Complexity and Logic lecture

Part II. Linear logic, lambda-calculus and polynomial time complexity

GEOCAL'06 Winter School

Patrick Baillot

CNRS-LIPN, Université Paris 13

patrick.baillot@lipn.univ-paris13.fr

feasible proofs-as-programs ?

we want to examine Implicit computational (ICC) complexity through a proof-theoretical lens:
framework:

proofs-as-programs
computation=normalization

solid foundations: lambda-calculus, proof-theory, semantics . . .

applications: type systems, proof assistents . . .

how can we delineate complexity classes in this approach?
in particular, how could feasible/Ptime computation fit into it?

feasible proofs-as-programs ?

how can we delineate complexity classes in this approach?
in particular, how could feasible/Ptime computation fit into it?

$\longrightarrow$ **regulate logical operations**

- within second-order approaches:

  2nd order quantification (Leivant 91)
  duplication (Girard 95)

- other proof-theory approaches, using 1st-order logic or arithmetic systems:
  - Bounded arithmetic (Buss 85).
  - ICC feasible arithmetics:
    Marion ’01, Bellantoni-Hofmann’02, Aehlig *et al.* ’04 . . .

. . . which method ?

within the proofs-as-programs approach, our reference language:

2nd order intuitionistic logic,

corresponding to

system F types for lambda-calculus
(polymorphic types)

our analysis will be focused on the duplication regulation approach

we would like complexity properties to result from the *internal* dynamic of the logic (cut-elimination)

$\longrightarrow$ *Linear logic* is a sharp tool to study system F dynamics

. . . what can be changed about LL?

with Linear logic, duplication is managed through a *protocol* specified by modality !:

duplication operation / duplicability

this protocol can be tuned via the choice of ! rules

this way: delineate fragments of F corresponding to Ptime complexity

this presentation will be based on results by

Girard, Asperti, Roversi, Lafont, and others.

about the goals

let us anticipate on what we cannot/can expect.

- lambda-calculus is a basic language, but it allows for general features (higher-order, polymorphism . . .)
  $\longrightarrow$ this approach is *not* directed towards a fine-grained *intensional* characterization
  but we can hope for *robust* conditions
- *internal* chacterization:
  the complexity bounds will result from basic logical/programming operations
  $\longrightarrow$ analogous to safe recursion, tiering approaches
  but different from the QI approach (somehow external condition)

comparison with safe recursion

safe/tiered recursion: *data types* and associated *methods* as primitives

LL aproach: logical operations as primitives

data structures are constructed

=*lambda-calculus-style* approach with

homogeneity between programs and data

Plan

1. system F, Linear logic and overview of LL variants for ICC
2. Elementary and Soft Linear Logics
3. Light Linear Logic

Part 1: system F, LL and overview of LL variants for complexity

- 1.1 Background on λ-calculus, Curry-Howard correspondence
- 1.2 core Linear logic. Proof-nets.
- 1.3 what are the options for duplication ? variants of LL

Setting

From system F to Linear logic

proofs-as-programs approach

- Intuitionistic 2nd-order logic (IL) $\xrightarrow{\text{Curry-Howard.}}$ system F

Setting

From system F to Linear logic

proofs-as-programs approach

- Intuitionistic 2nd-order logic (IL) $\xrightarrow{\text{Curry-Howard.}}$ system F
- (Intuitionistic) linear logic (LL2): refinement of IL
  by Curry-Howard corresp.: formulas=types, proofs=programs

$$\multimap,\ \otimes,\ !(.),\ \forall^{(2)}$$

Examples of LL types:

$N \multimap N$ $\qquad$ $(!N \multimap N) \multimap N$

$!N \multimap N$

Setting

From system F to Linear logic

proofs-as-programs approach

- Intuitionistic 2nd-order logic (IL) $\xrightarrow{\text{Curry-Howard.}}$ system F
- (Intuitionistic) linear logic (LL2): refinement of IL
  by Curry-Howard corresp.: formulas=types, proofs=programs

$$\multimap,\ \otimes,\ !(.),\ \forall^{(2)}$$

Examples of LL types:

$N \multimap N$ $\qquad$ $(!N \multimap N) \multimap N$

$!N \multimap N$

- rules for modality ! responsible for duplication
  $\Rightarrow$ restrict the rules for ! to tame complexity.

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: $t[u/x]$

- 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 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-programs correspondence (Curry-Howard)

typed term	$\Rightarrow$	**2nd-order intuitionistic logic proo**
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)

Sequent calculus for Intuitionistic logic

sequents of the form $B_1, \ldots, B_n \vdash A$

$$\frac{}{A \vdash A}\ Id \qquad \frac{\Gamma_1 \vdash A \quad A, \Gamma_2 \vdash C}{\Gamma_1, \Gamma_2 \vdash C}\ Cut$$

$$\frac{\Gamma_1 \vdash A_1 \quad A_2, \Gamma_2 \vdash C}{\Gamma_1, A_1 \to A_2, \Gamma_2 \vdash C} \to l \qquad \frac{A_1, \Gamma \vdash A_2}{\Gamma \vdash A_1 \to A_2} \to r$$

$$\frac{A[B/\alpha], \Gamma \vdash C}{\forall \alpha.A, \Gamma \vdash C}\ \forall l \qquad \frac{\Gamma \vdash A}{\Gamma \vdash \forall \alpha.A}\ \forall r \ (\alpha \text{ not free in } \Gamma)$$

$$\frac{\Gamma \vdash C}{A, \Gamma \vdash C}\ Weak \qquad \frac{A, A, \Gamma \vdash t : C}{A, \Gamma \vdash C}\ Cntr$$

Sequent calculus and lambda-terms

judgements of the form $x_1 : B_1, \ldots, x_n : B_n \vdash t : A$, where $FV(t) \subseteq \{x_1, \ldots, x_n\}$.

$$\frac{}{x{:}A \vdash x{:}A}\ Id \qquad \frac{\Gamma_1 \vdash u{:}A \quad x{:}A, \Gamma_2 \vdash t{:}C}{\Gamma_1, \Gamma_2 \vdash t[u/x]{:}C}\ Cut$$

$$\frac{\Gamma_1 \vdash u{:}A_1 \quad x{:}A_2, \Gamma_2 \vdash t{:}C}{\Gamma_1, y{:}A_1 \to A_2, \Gamma_2 \vdash t[(y)u/x]{:}C} \to l \qquad \frac{x{:}A_1, \Gamma \vdash t{:}A_2}{\Gamma \vdash \lambda x.t{:}A_1 \to A_2} \to r$$

$$\frac{x{:}A[B/\alpha], \Gamma \vdash t{:}C}{x{:}\forall\alpha.A, \Gamma \vdash t{:}C} \forall l \qquad \frac{\Gamma \vdash t{:}A}{\Gamma \vdash t{:}\forall\alpha.A} \forall r \ (\alpha \text{ not free in } \Gamma)$$

$$\frac{\Gamma \vdash t{:}C}{A, \Gamma \vdash t{:}C}\ Weak \qquad \frac{x{:}A, y{:}A, \Gamma \vdash t{:}C}{z{:}A, \Gamma \vdash t[z/x, z/y]{:}C}\ Cntr$$

In binary rules we assume Γ_1 and Γ_2 have disjoint variables.

Sequent calculus and lambda-terms

judgements of the form $x_1 : B_1, \ldots, x_n : B_n \vdash t : A$, where $FV(t) \subseteq \{x_1, \ldots, x_n\}$.

$$\frac{}{x{:}A \vdash x{:}A}\ Id \qquad \frac{\Gamma_1 \vdash u{:}A \quad x{:}A, \Gamma_2 \vdash t{:}C}{\Gamma_1, \Gamma_2 \vdash t[u/x]{:}C}\ Cut$$

$$\frac{\Gamma_1 \vdash u{:}A_1 \quad x{:}A_2, \Gamma_2 \vdash t{:}C}{\Gamma_1, y{:}A_1 \to A_2, \Gamma_2 \vdash t[(y)u/x]{:}C} \to l \qquad \frac{x{:}A_1, \Gamma \vdash t{:}A_2}{\Gamma \vdash \lambda x.t{:}A_1 \to A_2} \to r$$

$$\frac{x{:}A[B/\alpha], \Gamma \vdash t{:}C}{x{:}\forall\alpha.A, \Gamma \vdash t{:}C} \forall l \qquad \frac{\Gamma \vdash t{:}A}{\Gamma \vdash t{:}\forall\alpha.A} \forall r \ (\alpha \text{ not free in } \Gamma)$$

$$\frac{\Gamma \vdash t{:}C}{A, \Gamma \vdash t{:}C}\ Weak \qquad \frac{x{:}A, y{:}A, \Gamma \vdash t{:}C}{z{:}A, \Gamma \vdash t[z/x, z/y]{:}C}\ Cntr$$

In binary rules we assume Γ_1 and Γ_2 have disjoint variables.

a derivable rule:

$$\frac{\Gamma_1 \vdash v{:}A_1 \to A_2 \quad \Gamma_2 \vdash u{:}A_1}{\Gamma_1, \Gamma_2 \vdash (v)\ u{:}A_2}$$

Examples of F types

Polymorphic identity:

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

Tally integers:

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

example

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

Binary words:

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

example

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

Example of derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{x:\alpha \vdash x:\alpha \qquad y:\alpha \vdash y:\alpha}{f_2:\alpha\to\alpha, x:\alpha \vdash (f_2)\, x:\alpha}\to \mathsf{I} \qquad z:\alpha \vdash z:\alpha}{f_1:\alpha\to\alpha, f_2:\alpha\to\alpha, x:\alpha \vdash (f_1)\,(f_2)\, x:\alpha}\to \mathsf{I}}{f:\alpha\to\alpha, f:\alpha\to\alpha, x:\alpha \vdash (f)\,(f)\, x:\alpha}\ \text{contr}}{\vdash \lambda f x.(f)\,(f)\, x : (\alpha\to\alpha)\to(\alpha\to\alpha)}\ 2}{\vdash \underline{2} : \forall\alpha.(\alpha\to\alpha)\to(\alpha\to\alpha)}\ \forall$$

so

$$\vdash \underline{2} : N$$

.

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 for any A an iterator $iter_A$:

$$iter_A = \lambda f x n.\ (n)\ f\ x\ \ : (A \to A) \to A \to N \to A$$

then $(iter_A)\ F\ t\ \underline{n} \to (F)\ (F) \ldots (F)\ t$ (n times)

example:

$double : N \to N$

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

Examples of terms

concatenation

$$conc \quad = \quad \lambda u^W.\lambda v^W.\lambda o.\lambda z.\lambda x.((u)\ o\ z)\ (v)\ o\ z\ x$$

$$: \quad W \to W \to W$$

length

$$length \quad = \quad \lambda u^W.\lambda f^{\alpha\to\alpha}.(u)\ f\ f^{\alpha\to\alpha}$$

$$: \quad W \to N$$

'multiplication'

$$mult \quad = \quad \lambda n^N.\lambda v^W.[(n)\ (conc)\ v)]\ \underline{nil}^W$$

$$: \quad N \to W \to W$$

System F and termination

Theorem 0 (Girard 1972) *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?

Core Intuitionistic Linear logic

Core Intuitionistic Linear logic (IMLL) formulas:

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

$A \multimap B$: *use-once* implication

$A \otimes B$: conjunction (pair of resources)

IMLL$_{\multimap}$: fragment without $\otimes$

Core Intuitionistic Linear Logic: rules

system IMLL$_{\multimap}$:

$$\frac{}{x:A \vdash x:A}\ Id \qquad \frac{\Gamma_1 \vdash u:A \quad x:A, \Gamma_2 \vdash t:C}{\Gamma_1, \Gamma_2 \vdash t[u/x]:C}\ Cut$$

$$\frac{\Gamma_1 \vdash u:A_1 \quad x:A_2, \Gamma_2 \vdash t:C}{\Gamma_1, y:A_1 \multimap A_2, \Gamma_2 \vdash t[(y\ u)/x]:C}\ \multimap l \qquad \frac{x:A_1, \Gamma \vdash t:A_2}{\Gamma \vdash \lambda x.t:A_1 \multimap A_2}\ \multimap r$$

$$\frac{x:A[B/\alpha], \Gamma \vdash t:C}{x:\forall\alpha.A, \Gamma \vdash t:C}\ \forall l \qquad \frac{\Gamma \vdash t:A}{\Gamma \vdash t:\forall\alpha.A}\ \forall r \ (\alpha \text{ non free in } \Gamma)$$

Typeable terms are . . . *linear*, in the sense that each variable occurs exactly once.

