A Geometric Theory of Computational Complexity

Thomas Seiller
Department of Computer Science, University of Copenhagen
(soon) CNRS – LIPN (Paris 13 Univ.)

seiller@di.ku.dk seiller@lipn.fr

Thirty Years of Linear Logic, Rome
October 25th, 2017

Plan of the Talk

- ▶ **Linear Logic and Complexity**
- Geometry of Interaction and Complexity
- Interaction Graphs and Complexity

  A Danish Winter
- Complexity Theory: the Barriers
- What is a Program?
- Interaction Graphs and Complexity, Revisited

Logic in Complexity

- Descriptive Complexity (DC) studies types of logics whose individual sentences characterise exactly particular complexity classes. E.g. (Fagin 74) NPTIME is the set of problems definable by existential second order boolean formulas. DC led to the proof that NL = coNL (Immermann 88).
- Implicit Computational Complexity (ICC) studies algorithmic complexity only in terms of restrictions of languages and computational principles, e.g considering restrictions on recursion schemes (Bellantoni and Cook 92).
- Constrained Linear Logics is a Curry-Howard approach to ICC, where one defines "subsystems" of Girard's linear logic which capture complexity classes (i.e. one can write less proofs, thus less programs, thus one can compute less functions).

Proofs as Programs – Curry-Howard – correspondence

Proof Theory	Computer Science
Proof	Program
Proof	Data
Formulas	Types
Cut rule	Application
Cut Elimination	Execution (Computation)
...	...

Proofs as Programs – Curry-Howard – correspondence

- Integers: $\mathrm{nat} := \forall X \;\; (X \to X) \to (X \to X)$
- Functions from integers to integers: $\mathrm{nat} \to \mathrm{nat}$
- If $[n]$ is a (cut-free) proof of nat, and $[f]$ a proof of $\mathrm{nat} \to \mathrm{nat}$, we can define the proof $[f][n]$:

$$\dfrac{\begin{array}{cc}[f] & [n] \\ \vdots & \vdots \\ \mathrm{nat} \vdash \mathrm{nat} & \vdash \mathrm{nat}\end{array}}{\vdash \mathrm{nat}}\ \mathrm{cut}$$

- The cut elimination procedure applied to $[f][n]$ corresponds to the computation of $f(n)$. The cut-free proof it produces is equal to $[f(n)]$.

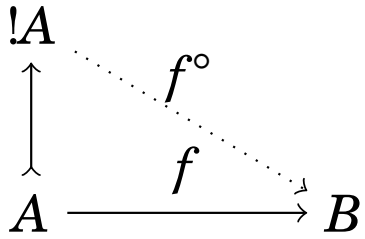
Linear Logic and Implicit Computational Complexity

In *Bounded Linear Logic* (Girard, Scedrov, Scott):

> The activity of manipulating types was recognized a long time ago as analogous to proving theorems in intuitionistic logic—this is now technically known as the Curry-Howard isomorphism (or the propositions-as-types paradigm)—but the origins of this idea date back to old intuitionism in the early 1900s and especially to Heyting and Kolmogorov in the 1920s. The situation is actually more involved: the idea makes better sense when combined with formalist tradition: logic offers not only a paradigm for basic specifications but also a mode of execution, namely through cut-elimination or its variants, e.g., natural deduction, all dating back to Gentzen's work in the 1930s. Here we show that this paradigm "computation as cut-elimination" is flexible enough to express a notion of *feasible computation.*

Linear Logic and Implicit Computational Complexity

- **Normal functors (Girard).** A model of programs as functors, in which functors representing programs are analytical, i.e. can be factored through a linear functor on a kind of tensor algebra $!A := \bigoplus_{k \leqslant 0} A \otimes A \otimes \cdots \otimes A$



- **Linear logic** is obtained by acknowledging this decomposition into the syntax, i.e. the usual implication $A \Rightarrow B$ becomes $!A \multimap B$;
- **LL for Complexity.** The rules governing the modality ! can be modified to define sub-systems characterising complexity classes. E.g. Elementary Linear Logic (ELL), Light Linear Logic (LLL).
  - BLL (Girard, Scedrov, Scott 92) characterises FP
  - ELL/LLL (Girard 98) characterise ELEM/FP
  - SLL (Lafont 2004) characterises FP
  - L^3/L^4 (Baillot, Mazza 2010) characterise ELEM/FP

