

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

Patrick Baillot

LIPN, CNRS Université Paris 13

based on joint works with V. Atassi, P. Coppola, U. Dal Lago, D. Mazza, K. Terui

Typed Lambda Calculi and Applications TLCA'07

Paris, June 28th 2007

(partly supported by project NOCoST (ANR))

From lambda calculus to complexity ?

- *Semantics and logic* aim at providing a structured understanding of computing, amenable to formal reasoning,
- *Algorithm analysis* provides accurate analysis of time/space properties of given algorithms

Can the semantical approach be tuned so as to take into account time/space properties ?

Can we refine it to improve our understanding of *feasible computing* (polynomial time), or more generally of time/space bounded computation ?

Implicit computational complexity

Can the semantical approach be tuned so as to take into account time/space properties ?

This is the purpose of *implicit computational complexity* (ICC).
Define calculi for which all programs have a certain complexity property (e.g. Ptime).

Born from works by Leivant, Bellantoni-Cook, Jones, Girard ...

Goals:

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

Various approaches in ICC

- **ramified/safe recursion**: primitive recursive definitions, with discipline for nesting (Leivant, Bellantoni-Cook);
- **term rewriting**: *quasi-interpretations* (Bonfante-Marion-Moyen);
- **non-size-increasing linear types**: lambda calculus with iterator, with linear type system (Hofmann);
- **linear logic** ...

Linear logic

We consider here the Linear logic (LL) approach to ICC, based on the *proofs-as-programs* paradigm (Curry-Howard correspondence).

- LL gives a logical status to duplication, by specific connectives (*exponentials* !, ?),
- various choices of rules for the modalities give variants of LL with different bounds on proof normalization (evaluation).

→ for Ptime computation:

Bounded (BLL) , *Soft* (SLL) and *Light* linear logics (LLL, LAL).

programming language = *proof-nets*

Linear logic

We consider here the Linear logic (LL) approach to ICC, based on the *proofs-as-programs* paradigm (Curry-Howard correspondence).

- LL gives a logical status to duplication, by specific connectives (*exponentials* $!$, $?$),
- various choices of rules for the modalities give variants of LL with different bounds on proof normalization (evaluation).

→ for Ptime computation:

Bounded (BLL) , *Soft* (SLL) and *Light* linear logics (LLL, LAL).

programming language = *proof-nets*

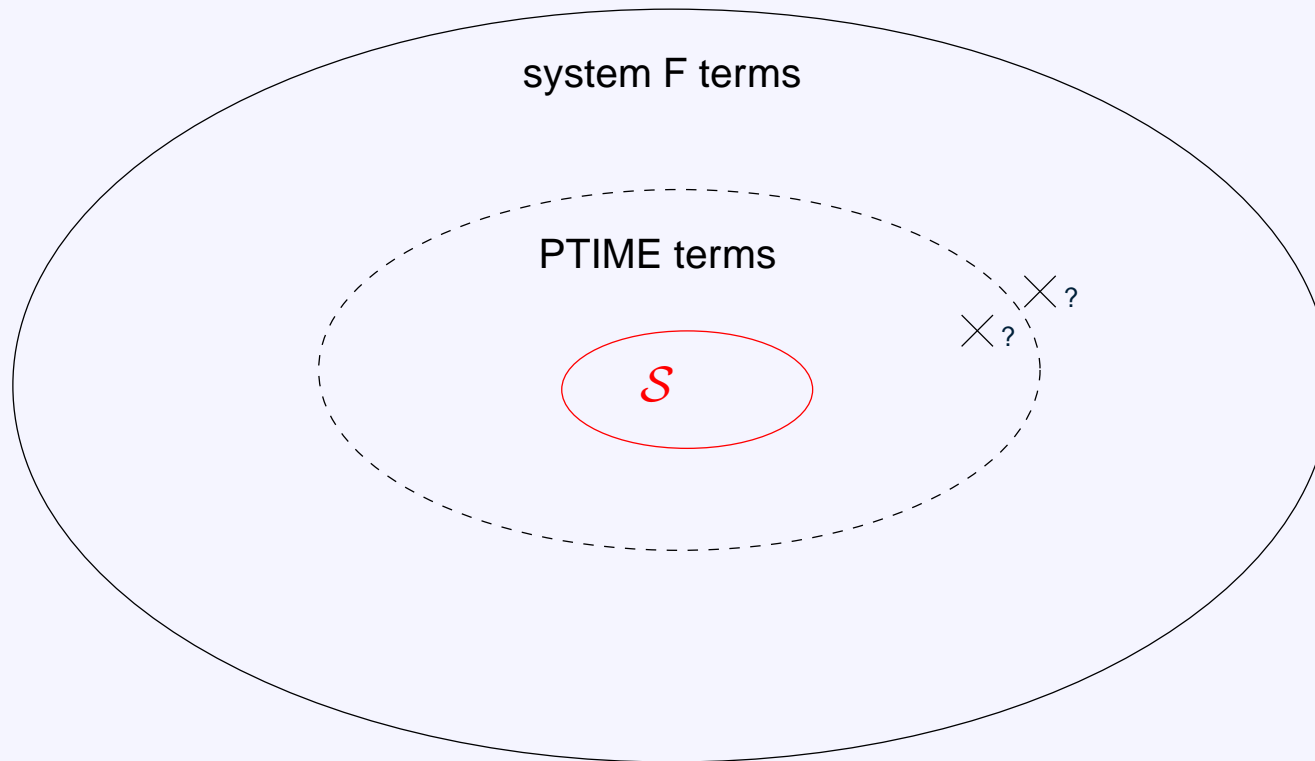
BLL: Girard-Scedrov-Scott 92

SLL: Lafont 04

LLL: Girard 95, LAL: Asperti 98

Feasibility in lambda calculus ?

We want to recast the linear logic approach to ICC as a *study of lambda calculus*.



Note that in general, testing if a system F term is Ptime is undecidable.

A subclass of feasible lambda terms ?

Delineate a subclass \mathcal{S} of lambda terms such that:

1. *terms of \mathcal{S} acting on data-types are feasible (Ptime),*
2. *testing if a term is in \mathcal{S} should be decidable (and possibly Ptime),*
3. \mathcal{S} enjoys certain closure properties
4. \mathcal{S} contains an algorithm for each Ptime *function* (*extensional completeness*) and ideally, as many algorithms as possible (*intensional expressivity*).

Proof-nets are an essential *tool* for that:

- find out the key invariants to obtain the complexity bounds,
- allow for path-based analysis of computation.

From LL systems to types

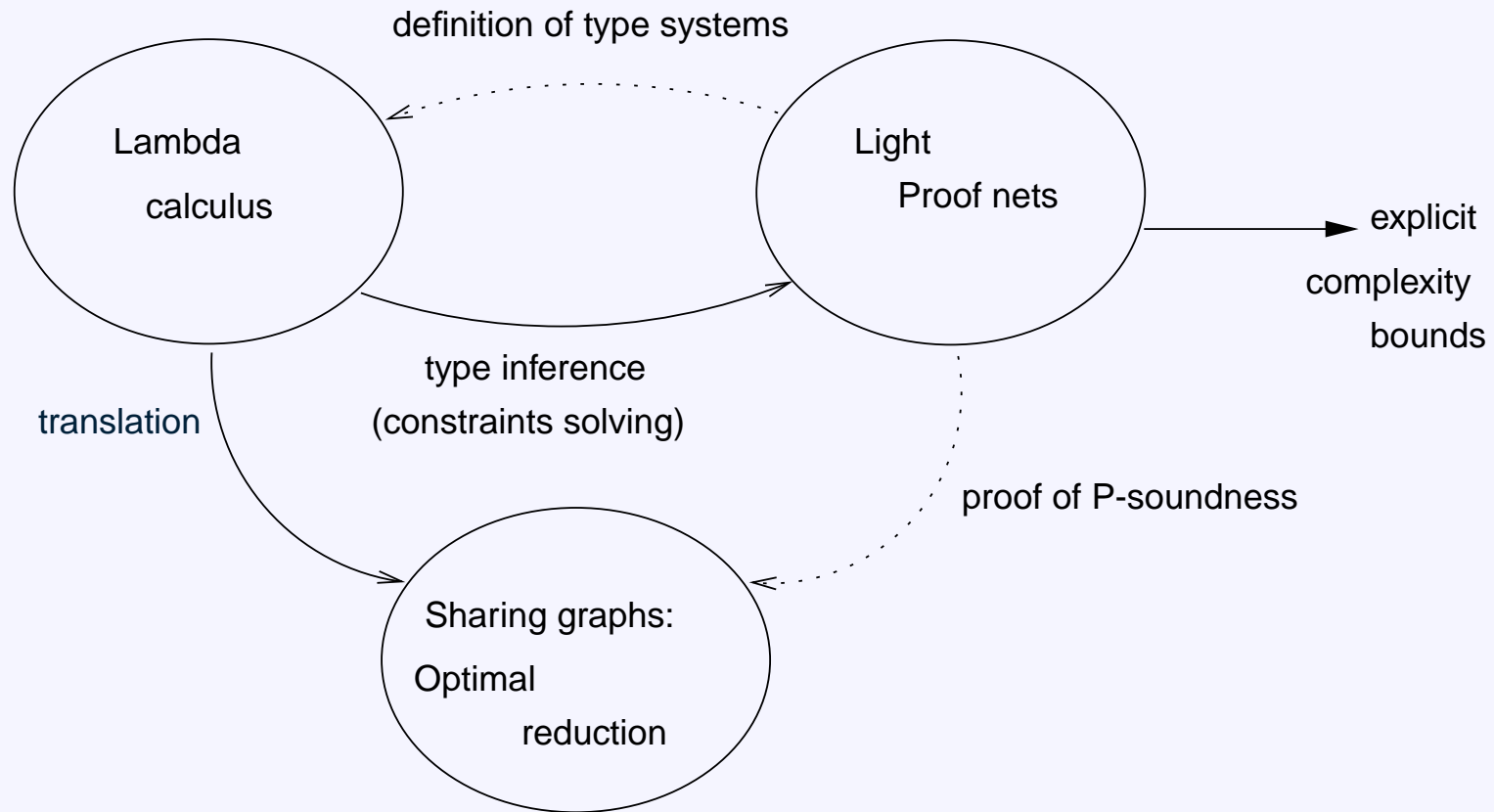
In this talk we concentrate on a particular approach to this problem, based on LLL and *typing*.

The logical systems mentioned can lead to type systems for guaranteeing Ptime bounds on lambda calculus:

if a term is well-typed, then it enjoys a polynomial time bound.

We will consider here a type system derived from Light linear logic.