Core Intuitionistic Linear Logic: rules

system IMLL$_{\multimap}$:

$$\frac{}{x:A \vdash x:A}\ Id \qquad \frac{\Gamma_1 \vdash u:A \quad x:A, \Gamma_2 \vdash t:C}{\Gamma_1, \Gamma_2 \vdash t[u/x]:C}\ Cut$$

$$\frac{\Gamma_1 \vdash u:A_1 \quad x:A_2, \Gamma_2 \vdash t:C}{\Gamma_1, y:A_1 \multimap A_2, \Gamma_2 \vdash t[(y\ u)/x]:C}\ \multimap l \qquad \frac{x:A_1, \Gamma \vdash t:A_2}{\Gamma \vdash \lambda x.t:A_1 \multimap A_2}\ \multimap r$$

$$\frac{x:A[B/\alpha], \Gamma \vdash t:C}{x:\forall\alpha.A, \Gamma \vdash t:C}\ \forall l \qquad \frac{\Gamma \vdash t:A}{\Gamma \vdash t:\forall\alpha.A}\ \forall r\ \ (\alpha \text{ non free in } \Gamma)$$

Typeable terms are . . . *linear*, in the sense that each variable occurs exactly once.

We can also consider the *affine* variant with weakening:

$$\frac{\Gamma \vdash t:C}{x:A, \Gamma \vdash t:C}\ Weak$$

corresponds to lambda-terms where variables occur at most once.

Classical multiplicative Linear logic (MLL)

Classical MLL formulas:

$$A, B ::= \alpha \mid \alpha^{\perp} \mid A \wp B \mid A \otimes B \mid \forall \alpha.A \mid \exists \alpha.A$$

linear negation $A^{\perp}$.

$A \wp B$: *par* (disjunction)

$A \multimap B =^{def} A^{\perp} \wp B$

$A^{\perp}$ defined by:

$$(A \otimes B)^{\perp} = A^{\perp} \wp B^{\perp} \quad (A \wp B)^{\perp} = A^{\perp} \otimes B^{\perp}$$
$$(\forall \alpha.A)^{\perp} = \exists \alpha.A^{\perp} \quad (\exists \alpha.A)^{\perp} = \forall \alpha.A^{\perp}$$

replace sequents $A_1, \ldots, A_n \vdash B$ by $\vdash A_1^{\perp}, \ldots, A_n^{\perp}, B$

Classical multiplicative Linear logic

- **Identity Group**

Axiom $\dfrac{}{\vdash A^{\perp}, A}$ **ax**

Cut $\dfrac{\vdash A, \Delta \quad \vdash A^{\perp}, \Delta'}{\vdash \Delta, \Delta'}$ **cu**

- **Logical Rules** *(multiplicatives and second-order quantifiers)*

Par $\dfrac{\vdash A, B, \Delta}{\vdash A \wp B, \Delta}\ \wp$

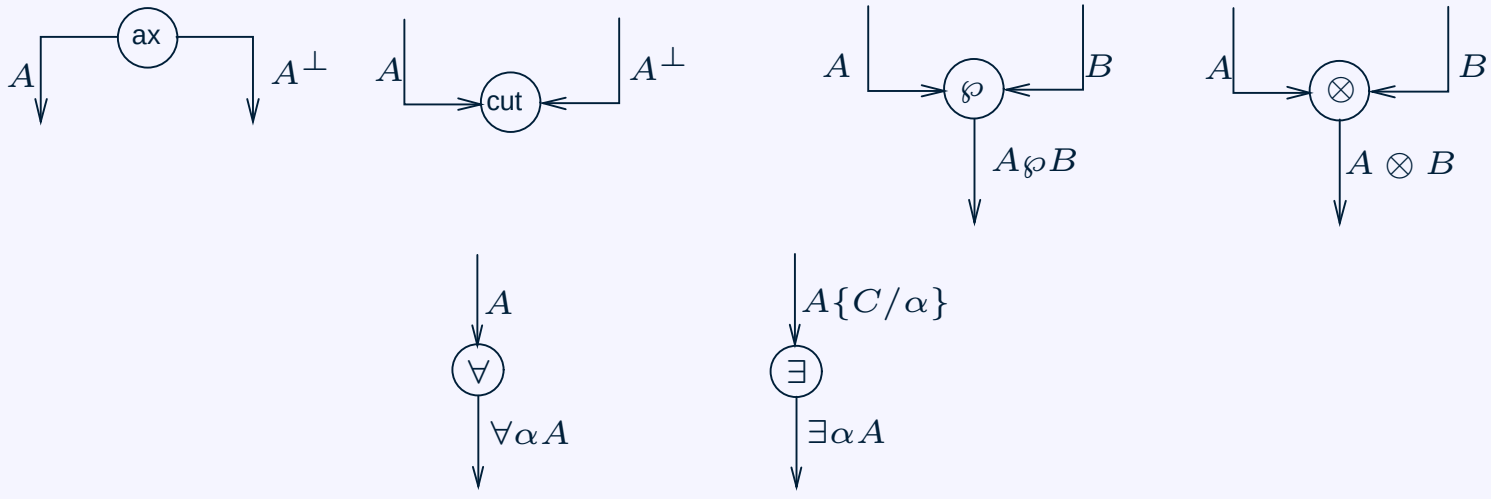
Tensor $\dfrac{\vdash A, \Delta \quad \vdash B, \Delta'}{\vdash A \otimes B, \Delta, \Delta'}\ \otimes$

Universal $\dfrac{\vdash A, \Delta}{\vdash \forall \alpha A, \Delta}\ \forall$

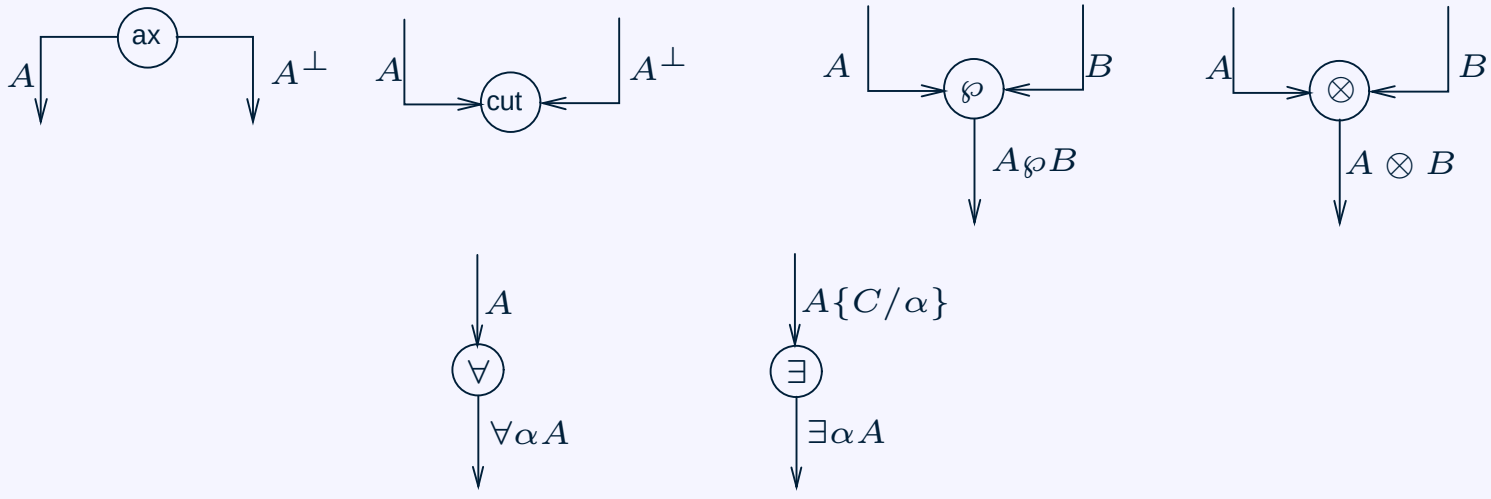
Existential $\dfrac{\vdash A[C/\alpha], \Delta}{\vdash \exists \alpha A, \Delta}\ \exists$

provided α is not free in Δ

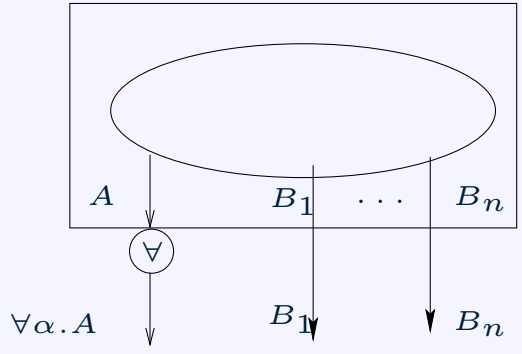
MLL proof-structures



MLL proof-structures



each $\forall$ node is associated with a *box*:



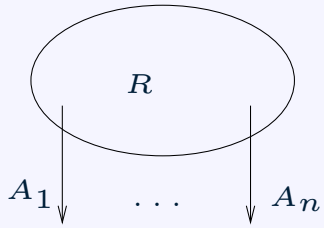
2 boxes are either disjoint, or one is included in the other

Translation of proofs

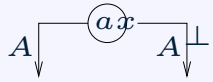
proof Π

translation $\Pi^* = R$

$$\frac{\Pi}{\vdash A_1, \ldots, A_n}$$

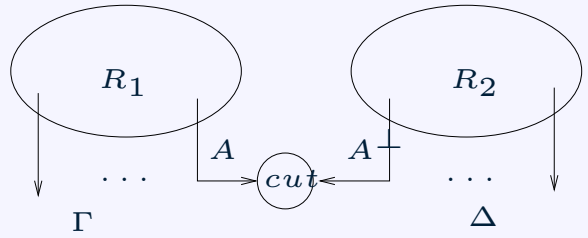


$$\frac{}{\vdash A, A^{\perp}}\ ax$$



$$\frac{\dfrac{\Pi_1}{\vdash \Gamma, A} \quad \dfrac{\Pi_2}{\vdash A^{\perp}, \Delta}}{\vdash \Gamma, \Delta}\ \text{cut}$$

$\Pi_i^* = R_i,\ i = 1, 2$



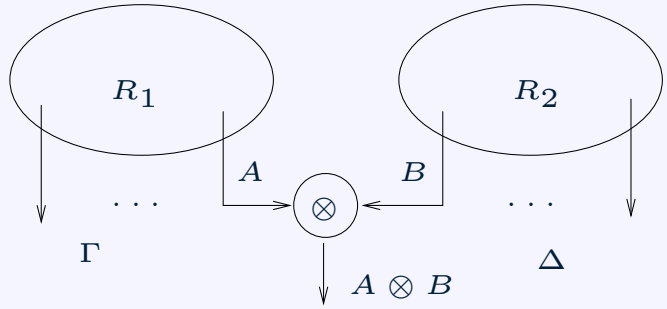
Translation of proofs (continued)

proof Π

translation $\Pi^* = R$

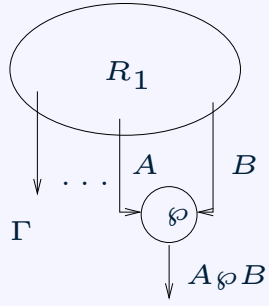
$$\frac{\dfrac{\Pi_1}{\vdash \Gamma, A} \quad \dfrac{\Pi_2}{\vdash B, \Delta}}{\vdash \Gamma, A \otimes B, \Delta}\ \otimes$$

$\Pi_i^* = R_i,\ i = 1, 2$



$$\frac{\dfrac{\Pi_1}{\vdash \Gamma, A, B}}{\vdash \Gamma, A \wp B}\ \wp$$

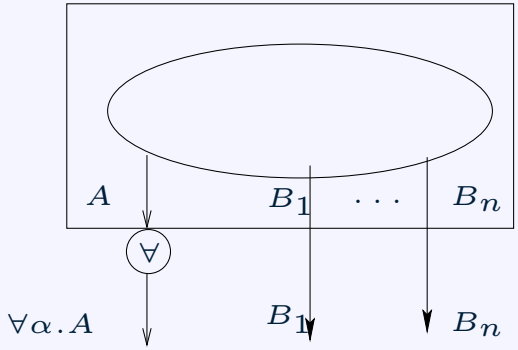
$\Pi_1^* = R_1$



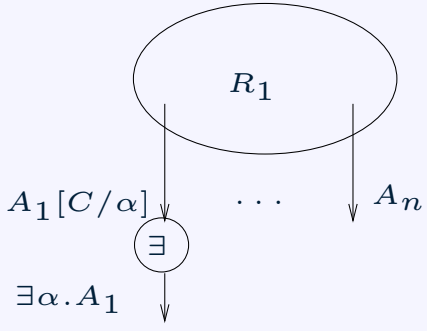
Translation of proofs (continued)

proof Π translation $\Pi^* = R$

$$\dfrac{\dfrac{\Pi_1}{\vdash A, B_1, \ldots, B_n}}{\vdash \forall\alpha.A, B_1, \ldots, B_n}\ \forall \qquad \Pi_1^* = R_1$$



$$\dfrac{\dfrac{\Pi_1}{\vdash A_1[C/\alpha], A_2, \ldots, A_n}}{\vdash \exists\alpha.A_1, A_2, \ldots, A_n}\ \exists \qquad \Pi_1^* = R_1$$

