL type system

Natural deduction presentation with associated proof term (affine variant: unrestricted weakening):

$$\frac{}{x : A \vdash x : A}\ (\textsc{var}) \qquad \frac{\Gamma \vdash M : B}{\Gamma, x : A \vdash M : B}\ (\textsc{weak})$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \multimap B}\ (\textsc{abs}) \qquad \frac{\Gamma_1 \vdash M_1 : A \multimap B \qquad \Gamma_2 \vdash M_2 : A}{\Gamma_1, \Gamma_2 \vdash (M_1)\ M_2 : B}\ (\textsc{appl})$$

$$\frac{\Gamma_1 \vdash t_1 : !A_1 \ldots \Gamma_n \vdash t_n : !A_n \qquad x_1 : A_1, \ldots, x_n : A_n \vdash M : B}{\Gamma_1, \ldots, \Gamma_n \vdash M\{t_i/x_i\} : !B}\ (\textsc{prom})$$

$$\frac{x_1 : !A, ..., x_n !A, \Gamma \vdash M : B}{!x : A, \Gamma \vdash M\{x/x_1, ..., x/x_n\} : B}\ \textsc{contr}$$

Figure: EAL type system

$\Rightarrow$ Slight restriction of previous Sequent calculus presentation

$\Rightarrow$ (prom) and (contr) rule are not syntax directed — it is no big problem for (contr) but is for (prom)

PROM rule applications are boxes

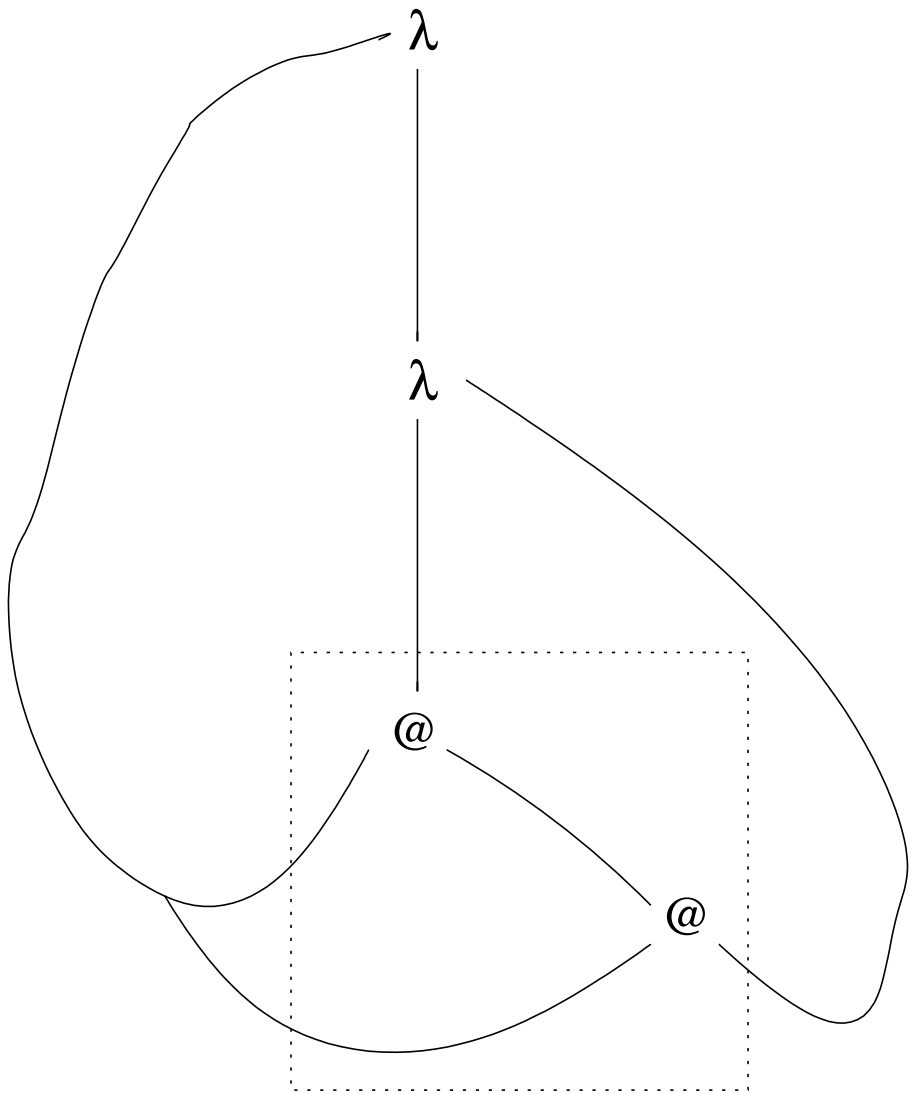

Figure: Box placement for Church integer 2, resulting type is $!(A \multimap A) \multimap !A \multimap !A$