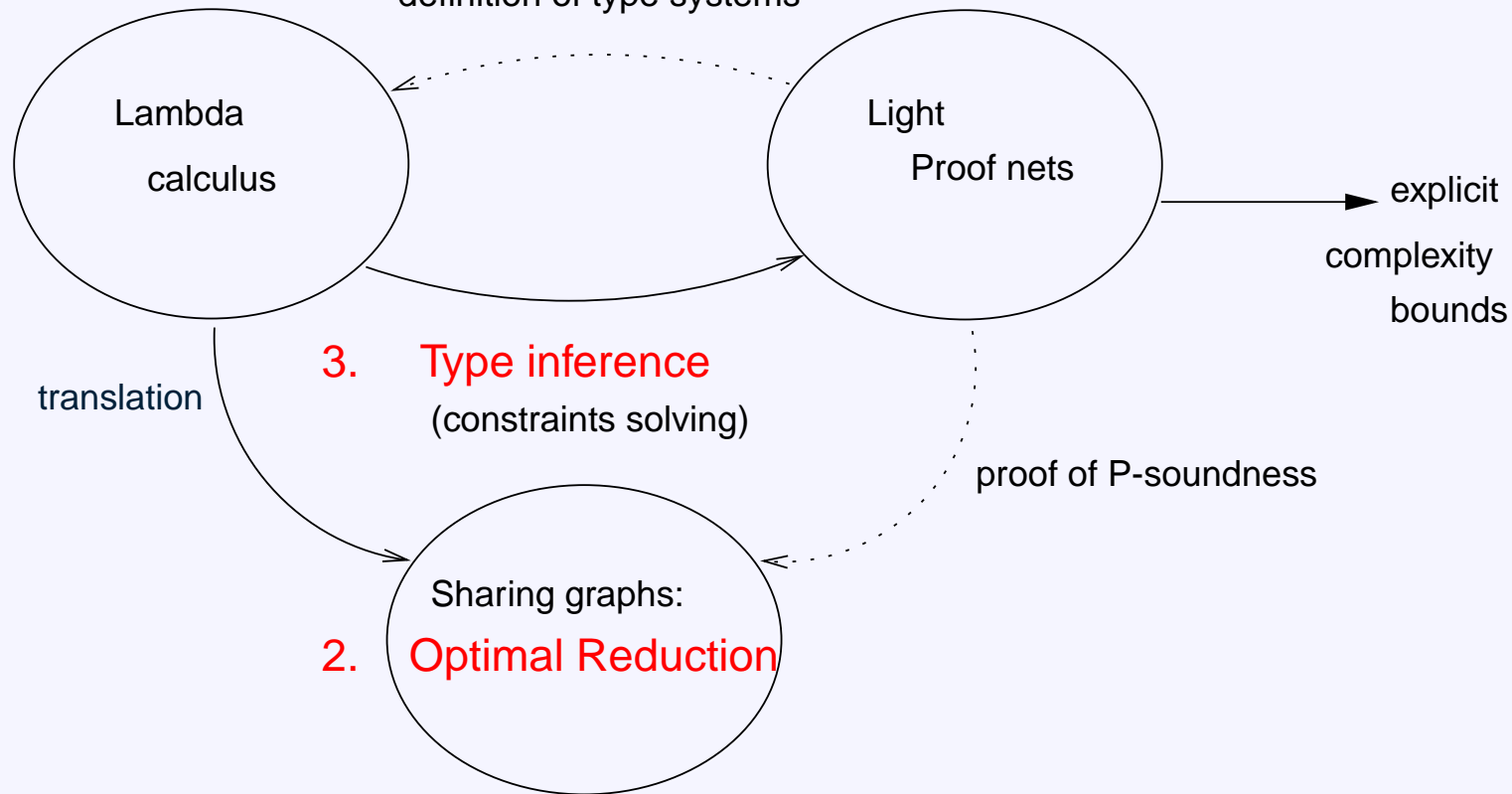
Outline of the talk



Outline of the talk

1. Type system DLAL and proof-nets

definition of type systems

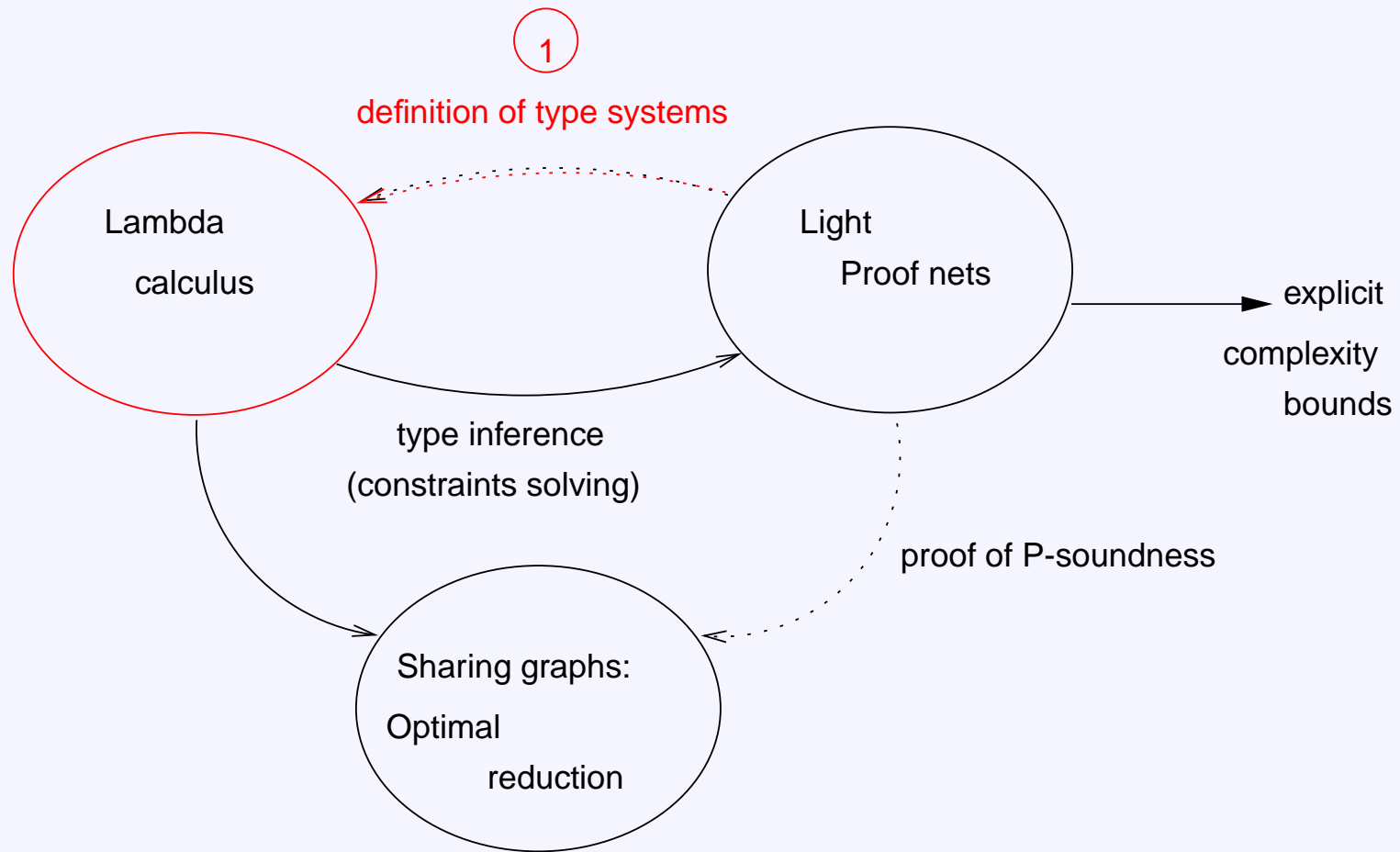


3. Type inference (constraints solving)

2. Optimal Reduction

4. Extending LAL proof-nets (work in progress, with D. Mazza)

1.Type system DLAL and proof-nets



Exponential blow up

2 easy ways to cause exponential blow-up:

■ basic functions: $0 : \mathbb{N}$, $s : \mathbb{N} \rightarrow \mathbb{N}$, $+ : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}$.

■ exponential blow-up can be caused by:

1. iteration-iteration

$$\begin{array}{ll} \text{dbl}(0) & = 0 \\ \text{dbl}(s(x)) & = \text{dbl}(x) + 2 \end{array} \qquad \begin{array}{ll} \text{exp}(0) & = 1 \\ \text{exp}(s(x)) & = \text{dbl}(\text{exp}(x)) \end{array}$$

2. contraction-iteration

$$\begin{array}{ll} \text{dbl}(x) & = x + x \\ \text{exp}(0) & = 1 \\ \text{exp}(s(x)) & = \text{dbl}(\text{exp}(x)) \end{array}$$

■ to keep contraction and iteration, we need to forbid bad combinations of these.

Linear logic

Linear logic (LL) arises from the decomposition $A \Rightarrow B = !A \multimap B$
the ! modality accounts for duplication (contraction)

principles:

$$\begin{array}{ccc} !A \multimap !A \otimes !A & \frac{A \vdash B}{!A \vdash !B} & !A \multimap A \\ !A \otimes !B \multimap !(A \otimes B) & & !A \multimap !!A \end{array}$$

Light linear logic, LLL (Girard)

$$!A \multimap !A \otimes !A$$
$$\frac{A \vdash B}{!A \vdash !B}$$
$$!A \otimes !B \multimap \S(A \otimes B)$$

new modality \S , with: $!A \multimap \S A$

\S is a functor and $\S A \otimes \S B \multimap \S(A \otimes B)$

—→ manages to avoid both exponentiation schemes

Light affine logic (LAL) is the variant with full weakening.

Theorem 1 (Girard) *Light linear logic proof-nets admit a polynomial time cut elimination (at fixed depth).*

Theorem 2 (Completeness. Girard/Asperti-Roversi) *All polynomial time functions $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ can be represented in Light Linear Logic (resp. Light Affine Logic).*

Light linear logic and typing

Can we use LLL or LAL directly as type systems for lambda calculus ?

There are two pitfalls:

- they do not give subject-reduction,
- no polynomial bound on the number of β -reduction steps for typed terms (even if there is one on proof-net normalization).

Type system DLAL

To overcome the problems with typing in LAL:
we can restrict in Light affine logic the use of ! to $!A \multimap B$, denoted $A \Rightarrow B$
the DLAL (*Dual Light Affine Logic*) type system [Baillot-Terui04]:

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

typing judgements of the form: $\Gamma; \Delta \vdash t : A$, where

Γ contains duplicable variables,

Δ contains linear variables.

DLAL as a type system

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

$$\frac{\Gamma_1; \Delta_1, x : A \vdash t : B}{\Gamma_1; \Delta_1 \vdash \lambda x. t : A \multimap B} \text{ (}\multimap\text{ i)}$$

$$\frac{\Gamma_1, x : A; \Delta_1 \vdash t : B}{\Gamma_1; \Delta_1 \vdash \lambda x. t : A \Rightarrow B} \text{ (}\Rightarrow\text{ i)}$$

$$\frac{\Gamma_1; \Delta_1 \vdash t : A}{\Gamma_1, \Gamma_2; \Delta_1, \Delta_2 \vdash t : A} \text{ (Weak)}$$

$$\frac{; \Gamma, x_1 : B_1, \dots, x_n : B_n \vdash t : A}{\Gamma; x_1 : \S B_1, \dots, x_n : \S B_n \vdash t : \S A} \text{ (§ i)}$$

$$\frac{\Gamma_1; \Delta_1 \vdash t : A \multimap B \quad \Gamma_2; \Delta_2 \vdash u : A}{\Gamma_1, \Gamma_2; \Delta_1, \Delta_2 \vdash (t u) : B} \text{ (}\multimap\text{ e)}$$

$$\frac{\Gamma_1; \Delta_1 \vdash t : A \Rightarrow B \quad ; z : C \vdash u : A}{\Gamma_1, z : C; \Delta_1 \vdash (t u) : B} \text{ (}\Rightarrow\text{ e)}$$

$$\frac{x_1 : A, x_2 : A, \Gamma_1; \Delta_1 \vdash t : B}{x : A, \Gamma_1; \Delta_1 \vdash t[x/x_1, x/x_2] : B} \text{ (Cntr)}$$

$$\frac{\Gamma_1; \Delta_1 \vdash u : \S A \quad \Gamma_2; x : \S A, \Delta_2 \vdash t : B}{\Gamma_1, \Gamma_2; \Delta_1, \Delta_2 \vdash t[u/x] : B} \text{ (§ e)}$$

