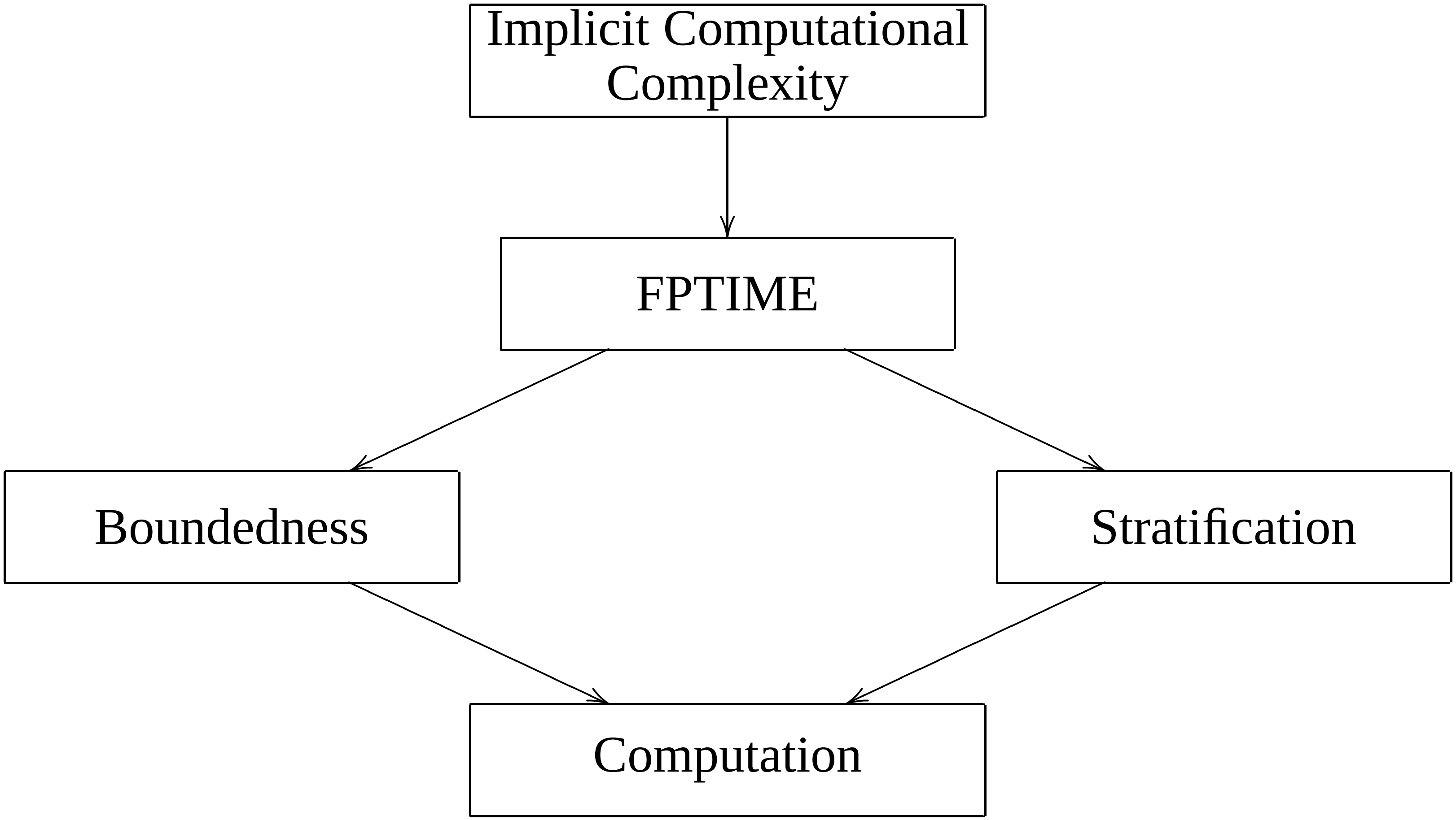
Light Logics for Polynomial Time Computations

Marco Gaboardi
University of Pennsylvania
Università di Bologna
Inria Focus Team

Outline



Implicit Computational Complexity
FPTIME
Boundedness
Stratification
Computation

Implicit Computational Complexity

Implicit Computational Complexity: Motivations

- ICC aims at describing complexity classes without:
  - explicit reference to a specific machine model
  - without an explicit cost bound

Implicit Computational Complexity: Motivations

- ICC aims at describing complexity classes without:
  - explicit reference to a specific machine model
  - without an explicit cost bound
- It generally borrows techniques from Mathematical Logic :
  - Recursion Theory, FPTIME = Predicative Recursion on Notation
  - Structural Proof Theory, FPTIME = Bounded/Light Linear Logic
  - Model Theory, FPTIME = PR Functions over Finite Structures

Implicit Computational Complexity: Motivations

- ICC aims at describing complexity classes without:
  - explicit reference to a specific machine model
  - without an explicit cost bound
- It generally borrows techniques from Mathematical Logic :
  - Recursion Theory, FPTIME = Predicative Recursion on Notation
  - Structural Proof Theory, FPTIME = Bounded/Light Linear Logic
  - Model Theory, FPTIME = PR Functions over Finite Structures
- Our approach:

Characterizing the class FPTIME

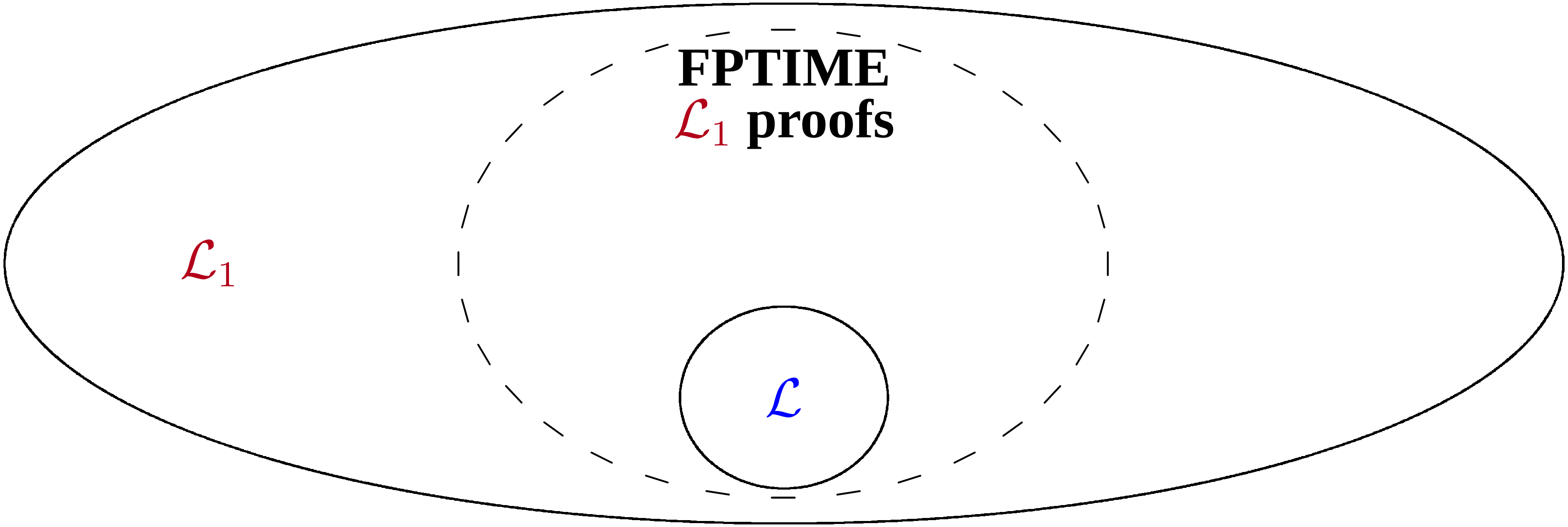
FPTIME is the class of functions (and not only decision problems) computable in polynomial time by a deterministic Turing Machine.