PROM rule applications are boxes

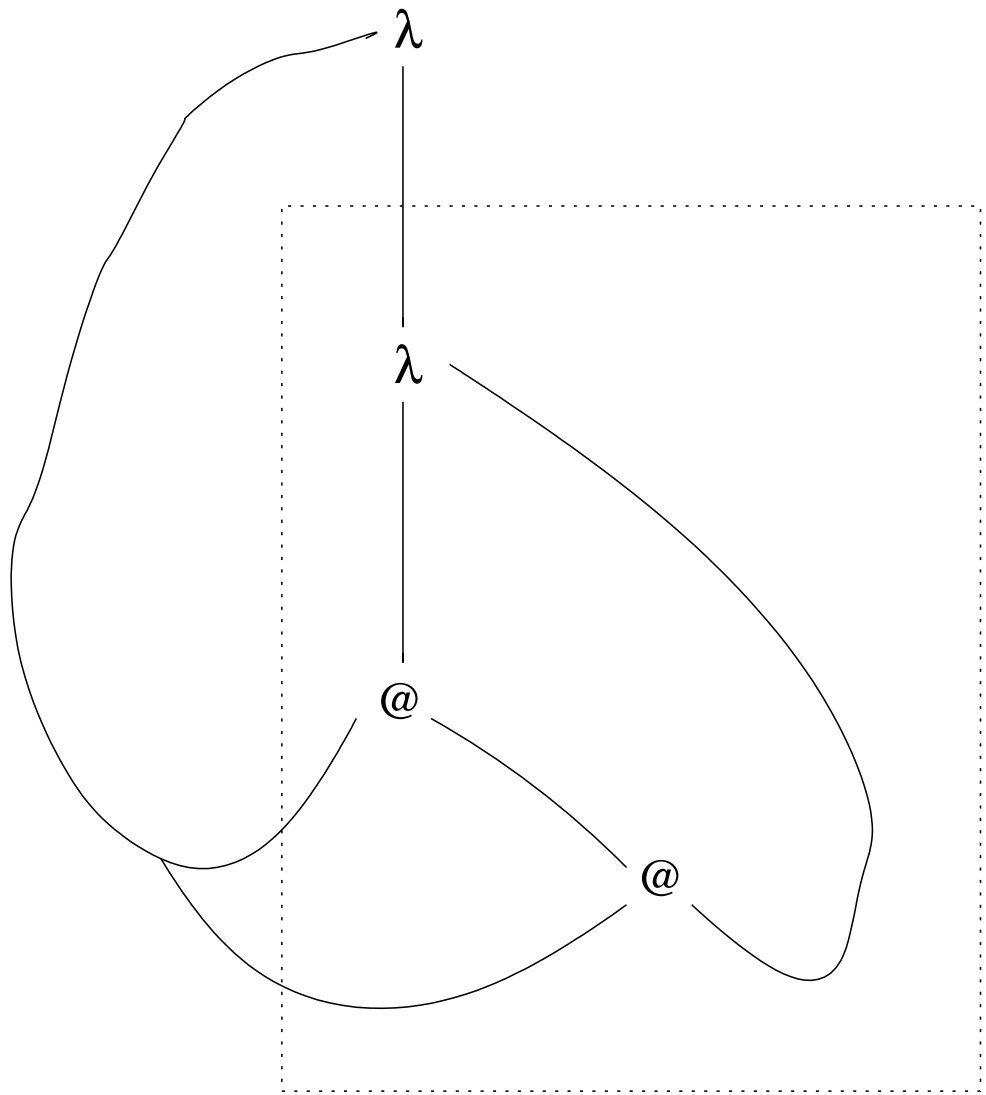

Figure: Box placement (2) for Church integer 2, resulting type is $!(A \multimap A) \multimap !(A \multimap A)$