DLAL as a type system

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

$$\frac{\Gamma_1; \Delta_1, x : A \vdash t : B}{\Gamma_1; \Delta_1 \vdash \lambda x. t : A \multimap B} \text{ (}\multimap\text{ i)}$$

$$\frac{\Gamma_1; \Delta_1 \vdash t : A \multimap B \quad \Gamma_2; \Delta_2 \vdash u : A}{\Gamma_1, \Gamma_2; \Delta_1, \Delta_2 \vdash (t u) : B} \text{ (}\multimap\text{ e)}$$

$$\frac{\Gamma_1, x : A; \Delta_1 \vdash t : B}{\Gamma_1; \Delta_1 \vdash \lambda x. t : A \Rightarrow B} \text{ (}\Rightarrow\text{ i)}$$

$$\frac{\Gamma_1; \Delta_1 \vdash t : A \Rightarrow B \quad ; z : C \vdash u : A}{\Gamma_1, z : C; \Delta_1 \vdash (t u) : B} \text{ (}\Rightarrow\text{ e)}$$

$$\frac{\Gamma_1; \Delta_1 \vdash t : A}{\Gamma_1, \Gamma_2; \Delta_1, \Delta_2 \vdash t : A} \text{ (Weak)}$$

$$\frac{x_1 : A, x_2 : A, \Gamma_1; \Delta_1 \vdash t : B}{x : A, \Gamma_1; \Delta_1 \vdash t[x/x_1, x/x_2] : B} \text{ (Cntr)}$$

$$\frac{; \Gamma, x_1 : B_1, \dots, x_n : B_n \vdash t : A}{\Gamma; x_1 : \S B_1, \dots, x_n : \S B_n \vdash t : \S A} \text{ (§ i)}$$

$$\frac{\Gamma_1; \Delta_1 \vdash u : \S A \quad \Gamma_2; x : \S A, \Delta_2 \vdash t : B}{\Gamma_1, \Gamma_2; \Delta_1, \Delta_2 \vdash t[u/x] : B} \text{ (§ e)}$$

Lemma 3 *If $\Gamma; \Delta \vdash_{DLAL} t : A$ and $x \in \Delta$ then x has at most one occurrence in t .*

DLAL as a type system

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

$$\frac{\Gamma_1; \Delta_1, x : A \vdash t : B}{\Gamma_1; \Delta_1 \vdash \lambda x. t : A \multimap B} \text{ (}\multimap\text{ i)}$$

$$\frac{\Gamma_1, x : A; \Delta_1 \vdash t : B}{\Gamma_1; \Delta_1 \vdash \lambda x. t : A \Rightarrow B} \text{ (}\Rightarrow\text{ i)}$$

$$\frac{\Gamma_1; \Delta_1 \vdash t : A}{\Gamma_1, \Gamma_2; \Delta_1, \Delta_2 \vdash t : A} \text{ (Weak)}$$

$$\frac{; \Gamma, x_1 : B_1, \dots, x_n : B_n \vdash t : A}{\Gamma; x_1 : \S B_1, \dots, x_n : \S B_n \vdash t : \S A} \text{ (§ i)}$$

$$\frac{\Gamma_1; \Delta_1 \vdash t : A \multimap B \quad \Gamma_2; \Delta_2 \vdash u : A}{\Gamma_1, \Gamma_2; \Delta_1, \Delta_2 \vdash (t u) : B} \text{ (}\multimap\text{ e)}$$

$$\frac{\Gamma_1; \Delta_1 \vdash t : A \Rightarrow B \quad ; z : C \vdash u : A}{\Gamma_1, z : C; \Delta_1 \vdash (t u) : B} \text{ (}\Rightarrow\text{ e)}$$

$$\frac{x_1 : A, x_2 : A, \Gamma_1; \Delta_1 \vdash t : B}{x : A, \Gamma_1; \Delta_1 \vdash t[x/x_1, x/x_2] : B} \text{ (Cntr)}$$

$$\frac{\Gamma_1; \Delta_1 \vdash u : \S A \quad \Gamma_2; x : \S A, \Delta_2 \vdash t : B}{\Gamma_1, \Gamma_2; \Delta_1, \Delta_2 \vdash t[u/x] : B} \text{ (§ e)}$$

(\multimap i) (resp. (\Rightarrow i)) corresponds to abstraction on a linear (resp. non-linear) variable,

DLAL as a type system

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

$$\frac{\Gamma_1; \Delta_1, x : A \vdash t : B}{\Gamma_1; \Delta_1 \vdash \lambda x.t : A \multimap B} \text{ (}\multimap\text{ i)}$$

$$\frac{\Gamma_1; \Delta_1 \vdash t : A \multimap B \quad \Gamma_2; \Delta_2 \vdash u : A}{\Gamma_1, \Gamma_2; \Delta_1, \Delta_2 \vdash (t u) : B} \text{ (}\multimap\text{ e)}$$

$$\frac{\Gamma_1, x : A; \Delta_1 \vdash t : B}{\Gamma_1; \Delta_1 \vdash \lambda x.t : A \Rightarrow B} \text{ (}\Rightarrow\text{ i)}$$

$$\frac{\Gamma_1; \Delta_1 \vdash t : A \Rightarrow B \quad ; z : C \vdash u : A}{\Gamma_1, z : C; \Delta_1 \vdash (t u) : B} \text{ (}\Rightarrow\text{ e)}$$

$$\frac{\Gamma_1; \Delta_1 \vdash t : A}{\Gamma_1, \Gamma_2; \Delta_1, \Delta_2 \vdash t : A} \text{ (Weak)}$$

$$\frac{x_1 : A, x_2 : A, \Gamma_1; \Delta_1 \vdash t : B}{x : A, \Gamma_1; \Delta_1 \vdash t[x/x_1, x/x_2] : B} \text{ (Cntr)}$$

$$\frac{; \Gamma, x_1 : B_1, \dots, x_n : B_n \vdash t : A}{\Gamma; x_1 : \S B_1, \dots, x_n : \S B_n \vdash t : \S A} \text{ (§ i)}$$

$$\frac{\Gamma_1; \Delta_1 \vdash u : \S A \quad \Gamma_2; x : \S A, \Delta_2 \vdash t : B}{\Gamma_1, \Gamma_2; \Delta_1, \Delta_2 \vdash t[u/x] : B} \text{ (§ e)}$$

an argument u of a term t of type $A \Rightarrow B$ must have at most one free variable z , which is moreover linear.

→ prevents 2nd exponentiation scheme (contraction-iteration).

DLAL as a type system

$$\begin{array}{c}
 \overline{} \quad (\text{Id}) \\
 \Gamma_1; \Delta_1, x : A \vdash t : B \\
 \hline
 \Gamma_1; \Delta_1 \vdash \lambda x.t : A \multimap B \quad (\multimap \text{ i})
 \end{array}
 \qquad
 \begin{array}{c}
 \Gamma_1; \Delta_1 \vdash t : A \multimap B \quad \Gamma_2; \Delta_2 \vdash u : A \\
 \hline
 \Gamma_1, \Gamma_2; \Delta_1, \Delta_2 \vdash (t u) : B \quad (\multimap \text{ e})
 \end{array}$$

$$\begin{array}{c}
 \Gamma_1, x : A; \Delta_1 \vdash t : B \\
 \hline
 \Gamma_1; \Delta_1 \vdash \lambda x.t : A \Rightarrow B \quad (\Rightarrow \text{ i})
 \end{array}
 \qquad
 \begin{array}{c}
 \Gamma_1; \Delta_1 \vdash t : A \Rightarrow B \quad ; z : C \vdash u : A \\
 \hline
 \Gamma_1, z : C; \Delta_1 \vdash (t u) : B \quad (\Rightarrow \text{ e})
 \end{array}$$

$$\begin{array}{c}
 \Gamma_1; \Delta_1 \vdash t : A \\
 \hline
 \Gamma_1, \Gamma_2; \Delta_1, \Delta_2 \vdash t : A \quad (\text{Weak})
 \end{array}
 \qquad
 \begin{array}{c}
 x_1 : A, x_2 : A, \Gamma_1; \Delta_1 \vdash t : B \\
 \hline
 x : A, \Gamma_1; \Delta_1 \vdash t[x/x_1, x/x_2] : B \quad (\text{Cntr})
 \end{array}$$

$$\begin{array}{c}
 \Gamma, x_1 : B_1, \dots, x_n : B_n \vdash t : A \\
 \hline
 \Gamma; x_1 : \S B_1, \dots, x_n : \S B_n \vdash t : \S A \quad (\S \text{ i})
 \end{array}
 \qquad
 \begin{array}{c}
 \Gamma_1; \Delta_1 \vdash u : \S A \quad \Gamma_2; x : \S A, \Delta_2 \vdash t : B \\
 \hline
 \Gamma_1, \Gamma_2; \Delta_1, \Delta_2 \vdash t[u/x] : B \quad (\S \text{ e})
 \end{array}$$

the rule (\S i) allows to turn linear variables (in Γ) into non-linear ones.

→ prevents 1st exponentiation scheme (iteration-iteration).

DLAL as a type system

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

$$\frac{\Gamma_1; \Delta_1, x : A \vdash t : B}{\Gamma_1; \Delta_1 \vdash \lambda x. t : A \multimap B} \text{ (}\multimap\text{ i)}$$

$$\frac{\Gamma_1; \Delta_1 \vdash t : A \multimap B \quad \Gamma_2; \Delta_2 \vdash u : A}{\Gamma_1, \Gamma_2; \Delta_1, \Delta_2 \vdash (t u) : B} \text{ (}\multimap\text{ e)}$$

$$\frac{\Gamma_1, x : A; \Delta_1 \vdash t : B}{\Gamma_1; \Delta_1 \vdash \lambda x. t : A \Rightarrow B} \text{ (}\Rightarrow\text{ i)}$$

$$\frac{\Gamma_1; \Delta_1 \vdash t : A \Rightarrow B \quad ; z : C \vdash u : A}{\Gamma_1, z : C; \Delta_1 \vdash (t u) : B} \text{ (}\Rightarrow\text{ e)}$$

$$\frac{\Gamma_1; \Delta_1 \vdash t : A}{\Gamma_1, \Gamma_2; \Delta_1, \Delta_2 \vdash t : A} \text{ (Weak)}$$

$$\frac{x_1 : A, x_2 : A, \Gamma_1; \Delta_1 \vdash t : B}{x : A, \Gamma_1; \Delta_1 \vdash t[x/x_1, x/x_2] : B} \text{ (Cntr)}$$

$$\frac{; \Gamma, x_1 : B_1, \dots, x_n : B_n \vdash t : A}{\Gamma; x_1 : \S B_1, \dots, x_n : \S B_n \vdash t : \S A} \text{ (§ i)}$$

$$\frac{\Gamma_1; \Delta_1 \vdash u : \S A \quad \Gamma_2; x : \S A, \Delta_2 \vdash t : B}{\Gamma_1, \Gamma_2; \Delta_1, \Delta_2 \vdash t[u/x] : B} \text{ (§ e)}$$