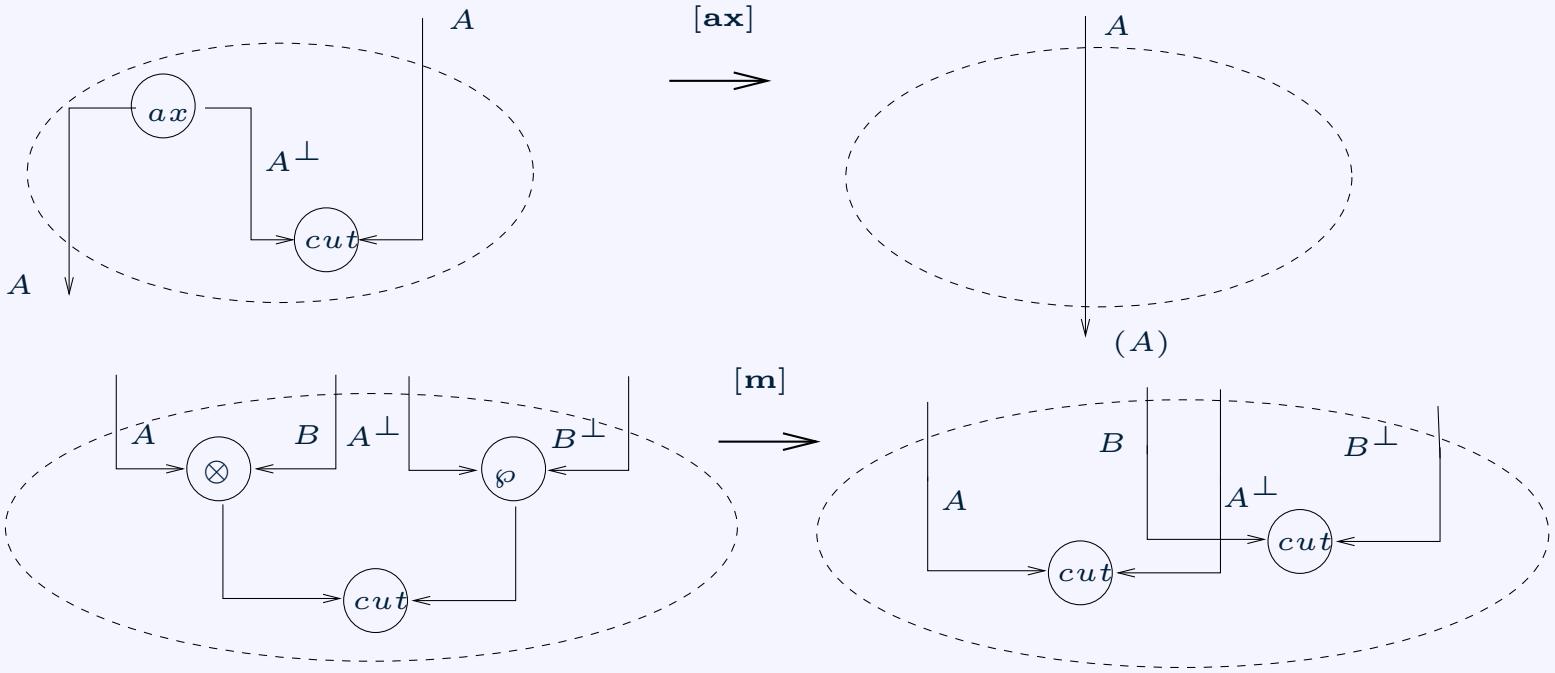


MLL proof-nets

A proof-structure R is called a *proof-net* iff there exists a proof Π such that $\Pi^* = R$.

Proofs-nets can also be characterized among proof-structures in an intrinsic way, by *correctness criterions*. However, this will not be our concern here.

Reduction of proof-nets (cut-elimination)



Example

$t = (\lambda x.(x)y)\ \lambda z.z$

$$\dfrac{\dfrac{\dfrac{y:\alpha \vdash y:\alpha \qquad a:\alpha \vdash a:\alpha}{x:\alpha \multimap \alpha, y:\alpha \vdash (x)\ y:\alpha}\multimap \mathsf{I}}{y:\alpha \vdash \lambda x.(x)\ y:(\alpha \multimap \alpha) \multimap \alpha} \qquad \dfrac{\dfrac{z:\alpha \vdash z:\alpha}{\vdash \lambda z.z:\alpha \multimap \alpha} \qquad w:\alpha \vdash w}{z':(\alpha \multimap \alpha) \multimap \alpha \vdash (z')\ \lambda z.z}}{y:\alpha \vdash (\lambda x.(x)y)\ \lambda z.z:\alpha}\ \text{cut}$$

Example

$t = (\lambda x.(x)y)\ \lambda z.z$

$$\dfrac{\dfrac{\dfrac{\alpha \vdash \alpha \qquad \alpha \vdash \alpha}{\alpha \multimap \alpha, \alpha \vdash \alpha}\ \multimap \mathsf{l}}{\alpha \vdash (\alpha \multimap \alpha) \multimap \alpha}\ \multimap \mathsf{r} \qquad \dfrac{\dfrac{\dfrac{\alpha \vdash \alpha}{\vdash \alpha \multimap \alpha}\ \multimap \mathsf{r} \qquad \alpha \vdash \alpha}{(\alpha \multimap \alpha) \multimap \alpha \vdash \alpha}\ \multimap \mathsf{l}}{\alpha \vdash \alpha}\ \mathsf{cut}$$

Example

$t = (\lambda x.(x)y)\ \lambda z.z$

$$\dfrac{\dfrac{\dfrac{\alpha \vdash \alpha \qquad \alpha \vdash \alpha}{\alpha \multimap \alpha, \alpha \vdash \alpha}\ \multimap \mathsf{l}}{\alpha \vdash (\alpha \multimap \alpha) \multimap \alpha}\ \multimap \mathsf{r} \qquad \dfrac{\dfrac{\alpha \vdash \alpha}{\vdash \alpha \multimap \alpha} \qquad \alpha \vdash \alpha}{(\alpha \multimap \alpha) \multimap \alpha \vdash \alpha}\ \multimap \mathsf{l}}{\alpha \vdash \alpha}\ \mathsf{cut}\ \multimap \mathsf{r}$$

in MLL:

$$\dfrac{\dfrac{\dfrac{\vdash \alpha^{\perp}, \alpha \qquad \vdash \alpha^{\perp}, \alpha}{\vdash (\alpha \multimap \alpha)^{\perp}, \alpha^{\perp}, \alpha}}{\vdash \alpha^{\perp}, (\alpha \multimap \alpha)^{\perp} \wp \alpha} \qquad \dfrac{\dfrac{\vdash \alpha^{\perp}, \alpha}{\vdash \alpha^{\perp} \wp \alpha} \qquad \vdash \alpha^{\perp}, \alpha}{\vdash ((\alpha \multimap \alpha) \multimap \alpha)^{\perp}, \alpha}}{\vdash \alpha^{\perp}, \alpha}\ \mathsf{cut}$$

Example

$t = (\lambda x.(x)y)\ \lambda z.z$

$$\dfrac{\dfrac{\dfrac{\alpha \vdash \alpha \qquad \alpha \vdash \alpha}{\alpha \multimap \alpha, \alpha \vdash \alpha}\ \multimap \mathsf{l}}{\alpha \vdash (\alpha \multimap \alpha) \multimap \alpha}\ \multimap \mathsf{r} \qquad \dfrac{\dfrac{\alpha \vdash \alpha}{\vdash \alpha \multimap \alpha} \qquad \alpha \vdash \alpha}{(\alpha \multimap \alpha) \multimap \alpha \vdash \alpha}\ \multimap \mathsf{l}}{\alpha \vdash \alpha}\ \mathsf{cut}$$

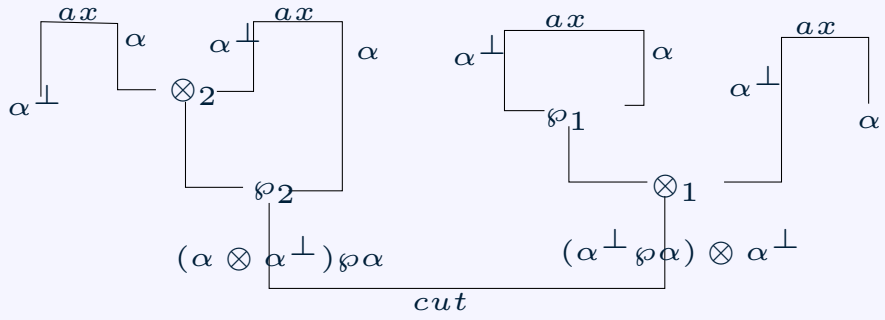
in MLL:

$$\dfrac{\dfrac{\dfrac{\vdash \alpha^\perp, \alpha \qquad \vdash \alpha^\perp, \alpha}{\vdash (\alpha \otimes \alpha^\perp), \alpha^\perp, \alpha}\ \otimes}{\vdash \alpha^\perp, (\alpha \otimes \alpha^\perp) \wp \alpha} \qquad \dfrac{\dfrac{\vdash \alpha^\perp, \alpha}{\vdash \alpha^\perp \wp \alpha} \qquad \vdash \alpha^\perp, \alpha}{\vdash ((\alpha^\perp \wp \alpha) \otimes \alpha^\perp), \alpha}\ \otimes}{\vdash \alpha^\perp, \alpha}\ \mathsf{cut}$$

Example

$$t = (\lambda x^{\alpha \multimap \alpha}.[(x)y]^{\alpha})\ \lambda z^{\alpha}.z\ : \alpha$$

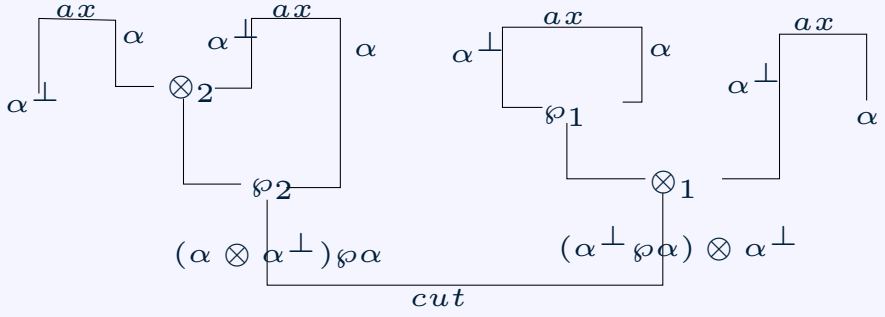
$$\text{cut}\ \dfrac{\dfrac{\dfrac{\vdash\ \alpha^{\perp}, \alpha \qquad \vdash\ \alpha^{\perp}, \alpha}{\vdash\ (\alpha \otimes \alpha^{\perp}), \alpha^{\perp}, \alpha}\otimes_2}{\vdash\ \alpha^{\perp}, (\alpha \otimes \alpha^{\perp}) \wp \alpha}\wp_2 \qquad \dfrac{\dfrac{\dfrac{\vdash\ \alpha^{\perp}, \alpha}{\vdash\ \alpha^{\perp} \wp \alpha}\wp_1 \qquad \vdash\ \alpha^{\perp}, \alpha}{\vdash\ ((\alpha^{\perp} \wp \alpha) \otimes \alpha^{\perp}), \alpha}\otimes_1}{\vdash\ \alpha^{\perp}, \alpha}$$

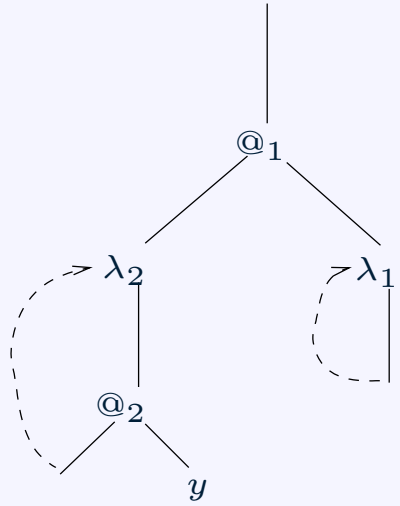


Example

$$t = (\lambda x^{\alpha \multimap \alpha}.[(x)y]^{\alpha})\ \lambda z^{\alpha}.z\ : \alpha$$

$$\text{cut}\ \dfrac{\dfrac{\dfrac{\vdash\ \alpha^{\perp}, \alpha \qquad \vdash\ \alpha^{\perp}, \alpha}{\vdash\ (\alpha \otimes \alpha^{\perp}), \alpha^{\perp}, \alpha}\otimes_2}{\vdash\ \alpha^{\perp}, (\alpha \otimes \alpha^{\perp}) \wp \alpha}\wp_2 \qquad \dfrac{\dfrac{\dfrac{\vdash\ \alpha^{\perp}, \alpha}{\vdash\ \alpha^{\perp} \wp \alpha}\wp_1 \qquad \vdash\ \alpha^{\perp}, \alpha}{\vdash\ ((\alpha^{\perp} \wp \alpha) \otimes \alpha^{\perp}), \alpha}\otimes_1}{\vdash\ \alpha^{\perp}, \alpha}$$





Bound on reduction of MLL proof-nets

Theorem 0 (Small normalization theorem) *Any MLL proof-net R can be reduced in less than $|R|$ steps.*

. . . because each reduction step makes $|R|$ decrease.