PROM rule applications are boxes

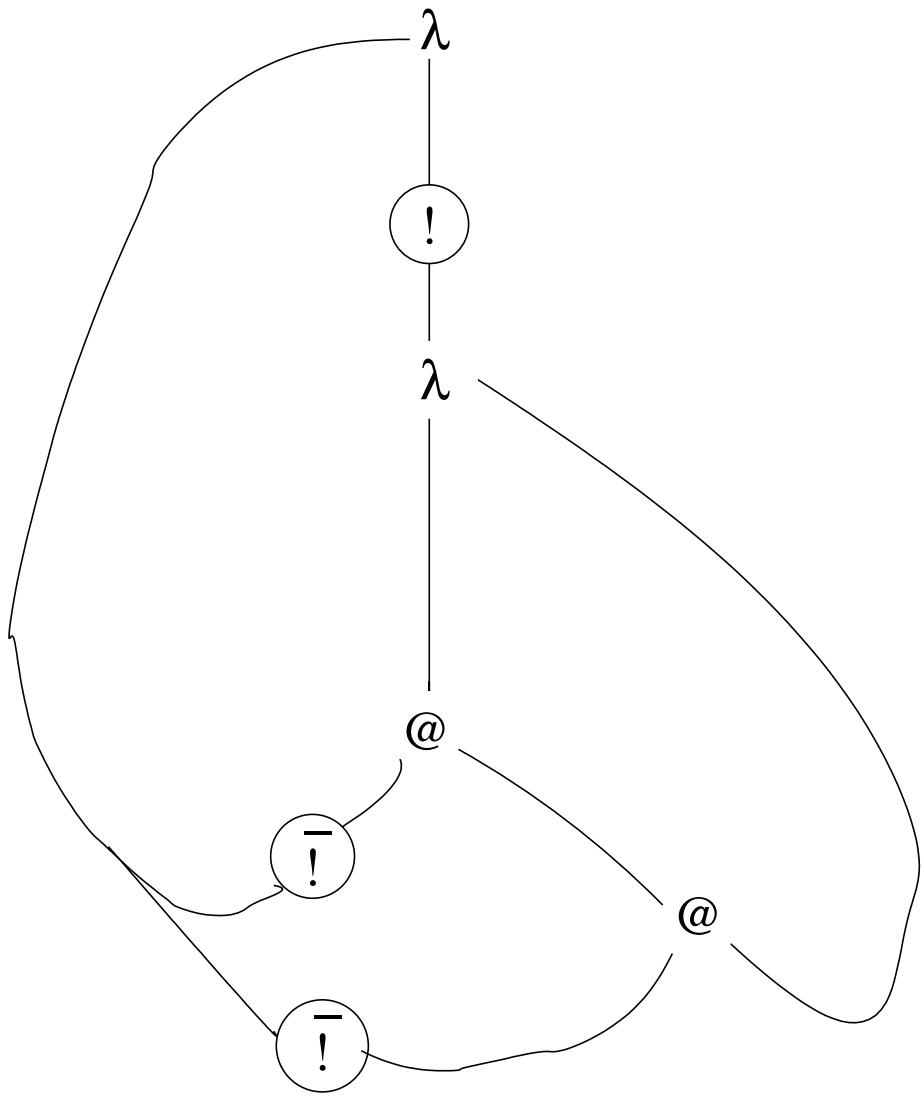

Figure: Box placement for Church integer 2, resulting type is $!(A \multimap A) \multimap !(A \multimap A)$

Typability rephrased

Constraints on modalities and box placement:

bracketing Boxes must be well-formed:

- Each auxiliary door ("$\bar{!}$") matches a main door
- Any variable is at the same level as its binder

typing Modalities consistency:

- Each applied function must be an arrow type (ie. $A \multimap B$ and not $!(A \multimap B)$)
- Unification is extended to banged types : $((t : (!^n A \multimap B))t_2 : (!^m A) \Rightarrow m = n)$

contraction Any contracted variable must have at least one "!"