A Logic $\mathcal{L}$ characterizes FPTIME if:

- **Soundness**: Every proof/program in $\mathcal{L}$ can be evaluated in polynomial time.
- **Extensional Completeness**: Every TM computing a function in FPTIME can be simulated by means of a proof/program in $\mathcal{L}$.

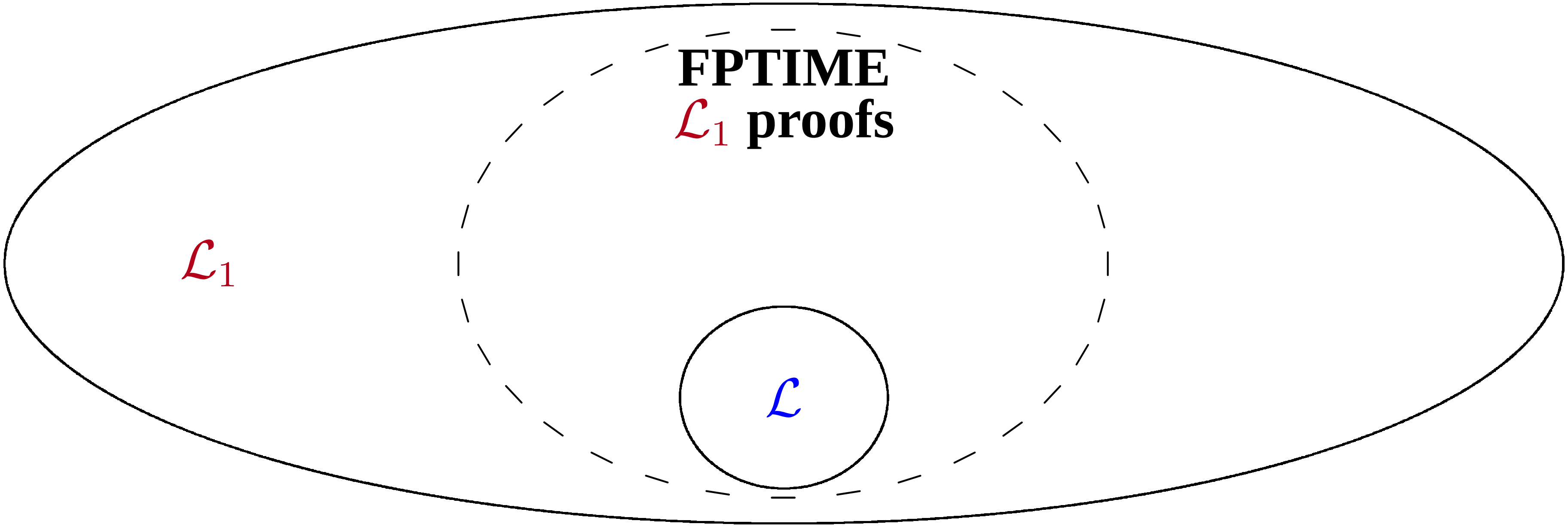
Characterizing the class FPTIME

Extensional Completeness does not say anything about the expressivity of $\mathcal{L}$. For instance:



Characterizing the class FPTIME

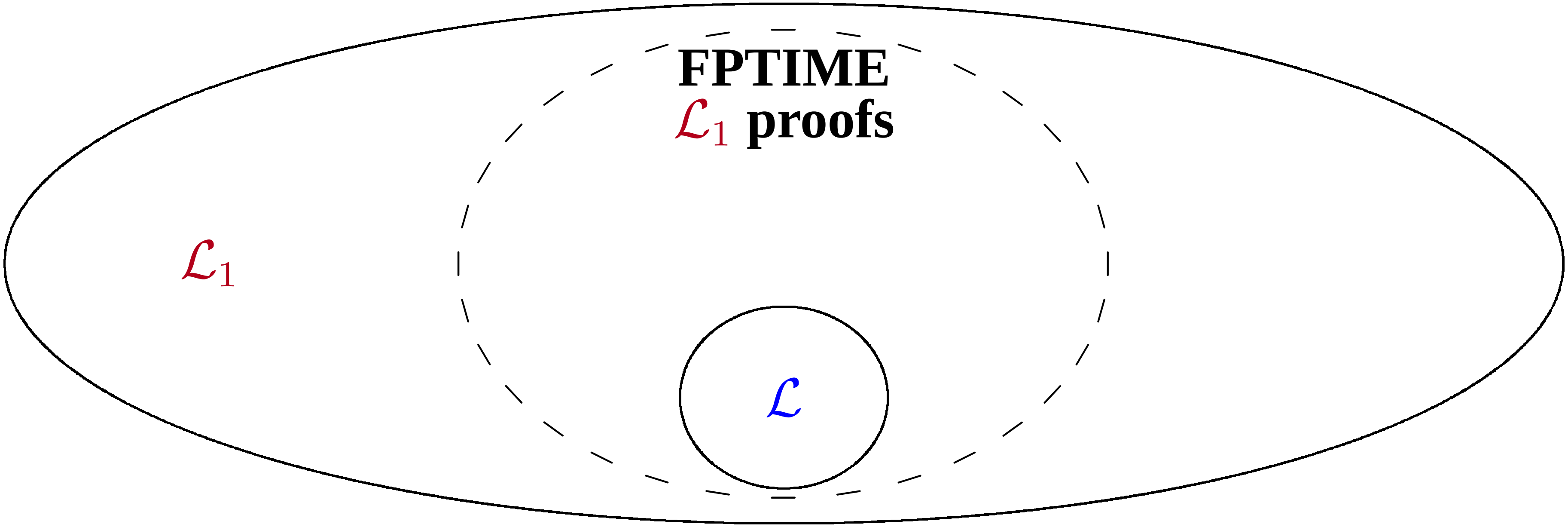
Extensional Completeness does not say anything about the expressivity of $\mathcal{L}$. For instance:



Intensional Completeness:
The Logic $\mathcal{L}$ captures all the FPTIME proofs of $\mathcal{L}_1$.

Characterizing the class FPTIME

Extensional Completeness does not say anything about the expressivity of $\mathcal{L}$. For instance:



Intensional Completeness: (Usually undecidable!)
The Logic $\mathcal{L}$ captures all the FPTIME proofs of $\mathcal{L}_1$.

FPTIME

- FPTIME is the class of functions (and not only decision problems) computable in polynomial time by a deterministic Turing Machine.

FPTIME

- FPTIME is the class of functions (and not only decision problems) computable in polynomial time by a deterministic Turing Machine.
- Intuitively, FPTIME is the class of programs obtained by permitting arbitrary compositions of polynomial iterations.

$$r(x) \circ q(x) = p(x) \qquad \underbrace{p(x) \circ p(x) \circ \cdot \circ p(x)}_{x\text{-times}} = e(x)$$

FPTIME

- FPTIME is the class of functions (and not only decision problems) computable in polynomial time by a deterministic Turing Machine.
- Intuitively, FPTIME is the class of programs obtained by permitting arbitrary compositions of polynomial iterations.

$$r(x) \circ q(x) = p(x) \qquad \underbrace{p(x) \circ p(x) \circ \cdot \circ p(x)}_{x\text{-times}} = e(x)$$

- **Recurrent idea:** Allows definitions by iteration of polynomials but forbids dangerous iterations of polynomials.