Thus MLL admits a polynomial time dynamics.
However it is not expressive enough, while on the other hand F is *too* expressive.

In between MLL and F: reintroduce duplication

Linear logic idea to control duplication: specific connective= modality !

(called *exponential*)

2 issues:

In between MLL and F: reintroduce duplication

Linear logic idea to control duplication: specific connective= modality !
(called *exponential*)
2 issues:

	how to use duplicable ressources	how to produce them
Linear Logic:	*contraction*	
	$!A \multimap (!A \otimes !A)$ *(contr)*	$\dfrac{A_1 \otimes \ldots \otimes A_n \multimap B}{!A_1 \otimes \ldots \otimes !A_n \multimap !B}$ (!)
	$!A \multimap 1$	
	$!A \multimap A$	$!A \multimap !!A$
	dereliction	*digging*

In between MLL and F: reintroduce duplication

Linear logic idea to control duplication: specific connective= modality !
(called *exponential*)
2 issues:

	how to use duplicable ressources	how to produce them
Elementary LL:	$!A \multimap (!A \otimes !A)$ *(contr)*	
	$!A \multimap 1$	$\dfrac{A_1 \otimes \ldots \otimes A_n \multimap B}{!A_1 \otimes \ldots \otimes !A_n \multimap !B}$ (!)
Soft LL:	*multiplexing*	
	$!A \multimap \underbrace{(A \otimes \ldots \otimes A)}$ *(mplex)*	
	n copies of A, for any $n \geq 0$	$\dfrac{A_1 \otimes \ldots \otimes A_n \multimap B}{!A_1 \otimes \ldots \otimes !A_n \multimap !B}$ (!)
Light LL:	$!A \multimap (!A \otimes !A)$ *(contr)*	$\dfrac{A_1 \multimap B}{!A_1 \multimap !B}$ (u !)
	$!A \multimap 1$	and a new modality §

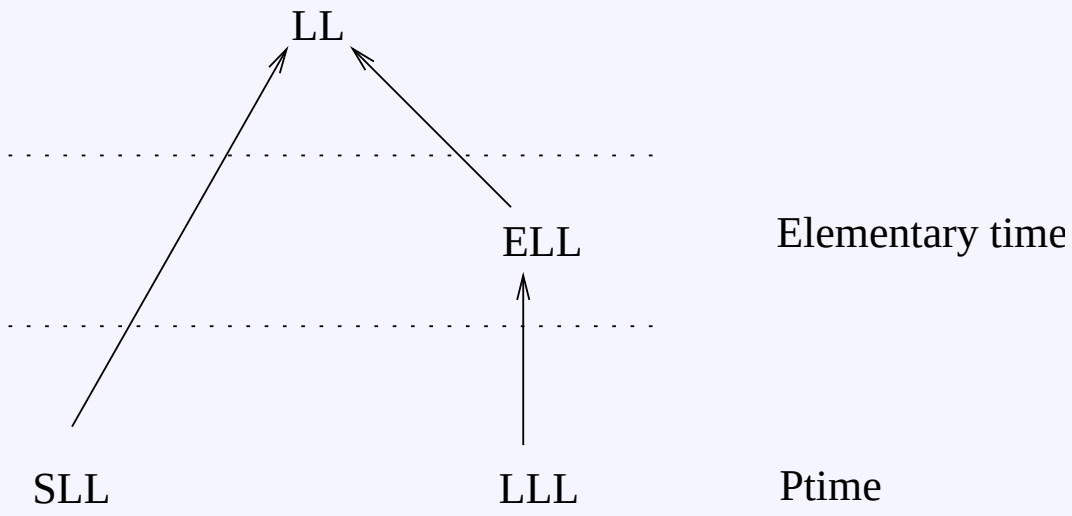
Properties

- ELL has an *elementary time* cut-elimination procedure, that is to say with time bounded by a function $2^{2^{\cdots^{2^n}}}$, with fixed height
  $\Rightarrow$ represents only elementary functions
- SLL, LLL have polynomial time cut-elimination procedure
  $\Rightarrow$ represent only Ptime functions
- LL is as expressive as system F

to make these claims precise we will need to specify the parameters varying in proofs.

Remark: another (earlier) variant of LL for Ptime: *Bounded Linear Logic* ([GSS92]).

Relations between these systems



Relating these systems to F

- LL (or ILL) is as expressive as system F:
  Girard translation:

$$\begin{array}{ccc} F & \longrightarrow & ILL \\ A \to B & & !A^\star \multimap B^\star \end{array}$$

  gives a simulation
- For all these systems there is a forgetful map:

$$\begin{array}{ccc} ILL & \longrightarrow & F \\ ELL & & \\ SLL & & \\ LLL & & \end{array}$$

  obtained by erasing modalities and replacing $\multimap$ with $\to$.

Method

we adopt the following viewpoint:

source language:	lambda-calculus
type system:	ELL/SLL/LLL. . .
intermediary language:	proof-nets

as affine variants are easier to handle we will consider
Elementary (res. Soft, Light) Affine Logic: EAL (resp. SAL, LAL).

Summary

$$\begin{array}{rcrcrcl}
\text{(contr)} & + & \text{(!)} & & & \equiv & \text{Elementary} \quad \text{(ELL)} \\
\text{(mplex)} & + & \text{(!)} & & & \equiv & \text{Ptime} \quad \text{(SLL)} \\
\text{(contr)} & + & \text{(u !)} & + & (\S) & \equiv & \text{Ptime} \quad \text{(LLL)}
\end{array}$$

how to use duplicable resources / how to produce them

Part 2: Elementary and Soft Linear Logics

- 2.1 Elementary Linear logic (ELL)
  - the system
  - proof-nets and normalization
  - representation issues
- 2.2 Soft Linear logic (SLL)

Presentation

for each system (ELL, SLL . . .) we will proceed as follows:

- define it as a sequent calculus/ typing system
- give data types
- study its dynamics with proof-nets
- state soundness and completeness complexity results

Elementary Linear Logic (ELL) [Girard98]

- corresponds to *elementary recursive* functions : time $2^{2^{\cdots^{2^n}}}$, with fixed height. . .
- applications to *optimal reduction*: simplification of Lamping's algorithm.

ELL : definition

we consider here the intuitionistic version of ELL, with second-order (à la système F).
the language of formulas is that of ILL (without additives)

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

ELL is obtained by removing from ILL the rules for dereliction and digging
the other rules are unchanged.

actually this is the *multiplicative* fragment of ELL, but it will be sufficient

for our purposes.

ELL : remarks

- contrarily to ILL/CLL : infinite number of modalities (sequences of $!, ?$) up to equivalence;
- to have more flexibility, we can add general weakening: Elementary *Affine* Logic (EAL);
- for computational intuition: we see ELL/EAL as a type system for λ-calculus.

EAL as a sequent calculus

$$\frac{}{A \vdash A}\ Id \qquad \frac{\Gamma_1 \vdash A \quad A, \Gamma_2 \vdash C}{\Gamma_1, \Gamma_2 \vdash C}\ Cut$$

$$\frac{\Gamma_1 \vdash A_1 \quad A_2, \Gamma_2 \vdash C}{\Gamma_1, A_1 \multimap A_2, \Gamma_2 \vdash C}\ \multimap l \qquad \frac{A_1, \Gamma \vdash A_2}{\Gamma \vdash A_1 \multimap A_2}\ \multimap r$$

$$\frac{A[B/\alpha], \Gamma \vdash C}{\forall \alpha.A, \Gamma \vdash C}\ \forall l \qquad \frac{\Gamma \vdash A}{\Gamma \vdash \forall \alpha.A}\ \forall r \ (\alpha \text{ not free in } \Gamma)$$

$$\frac{\Gamma \vdash C}{\Delta, \Gamma \vdash C}\ Weak \qquad \frac{!A, !A, \Gamma \vdash C}{!A, \Gamma \vdash C}\ Cntr$$

$$\frac{\Gamma \vdash A}{!\Gamma \vdash !A}\ !r$$

EAL and λ-calculus

$$\frac{}{x:A \vdash x:A}\ Id \qquad \frac{\Gamma_1 \vdash u:A \quad x:A, \Gamma_2 \vdash t:C}{\Gamma_1, \Gamma_2 \vdash t[u/x]:C}\ Cut$$

$$\frac{\Gamma_1 \vdash u:A_1 \quad x:A_2, \Gamma_2 \vdash t:C}{\Gamma_1, y:A_1 \multimap A_2, \Gamma_2 \vdash t[(y)u/x]:C}\ \multimap l \qquad \frac{x:A_1, \Gamma \vdash t:A_2}{\Gamma \vdash \lambda x.t:A_1 \multimap A_2}\ \multimap r$$

$$\frac{x:A[B/\alpha], \Gamma \vdash t:C}{x:\forall\alpha.A, \Gamma \vdash t:C}\ \forall l \qquad \frac{\Gamma \vdash t:A}{\Gamma \vdash t:\forall\alpha.A}\ \forall r \ (\alpha \text{ not free in } \Gamma)$$

$$\frac{\Gamma \vdash t:C}{\Delta, \Gamma \vdash t:C}\ Weak \qquad \frac{x:!A, y:!A, \Gamma \vdash t:C}{z:!A, \Gamma \vdash t[z/x, z/y]:C}\ Cntr$$

$$\frac{\Gamma \vdash t:A}{!\Gamma \vdash t:!A}\ !r$$

Forgetful map: from EAL to F

map $(.)^- : EAL \to F$ defined by:

$$(!A)^- = A^-, \quad (A \multimap B)^- = A^- \to B^-,$$

Proposition 0 *If* $\Gamma \vdash_{EAL} t : A$ *then* $\Gamma^- \vdash_F t : A^-$.

If $A^- = T \quad : A$ is called a *décoration* of T in EAL.