DLAL as a type system

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

$$\frac{\Gamma_1; \Delta_1, x : A \vdash t : B}{\Gamma_1; \Delta_1 \vdash \lambda x. t : A \multimap B} \text{ (}\multimap\text{ i)}$$

$$\frac{\Gamma_1; \Delta_1 \vdash t : A \multimap B \quad \Gamma_2; \Delta_2 \vdash u : A}{\Gamma_1, \Gamma_2; \Delta_1, \Delta_2 \vdash (t u) : B} \text{ (}\multimap\text{ e)}$$

$$\frac{\Gamma_1, x : A; \Delta_1 \vdash t : B}{\Gamma_1; \Delta_1 \vdash \lambda x. t : A \Rightarrow B} \text{ (}\Rightarrow\text{ i)}$$

$$\frac{\Gamma_1; \Delta_1 \vdash t : A \Rightarrow B \quad ; z : C \vdash u : A}{\Gamma_1, z : C; \Delta_1 \vdash (t u) : B} \text{ (}\Rightarrow\text{ e)}$$

$$\frac{\Gamma_1; \Delta_1 \vdash t : A}{\Gamma_1, \Gamma_2; \Delta_1, \Delta_2 \vdash t : A} \text{ (Weak)}$$

$$\frac{x_1 : A, x_2 : A, \Gamma_1; \Delta_1 \vdash t : B}{x : A, \Gamma_1; \Delta_1 \vdash t[x/x_1, x/x_2] : B} \text{ (Cntr)}$$

$$\frac{; \Gamma, x_1 : B_1, \dots, x_n : B_n \vdash t : A}{\Gamma; x_1 : \S B_1, \dots, x_n : \S B_n \vdash t : \S A} \text{ (§ i)}$$

$$\frac{\Gamma_1; \Delta_1 \vdash u : \S A \quad \Gamma_2; x : \S A, \Delta_2 \vdash t : B}{\Gamma_1, \Gamma_2; \Delta_1, \Delta_2 \vdash t[u/x] : B} \text{ (§ e)}$$

Depth d of a derivation \mathcal{D} : maximal number of (§ i) and r.h.s. premises of (\Rightarrow e) in a branch of \mathcal{D} .

DLAL as a type system

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

$$\frac{\Gamma_1; \Delta_1, x : A \vdash t : B}{\Gamma_1; \Delta_1 \vdash \lambda x.t : A \multimap B} \text{ (}\multimap\text{ i)}$$

$$\frac{\Gamma_1, x : A; \Delta_1 \vdash t : B}{\Gamma_1; \Delta_1 \vdash \lambda x.t : A \Rightarrow B} \text{ (}\Rightarrow\text{ i)}$$

$$\frac{\Gamma_1; \Delta_1 \vdash t : A}{\Gamma_1, \Gamma_2; \Delta_1, \Delta_2 \vdash t : A} \text{ (Weak)}$$

$$\frac{; \Gamma, x_1 : B_1, \dots, x_n : B_n \vdash t : A}{\Gamma; x_1 : \S B_1, \dots, x_n : \S B_n \vdash t : \S A} \text{ (§ i)}$$

$$\frac{\Gamma_1; \Delta_1 \vdash t : A}{\Gamma_1; \Delta_1 \vdash t : \forall \alpha. A} \text{ (}\forall\text{ i) (*)}$$

$$\frac{\Gamma_1; \Delta_1 \vdash t : A \multimap B \quad \Gamma_2; \Delta_2 \vdash u : A}{\Gamma_1, \Gamma_2; \Delta_1, \Delta_2 \vdash (t u) : B} \text{ (}\multimap\text{ e)}$$

$$\frac{\Gamma_1; \Delta_1 \vdash t : A \Rightarrow B \quad ; z : C \vdash u : A}{\Gamma_1, z : C; \Delta_1 \vdash (t u) : B} \text{ (}\Rightarrow\text{ e)}$$

$$\frac{x_1 : A, x_2 : A, \Gamma_1; \Delta_1 \vdash t : B}{x : A, \Gamma_1; \Delta_1 \vdash t[x/x_1, x/x_2] : B} \text{ (Cntr)}$$

$$\frac{\Gamma_1; \Delta_1 \vdash u : \S A \quad \Gamma_2; x : \S A, \Delta_2 \vdash t : B}{\Gamma_1, \Gamma_2; \Delta_1, \Delta_2 \vdash t[u/x] : B} \text{ (§ e)}$$

$$\frac{\Gamma_1; \Delta_1 \vdash t : \forall \alpha. A}{\Gamma_1; \Delta_1 \vdash t : A[B/\alpha]} \text{ (}\forall\text{ e)}$$

(*) $\alpha \notin \Gamma_1, \Delta_1$.

DLAL and system F

- Forgetful map $(.)^-$ from DLAL to F:

- remove occurrences of \S ,
- replace \multimap and \Rightarrow with \rightarrow .

$(.)^-$ gives a map from DLAL derivations to system F derivations.

- Data types in DLAL:

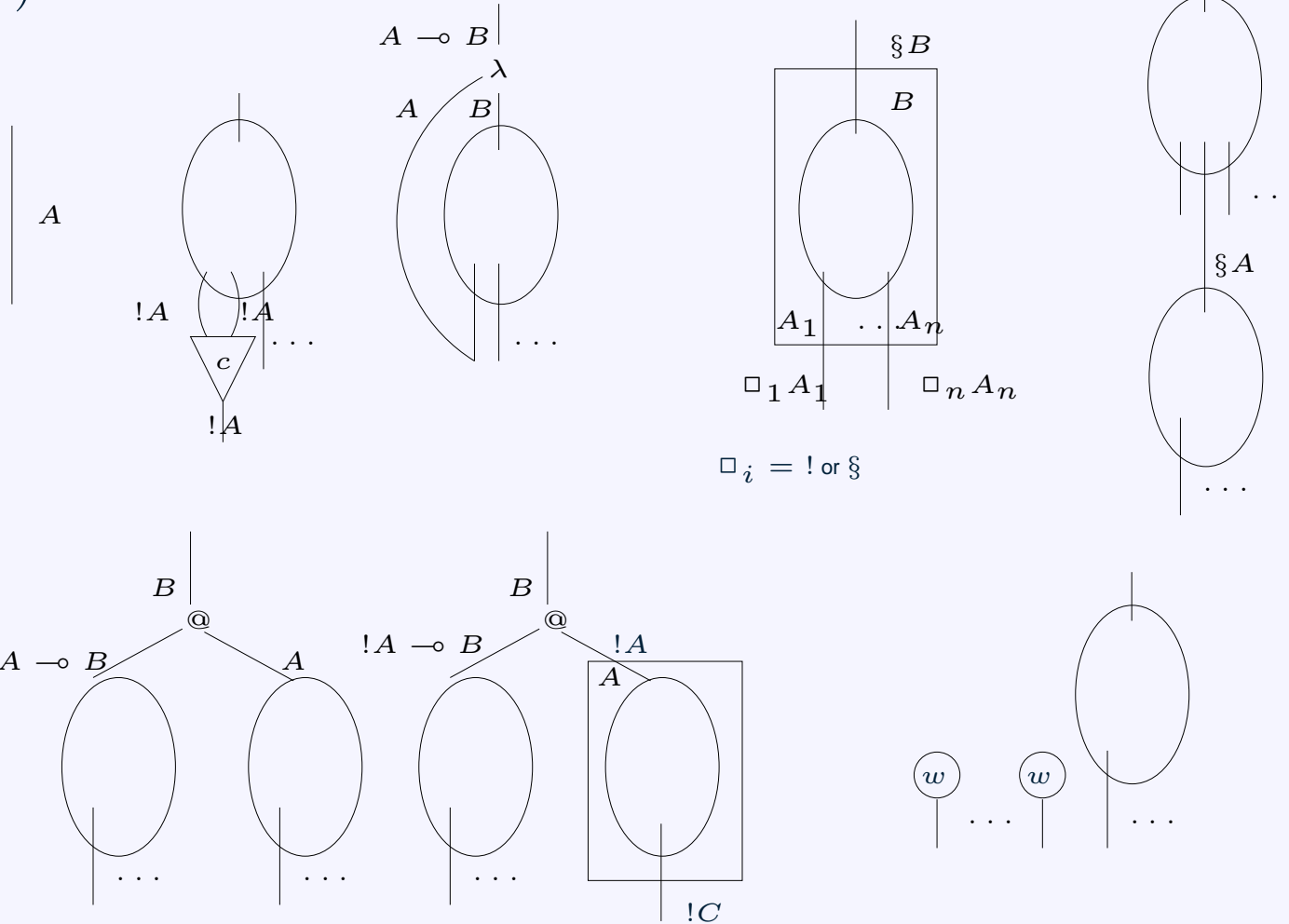
	F	DLAL
Church integers	$\forall\alpha.(\alpha \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha)$	$\forall\alpha.(\alpha \multimap \alpha) \Rightarrow \S(\alpha \multimap \alpha)$
binary lists	$\forall\alpha.(\alpha \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha)$	$\forall\alpha.(\alpha \multimap \alpha) \Rightarrow (\alpha \multimap \alpha) \Rightarrow \S(\alpha \multimap \alpha)$

Example for binary lists: $w = \langle 1, 0, 0 \rangle$:

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

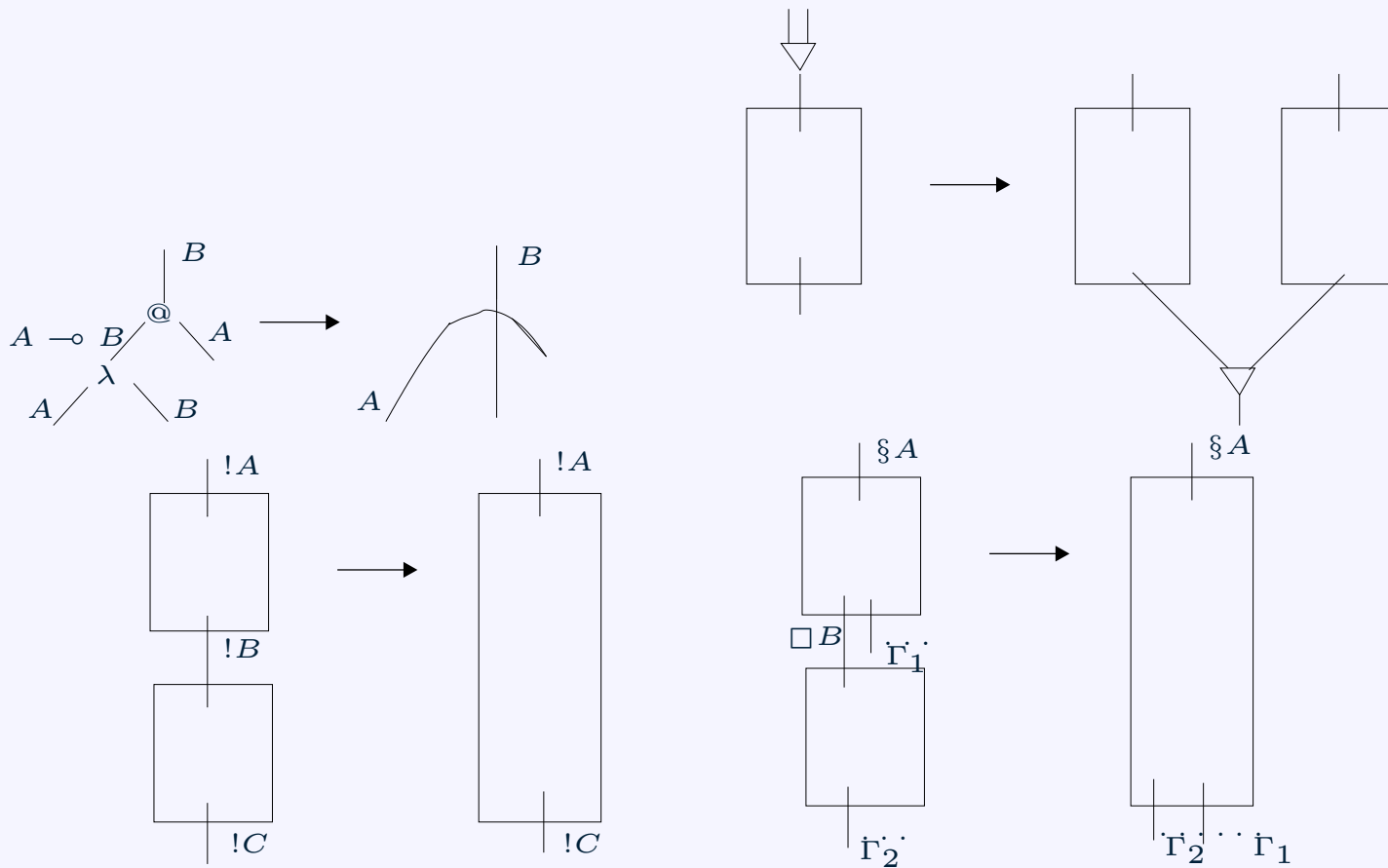
Intuitionistic Proof-Nets for DLAL terms

$$(A \Rightarrow B)^* = !A^* \multimap B^*$$



depth of an edge: number of boxes it is contained in.
depth of proof-net: maximal depth of its edges.