$\Rightarrow$ A pseudo-term is EAL-typable iff it has a simple type and conforms to those conditions

$\Rightarrow$ A λ-term t is EAL-typable iff there exists a pseudo term p conforming to those conditions s.t. $(p)^- = t$

Typability to type inference

Determining wether a λ-term is EAL-typable:

- Simple type inference
- Term and types free decoration: fresh parameters at all term nodes, fresh parameters for each subtype of occuring types
- Term and type derivation explorations for *linear contraints* generation on those parameters, which will correspond to the previous criterions
- Solvability of the generated constraints $\Rightarrow$ typability, and a solution to the constraints yields a type assignation

Example

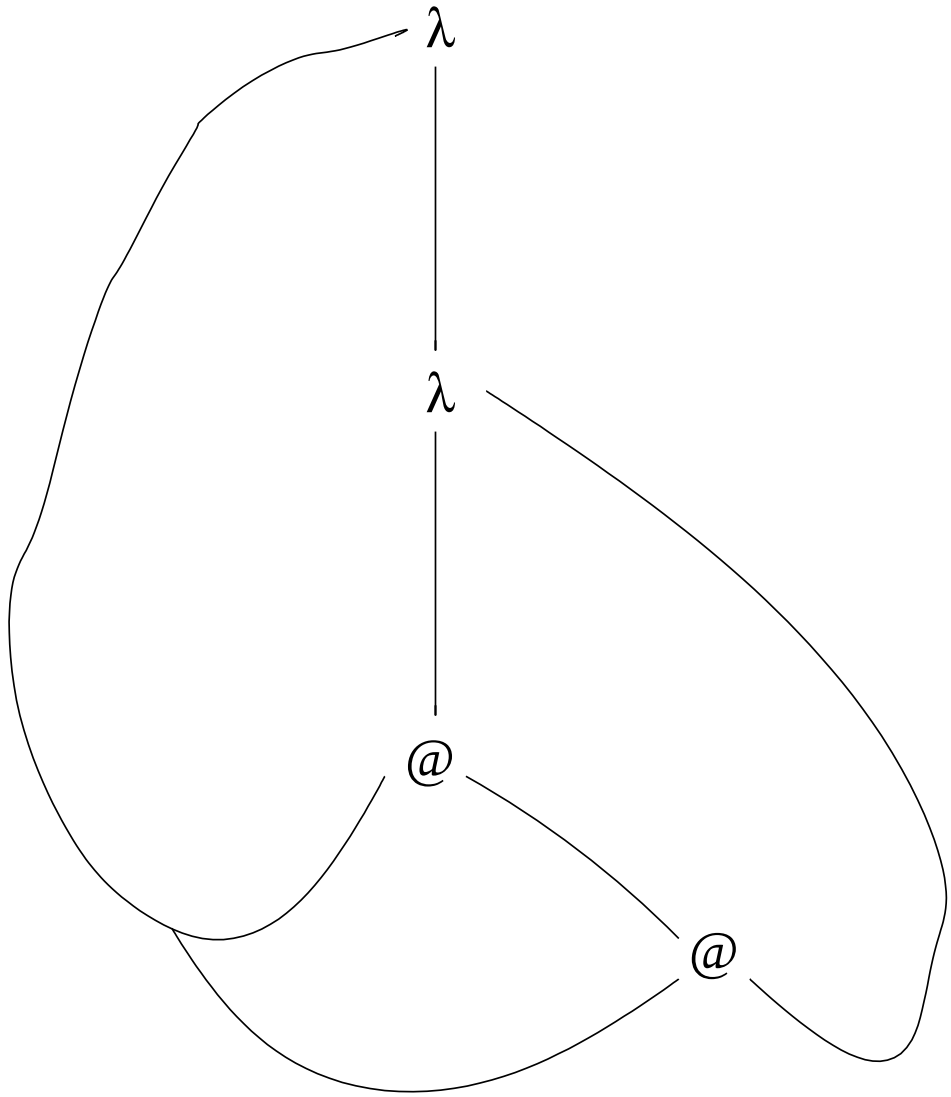

Figure: Abtract syntax tree for Church integer 2