Data types in EAL

- unary integers

F:

$$N^F$$

$$\forall\alpha.(\alpha \to \alpha) \to (\alpha \to \alpha)$$

EAL:

$$N^{EAL}$$

$$\forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha)$$

Example: 2, in F: $\underline{2} = \lambda f^{(\alpha\to\alpha)}.\lambda x^{\alpha}.(f)\ (f)\ x\ .$

- binary lists

F:

$$W^F$$

$$\forall\alpha.(\alpha \to \alpha) \to (\alpha \to \alpha) \to (\alpha \to \alpha)$$

EAL:

$$W^{EAL}$$

$$\forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha)$$

Example: $w = [1, 0, 0]$, in F:

$$\underline{w} = \lambda z^{(\alpha\to\alpha)}.\lambda u^{(\alpha\to\alpha)}.\lambda x^{\alpha}.(u)(z)(z)x\ .$$

Classical ELL formulas

we will represent proofs by proof-structures of *classical* ELL
the grammar of classical ELL formulas is given by

$$A, B ::= \alpha \mid \alpha^{\perp} \mid A \wp B \mid A \otimes B \mid !A \mid ?A \mid \forall \alpha.A \mid \exists \alpha.A$$

$A \multimap B = A^{\perp} \wp B$

$A^{\perp}$ is defined by:

$$(A \otimes B)^{\perp} = A^{\perp} \wp B^{\perp} \qquad (A \wp B)^{\perp} = A^{\perp} \otimes B^{\perp}$$
$$(\forall \alpha.A)^{\perp} = \exists \alpha.A^{\perp} \qquad (\exists \alpha.A)^{\perp} = \forall \alpha.A^{\perp}$$
$$(!A)^{\perp} = ?A^{\perp} \qquad (?A)^{\perp} = !A^{\perp}$$
$$\alpha^{\perp\perp} = \alpha$$

Modality rules for classical ELL

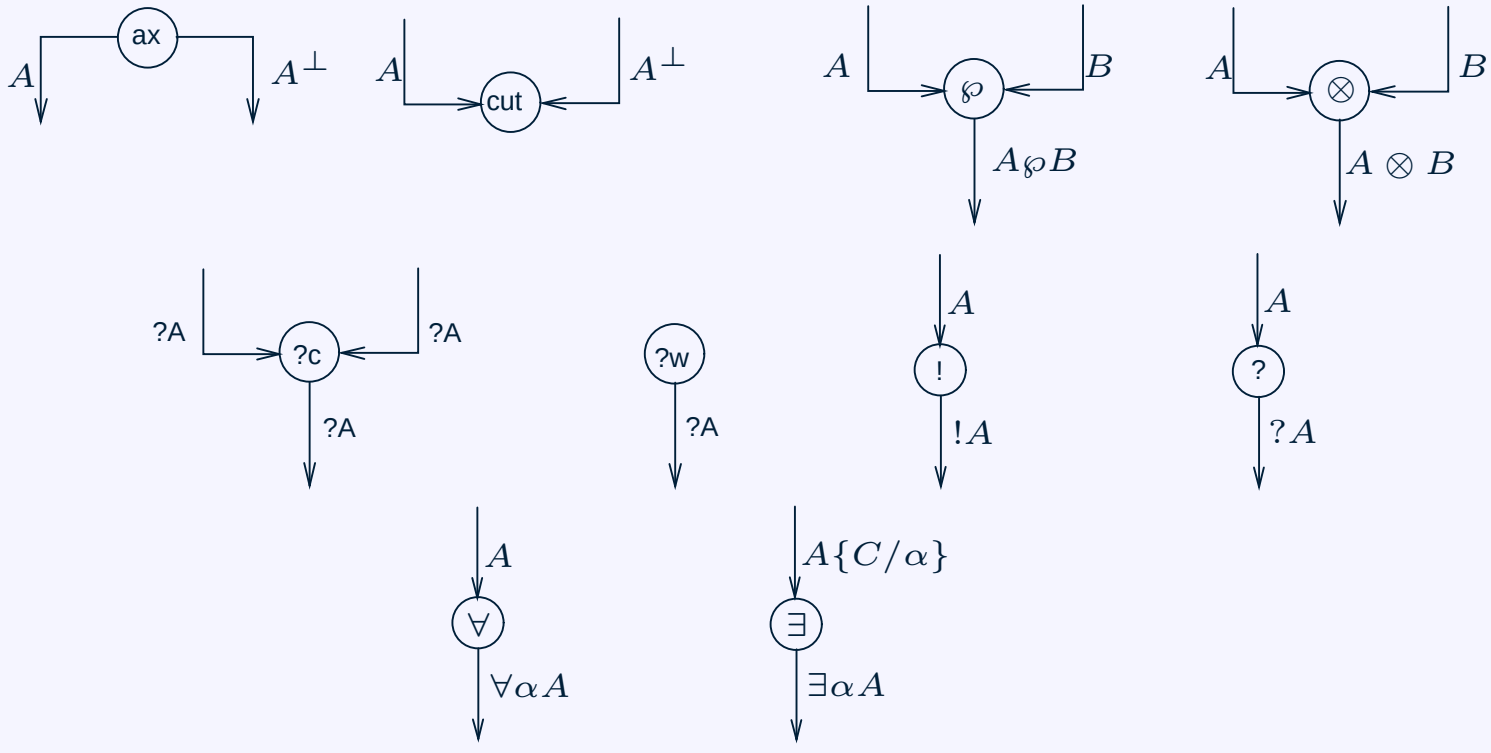
$$\frac{\vdash \Gamma}{\vdash ?A, \Gamma}\ Weak \qquad \frac{\vdash ?A, ?A, \Gamma}{\vdash ?A, \Gamma}\ Cntr$$

$$\frac{\vdash A, B_1, \ldots, B_n}{\vdash\ !A, ?B_1, \ldots, ?B_n}\ !$$

We translate (intuitionistic) ELL derivations into proof-nets, via their translation into classical ELL derivations:

$$B_1, \ldots, B_n \vdash A \quad \longrightarrow \quad \vdash B_1^{\perp}, \ldots, B_n^{\perp}, A$$

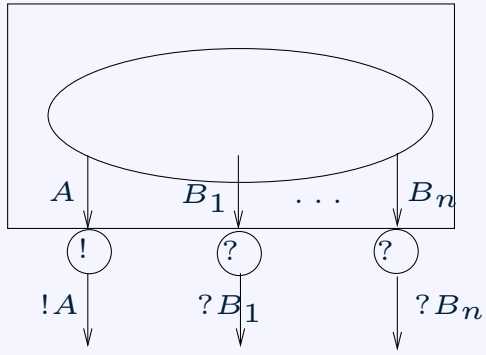
ELL proof-structures



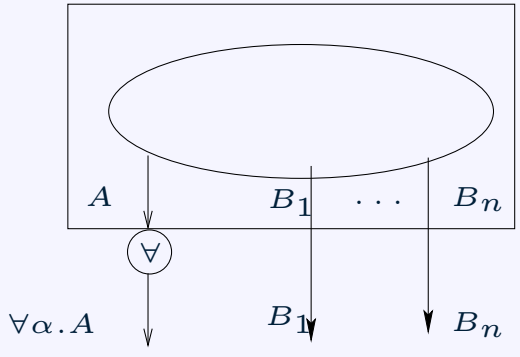
each !, ? node (resp. $\forall$ node) is a door of an exponential (resp. quantifier) box

Proof-structures: boxes

- exponential box (!-box)



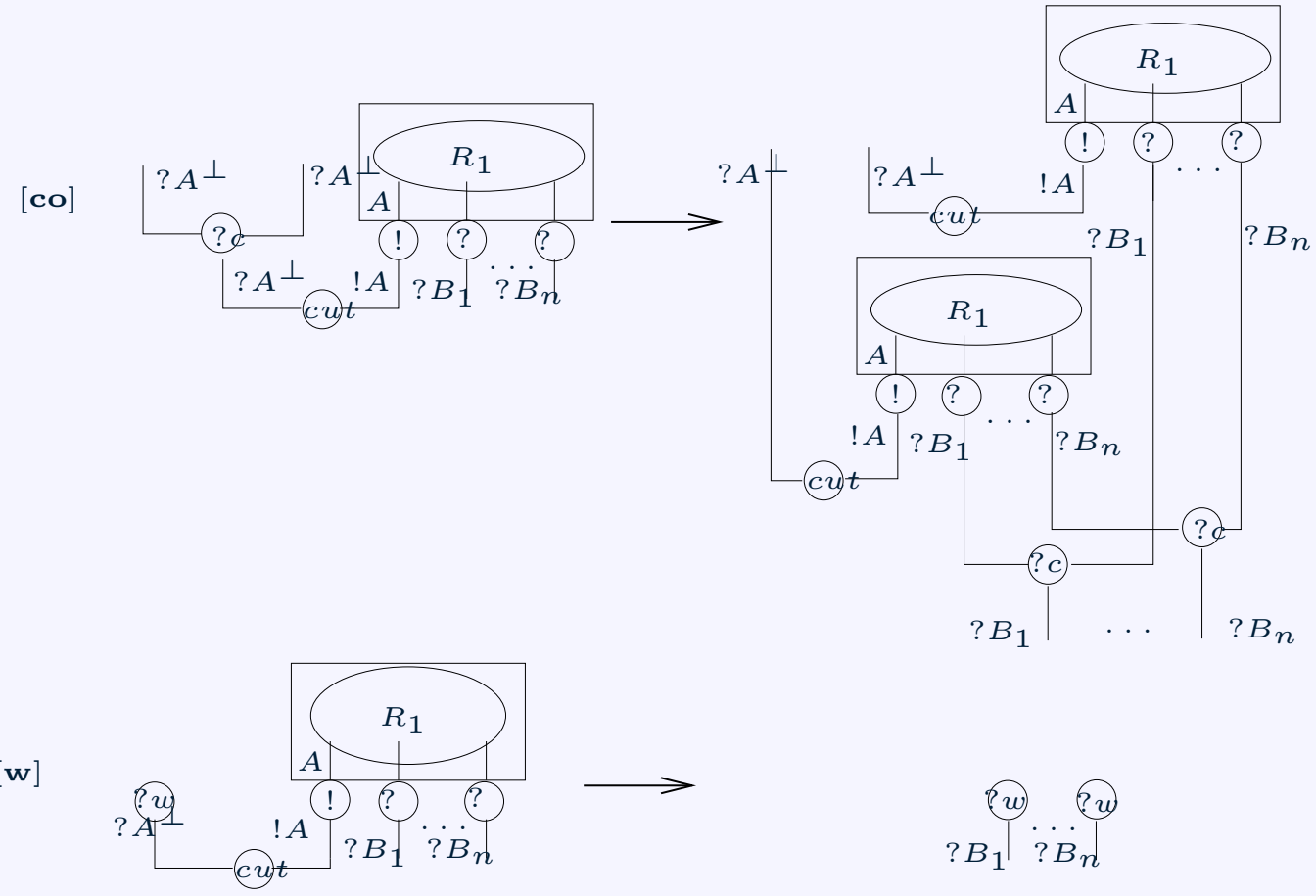
- box $\forall$: $\alpha \notin FV(B_i), 1 \leq i \leq n$



two boxes are disjoint, or one is included in the other

Proof-net reduction

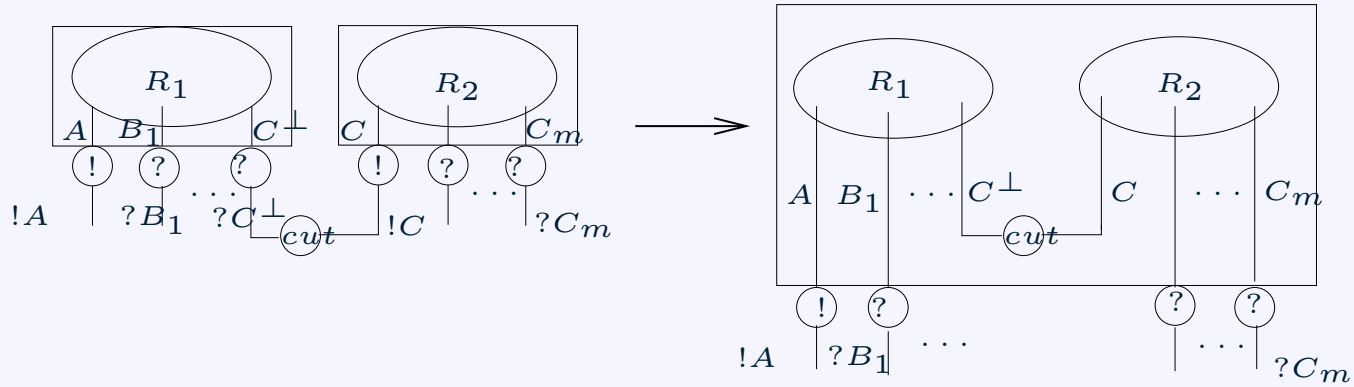
contraction or weakening cut reduction step: as in LL proof-nets



Proof-net reduction (continued)

box-box cut reduction step: merging

[cc]



one could handle full weakening (EAL) by using suitable proof-nets, but we stick here to ELL for simplicity

Examples of terms

addition

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

multiplication

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

square

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

Proof-net normalization and term reduction

Proposition 0 (simulation ELL-Λ) *If R is an ELL proof-net corresponding to a proof of conclusion $\Gamma \vdash t : A$ and if $R \to R'$ by reduction, then R' corresponds to a proof of conclusion $\Gamma \vdash t' : A$, with $t \to t'$*

Proposition 0 *If R is an ELL proof-net corresponding to a proof of $\Gamma \vdash t : A$ and if R is cut-free, then t is in normal form.*

Proposition 0 *If $\Gamma \vdash t : A$, $t \to t'$ and t' is in normal form then $\Gamma \vdash t' : A$.*

Normalization of proof-nets

R ELL proof-net. e edge of R.

depth of e, $d(e)$: number of exponential boxes containing e.

depth of node N: similar.

depth of R, $d(R)$: maximum depth of its nodes.

size of R, $|R|$: number of nodes of R

$|R|_i$: number of nodes at depth i

$|R|_{i+}$: number of nodes at depth $\geq i$

Proposition 0 (Stratification) *The depth of an edge of an ELL proof-net does not change after a reduction step.*

Note that this is not true in LL: the depth can decrease (dereliction step)

or increase (box-box step).