Road-map

System	Soundness	Extensional Completeness	Intensional Completeness	Decidability
Bounded Recursion on Notation	X	X	X	
Predicative Recursion on Notation	X	X		X
Light Affine Logic	X	X		X
Bounded Linear Logic	X	X	$\approx$	

First approach: Bounded Recursion on notation

The first implicit characterization of FPTIME [Cobham65] uses BRN:

$$\begin{array}{rcl} f(\epsilon, \vec{y}) & = & g(\vec{y}) \\ f(\mathbf{0}x, \vec{y}) & = & h_0(x, \vec{y}, f(x, \vec{y})) \\ f(\mathbf{1}x, \vec{y}) & = & h_1(x, \vec{y}, f(x, \vec{y})) \\ f(x, \vec{y}) & \leq & k(x, \vec{y}) \end{array}$$

with a smash function $x\#y = \underbrace{\mathbf{10}\cdots\mathbf{0}}_{|x|\cdot|y|}$ and few other basic functions.

First approach: Bounded Recursion on notation

The first implicit characterization of FPTIME [Cobham65] uses BRN:

$$\begin{array}{rcl} f(\epsilon, \vec{y}) & = & g(\vec{y}) \\ f(\mathbf{0}x, \vec{y}) & = & h_0(x, \vec{y}, f(x, \vec{y})) \\ f(\mathbf{1}x, \vec{y}) & = & h_1(x, \vec{y}, f(x, \vec{y})) \\ f(x, \vec{y}) & \leq & k(x, \vec{y}) \end{array}$$

with a smash function $x \# y = \underbrace{\mathbf{1}\mathbf{0} \cdots \mathbf{0}}_{|x| \cdot |y|}$ and few other basic functions.

Pros:

- \+ No explicit machine model.
- \+ Very expressive.

Cons:

- \- The bound is not really implicit.
- \- The bound is difficult to check (undecidable!).

Second approach: Predicative Recursion on notation

Another approach is by using PRN [Bellantoni& Cook91,Leivant91]:

$$\begin{array}{rcl} f(\epsilon, \vec{z}\,;\vec{y}) & = & g(\vec{z}\,;\vec{y}) \\ f(\mathbf{0}x, \vec{z}\,;\vec{y}) & = & h_0(x, \vec{z}\,;\vec{y}, f(x, \vec{z}\,;\vec{y})) \\ f(\mathbf{1}x, \vec{z}\,;\vec{y}) & = & h_1(x, \vec{z}\,;\vec{y}, f(x, \vec{z}\,;\vec{y})) \end{array}$$

Every function $f(\vec{x}\,;\vec{y})$ has normal arguments $\vec{x}$ and safe arguments $\vec{y}$.
Soundness: the result of an iteration cannot be a recurrence argument.

Second approach: Predicative Recursion on notation

Another approach is by using PRN [Bellantoni& Cook91,Leivant91]:

$$\begin{array}{rcl} f(\epsilon, \vec{z} \,\mathbf{;}\, \vec{y}) & = & g(\vec{z} \,\mathbf{;}\, \vec{y}) \\ f(\mathbf{0}x, \vec{z} \,\mathbf{;}\, \vec{y}) & = & h_0(x, \vec{z} \,\mathbf{;}\, \vec{y}, f(x, \vec{z} \,\mathbf{;}\, \vec{y})) \\ f(\mathbf{1}x, \vec{z} \,\mathbf{;}\, \vec{y}) & = & h_1(x, \vec{z} \,\mathbf{;}\, \vec{y}, f(x, \vec{z} \,\mathbf{;}\, \vec{y})) \end{array}$$

Every function $f(\vec{x} \,\mathbf{;}\, \vec{y})$ has normal arguments $\vec{x}$ and safe arguments $\vec{y}$.
Soundness: the result of an iteration cannot be a recurrence argument.

Pros:

- **+** The bound is no more explicit.
- **+** Simple syntactic criterion.

Cons:

- **-** Poor expressivity.
- **-** Inherently first order.

Stratification

Paradox as the worst complexity...

Through the proofs-as-programs correspondence Intuitionistic Logic with type fixpoints corresponds to a system of recursive types.

$$\frac{}{\mathtt{x} : A \vdash \mathtt{x} : A}\ (Ax) \qquad \frac{\Gamma \vdash \mathtt{N} : A \qquad \Delta, \mathtt{x} : A \vdash \mathtt{M} : B}{\Gamma, \Delta \vdash \mathtt{M}[\mathtt{x}/\mathtt{N}]B}\ (cut)$$

$$\frac{\Gamma, \mathtt{x} : A \vdash \mathtt{M} : B \quad \mathtt{x} \notin FV(\Gamma)}{\Gamma \vdash \lambda \mathtt{x}.\mathtt{M} : A \Rightarrow B}\ (\Rightarrow R) \qquad \frac{\Gamma \vdash \mathtt{N} : A \quad \mathtt{x} : B, \Delta \vdash \mathtt{M} : C}{\mathtt{y} : A \Rightarrow B, \Gamma, \Delta \vdash \mathtt{M}[\mathtt{yN}/\mathtt{x}] : C}\ (\Rightarrow L)$$

$$\frac{\Gamma \vdash \mathtt{M} : A}{\Gamma, \mathtt{x} : B \vdash \mathtt{M} : A}\ (w) \qquad \frac{\Gamma, \mathtt{x} : B, \mathtt{x} : B \vdash \mathtt{M} : A}{\Gamma, \mathtt{x} : B \vdash \mathtt{M} : A}\ (c)$$

$$\frac{\Gamma \vdash \mathtt{M} : A \quad A = B}{\Gamma \vdash \mathtt{M} : B}\ (= R) \qquad \frac{\Gamma, \mathtt{x} : A \vdash \mathtt{M} : C \quad A = B}{\Gamma, \mathtt{x} : B \vdash \mathtt{M} : C}\ (= L)$$

Non termination is the worst use of resource

For what follows it is instructive to look how $(\lambda \mathtt{x}.\mathtt{xx})\lambda \mathtt{x}.\mathtt{xx}$ can be typed using the fixpoint $A = A \Rightarrow \bot$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathtt{x} : A \vdash \mathtt{x} : A \quad \mathtt{x} : \bot \vdash \mathtt{x} : \bot}{\mathtt{x} : A, \mathtt{x} : A \Rightarrow \bot \vdash \mathtt{xx} : \bot}\,(\Rightarrow L)}{\mathtt{x} : A, \mathtt{x} : A \vdash \mathtt{xx} : \bot}\,(= L)}{\mathtt{x} : A \vdash \mathtt{xx} : \bot}\,(c)}{\vdash \lambda \mathtt{x}.\mathtt{xx} : A \Rightarrow \bot}\,(\Rightarrow R) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathtt{x} : A \vdash \mathtt{x} : A \quad \mathtt{x} : \bot \vdash \mathtt{x} : \bot}{\mathtt{x} : A, \mathtt{x} : A \Rightarrow \bot \vdash \mathtt{xx} : \bot}\,(\Rightarrow L)}{\mathtt{x} : A, \mathtt{x} : A \vdash \mathtt{xx} : \bot}\,(= L)}{\mathtt{x} : A \vdash \mathtt{xx} : \bot}\,(c)}{\vdash \lambda \mathtt{x}.\mathtt{xx} : A \Rightarrow \bot}\,(\Rightarrow R)}{\vdash \lambda \mathtt{x}.\mathtt{xx} : A}\,(= R)}{\vdash (\lambda \mathtt{x}.\mathtt{xx})\lambda \mathtt{x}.\mathtt{xx} : \bot}\,(cut)$$

Remark: contraction is necessary.