Proof-net reduction



During normalization: the depth of an edge is unchanged.

Example

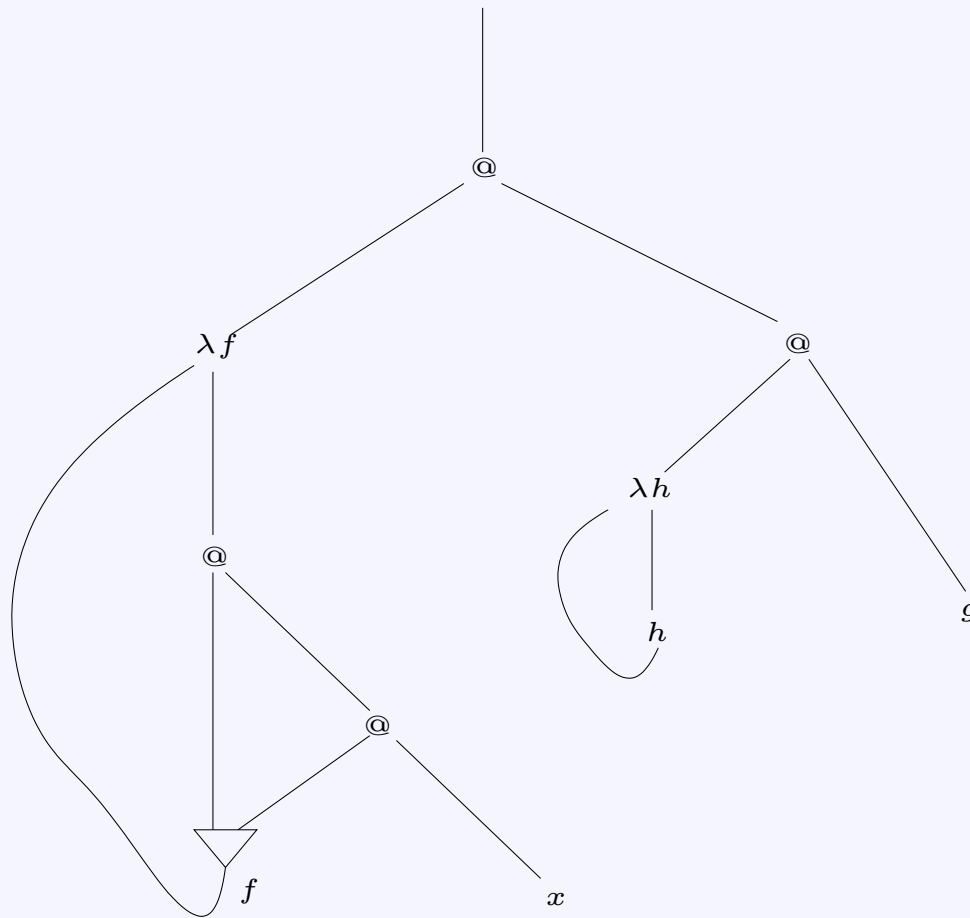
$$M = (\lambda f.(f (f x)))(\lambda h.h) g$$

It can be given the following simple type:

$$M = (\lambda f^{\alpha \rightarrow \alpha} . (f (f x^\alpha)))(\lambda h^{\alpha \rightarrow \alpha} . h) g^{\alpha \rightarrow \alpha} : \alpha$$

Example

$$M = (\lambda f.(f (f x)))(\lambda h.h) g$$



Example of type derivation

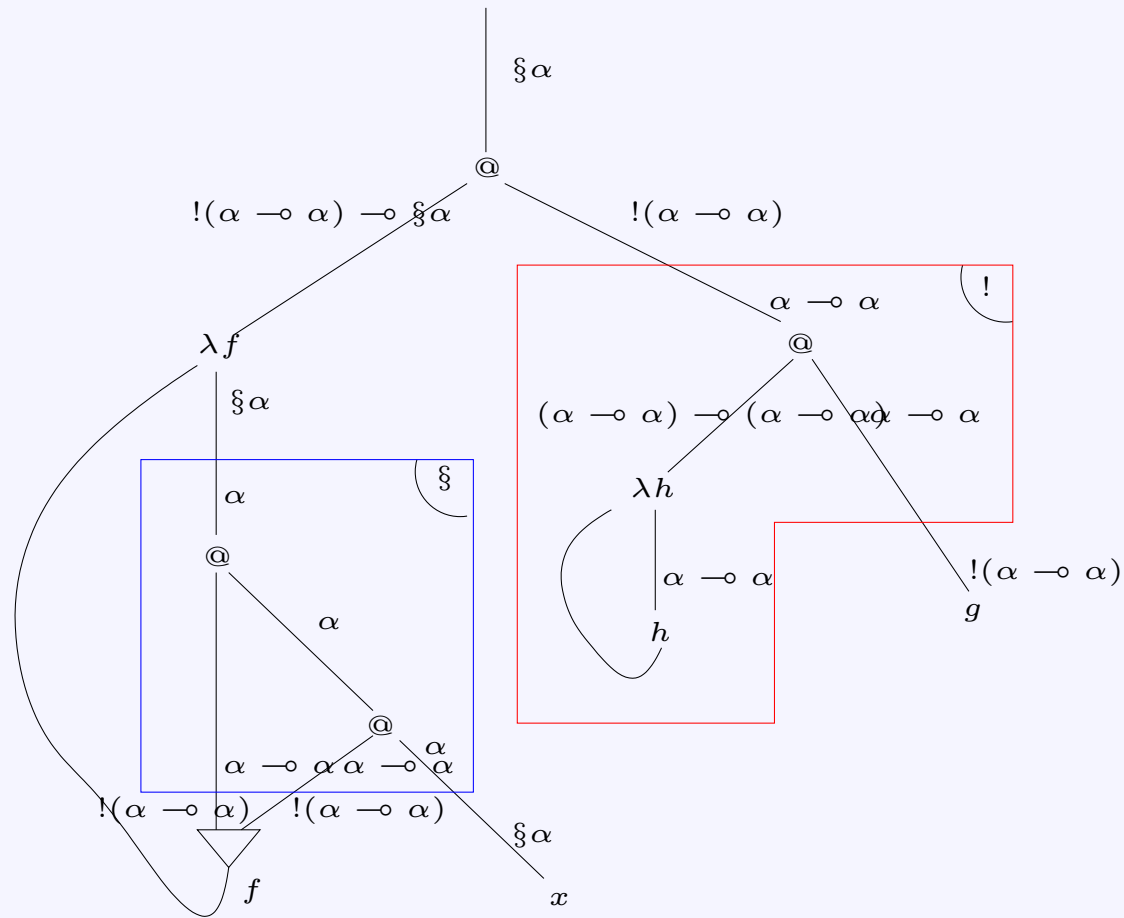
$$M = (\lambda f.(f (f x)))(\lambda h.h) g$$

$$\begin{array}{c}
 \dots \\
 \hline
 ; f_2 : \alpha \multimap \alpha, x : \alpha \vdash (f_2) x : \alpha \quad ; f_1 : \alpha \multimap \alpha \vdash f_1 : \alpha \multimap \alpha \\
 \hline
 (\S \text{ i}) \frac{; f_1 : \beta, f_2 : \beta, x : \alpha \vdash (f_1) ((f_2) x) : \alpha}{f_1 : \beta, f_2 : \beta; x : \xi\alpha \vdash (f_1) ((f_2) x) : \xi\alpha} \quad \frac{; h : \beta \vdash h : \beta}{; \vdash \lambda h.h : \beta \multimap \beta} \quad ; g : \beta \vdash g : \beta \\
 \hline
 (\Rightarrow \text{ e}) \frac{f : \beta; x : \xi\alpha \vdash (f) ((f) x) : \xi\alpha \quad ; x : \xi\alpha \vdash \lambda f.(f) ((f) x) : \beta \Rightarrow \xi\alpha \quad ; g : \beta \vdash (\lambda h.h) g : \beta}{g : \beta; x : \xi\alpha \vdash (\lambda f.(f) ((f) x))(\lambda h.h) g : \xi\alpha}
 \end{array}$$

where $\beta = \alpha \multimap \alpha$.

Example

$$M = (\lambda f.(f (f x)))((\lambda h.h) g)$$



Boxes

Boxes in linear logic proof-nets have 2 characteristics:

1. *logically*:

they correspond to a sequentiality information on the graph.

2. *dynamically*:

they delimitate portions of the graph that can be duplicated.

Note that in LAL, (2) is not relevant for \S boxes, since they cannot be duplicated.

In Light logics, the crucial point of boxes is that they allow to define some *invariants* of the dynamics of the graphs:

- depth of edges,
- quantitative invariant: the number of inputs of duplicable boxes (! boxes) is at most one.