Complexity bound on reduction

$K(0,n) = n$, $K(k+1,n) = 2^{K(k,n)}$.

Theorem 0 *If R is an ELL proof-net, then R can be reduced into its normal form in less than $K(d+1,|R|)$ steps.*

Remarks:

- the height of the tower only depends on the depth, and not on the size;
- no reference to the formulas occuring in the proof-net;
- this bound is obtained by applying a reduction strategy *by levels*.

Normalization strategy by levels

this strategy eliminates cuts by proceeding by increasing depth.
more precisely:

- one deals successively with *rounds* for each depth level $i = 0, 1, \ldots d$.
  at the end of round i: the proof-net does not have any more cut at depth $\leq i$

- **round** i:
  - phase (a): one reduces the multiplicative/axiom/quantifier cuts at depth i,
  - phase (b): one repeats the following step until there isn't any exponential cut left:
    at depth i:

    step: reduce a *maximal* exponential cut at depth i (for a certain order relation $\prec$ on cuts).

Bounds (continued)

Proposition 0 *A proof-net R can be reduced in time $O(K(d+1, |R|))$.*

notation: $!^k A = !\ldots!A$ (k times).

Consequence:

If $\vdash t : N \multimap !^k N$ then t represents an elementary recursive function.

Iteration in EAL

we can define for any A an iterator $iter_A$:

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

then $(iter_A)\ F\ t\ \underline{n} \to (F)\ (F) \ldots (F)\ t$ (n times)

examples:

$double : N \multimap N$

$exp = \lambda n.(iter_N)\ double\ \underline{1}\ n\ : N \multimap !N$

remark: exp cannot be iterated.

$coerc_1 = \lambda n.(iter_N)\ succ\ \underline{0}\ : N \multimap !N$

coercion from N to $!N$.

more generally: $coerc_i : N \multimap !^i N$, for $i \in \mathbb{N}$

consequence: a term $\vdash t : !^i N \multimap A$ can be replaced by $\vdash t' : N \multimap A$

extensionally equal:

$t' = \lambda n.(t)\ (coerc_i)\ n$

Representation of functions

Π proof of $x : N \vdash t : !^l N$.
we say π represents function $f : \mathbb{N} \to \mathbb{N}$ if:
for any integer n the proof obtained by cutting $\vdash \underline{n} : N$ with Π reduces to $\vdash \underline{n'} : !^l N$, where $n' = f(n)$.

Definition 0 *a function f is elementary recursive if there exists an integer h and a Turing machine $\mathcal{M}$ which computes f in time bounded by $K(h, n)$.*

Theorem 0 *The functions representable in EAL are exactly the elementary recursive functions.*

ELL and type fixpoints

as the normalization proof does not use formulas, ELL can be extended with type fixpoints while keeping the same complexity property

EAL$_\mu$ is obtained by extending the language with formulas $\mu\alpha.A$, and rules of EAL by:

$$\frac{\Gamma, A[\mu\alpha.A/\alpha] \vdash B}{\Gamma, \mu\alpha.A \vdash B}\ \mu\ l \qquad \frac{\Gamma \vdash A[\mu\alpha.A/\alpha]}{\Gamma \vdash \mu\alpha.A}\ \mu\ r$$

the following *unfolding* rules are then derivable:

$$\frac{\Gamma, \mu\alpha.A \vdash B}{\Gamma, A[\mu\alpha.A/\alpha] \vdash B} \qquad \frac{\Gamma \vdash \mu\alpha.A}{\Gamma \vdash A[\mu\alpha.A/\alpha]}$$

Proposition 0 *A proof Π of EAL$_\mu$ with depth d can be normalized in a number of steps bounded by $K(d+1, |\Pi|)$.*

this allows to define alternative data-types, as for integers:

$\mathcal{N} = \mu\alpha.(1 \oplus (\alpha \otimes 1))$.

however: no associated iteration scheme!

Soft Linear Logic (Lafont 04)

languages of formulas: same as ELL
SLL rules:

core ILL and

$$\frac{x_1 : A_1, \ldots, x_n : A_n, \vdash t : A}{x_1 : !A_1, \ldots, x_n : !A_n, \vdash t : !A} \; !$$

$$\frac{x_1 : A, \ldots, x_n : A, \Gamma \vdash t : C}{z : !A, \Gamma \vdash t[z/x_1, \ldots, z/x_n] : C} \; mplex \; (n \geq 0)$$

The rule (mplex) is *multiplexing*.

Soft *affine* logic: SAL.

With connective $\otimes$ we have:

non-valid: $!A \multimap !A \otimes !A$ (Contraction), $!A \multimap !!A$ (digging)

valid: $!A \multimap (A \otimes ... \otimes A)$ for any number $k \geq 0$ of A.

generic derivation: no multiplexing (linear term)

SAL

Type for Church integers:

$$N = \forall\alpha.!(\alpha \multimap \alpha) \multimap (\alpha \multimap \alpha)$$

For booleans:

$$B = \forall\alpha.\ \alpha \multimap \alpha \multimap \alpha$$

For binary lists:

$$W = \forall\alpha.!(B \multimap \alpha \multimap \alpha) \multimap (\alpha \multimap \alpha)$$

SAL: properties

Theorem 0 (Lafont) *If $\mathcal{D}$ is type derivation for t, with depth d, then $\mathcal{D}$ can be normalized in a number of steps bounded by $O(n^d)$, where $n = |\mathcal{D}|$.*

Consequence: if $\vdash_{SAL} t : !W \multimap B$, then t denotes a Ptime predicate.

Theorem 0 *If $f : \{0,1\}^\star \to \{0,1\}$ is a Ptime predicate then there exists a term t with a Soft affine logic derivation of $\vdash t : !W \multimap B$, representing f.*

Mairson-Terui 03: P-completeness of multiplicative SLL.

SAL: programming

idea: programs typed with generic derivations. multiplexing used for data.

however: successor cannot be typed with $N \multimap N$.

to type it:

$$N \multimap N\langle X+1\rangle, \qquad \text{where}$$

$N\langle X+1\rangle = \forall\alpha.!(\alpha \multimap \alpha) \otimes (\alpha \multimap \alpha) \multimap (\alpha \multimap \alpha)$.

More generally, for polynomial P:

$N\langle P\rangle$: type for integers $P(n)$

Proposition 0 *For any P, there is a (generic) derivation for:*

$$N^p \vdash N\langle P\rangle$$

where $p = deg(P)$ and N^p denotes $N, \ldots, N$ with p repetitions.

Encoding of Turing Machines: sketch

type for machine configurations: C.
$\mathcal{M}$ machine with polynomials P (for time) and Q for space.
the step function can be encoded with type: $C \multimap C$.
then: $C\langle Q\rangle \multimap C\langle Q\rangle$
besides we have:

$$\begin{aligned} W^p &\vdash N\langle P\rangle \\ W^q &\vdash C\langle Q\rangle \end{aligned}$$

by iteration we get: $N\langle P\rangle \vdash C\langle Q\rangle \multimap C\langle Q\rangle$,
(k=P(n) steps of transition on a configuration)
so finally:

$$\begin{aligned} W^{p+q} &\vdash C\langle Q\rangle \\ !W &\vdash C\langle Q\rangle \qquad \text{by multiplexing .} \end{aligned}$$

Part 3: Light Linear Logic

- 3.1 Light Linear logic (LLL)
  - the system
  - proof-nets and polynomial bound
  - expressivity and discussion
- 3.2 Light types for lambda-calculus (DLAL)
  - language and translation to LAL
  - coercions and expressivity
  - Ptime soundness
- 3.3 Type inference problems

Light Linear Logic (LLL)

language of formulas

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

rules:

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

$$\frac{\Gamma, \Delta \vdash A}{!\Gamma, \S\Delta \vdash \S A}$$

the other rules are unchanged from ELL.

- we will consider the affine version : light affine logic (LAL);
- as with EAL we will use LAL as a type system for λ–calculus.

LAL: remarks

- the rule $\S$ can be seen as a kind of multiple dereliction, with a *marker* $\S$;
- from a typing point of view: $!A$ subtype of $\S A$;
- from a semantical point of view:
  $!$, $\S$ are functors, and we have the principles
  $!A \multimap (!A \otimes !A)$
  $!A \multimap \S A \quad \S A \otimes \S B \multimap \S(A \otimes B)$

LAL sequent-calculus

$$\frac{}{A \vdash A}\ Id \qquad \frac{\Gamma_1 \vdash A \quad A, \Gamma_2 \vdash C}{\Gamma_1, \Gamma_2 \vdash C}\ Cut$$

$$\frac{\Gamma_1 \vdash A_1 \quad A_2, \Gamma_2 \vdash C}{\Gamma_1, A_1 \multimap A_2, \Gamma_2 \vdash C}\ \multimap l \qquad \frac{A_1, \Gamma \vdash A_2}{\Gamma \vdash A_1 \multimap A_2}\ \multimap r$$

$$\frac{A[B/\alpha], \Gamma \vdash C}{\forall \alpha.A, \Gamma \vdash C}\ \forall l \qquad \frac{\Gamma \vdash A}{\Gamma \vdash \forall \alpha.A}\ \forall r \ (\alpha \text{ not free in } \Gamma)$$

$$\frac{\Gamma \vdash C}{\Delta, \Gamma \vdash C}\ Weak \qquad \frac{!A, !A, \Gamma \vdash C}{!A, \Gamma \vdash C}\ Cntr$$

$$\frac{B \vdash A}{!B \vdash !A}\ !r \qquad \frac{\vdash A}{\vdash !A}\ !r$$

$$\frac{\Gamma, \Delta \vdash A}{!\Gamma, \S\Delta \vdash \S A}\ \S r$$

LAL and λ-calculus

$$\frac{}{x:A \vdash x:A}\ Id \qquad \frac{\Gamma_1 \vdash u:A \quad x:A, \Gamma_2 \vdash t:C}{\Gamma_1, \Gamma_2 \vdash t[u/x]:C}\ Cut$$

$$\frac{\Gamma_1 \vdash u:A_1 \quad x:A_2, \Gamma_2 \vdash t:C}{\Gamma_1, y:A_1 \multimap A_2, \Gamma_2 \vdash t[(y)u/x]:C}\ \multimap l \qquad \frac{x:A_1, \Gamma \vdash t:A_2}{\Gamma \vdash \lambda x.t:A_1 \multimap A_2}\ \multimap r$$

$$\frac{x:A[B/\alpha], \Gamma \vdash t:C}{x:\forall\alpha.A, \Gamma \vdash t:C}\ \forall l \qquad \frac{\Gamma \vdash t:A}{\Gamma \vdash t:\forall\alpha.A}\ \forall r \ (\alpha \text{ not free in } \Gamma)$$

$$\frac{\Gamma \vdash t:C}{\Delta, \Gamma \vdash t:C}\ Weak \qquad \frac{x:!A, y:!A, \Gamma \vdash t:C}{z:!A, \Gamma \vdash t[z/x, z/y]:C}\ Cntr$$

$$\frac{x:B \vdash t:A}{x:!B \vdash t:!A}\ !r \qquad \frac{\vdash t:A}{\vdash t:!A}\ !r$$

$$\frac{\Gamma, \Delta \vdash t:A}{!\Gamma, \S\Delta \vdash t:\S A}\ \S r$$

Translation LAL -> EAL

translation $(.)^o : LAL \longrightarrow EAL$:

- $(!A)^o = (\S A)^o = !A^o$,
- $(.)^o$ commutes with other connectives.

This gives a translation from LAL proofs to EAL proofs, and it is a simulation.
Of course, not all EAL proofs are in the image of $(.)^o$.

As for EAL, we have a forgetful map : $(.)^- : LAL \to F$.