Example

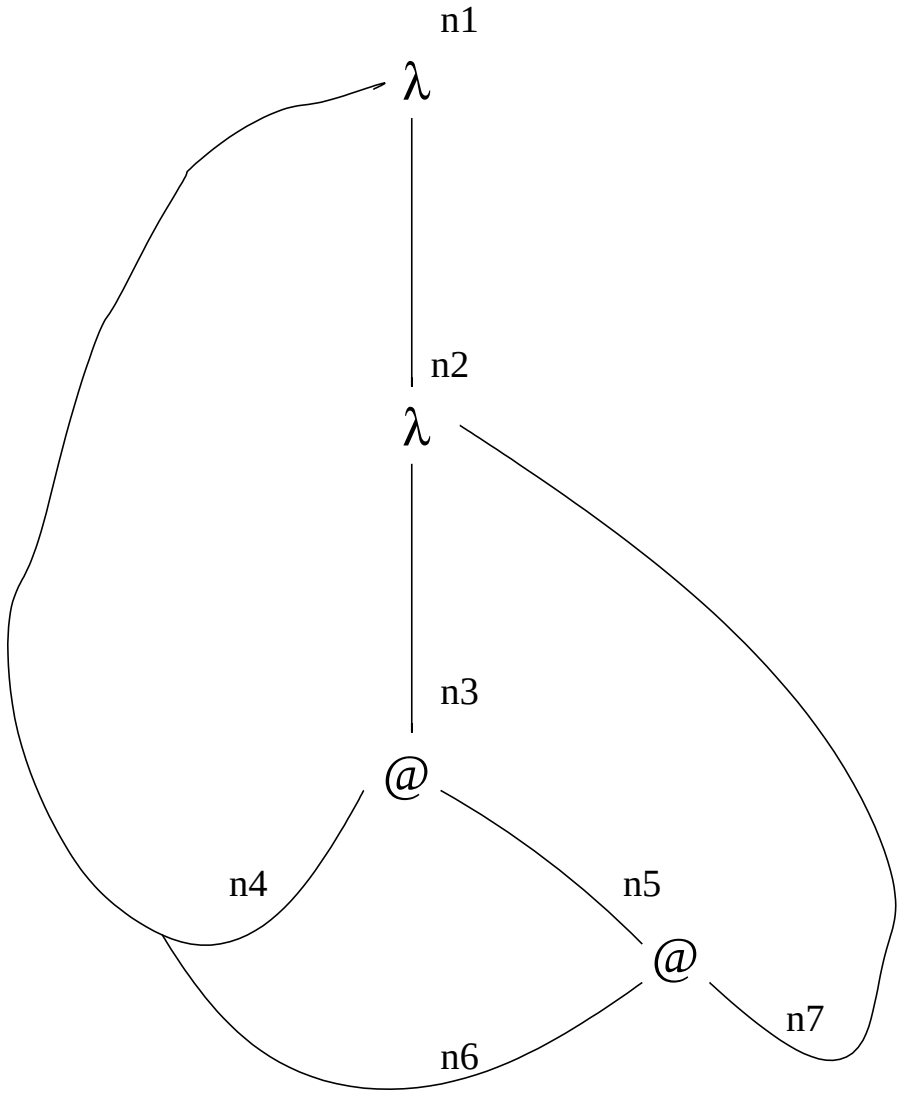

Figure: Parametrized abtract syntax tree for Church integer 2

Example

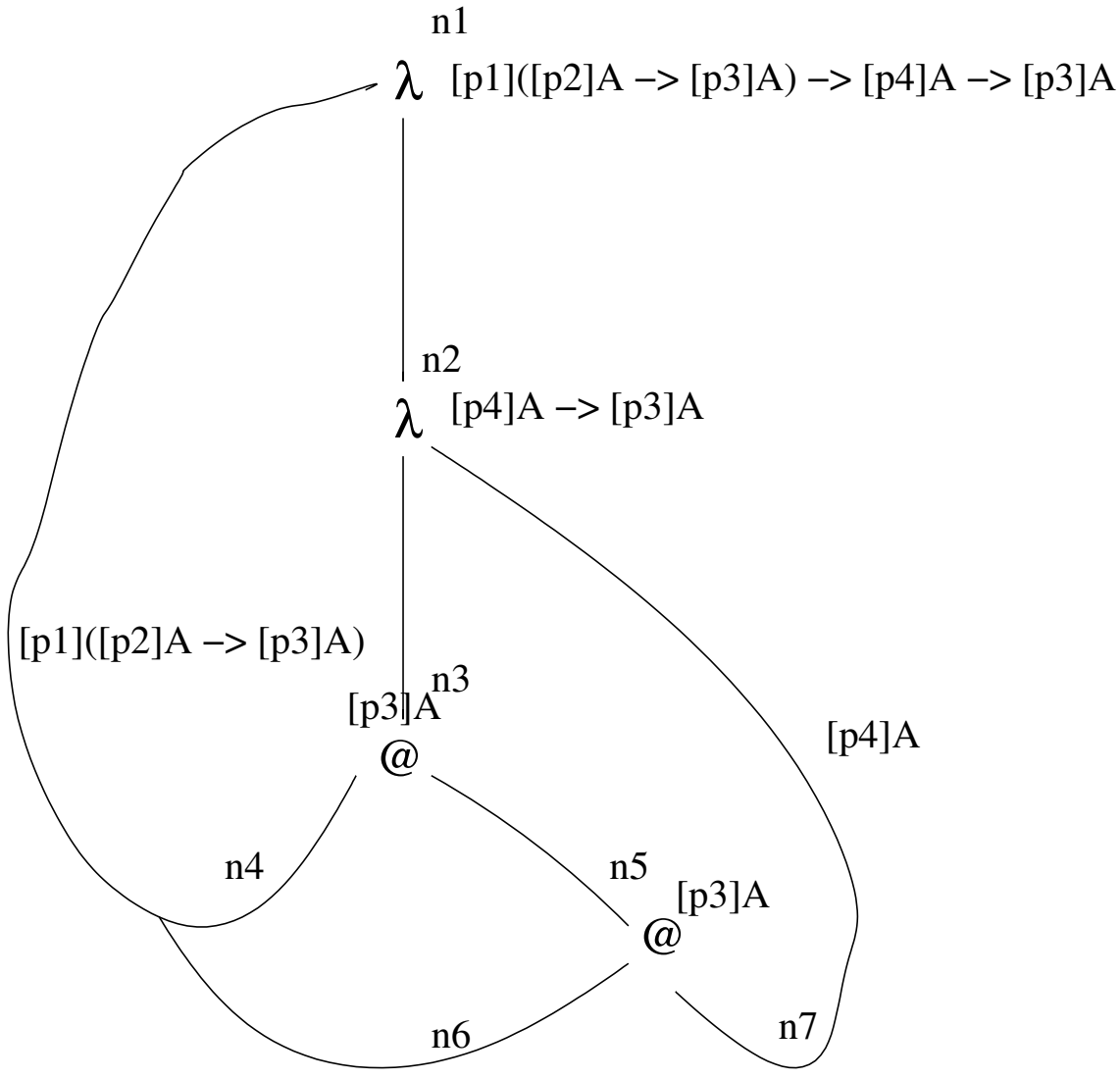

Figure: Simply typed parametrized abtract syntax tree

Example (2)

- Bracketing constraints:

$$n1 \geq 0 \quad n2 + n3 + n5 \geq 0$$
$$n2 + n3 + n5 + n6 = 0 \quad n3 + n5 + n7 = 0$$
$$n3 + n5 \geq 0 \quad n3 \geq 0$$
$$n2 + n3 + n4 = 0 \quad n2 + n3 \geq 0$$
$$n2 \geq 0$$

- Contraction constraints: $p1 \geq 1$
- Type consistency constraints:

$$p1 + n6 = 0 \quad p1 + n4 = 0$$
$$p2 = p3 + n5 \quad p2 = p4 + n7$$

- Plus some bookkeeping ones..

Type inference algorihtm

- Runs in polynomial time w.r.t. typed term size
- Has been implemented (in CAML) except for the resolution part (external LP solver)
- Gives an elementary bound: that is to say the height of an exponential tower $2^{2^{\cdot^{\cdot^{\cdot^{2^n}}}}}$
- But scales well to a polynomial logic and polymorphic types
- And characterises a large set of terms reducible by Lamping's abstract algorithm

ELL and cœrcions

$\Rightarrow$ In our framework a *cœrcion* will be any function of type $A \multimap !A$, with A a datatype.

Example

In EAL the square function $\lambda n.(mult)\ n\ n$ is *not* typable, only $\lambda n.(mult)\ n\ (coerc_N)\ n$ is.

$\Rightarrow$ One faces the same difficulty in LLL or LAL.
$\Rightarrow$ Now, in the context of type inference, we automatized boxes placement. A further step is to automatize cœrcions placement.
$\Rightarrow$ The same way we enriched the input terms with boxes, we will enrich them with cœrcive subterms.