DLAL: complexity bounds [Baillot-Terui04]

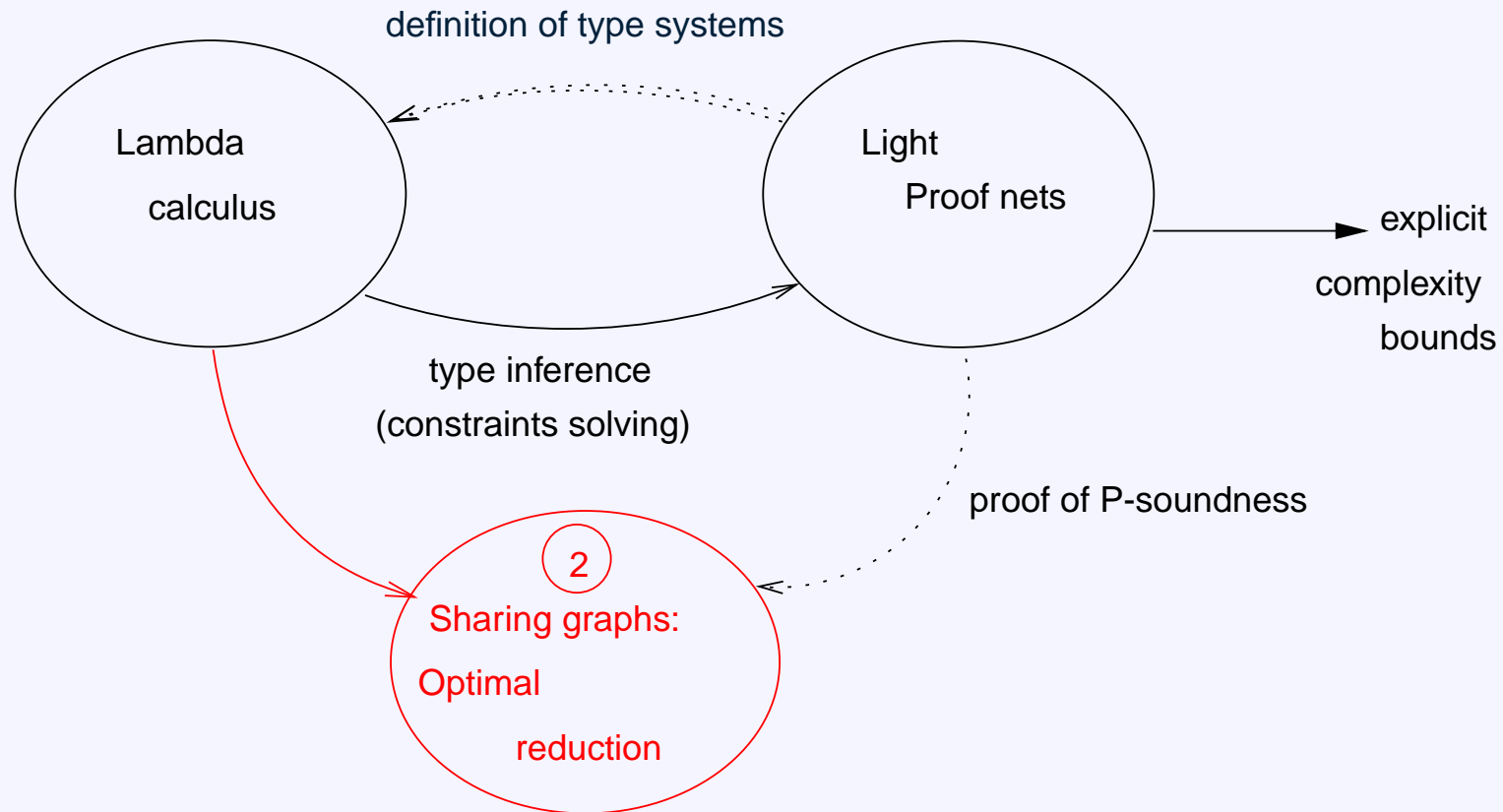
Theorem 4 (Strong Ptime bound) *If t is typable in DLAL with a derivation of depth d , then any β reduction of t can be performed in time $O((d + 1) \cdot |t|^{2^{d+1}})$.*

Note that here the bound is with respect to β reduction: the boxing structure is not needed for polynomial time evaluation.

Theorem 5 (Completeness) *For any polynomial time function $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$, there exists a term t representing it and typable in DLAL with a type $W \multimap \xi^k W$, for a certain integer $k \in \mathbb{N}$.*

However DLAL is not very expressive from an intensional point of view: some simple Ptime lambda terms might not be typable.

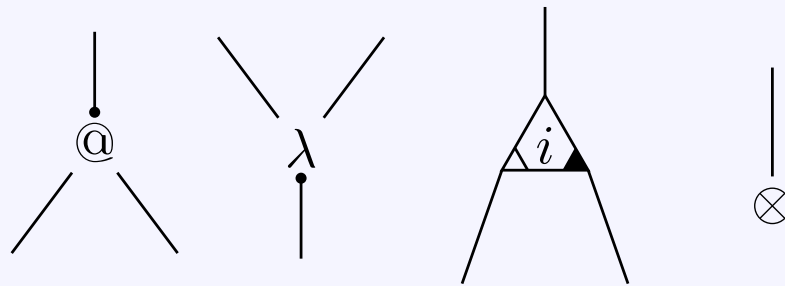
2. Optimal reduction: *Evaluating without boxes*



Sharing graphs

DLAL typable terms can be evaluated using a simple algorithm for *optimal reduction*: *Lamping's abstract algorithm*.

Abstract sharing graphs are built from the following nodes:



t a term. Let \mathcal{G}_t be the abstract sharing graph obtained by annotating each contraction in the syntax tree by a *distinct* integer.

Advantage of the abstract algorithm w.r.t. the general one ([Lamping90,GAL92]): no bookkeeping needed for managing the indices, which makes it particularly simple.

Lamping's abstract algorithm

Proposition 6 *Lamping's abstract algorithm is correct for DLAL typable terms.*

This is actually already true for EAL, LAL terms.

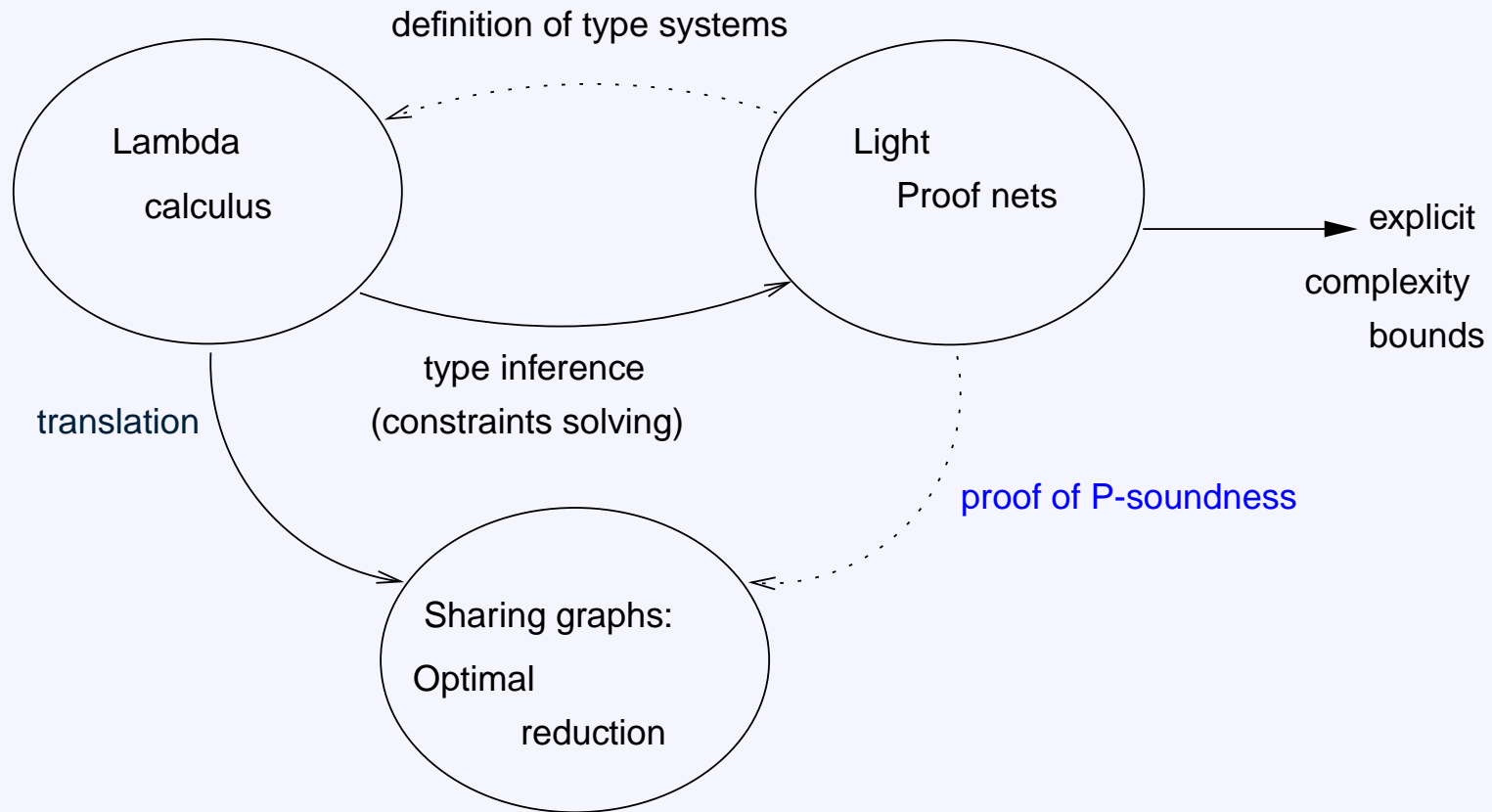
However this algorithm is not correct in general for arbitrary lambda terms.

Theorem 7 (Baillot- Coppola-Dal Lago 07) *If a term t is typable in DLAL with a derivation of depth d , then Lamping's abstract algorithm applied to t terminates in time $P(|t|)$, where P is a polynomial depending only on d .*

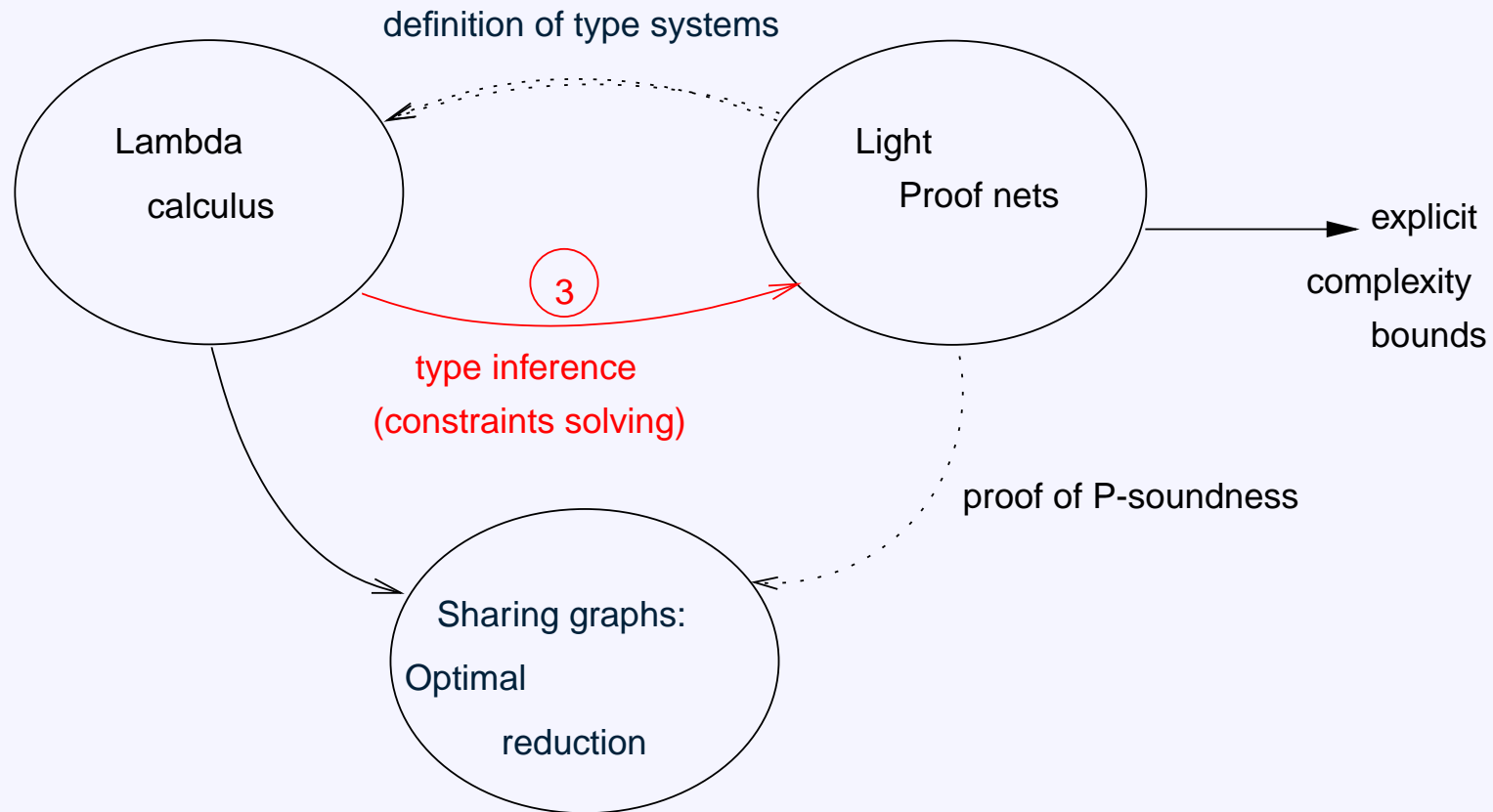
Lamping's abstract algorithm

- the translation of term into sharing graph does not need the DLAL type derivation: if we know a term is *typable* (without its type), we can evaluate it.
- the bound on optimal reduction from [Baillot-Coppola-DaLago07] is obtained by relating optimal reduction to proof-net reduction by using a form of *geometry of interaction* (context semantics): path-based analysis of computation.
It is actually more general than the result stated here (EAL, LAL).