LAL data types

- unary integers

EAL:

$$N^{EAL}$$

$$\forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha),$$

LAL:

$$N^{LAL}$$

$$\forall\alpha.!(\alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha)\ .$$

- binary lists

EAL:

$$W^{EAL}$$

$$\forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha),$$

LAL

$$W^{LAL}$$

$$\forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha)\ .$$

Examples

addition

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

double

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

Iteration in LAL

we have in LAL as type for the iterator:

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

$(iter_A)\ F\ t\ \underline{n} \to (F)\ (F) \ldots (F)\ t$ (n times)

examples:

$add : N \multimap N \multimap N, \quad (add)\ \underline{2} : N \multimap N$

$double' = \lambda n.[(iter_N)\ (add)\ \underline{2}]\ \underline{0}\ n\ \ : N \multimap \S N$

$double'$ cannot be iterated.

similarly:

$m : N \vdash (add)\ m : N \multimap N$, so $m : !N \vdash (add)\ m : !(N \multimap N)$

multiplication is obtained by:

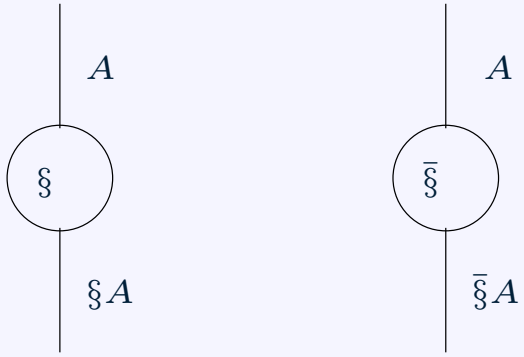
$mult' = \lambda n m.[(iter_N)\ (add)\ m]\ \underline{0}\ n\ \ : !N \multimap N \multimap \S N$

Proof-structures for LLL

we represent proofs by proof-structures for *classical* LLL.
the language of formula is extended with $\overline{\S}A$, and we have:

$$(\S A)^{\perp} = \overline{\S} A^{\perp} \quad (\overline{\S} A)^{\perp} = \S A^{\perp}$$

new nodes:

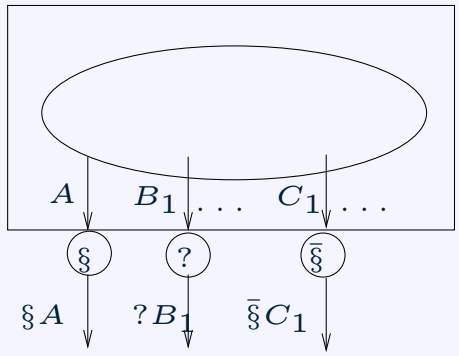


these nodes are used with boxes

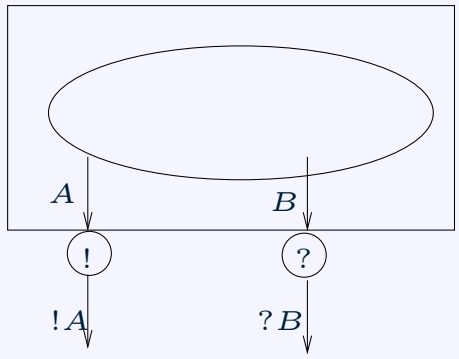
Proof-nets for LLL: boxes

two kinds of exponential boxes:

§-box:



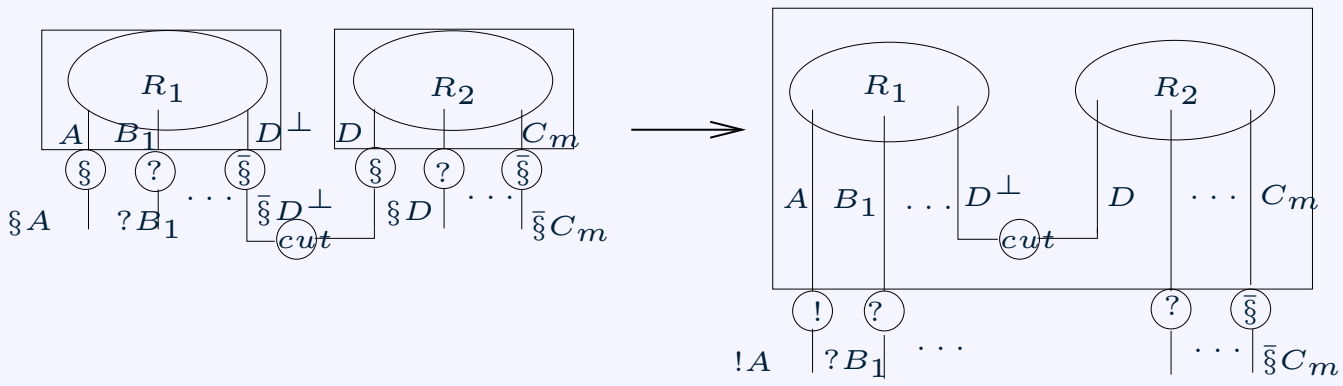
!-box:



Reduction of LLL proof-nets

as in ELL, with moreover the case of cut $\S/\overline{\S}$:

[cc]



Cut-elimination in LLL

Proposition 0 (simulation LLL-Λ) *If R is an LLL proof-net corresponding to a proof of conclusion $\Gamma \vdash t : A$ and if $R \to R'$ by reduction,then R' corresponds to a proof of conclusion $\Gamma \vdash t' : A$, with $t \to t'$.*

Complexity bound on reduction

As ELL proof-nets, the LLL proof-nets satisfy the stratification property.

Theorem 0 *If R is an LLL proof-net, with depth d, then the reduction of R into its normal form can be done in $O((d+1).|R|^{2^{d+1}})$ steps.*

Remarks:

- if the depth is fixed, then the number of steps is polynomial in $|R|$;
- this bound is obtained by a strategy *by levels*.

Normalization strategy

Strategy similar to the one for ELL:

- rounds at increasing depth levels, from $i = 0$ to $i = d$.
- **round** i:
  - phase (a): one reduces the multiplicative/axiom/quantifier / $\S$ cuts at depth i,
  - phase (b): repeat the following step until no more exponential cut at depth i:

    step: reduce a maximal exponential cut at depth i (for $\prec$).

Strong polynomial bound and other issues

- Actually the previous bound $O((d+1).|R|^{2^{d+1}})$ holds not only for the previous reduction strategy, but for *any* reduction strategy: *strongly polynomial normalization*, [Terui01].
- the interest of the strategy by levels is that it makes it easier to prove the bound.
- This strong polynomial normalization property is in contrast with some other ICC systems (BC safe recursion).
- Alternative intermediate language, instead of proof-nets: *light affine lambda-calculus* ([Terui01]). Essentially a lambda-calculus extended with constructions for managing !-boxes.
- another possible approach: define ELL, LLL within standard LL proof-nets. Danos-Joinet 99, Mazza 04.

Type coercions

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

We can define a coercion from N to $\S N$:

$coerc_1 = \lambda n.(iter_N)\ succ\ \underline{0}\ : N \multimap \S N$.

More generally: $coerc_{i,j} : N \multimap \S^{i+1} !^j N$, with $i, j \geq 0$.

From $mult' : !N \multimap N \multimap \S N$, we get

$mult'' : N \multimap N \multimap \S^2 N$,

then:

$square'' : !N \multimap \S^3 N$,

Similarly for type W. This way we can consider types of the form $W \multimap \S^k W$.

Representation of functions

for Ptime computation the choice of representation of integers is important: binary lists.

Π proof of $x : W \vdash t : \S^k W$.
We say Π represents function $f : \{0,1\}^\star \to \{0,1\}^\star$ if:
for any w of $\{0,1\}^\star$, the proof obtained by cutting $\vdash \underline{w} : W$ with Π reduces to $\vdash \underline{w'} : \S^k W$, where $w' = f(w)$.

Complexity of LAL

Theorem 0 *If $f : \{0,1\}^* \to \{0,1\}^*$ belongs to FP, then there exists an integer k and a proof Π of $x : W \vdash t : \S^k W$ representing f.*

Corollary 0 *The functions representable in LAL are exactly the functions of FP, that is to say computable in polynomial time on a Turing machine.*

Remarks:

- even if a term t is typeable in LAL, to execute it with the expected complexity bound we need to compile it into a proof-net;
- a term can be reducible in polynomial time and still not be typeable in LAL.

Simulation of Ptime Turing machines in LAL (sketch)

Girard98, Asperti-Roversi02.

$Config$: data type for Turing machine configurations (tape and state)

$\vdash init : W \multimap Config$

$\vdash length : W \multimap N$: length of a binary list

given a machine $\mathcal{M}$ with associated polynomial P, we build:

$\vdash step : Config \multimap Config$ representing 1 step of the machine

$\vdash t_P : N \multimap \S^k N$

then , with $c_0 : Config$ we define $s = (iter_{Config})\ step\ c_0$.

so we have:

$$c_0 : \S Config \vdash s : N \multimap \S Config$$

hence $(s)\underline{n}$ computes the configuration obtained after n steps of $\mathcal{M}$ starting from c_0.

Simulation of Turing machines: continued

We have:

$$c_0 : \S Config \quad \vdash \quad s : N \multimap \S Config$$

$$c_0 : \S^{k+1} Config \quad \vdash \quad s : \S^k N \multimap \S^{k+1} Config$$

$$\vdash \quad t_P : N \multimap \S^k N$$

so: $w_1 : W,\ c_0 : \S^{k+1} Config \quad \vdash \quad (s)\ (t_P)\ (length)\ w_1 : \S^{k+1} Config$

using $init : W \multimap Config$ we get a term s':

$$w_1 : W,\ w_0 : \S^{k+1} W \quad \vdash \quad s' : \S^{k+1} Config$$

with the coercions: $w_1 : W, w_0 : W \vdash s'' : \S^{k+1} Config$

then by contraction and abstraction:

$$\vdash \quad M : !W \multimap \S^{k+2} Config$$

composing with $\vdash extract : Config \multimap W$ and coercions we finally obtain $\vdash \quad M' : W \multimap \S^{k+3} W$

This term M' simulates the machine $\mathcal{M}$ we started from.

A simpler type system : DLAL

DLAL: *Dual Light Affine Logic*
langage of DLAL types:

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

translation $(.)^* : DLAL \longrightarrow LAL$:

- $(A \Rightarrow B)^* = !A^* \multimap B^*$,
- $(.)^*$ commutes to other connectives.

Note that the following types are not in the image of DLAL :

$$A \multimap !B, \quad \S !A \multimap B, \quad !A \ .$$

For typing in DLAL : mixed judgements $\Gamma ; \Delta \vdash t : C$,

where Δ is an *affine-linear* context, Γ is *non linear*.

DLAL

$$\frac{}{;x:A\vdash x:A}\ (Id) \qquad \frac{\Gamma_1;\Delta_1\vdash u:A \quad \Gamma_2;x:A,\Delta_2\vdash t:C}{\Gamma_1,\Gamma_2;\Delta_1,\Delta_2\vdash t[u/x]:C}\ (Cu$$

$$\frac{\Gamma_1;\Delta_1\vdash u:A \quad \Gamma_2;x:B,\Delta_2\vdash t:C}{\Gamma_1,\Gamma_2;y:A\multimap B,\Delta_1,\Delta_2\vdash t[(y)u/x]:C}\ (\multimap l) \qquad \frac{\Gamma;x:A,\Delta\vdash t:B}{\Gamma;\Delta\vdash \lambda x.t:A\multimap B}\ (\multimap r)$$

$$\frac{;z:D\vdash u:A \quad \Gamma;x:B,\Delta\vdash t:C}{z:D,\Gamma;y:A\Rightarrow B,\Delta\vdash t[(y)u/x]:C}\ (\Rightarrow l)(*) \qquad \frac{x:A,\Gamma;\Delta\vdash t:B}{\Gamma;\Delta\vdash \lambda x.t:A\Rightarrow B}\ (\Rightarrow r)$$

$$\frac{\Gamma;x:A[B/\alpha],\Delta\vdash t:C}{\Gamma;x:\forall\alpha.A,\Delta\vdash t:C}\ (\forall l) \qquad \frac{\Gamma;\Delta\vdash t:A}{\Gamma;\Delta\vdash t:\forall\alpha.A}\ (\forall r),\ \alpha \text{ not free in } \Gamma,\Delta$$

$$\frac{\Gamma;\Delta\vdash t:C}{\Sigma,\Gamma;\Pi,\Delta\vdash t:C}\ (Weak) \qquad \frac{x:A,y:A,\Gamma;\Delta\vdash t:C}{z:A,\Gamma;\Delta\vdash t[z/x,z/y]:C}\ (Cntr)$$

$$\frac{;\Gamma,x_1:B_1,\ldots,x_n:B_n\vdash t:A}{\Gamma;x_1:\S B_1,\ldots,x_n:\S B_n\vdash t:\S A}\ (\S)$$

(*) $z:D$ can be absent.

Type derivations in DLAL

Proposition 0 *if* $\Gamma; \Delta \vdash_{DLAL} t : A$ *and* $x \in \Delta$*, then* x *has at most one occurrence in* t *(*x *linear).*

Proposition 0 *If* $\Gamma; \Delta \vdash_{DLAL} t : A$ *then* $!\Gamma^*, \Delta^* \vdash_{LAL} t : A^*$*.*
We obtain in this way a simulation from DLAL to LAL.

The *depth* of a DLAL derivation is the depth of the corresponding LAL derivation.