Intuitionistic Linear Logic with fixpoints: $\mathbf{ILL}_\mu$

$$\frac{}{\mathtt{x} : A \vdash \mathtt{x} : A}\ (Ax) \qquad \frac{\Gamma \vdash \mathtt{N} : A \qquad \Delta, \mathtt{x} : A \vdash \mathtt{M} : B}{\Gamma, \Delta \vdash \mathtt{M}[\mathtt{N}/\mathtt{x}] : B}\ (cut)$$

$$\frac{\Gamma, \mathtt{x} : A \vdash \mathtt{M} : B \quad \mathtt{x} \notin FV(\Gamma)}{\Gamma \vdash \lambda \mathtt{x}.\mathtt{M} : A \multimap B}\ (\multimap R) \quad \frac{\Gamma \vdash \mathtt{N} : A \qquad \mathtt{x} : B, \Delta \vdash \mathtt{M} : C}{\mathtt{y} : A \multimap B, \Gamma, \Delta \vdash \mathtt{M}[\mathtt{yN}/\mathtt{x}] : C}\ (\multimap L)$$

$$\frac{\Gamma \vdash \mathtt{M} : A \quad A = B}{\Gamma \vdash \mathtt{M} : B}\ (= R) \qquad \frac{\Gamma, \mathtt{x} : A \vdash \mathtt{M} : C \quad A = B}{\Gamma, \mathtt{x} : B \vdash \mathtt{M} : C}\ (= L)$$

$$\frac{!\Gamma \vdash \mathtt{M} : A}{!\Gamma \vdash \mathtt{M} : !A}\ (p) \qquad \frac{\Gamma, \mathtt{x} : B \vdash \mathtt{M} : A}{\Gamma, \mathtt{x} : !B \vdash \mathtt{M} : A}\ (d)$$

$$\frac{\Gamma \vdash \mathtt{M} : A}{\Gamma, \mathtt{x} : !B \vdash \mathtt{M} : A}\ (w) \qquad \frac{\Gamma, \mathtt{x} : !B, \mathtt{x} : !B \vdash \mathtt{M} : A}{\Gamma, \mathtt{x} : !B \vdash \mathtt{M} : A}\ (c)$$

Non termination in $\mathbf{ILL}_\mu$

Obviously $(\lambda \mathtt{x}.\mathtt{xx})\lambda \mathtt{x}.\mathtt{xx}$ is typable in ILL_μ by means of the type fixpoint $A =!A \multimap \bot$ (which is the usual translation of $A = A \Rightarrow \bot$).

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{\mathtt{x} :!A \vdash \mathtt{x} :!A \qquad \mathtt{x} : \bot \vdash \mathtt{x} : \bot}{\mathtt{x}!A, \mathtt{x} :!A \multimap \bot \vdash \mathtt{xx} : \bot}\;(\multimap L)
}{\mathtt{x}!A, \mathtt{x} : A \vdash \mathtt{xx} : \bot}\;(= L)
}{\color{red}{\mathtt{x}!A, \mathtt{x} :!A \vdash \mathtt{xx} : \bot}}\;\color{red}{(d)}
}{\mathtt{x} :!A \vdash \mathtt{xx} : \bot}\;(c)
}{\vdash \lambda \mathtt{x}.\mathtt{xx} :!A \multimap \bot}\;(\multimap R)
\qquad
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{\mathtt{x} :!A \vdash \mathtt{x} :!A \qquad \mathtt{x} : \bot \vdash \mathtt{x} : \bot}{\mathtt{x}_1!A, \mathtt{x}_2 :!A \multimap \bot \vdash \mathtt{xx} : \bot}\;(\multimap L)
}{\mathtt{x}!A, \mathtt{x} : A \vdash \mathtt{xx} : \bot}\;(= L)
}{\color{red}{\mathtt{x}!A, \mathtt{x} :!A \vdash \mathtt{xx} : \bot}}\;\color{red}{(d)}
}{\mathtt{x} :!A \vdash \mathtt{xx} : \bot}\;(c)
}{\vdash \lambda \mathtt{x}.\mathtt{xx} :!A \multimap \bot}\;(\multimap R)
}{\vdash \lambda \mathtt{x}.\mathtt{xx} : A}\;(= R)
}{\color{blue}{\vdash \lambda \mathtt{x}.\mathtt{xx} :!A}}\;\color{blue}{(p)}
}{\vdash (\lambda \mathtt{x}.\mathtt{xx})\lambda \mathtt{x}.\mathtt{xx} : \bot}\;(cut)
$$

Remark: contraction, promotion and dereliction are necessary.

Towards Light Logics

The idea of Light Logics is to limit the power of the structural rules.

- First idea: forbid contraction.
  - \+ Consistent Set theory with a full comprehension scheme.
  - \- Too weak for polynomial time.

Towards Light Logics

The idea of Light Logics is to limit the power of the structural rules.

- □ First idea: forbid contraction.
  - \+ Consistent Set theory with a full comprehension scheme.
  - \- Too weak for polynomial time.
- □ Second idea: control the other structural rules:
  - i) Proof Stratification.
  - ii) Avoiding the monoidalness of !.

Stratification

We can think to the promotion rule as a **box**:

$$\boxed{\dfrac{\dfrac{\vdots}{!\Gamma \vdash \mathtt{M} : A}}{!\Gamma \vdash \mathtt{M} : !A}}$$

and we say **depth** of a rule, the number of boxes containing it.

Stratification

We can think to the promotion rule as a **box**:

$$\boxed{\dfrac{\dfrac{\vdots}{!\Gamma \vdash \mathtt{M} : A}}{!\Gamma \vdash \mathtt{M} : !A}}$$

and we say **depth** of a rule, the number of boxes containing it.

Non-stratification of a proof in Linear Logic.

The depth of any rule **can** change during the cut elimination.

$$!A \multimap !!A \qquad !A \multimap A \qquad !A \multimap !A \otimes !A \qquad !A_1 \otimes !A_2 \multimap !A$$

Stratification

If we remove the principles:

$$!A \multimap !!A \qquad !A \multimap A$$

we have instead:

Stratification: The depth of any rule **cannot** change during the cut elimination.

Stratification

If we remove the principles:

$$!A \multimap !!A \qquad !A \multimap A$$

we have instead:

Stratification: The depth of any rule **cannot** change during the cut elimination.

So, we can consider only the rules

$$\frac{\Gamma \vdash A}{!\Gamma \vdash !A}\ (\Box) \qquad \frac{!A, !A, \Gamma \vdash B}{!A, \Gamma \vdash B}\ (\nabla)$$

Reduction at depth i

A cut elimination step at depth i:

i) duplicate part of the proof at depth $i+1$,
ii) decrease the number of rules at depth i,
iii) does not affect the depths $j < i$,
iv) does not increase the global depth.

Reduction at depth i

A cut elimination step at depth i can only:

i) duplicate part of the proof at depth $i+1$,
ii) decrease the number of rules at depth i,
iii) does not affect the depths $j < i$,
iv) does not increase the global depth.

$$\dfrac{\boxed{\dfrac{\Gamma \vdash A}{!\Gamma \vdash !A}} \qquad \dfrac{\Delta, !A, !A \vdash B}{\Delta, !A \vdash B}\,(\nabla)}{!\Gamma, \Delta \vdash B}\,(cut) \quad \mapsto \quad \dfrac{\dfrac{\boxed{\dfrac{\Gamma \vdash A}{!\Gamma \vdash !A}} \qquad \dfrac{\boxed{\dfrac{\Gamma \vdash A}{!\Gamma \vdash !A}} \qquad \Delta, !A, !A \vdash B}{!\Gamma, \Delta, A \vdash B}\,(cut)}{!\Gamma, !\Gamma, \Delta \vdash B}\,(cut)}{!\Gamma, \Delta \vdash B}\,(\nabla)$$

This can be iterated to obtain an exponential at each fixed depth i.