Typing with cœrcions

We have terms such that $x : B \vdash t : !B$ is derivable, for any base type (integers, binary words, booleans).
Placing cœrcions is adding *cuts* in proofs: sounds problematic for type inference.
But those cuts have *no computational value*: their only effect is to add modalities on positive sides of arrows, and to remove some on the negative side.
Hence the idea of a subtyping relation on types for expressing cœrcions: $N \leq !N$ and $!N \multimap N \leq N \multimap !N$.

A subtyping relation for cœrcions

We define first the subtyping relation for base types which admit cœrcions.

Definition

$\preceq$ *is defined by:*

$$\forall n, m \text{ st } 0 \leq n \leq m, !^n N \preceq !^m N \text{ and } !^n B \preceq !^m B$$

Proposition

$\preceq$ *is reflexive and transitive.*

Lemma (Correction of $\preceq$)

If $A \preceq B$, *then* $\exists t$ *such that* $x : A \vdash t : B$ *is derivable and* $t \longrightarrow_\beta^* x$.

Subtyping relation

$$(\textsc{base})\ \frac{}{A \leq B}\ A \preceq B \qquad (\textsc{var})\ \frac{}{\alpha \leq \alpha}$$

$$(\textsc{prom})\ \frac{A \leq B}{!A \leq !B} \qquad (\textsc{arrow})\ \frac{A_1 \geq A_2 \qquad B_1 \leq B_2}{A_1 \multimap B_1 \leq A_2 \multimap B_2}$$

Figure: Subtyping system.

Lemma

$\leq$ *is reflexive and transitive.*

Proposition ($\leq$'s correction)

If $A \leq B$*, then exists* t *such that* $x : A \vdash t : B$ *is derivable in* EAL *and* $t \longrightarrow^*_{\beta\eta} x$*.*

A cœrcions enriched type system

You add the two following rules to EAL to obtain $\vdash_{\leq}$:

$$\frac{\Gamma, x : B \vdash t : C \quad A \leq B}{\Gamma, x : A \vdash t : C}\ coerc - L \qquad \frac{\Gamma \vdash t : A \quad A \leq B}{\Gamma \vdash t : B}\ coerc - R$$

The correction is given by:

Theorem

If $\Gamma \vdash_{\leq} t : A$ *then* $\exists t'$ *such that* $\Gamma \vdash_{\mathrm{EAL}} t' : A$ *is derivable and* $t' \longrightarrow^{*}_{\beta\eta} t$.

Proof.

Left side or right side cut with the term built from the subtyping derivation. □

Type inference with cœrcions

$$\frac{A \leq B}{x : A \vdash x : B}\ (\textsc{var}) \qquad \frac{\Gamma \vdash M : B}{\Gamma, x : A \vdash M : B}\ (\textsc{weak}) \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \multimap B}\ (\textsc{abs})$$

$$\frac{\Gamma_1 \vdash M_1 : B \multimap C \qquad \Gamma_2 \vdash M_2 : A \qquad A \leq B \quad C \leq D}{\Gamma_1, \Gamma_2 \vdash (M_1\ M_2) : D}\ (\textsc{appl})$$

$$\frac{x_1 : !C, \ldots, x_n : !C, \Delta \vdash M : B \qquad A \leq !C}{x : A, \Delta \vdash M\{x/x_1, \ldots, x/x_n\} : B}\ (\textsc{contr})$$

$$\frac{\Gamma_1 \vdash M_1 : !A_1 \ \ldots \Gamma_n \vdash M_n : !A_n \qquad x_1 : A_1, \ldots, x_n : A_n \vdash M : B}{\Gamma_1, \ldots, \Gamma_n \vdash !M\{\bar{!}M_i/x_i\} : !B}\ (\textsc{prom})$$

Figure: $\textsc{eal}_{\leq}$ inference-driven type system

Conclusion & perspectives

To sum up:

- Subsystems of LL provide original characterisations of complexity classes
- We can use those systems for typed λ-calculus static analysis, extending typing properties to complexity bounds certificate
- We presented an efficient type inference algorithm, and an extension to subtyping, which allows for more intensionality

$\Rightarrow$ Programming in those systems remains *very* tedious.
Some directions:

- Presented methods extend to a polynomial logic and polymorphic types (system F type decoration)
- Relations between complexity and Optimal Reduction
- Extending typing system (recursive types)