DLAL data types

- unary integers

LAL:

$$N^{LAL}$$

$$\forall\alpha.!(\alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha),$$

DLAL

$$N^{DLAL}$$

$$\forall\alpha.(\alpha \multimap \alpha) \Rightarrow \S(\alpha \multimap \alpha)\ .$$

- binary lists

LAL:

$$W^{LAL}$$

$$\forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha),$$

DLAL

$$W^{DLAL}$$

$$\forall\alpha.(\alpha \multimap \alpha) \Rightarrow (\alpha \multimap \alpha) \Rightarrow \S(\alpha \multimap \alpha)\ .$$

Types in DLAL

examples of types:

$$\vdash addition : \quad N \multimap N \multimap N$$
$$\vdash double' : \quad N \Rightarrow \S N$$

The functions representable in DLAL are given by types $W \multimap \S^k W$.

Typing of iterator:

$$iter_A = \lambda f x n.\ (n)\ f\ x\ : (A \multimap A) \Rightarrow \S A \multimap N \multimap \S A$$

Coercions in DLAL

in LAL: $coer^{p,q} : N \multimap \S^{p+1}!^{q}N$

in DLAL, the following rules are derivable:

$$\frac{n : N; \Delta \vdash t : A}{; m : N, \S\Delta \vdash C_1[t] : \S A}\ \text{(coerc1)} \qquad \frac{\Gamma; n : \S N, \Delta \vdash t : A}{\Gamma; m : N, \Delta \vdash C_2[t] : A}\ \text{(coerc2)}$$

with C_i contexts such that $C_i[t]$ is extensionally equivalent to t (represents the same function)

example:

$$\begin{array}{lcll} mult & : & N \Rightarrow (N \multimap \S N) & \\ mult' & : & N \multimap \S(N \multimap \S N) & \text{by (coerc1)} \\ mult'' & : & N \multimap N \multimap \S\S N & \text{by (coerc2)} \end{array}$$

Proposition 1 *If $P \in \mathbb{N}[X]$, then there exists a term t_P representing P and an integer k such that: $\vdash_{DLAL} t_P : N \multimap \S^k N$.*

DLAL: complexity bounds

DLAL satisfies the subject-reduction property.

Theorem 2 (strong polynomial bound) *If t is typeable in DLAL with a derivation of depth d, then any sequence of β-reducts of t has length bounded by $O((d+1).|t|^{2^{d+1}})$.*

Remarks:

- we are dealing here with β-réduction, and not anymore with proof-net reduction;
- this bound holds for any reduction strategy;
- in particular, if $\vdash t : W \multimap \S^k W$ then we can normalize $(t)\underline{w}$ in a number of steps polynomial in $longueur(w)$.

Theorem 3 *The functions representable by terms typeable in DLAL are exactly the functions of FP.*

Type inference issues

problem: given a term t, is it typeable in EAL/LAL/DLAL ?

for propositional systems the problem is decidable:

- EAL: Coppola, Martini, Ronchi della Rocca, Dal Lago, PB, Terui.
- LAL: PB.

also possible to infer coercions for EAL: Atassi.

typeability is reduced to a constraints system,
for EAL: linear programming problem
for LAL: word constraints.

decoration problem: given t typed in F, does it admit a decoration in EAL2/LAL2/DLAL2?

(subsumes propositional inference)

Type decoration for DLAL

Theorem 4 *There is a polynomial time algorithm which, given t typed in F, decides whether it admits a decoration in DLAL2.*

(joint work with Terui and Atassi)

a key property:

Proposition 5 *Let M be a system F term:*

$x_1 : A_1, \ldots, x_m : A_m; y_1 : B_1, \ldots, y_n : B_n \vdash M : C$ *is derivable in* $DLAL$

if and only if

there is a decoration t of M with type $C^\star$ and with free variables $x_1^{!A_1^\star}, \ldots, x_m^{!A_m^\star}$, $y_1^{B_1^\star}, \ldots, y_n^{B_n^\star}$ which is regular and satisfies the local typing, bracketing, λ-scope, bang and Λ-scope conditions.

Example

$$M = (\lambda g^{\alpha\to\alpha}.(g\ (g\ x^{\alpha})))\lambda y^{\alpha}.z^{\alpha} \quad : \alpha$$

$$\overline{M} = \mathbf{n_1}[(\mathbf{n_2}\lambda g^{\alpha\to\alpha}.\mathbf{n_3}(\mathbf{n_4}g\ \mathbf{n_6}(\mathbf{n_5}g\ \mathbf{n_7}x^{\alpha})))\ \mathbf{n_8}(\lambda y^{\alpha}.\mathbf{n_9}z^{\alpha})]$$

Example

$$\overline{M} = \mathbf{n_1}[(\mathbf{n_2}\lambda g^{\alpha\to\alpha}.\mathbf{n_3}(\mathbf{n_4}g\ \mathbf{n_6}(\mathbf{n_5}g\ \mathbf{n_7}x^{\alpha})))\ \mathbf{n_8}(\lambda y^{\alpha}.\mathbf{n_9}z^{\alpha})]$$

boxing conditions:

$$\begin{cases}
\mathbf{n_1} \geq 0 & \\
\mathbf{n_1}+\mathbf{n_2} \geq 0 & \\
\mathbf{n_1}+\mathbf{n_2}+\mathbf{n_3}+\mathbf{n_6}+\mathbf{n_7} = 0 & \text{bracketing } (x) \\
\mathbf{n_1}+\mathbf{n_8} \geq 0 & \\
\mathbf{n_1}+\mathbf{n_8}+\mathbf{n_9} = 0 & \text{bracketing } (z) \\
\mathbf{n_3} \geq 0 & \\
\mathbf{n_3}+\mathbf{n_4} = 0 & \lambda\text{-scope } (g^1) \\
\mathbf{n_3}+\mathbf{n_6} \geq 0 & \\
\mathbf{n_3}+\mathbf{n_6}+\mathbf{n_5} = 0 & \lambda\text{-scope } (g^2)
\end{cases}$$

Example

$$\overline{M} = \mathbf{n_1}[(\mathbf{n_2}\lambda g^{\alpha\to\alpha}.\mathbf{n_3}(\mathbf{n_4}g\ \mathbf{n_6}(\mathbf{n_5}g\ \mathbf{n_7}x^{\alpha})))\ \mathbf{n_8}(\lambda y^{\alpha}.\mathbf{n_9}z^{\alpha})]$$

p-types:

$$g : \S^{\mathbf{b_1},\mathbf{m_1}}(\S^{\mathbf{b_2},\mathbf{m_2}}\alpha \multimap \S^{\mathbf{m_3}}\alpha) \qquad x : \S^{\mathbf{b_4},\mathbf{m_4}}\alpha$$
$$y : \S^{\mathbf{b_5},\mathbf{m_5}}\alpha \qquad z : \S^{\mathbf{b_6},\mathbf{m_6}}\alpha$$

Local typing conditions:

$$\begin{cases} \mathbf{n_9}z^{\alpha} & : \quad \mathbf{n_9} + \mathbf{m_6} \geq 0 \\ \mathbf{n_8}(\lambda y^{\alpha}.\mathbf{n_9}z^{\alpha}) & : \quad \mathbf{n_8} \geq 0 \\ [(\mathbf{n_2}\lambda g^{\alpha\to\alpha}.\mathbf{n_3}(\mathbf{n_4}g\ \mathbf{n_6}(\mathbf{n_5}g\ \mathbf{n_7}x^{\alpha})))\ \mathbf{n_8}(\lambda y^{\alpha}.\mathbf{n_9}z^{\alpha})] & : \quad \mathbf{n_2} = 0, \mathbf{m_1} = \mathbf{n_8}, \mathbf{m_2} = \mathbf{m_5}, \\ & \qquad \mathbf{m_3} = \mathbf{n_9} + \mathbf{m_6}, \mathbf{b_2} = \mathbf{b_5} \\ \ldots & \\ (g \text{ has 2 occurrences}) & \qquad \mathbf{b_1} = 1 \end{cases}$$

Example

$$\overline{M} = \mathbf{n_1}[(\mathbf{n_2}\lambda g^{\alpha\to\alpha}.\mathbf{n_3}(\mathbf{n_4}g\ \mathbf{n_6}(\mathbf{n_5}g\ \mathbf{n_7}x^{\alpha})))\ \mathbf{n_8}(\lambda y^{\alpha}.\mathbf{n_9}z^{\alpha})]$$

p-types:

$$g : \S^{\mathbf{b_1},\mathbf{m_1}}(\S^{\mathbf{b_2},\mathbf{m_2}}\alpha \multimap \S^{\mathbf{m_3}}\alpha) \qquad x : \S^{\mathbf{b_4},\mathbf{m_4}}\alpha$$

$$y : \S^{\mathbf{b_5},\mathbf{m_5}}\alpha \qquad z : \S^{\mathbf{b_6},\mathbf{m_6}}\alpha$$

Bang conditions:

$$\begin{cases} \mathbf{b_2} = 1 \Rightarrow \mathbf{b_4} = 1 \\ \mathbf{b_2} = 0 \\ \mathbf{b_1} = 1 \Rightarrow \mathbf{b_6} = 1 \\ \mathbf{b_1} = 1 \Rightarrow \mathbf{n_8} \geq 1 \end{cases}$$

Example

$$\overline{M} = \mathbf{n_1}[(\mathbf{n_2}\lambda g^{\alpha\to\alpha}.\mathbf{n_3}(\mathbf{n_4}g\ \mathbf{n_6}(\mathbf{n_5}g\ \mathbf{n_7}x^{\alpha})))\ \mathbf{n_8}(\lambda y^{\alpha}.\mathbf{n_9}z^{\alpha})]$$

p-types:

$$g : \S^{\mathbf{b_1},\mathbf{m_1}}(\S^{\mathbf{b_2},\mathbf{m_2}}\alpha \multimap \S^{\mathbf{m_3}}\alpha) \quad x : \S^{\mathbf{b_4},\mathbf{m_4}}\alpha$$
$$y : \S^{\mathbf{b_5},\mathbf{m_5}}\alpha \qquad\qquad z : \S^{\mathbf{b_6},\mathbf{m_6}}\alpha$$

Boolean constraints:

$$\begin{aligned}\mathsf{Const}^b(\overline{M}) \quad = \quad & \{\mathbf{b_1} = 1,\ (\mathbf{b_2} = 1 \Rightarrow \mathbf{b_4} = 1), \mathbf{b_1} = 0,\\ & (\mathbf{b_1} = 1 \Rightarrow \mathbf{b_6} = 1),\ \mathbf{b_2} = \mathbf{b_5},\ \mathbf{b_2} = \mathbf{b_4}\}\end{aligned}$$

Minimal solution ψ^b:

$$\mathbf{b_1} = \mathbf{b_6} = 1, \quad \mathbf{b_2} = \mathbf{b_4} = \mathbf{b_5} = 0.$$

Example

$$\overline{M} = \mathbf{n_1}[(\mathbf{n_2}\lambda g^{\alpha\to\alpha}.\mathbf{n_3}(\mathbf{n_4}g\ \mathbf{n_6}(\mathbf{n_5}g\ \mathbf{n_7}x^{\alpha})))\ \mathbf{n_8}(\lambda y^{\alpha}.\mathbf{n_9}z^{\alpha})]$$

p-types:

$$g : \S^{\mathbf{b_1},\mathbf{m_1}}(\S^{\mathbf{b_2},\mathbf{m_2}}\alpha \multimap \S^{\mathbf{m_3}}\alpha) \quad x : \S^{\mathbf{b_4},\mathbf{m_4}}\alpha$$
$$y : \S^{\mathbf{b_5},\mathbf{m_5}}\alpha \qquad\qquad z : \S^{\mathbf{b_6},\mathbf{m_6}}\alpha$$

The linear system $\psi^b\mathsf{Const}^b(\overline{M})$ has solutions. One of them gives:

$$g : !(\alpha \multimap \alpha) \quad x : \S\alpha$$
$$y : \alpha \qquad\qquad z : !\alpha$$

$$\overline{M} = (\lambda g.\S\ (\ \bar{\S}g\ (\bar{\S}g\ \bar{\S}x)))\ \S(\lambda y.\bar{\S}z) \quad : \ \S\alpha$$

Conclusion

- LL brings a logical, proof-based approach to implicit computational complexity,
  This approach relates:
  ICC on the one hand / functional programming, typing, proofs on the other . . .
- also investigations in semantics:
  provability: Kanovich *et al*, Terui, Dal Lago-Martini
  denotational semantics: Murawski-Ong, PB, De Carvalho, Laurent-Tortora . . .
- however limited intensional expressivity
  all FP *functions* are representable, but some common polynomial time algorithms are not (directly) typeable
  Trade-off? intensional expressivity *vs* efficiency of decision procedure