Lamping's abstract algorithm



3. Type inference by linear programming: *Recovering boxes*



Type inference for DLAL

DLAL *typable* lambda terms can be evaluated efficiently without the type derivation. However, typing is important to get an explicit bound and for modularity.

DLAL type inference problem:

input: system F term t

problem: does there exist a DLAL derivation for t ?

main issue:

- decorate the F derivation with modalities
 - 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

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

For that

1. *symbolic approach*: use parameters n_i ranging over \mathbb{Z}

(also in algorithms by Coppola-Martini, Coppola-Ronchi Della Rocca, for EAL)

2. concentrate on *doors* rather than boxes: linear number of door positions.
3. ...

From term with doors to proof-net

if we have a term with doors, a *candidate proof-net* R can be constructed by matching closing doors with opening doors.

For R to be correct however, some conditions are needed:

1. closing and opening doors should match
2. boxes should be compatible with λ (and Λ) binding (akin to *sequentiality conditions*)
3. at each node (doors, @ etc) some *local typing condition* should hold
4. !-boxes should satisfy the *Light condition* (at most 1 input)

DLAL Type inference algorithm

The algorithm of [AtassiBaillotTerui06] relies on two ingredients:

1. analysing where non-linear application (and hence !-box) is needed: *boolean constraints*;
2. searching for a suitable distribution of boxes (! or § boxes).

Key technique for (2): assign integer parameters for the number of (box) *doors* on each edge, and search for instantiations of these parameters for which valid boxes can be reconstructed.

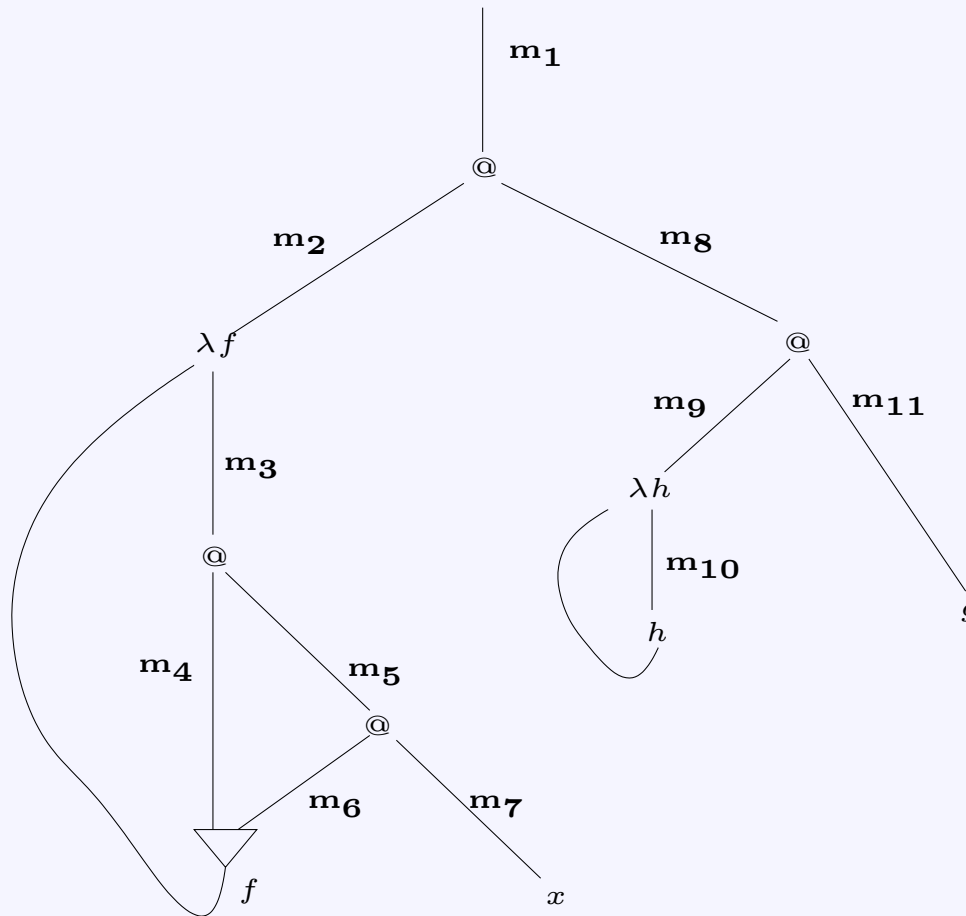
In this way a set of constraints on boolean (b_i) and integer (n_i) parameters is associated to a term, expressing its typability:

boolean constraints, *e.g.* $b_1 = b_2, \quad b_1 = 0$

linear constraints, *e.g.* $\sum_i n_i \geq 0$

mixed boolean/linear constraints, *e.g.* $b_1 = 1 \Rightarrow \sum_i n_i \geq 0.$

Example: Parameterized term



example of parameterized type: $\xi^{m_1} (\xi^{b_2, m_2} \alpha \rightarrow \xi^{m_3} \alpha)$

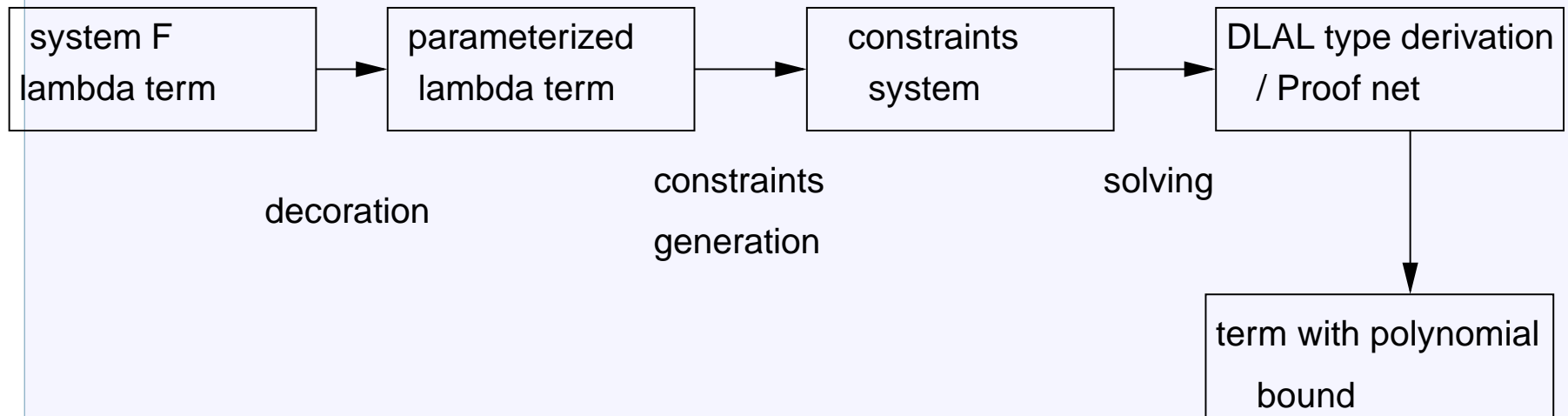
Constraints solving

A resolution method for solving the constraints system is given by a 2-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.

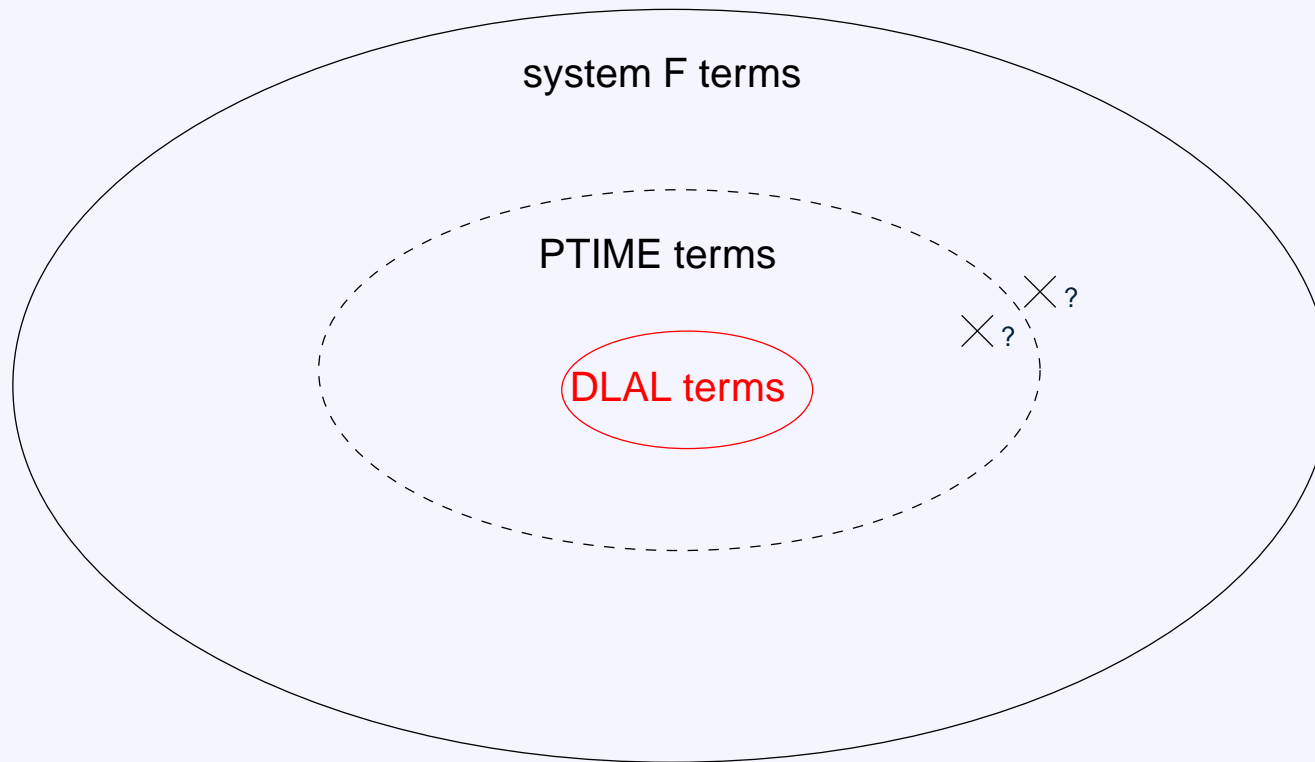
This resolution procedure is correct and complete and it can be performed in polynomial time w.r.t. the size of the original system F term.