Reduction at depth i

A cut elimination step at depth i can only:

i) duplicate part of the proof at depth $i+1$,
ii) decrease the number of rules at depth i,
iii) does not affect the depths $j < i$,
iv) does not increase the global depth.

$$\dfrac{\dfrac{\Gamma, A \vdash C}{\Gamma \vdash A \multimap C} \quad \dfrac{\Delta \vdash A \quad C, \Sigma \vdash B}{\Delta, A \multimap C, \Sigma \vdash B}}{!\Gamma, \Delta, \Sigma \vdash B}\ (cut) \mapsto \dfrac{\dfrac{\Delta \vdash A \quad \Gamma, A \vdash C}{\Delta, \Gamma \vdash C}\ (cut) \quad C, \Sigma \vdash B}{\Delta, \Gamma, \Sigma \vdash B}\ (cut)$$

Reduction at depth i

A cut elimination step at depth i can only:

i) duplicate part of the proof at depth $i+1$,
ii) decrease the number of rules at depth i,
iii) does not affect the depths $j < i$,
iv) does not increase the global depth.

$$\boxed{\dfrac{\dfrac{\Gamma, A \vdash C}{\Gamma \vdash A \multimap C} \quad \dfrac{\Delta \vdash A \quad C, \Sigma \vdash B}{\Delta, A \multimap C, \Sigma \vdash B}}{!\Gamma, \Delta, \Sigma \vdash B}\ (cut)} \mapsto \boxed{\dfrac{\dfrac{\Delta \vdash A \quad \Gamma, A \vdash C}{\Delta, \Gamma \vdash C}\ (cut) \quad C, \Sigma \vdash B}{\Delta, \Gamma, \Sigma \vdash B}\ (cut)}$$

Reduction at depth i

A cut elimination step at depth i can only:

i) duplicate part of the proof at depth $i + 1$,
ii) decrease the number of rules at depth i,
iii) does not affect the depths $j < i$,
iv) does not increase the global depth.

Because of the **stratification!**

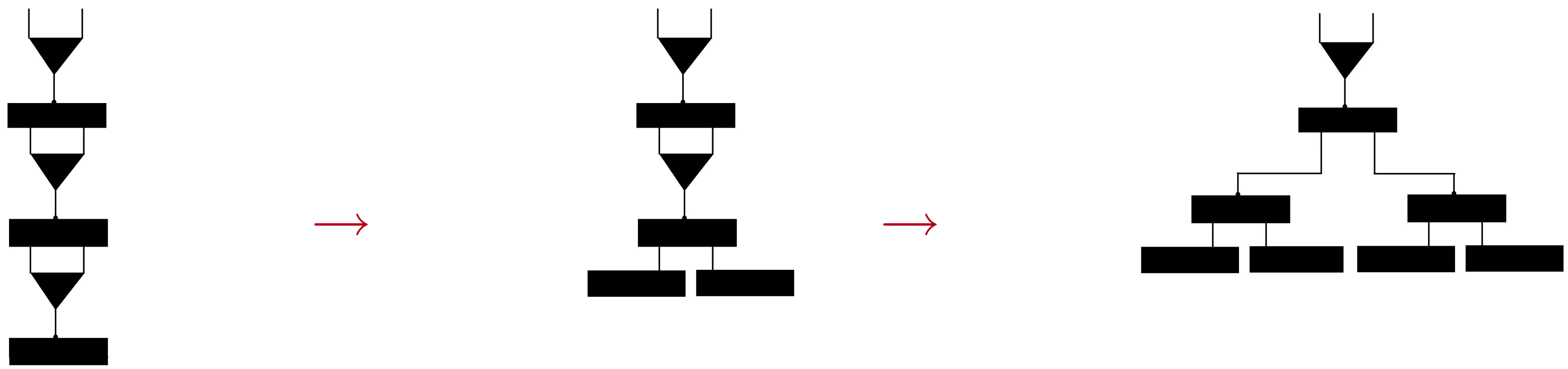
Stratified Reduction

We can perform a **depth-by-depth** reduction:

$$
\left\|\begin{array}{l} |\Pi|_0 \\ |\Pi|_1 \\ |\Pi|_2 \\ |\Pi|_3 \\ \vdots \\ |\Pi|_d \end{array}\right\|
\to_0^*
\left\|\begin{array}{l} |\Pi|_0 \\ 2^{|\Pi|_1} \\ 2^{|\Pi|_2} \\ 2^{|\Pi|_3} \\ \\ 2^{|\Pi|_d} \end{array}\right\|
\to_1^*
\left\|\begin{array}{l} |\Pi|_0 \\ 2^{|\Pi|_1} \\ 2^{2^{|\Pi|_2}} \\ 2^{2^{|\Pi|_3}} \\ \\ 2^{2^{|\Pi|_d}} \end{array}\right\|
\to_2^*
\left\|\begin{array}{l} |\Pi|_0 \\ 2^{|\Pi|_1} \\ 2^{2^{|\Pi|_2}} \\ 2^{2^{2^{|\Pi|_3}}} \\ \\ 2^{2^{2^{|\Pi|_d}}} \end{array}\right\|
\ldots
\left\|\begin{array}{l} |\Pi|_0 \\ 2^{|\Pi|_1} \\ 2^{2^{|\Pi|_2}} \\ 2^{2^{2^{|\Pi|_3}}} \\ \vdots \\ 2^{\cdot^{\cdot^{\cdot^{2^{2^{|\Pi|_d}}}}}} \end{array}\right\|
$$

The exponential blow-up

Most complex element **Most complex element reduced**



- The duplication of boxes depending over (more than one) free variables allows exponential time normalization.
- Limiting this kind of behavior corresponds to reduce the complexity of the normalization to polynomial time.

Light Affine Logic

$$\frac{\Gamma \vdash A \quad \Gamma \subseteq \{B\}}{!\Gamma \vdash !A}\ (p) \qquad \frac{!A, !A, \Gamma \vdash B}{!A, \Gamma \vdash B}\ (c) \qquad \frac{\Gamma, \Delta \vdash A}{\S\Gamma, !\Delta \vdash \S A}\ (d)$$

Light Affine Logic

$$\frac{\Gamma \vdash A \quad \Gamma \subseteq \{B\}}{!\Gamma \vdash !A}\ (p) \qquad \frac{!A, !A, \Gamma \vdash B}{!A, \Gamma \vdash B}\ (c) \qquad \frac{\Gamma, \Delta \vdash A}{\S\Gamma, !\Delta \vdash \S A}\ (d)$$

The restriction on the (p) rule corresponds to rule out the law:

$$!A_1 \otimes \ldots \otimes !A_n \multimap !A$$

The rules (d) corresponds to re-introduce a weak version of it:

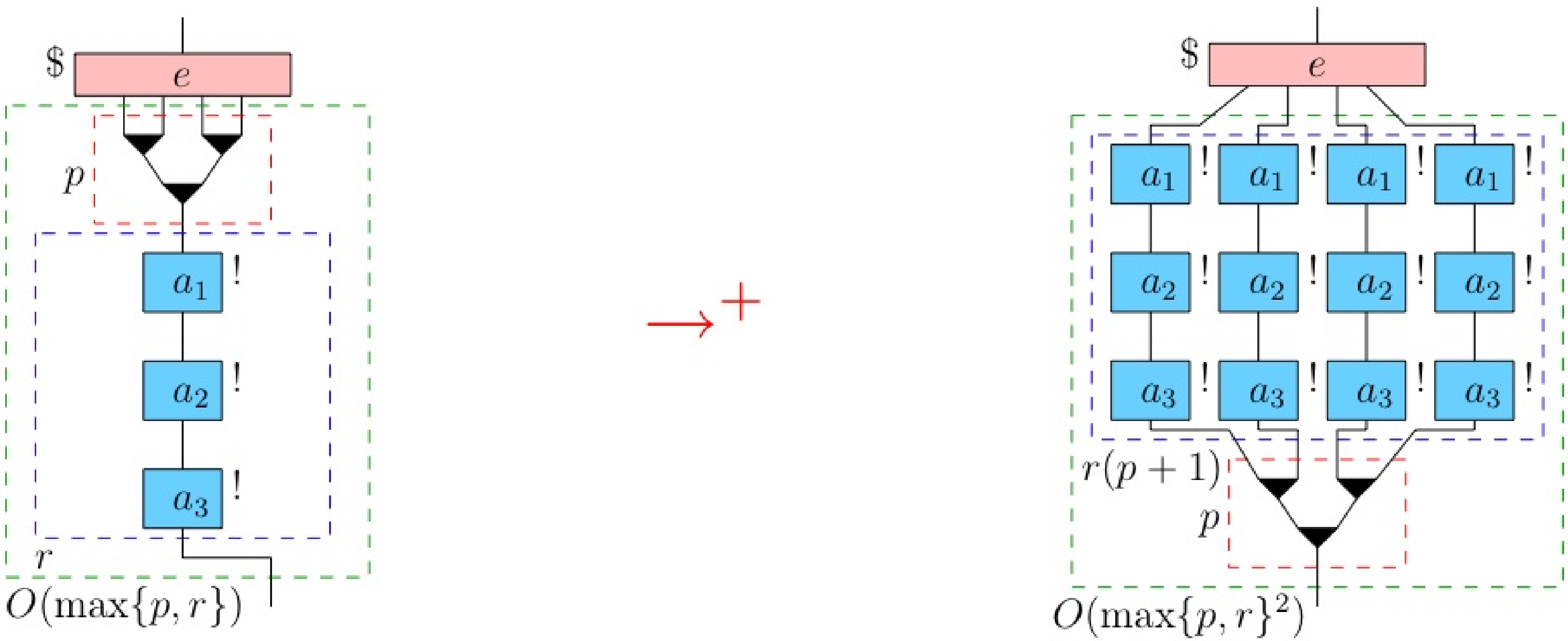
$$!A_1 \otimes \ldots \otimes !A_n \multimap \S A$$

Light Affine Logic

$$\boxed{\dfrac{\Gamma \vdash A \quad \Gamma \subseteq \{B\}}{!\Gamma \vdash !A}\ (p)} \qquad \dfrac{!A, !A, \Gamma \vdash B}{!A, \Gamma \vdash B}\ (c) \qquad \boxed{\dfrac{\Gamma, \Delta \vdash A}{\S\Gamma, !\Delta \vdash \S A}\ (d)}$$

- !-boxes ((p) rules) can be duplicated.
- §-boxes ((d) rules) cannot be duplicated

A sketch of LAL soundness - 1



The size $|\Pi|_j$ of depth $i < j$ becomes (at most) $|\Pi|_j^2$ at each round i.

A sketch of LAL soundness - 2

We can perform a **depth-by-depth** reduction:

$$\left\| \begin{array}{l} |\Pi|_0 \\ |\Pi|_1 \\ |\Pi|_2 \\ |\Pi|_3 \\ \vdots \\ |\Pi|_d \end{array} \right\| \rightarrow_0^* \left\| \begin{array}{l} |\Pi|_0 \\ (|\Pi|_1)^2 \\ (|\Pi|_2)^2 \\ (|\Pi|_3)^2 \\ \\ (|\Pi|_d)^2 \end{array} \right\| \rightarrow_1^* \left\| \begin{array}{l} |\Pi|_0 \\ (|\Pi|_1)^2 \\ ((|\Pi|_2)^2)^2 \\ ((|\Pi|_3)^2)^2 \\ \\ ((|\Pi|_d)^2)^2 \end{array} \right\| \rightarrow_2^* \left\| \begin{array}{l} |\Pi|_0 \\ (|\Pi|_1)^2 \\ ((|\Pi|_2)^2)^2 \\ (((|\Pi|_3)^2)^2)^2 \\ \\ (((|\Pi|_d)^2)^2)^2 \end{array} \right\| \cdots \left\| \begin{array}{l} |\Pi|_0 \\ (|\Pi|_1)^2 \\ ((|\Pi|_2)^2)^2 \\ (((|\Pi|_3)^2)^2)^2 \\ \vdots \\ (((|\Pi|_d)^2)^2 \cdots)^2 \end{array} \right\|$$

A sketch of LAL soundness - 3

The size of the proof Π after the stratified reduction is bounded by:

$$O\left(|\Pi|^{2^d}\right)$$

- d is the maximal depth of a rule in Π.

The same method can be used to reason about reduction steps, hence this gives a bound on the number of β-normalization steps.

Note: in LAL, data types have a fixed depth. This means that the depth depends just on the program part, hence we can work in polynomial time.

A sketch of LAL Completeness

- We have addition and multiplication in LAL as:

$$\mathbf{add} : \mathbf{N} \multimap \mathbf{N} \multimap \mathbf{N} \qquad \mathbf{mult} : \mathbf{N} \multimap \mathbf{N} \multimap \S\mathbf{N}$$

So, we can program a polynomial $p(x)$ of degree d as:

$$\mathbf{p} : \mathbf{N} \multimap \S^{2d}\mathbf{N}$$

A sketch of LAL Completeness

- We have addition and multiplication in LAL as:

$$\mathbf{add} : \mathbf{N} \multimap \mathbf{N} \multimap \mathbf{N} \qquad \mathbf{mult} : \mathbf{N} \multimap \mathbf{N} \multimap \S\mathbf{N}$$

  So, we can program a polynomial $p(x)$ of degree d as:

$$\mathbf{p} : \mathbf{N} \multimap \S^{2d}\mathbf{N}$$

- Transition on Turing Machines (TM) can be programmed as:

$$\mathbf{tr} : \mathbf{TM} \multimap \mathbf{TM}$$

A sketch of LAL Completeness

- We have addition and multiplication in LAL as:

$$\mathbf{add} : \mathbf{N} \multimap \mathbf{N} \multimap \mathbf{N} \qquad \mathbf{mult} : \mathbf{N} \multimap \mathbf{N} \multimap \S\mathbf{N}$$

  So, we can program a polynomial $p(x)$ of degree d as:

$$\mathbf{p} : \mathbf{N} \multimap \S^{2d}\mathbf{N}$$

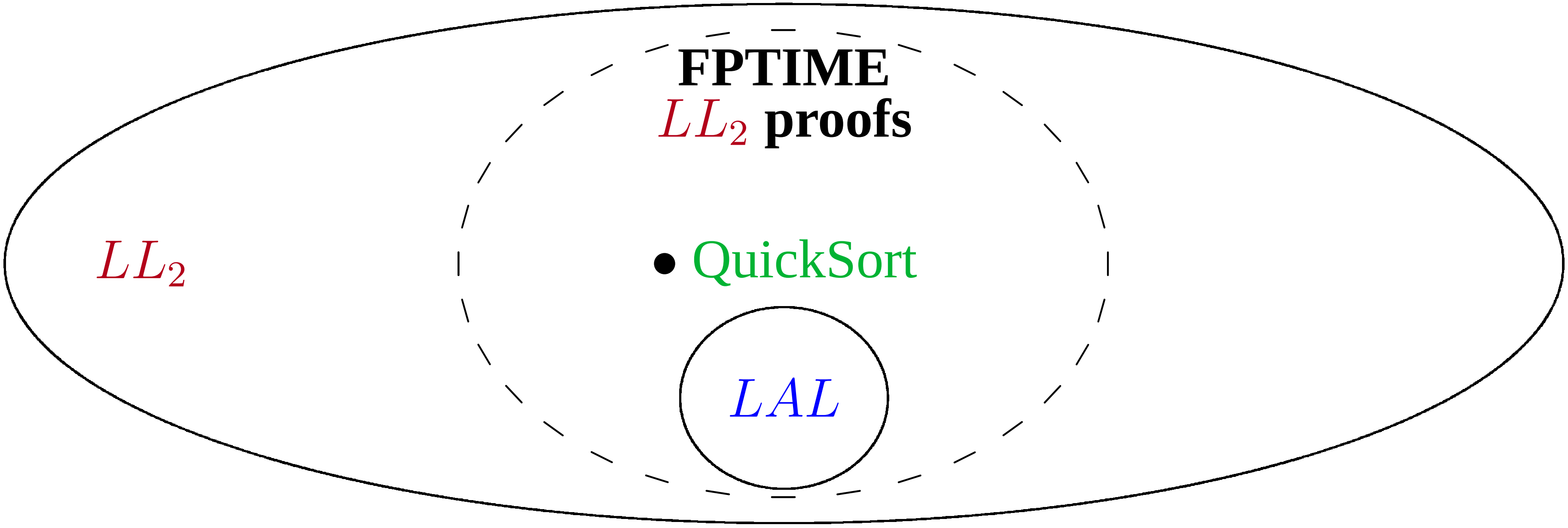
- Transition on Turing Machines (TM) can be programmed as:

$$\mathbf{tr} : \mathbf{TM} \multimap \mathbf{TM}$$

- By some manipulations we can obtain FPTIME Turing machines as:

$$\mathbf{t} : \mathbf{W} \multimap \S^{2d+1}\mathbf{W}$$

Lack of Expressivity



Back to Boundedness

Bounded Linear Logic - 1

One of the earliest examples of a Linear Logic system capturing FPTIME.
Formulas:

$$A ::= \alpha(p_1, \ldots, p_n) \mid A \otimes A \mid A \multimap A \mid \forall \alpha . A \mid \; !_{x<p} A$$

The idea is very simple:

$$!_{x<5} A = A[0/x] \otimes \cdots \otimes A[4/x]$$

However, the technicalities are increased by the fact that p is in general a polynomial expression, e.g.

$$!_{x<y^2} A = A[0/x] \otimes \cdots \otimes A[y^2 - 1/x]$$

This is however also it strength.

Bounded Linear Logic - 2

Rules:

$$\frac{\Gamma \vdash B}{\Gamma, !_{x<w} A \vdash B}\ (w) \qquad \frac{\Gamma, A[x := 0] \vdash B}{\Gamma, !_{x<1+w} A \vdash B}\ (d)$$

$$\frac{\Gamma, !_{x<p} A, !_{x<q} A[x := p + x] \vdash B}{\Gamma, !_{x<p+q+w} A \vdash B}\ (c)$$

$$\frac{!_{z<q_1(x)} A_1[y := v_1(x) + z], \ldots, !_{z<q_n(x)} A_n[y := v_n(x) + z] \vdash B}{!_{y<v_i(p)+w_1} A_1, \ldots, !_{y<v_n(p)+w_n} A_n \vdash !_{x<p} B}\ (p)$$

where $v_i(x) = \sum_{z<x} q_i(z)$.

A source of inspiration: Bounded Linear Logic - 2

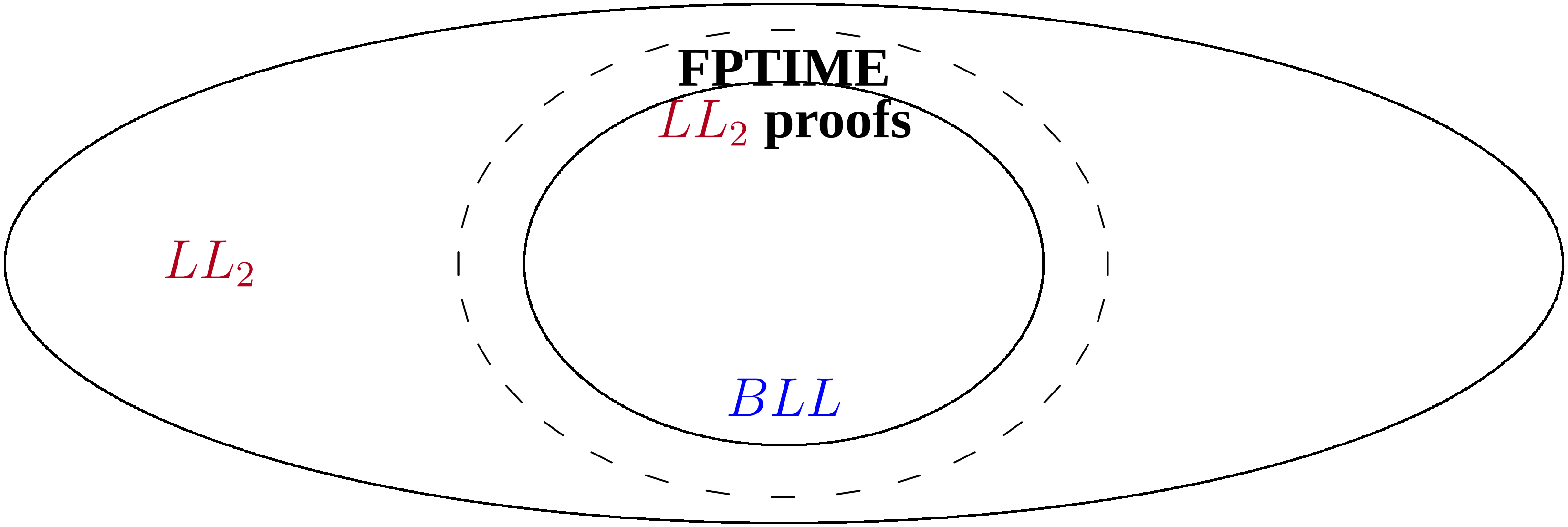
Bounded Linear Logic is sound and complete for FPTIME.

Moreover it has some interesting properties.

A source of inspiration: Bounded Linear Logic - 2

Bounded Linear Logic is sound and complete for FPTIME.

Moreover it has some interesting properties.



A source of inspiration: Bounded Linear Logic - 2

Bounded Linear Logic is sound and complete for FPTIME.

Moreover it has some interesting properties.

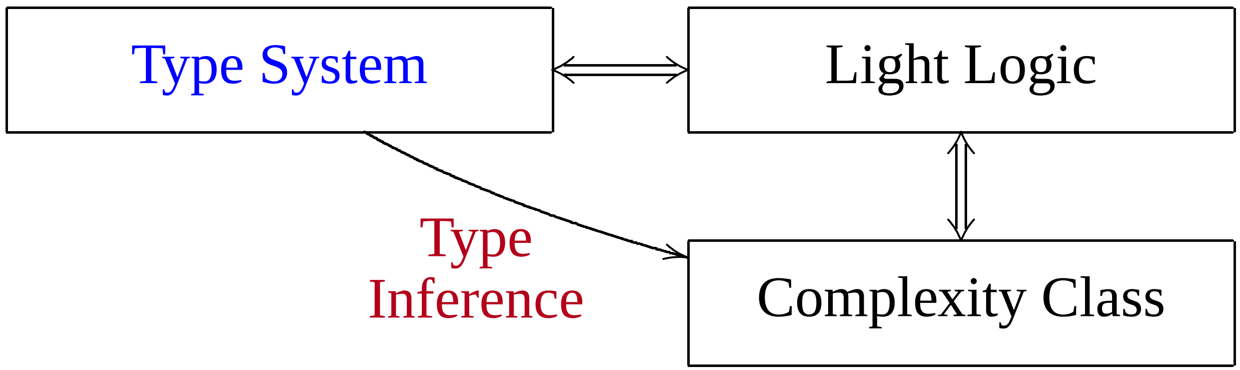
- The "bounding" part can be seen as a kind of extra-conditions. As an example:

$$\frac{\Gamma, !_{x<p}A, !_{x<q}A[x := p + x] \vdash B \quad p + q \leq r}{\Gamma, !_{x<r}A \vdash B}\ (c)$$