Type inference procedure



Theorem 8 *The type inference problem for system F terms in DLAL is decidable in polynomial time.*

The subclass of DLAL typable terms



Proof-nets and boxes

Boxes are usually needed to perform proof-net normalization in LL.

We emphasized here for DLAL and Light logic:

1. from a *methodological point of view*:

boxes are a key feature in Light logics because they enforce some *invariants* which guarantee the complexity bound;

2. from an *operational point of view*:

boxes can somehow be forgotten for evaluation of (typable) terms.

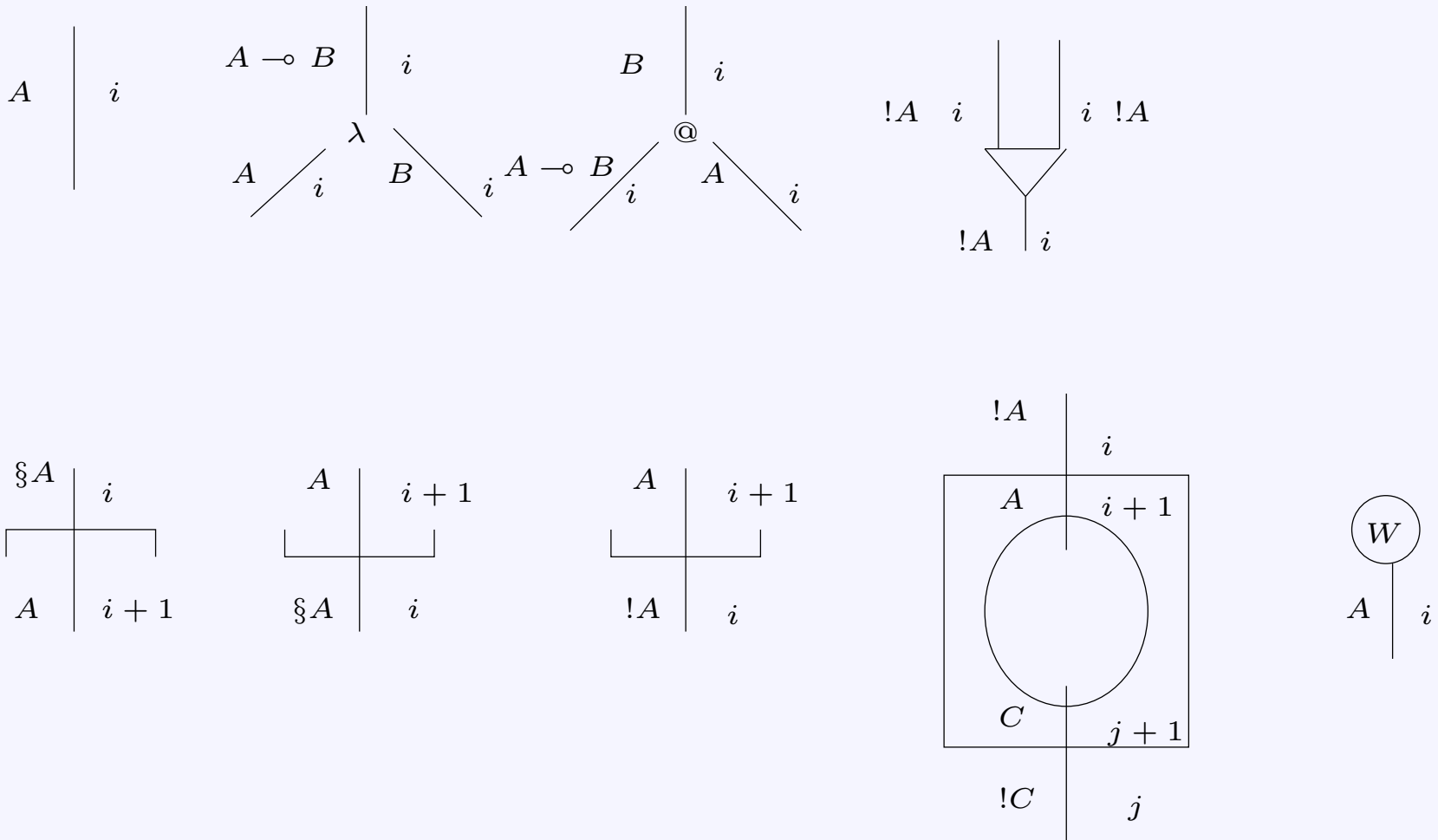
4. Extending LAL: LAL_{lev} ; work in progress, with D.Mazza

the techniques at work for bounding the complexity of LAL proof-net normalization suggest that:

- one might relax conditions on boxes ...
- provided that invariants are maintained in another way.

—→ an integer i (*level*) is attached to each edge: *LAL by levels* (LAL_{lev}).

LAL_{lev} proof-nets



LAL_{lev} and LAL

Embedding of LAL into LAL_{lev} proof-nets:

- assign to each edge as level its depth,
- remove \S boxes.

but LAL_{lev} contains more proof-nets.

in LAL_{lev} we have:

$$\S A \otimes \S B \dashv\vdash \S(A \otimes B),$$

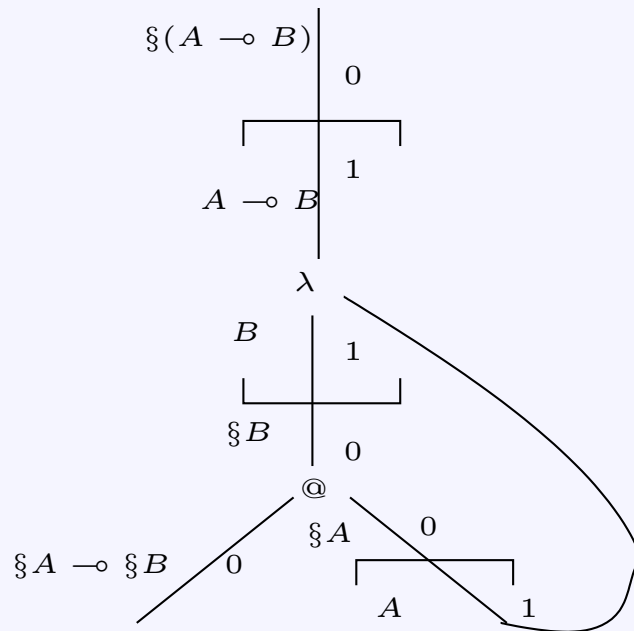
$$\text{so } \S(A \multimap B) \simeq \S A \multimap \S B.$$

Normalization of LAL_{lev} proof-nets: the level of an edge remains unchanged (even if the depth can change).

Example of LAL_{lev} proof-net

$\S A \multimap \S B \vdash \S(A \multimap B)$ provable in LAL_{lev} , but not in LAL .

$f : \S A \multimap \S B \vdash \lambda x.(f x) : \S(A \multimap B)$

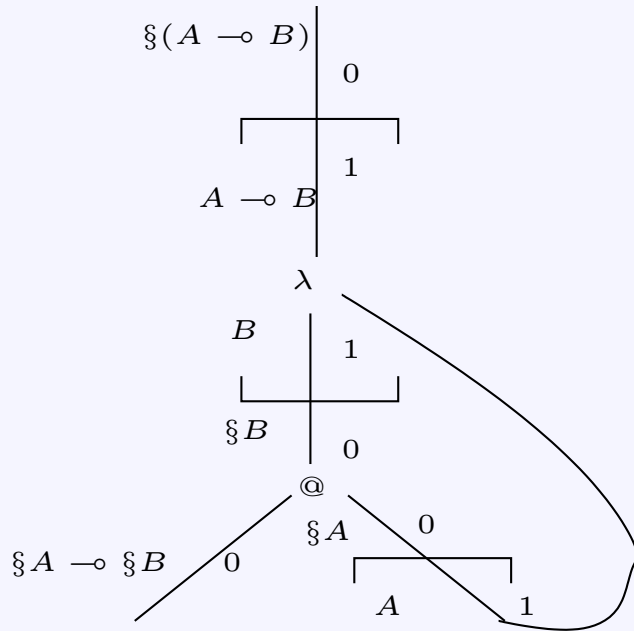


in LAL_{lev}

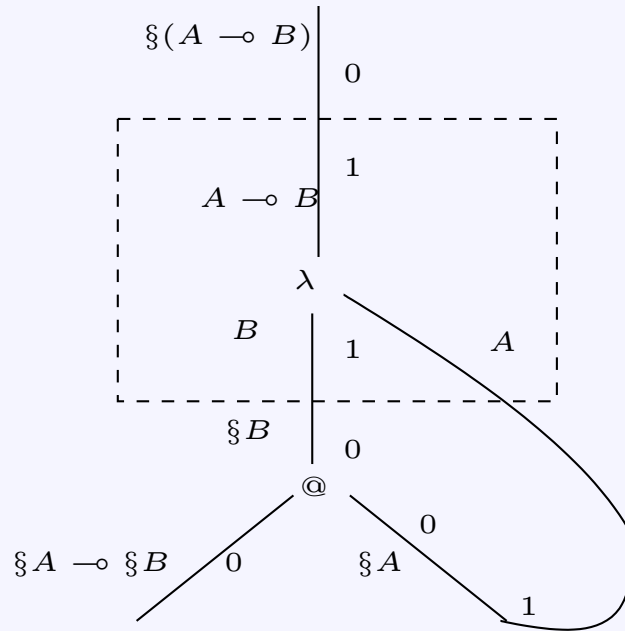
Example of LAL_{lev} proof-net

$\S A \multimap \S B \vdash \S(A \multimap B)$ provable in LAL_{lev} , but not in LAL.

$f : \S A \multimap \S B \vdash \lambda x.(f x) : \S(A \multimap B)$



in LAL_{lev}



tentative box in LAL ? incorrect.

Polynomial time bound

R proof-net.

$l(R)$: maximal level in R , called *level of the proof-net*.

Theorem 9 *Let R be an LAL_{lev} proof-net. It can be normalized in time $O(|R|^{2^{l(R)+1}})$.*

In the case where R comes from an LAL proof-net R_0 we have $l(R) = d(R_0)$ and we recover the previous bound.

As LAL_{lev} contains LAL, it is also complete for Ptime functions.

Moreover:

- more terms are typable,
- type inference should be easier (smaller number of constraints).

Some open questions and further issues

- characterization of other complexity classes (Pspace, Logspace. . .) in lambda calculus ? (Schoepp 07)
- other type systems for Ptime, more expressive intensionally ? (non-size-increasing linear types: Hofmann)
- denotational semantics, game semantics,
- proof systems with extraction to Ptime programs ? (naive Light set theory: Girard, Terui)

Conclusion

- Light logics allow to define subsets of lambda terms with Ptime complexity bounds and efficiently checkable,
- proof-nets play a key role in the design of Light logics or type inference,
- they allow to use path-based techniques to prove complexity bounds, *e.g.* for optimal reduction with Light logics,
- they can guide the design of extensions of the type systems for Ptime complexity.

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.