Computation

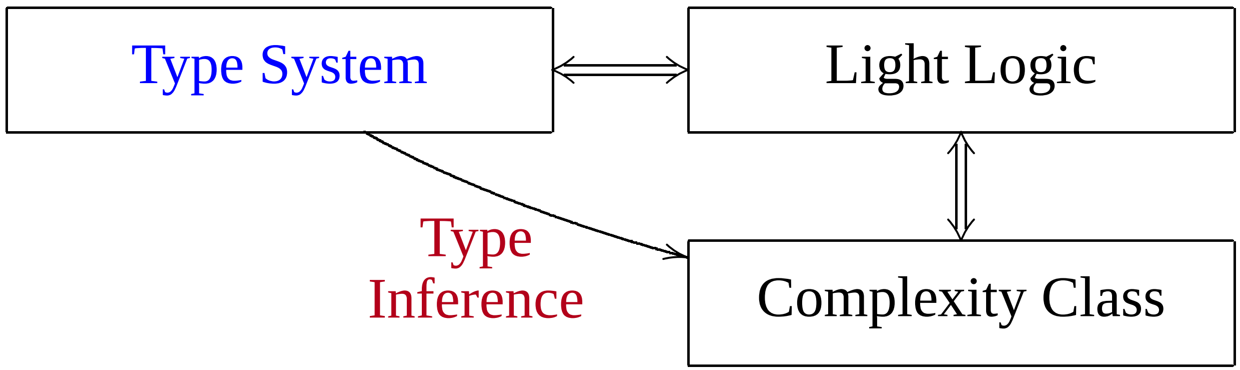
Light Type Systems

Stratified Light Logics have been used to design type systems:



Light Type Systems

Stratified Light Logics have been used to design type systems:

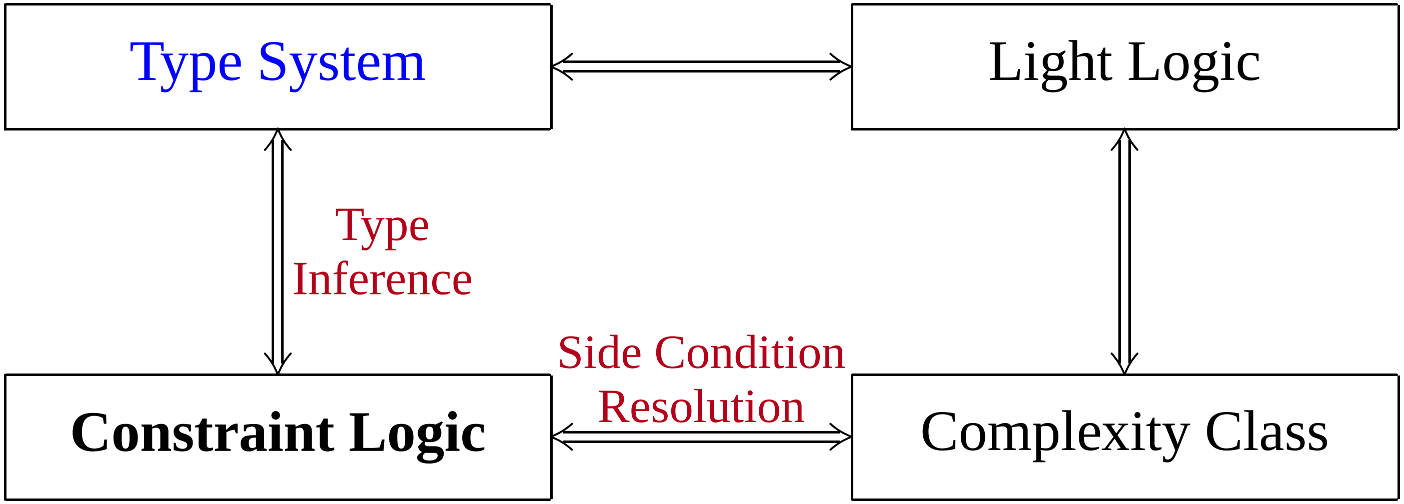


This approach is effective, flexible and robust:

- □ Type Inference usually decidable in polynomial time.
- □ Useful to characterize different complexity classes: FPTIME, ELEMENTARY, FPSPACE, FNPTIME, FEXPTIME.
- □ With minor modifications they are useful for: higher-order, different recursion schemes, control operators, multithreding and side effects.

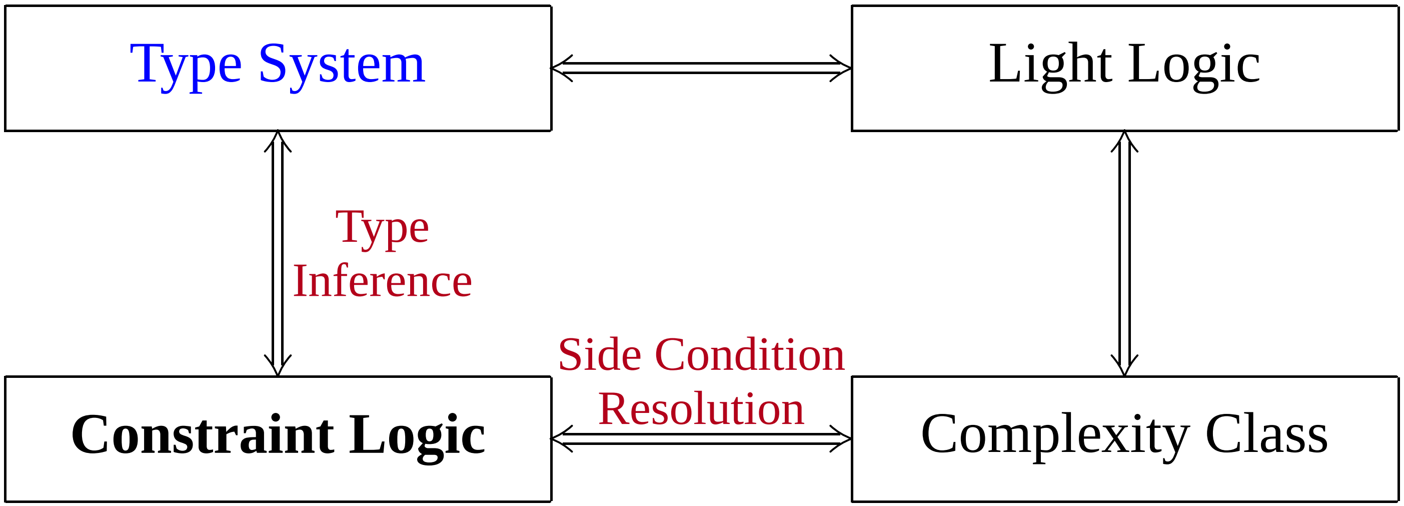
Relative Complete Type Systems

Boundedness can be used in a slightly different way.



Relative Complete Type Systems

Boundedness can be used in a slightly different way.



- This approach can give a system that is relatively complete with respect to the solvability of the side conditions by an oracle.
- This gives a general method to analyze program complexity.
- A similar approach has been used in an interactive framework for the class LOGSPACE.

THANKS