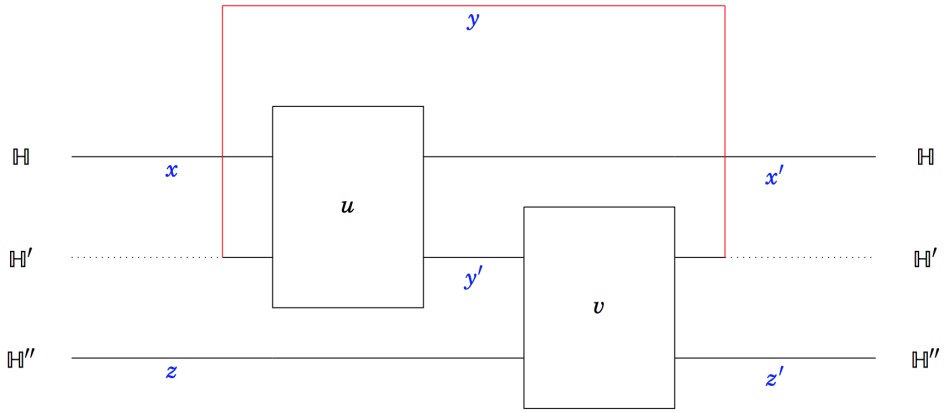
Plan of the Talk

- Linear Logic and Complexity
- ▶ **Geometry of Interaction and Complexity**
- Interaction Graphs and Complexity

  A Danish Winter
- Complexity Theory: the Barriers
- What is a Program?
- Interaction Graphs and Complexity, Revisited

Geometry of Interaction

Logic	Computer Science	Algebras
Proof $\pi \vdash \mathbf{Nat} \Rightarrow \mathbf{Nat}$	Program $f : \mathrm{nat} \to \mathrm{nat}$	Operator* $F \in \mathscr{L}(\mathbb{H} \oplus \mathbb{H})$
Proof $\rho \vdash \mathbf{Nat}$	Data $n : \mathrm{nat}$	Operator $N \in \mathscr{L}(\mathbb{H})$
Cut Rule $\mathrm{cut}(\pi, \rho)$	Application $f(n)$	Functional Equation $\begin{cases} F(x \oplus y) = x' \oplus y' \\ N(x') = x \end{cases}$
Cut elimination $\mathrm{cut}(\pi, \rho) \rightsquigarrow \mu \vdash \mathbf{Nat}$	Computation $f(n) \rightsquigarrow m : \mathrm{nat}$	Construction of a solution $\mathrm{Ex}(F, A)(y) = y' \in \mathscr{L}(\mathbb{H})$



*Bounded/Continuous linear map. Think of matrices.

Algebras as a model of computation

Theorem (Girard '06)

If $a \in \mathscr{L}(\mathbb{H} \oplus \mathbb{K})$, $b \in \mathscr{L}(\mathbb{H})$ *are operators of norm at most* 1, *the solution to the feedback equation involving* a *and* b *exists, is unique, and is an operator of norm at most* 1 *in the von Neumann algebra generated by* a *and* b.

Translation

(the unit ball of) a von neumann algebra = set of (untyped) programs.

Geometry of Interaction: Girard's Normativity Paper

In 2012, Girard proposes the following, based on the hyperfinite factor model of GoI.

- Consider the interpretation of ELL binary integers, i.e. elements of the type $\forall X, !(X \multimap X) \to (!(X \multimap X) \to !(X \multimap X))$;
- Restrict your attention to operators in a specific subalgebra $\mathfrak{S}$;
- Provide a characterisation of coNL as a subset of those operators.

This was later:

- reworked (Aubert, Seiller 2016);
- adapted to characterise L (Aubert, Seiller 2016);
- adapted to the more syntactic unification-based GoI (Aubert, Bagnol, Pistone, Seiller 2014);
- extended in this setting to P (Aubert, Bagnol, Seiller 2016).

Algebras as a model of computation

Theorem (Girard '06)

If $a \in \mathscr{L}(\mathbb{H} \oplus \mathbb{K})$, $b \in \mathscr{L}(\mathbb{H})$ *are operators of norm at most* 1, *the solution to the feedback equation involving* a *and* b *exists, is unique, and is an operator of norm at most* 1 *in the von Neumann algebra generated by* a *and* b.

Translation

(the unit ball of) a von neumann algebra = set of (untyped) programs.

Conjecture

Different von Neumann algebras = different degrees of expressivity.

This wild idea rests upon two results: the algebras $\mathscr{L}(\mathbb{H})$ and $\mathfrak{R} \subset \mathscr{L}(\mathbb{H})$ (the type II_∞ hyperfinite factor) model respectively pure lambda-calculus (Turing-complete) and "elementary programs".

Sub-algebras and complexity classes

Reality is a bit more subtle: the set of programs represented by a von neumann algebra $\mathfrak{N}$ depends also on a (maximal commutative) sub-algebra $\mathfrak{A} \subset \mathfrak{N}$.

Theorem (Seiller)

Depending on the (maximal abelian) subalgebra $\mathfrak{A} \subset \mathfrak{R}$ chosen, the expressivity of the set of programs modelled by the von Neumann algebra $\mathfrak{R}$ (the type II_∞ hyperfinite factor) varies.

The theorem is more precise than that, but the main point of interest is that the conjecture, as stated above is false. It can be corrected as follows.

(Corrected) Conjecture

Different pairs $\mathfrak{A} \subset \mathfrak{N}$ = different degrees of expressivity.

Limits of the algebraic approach

- **It (seems to) lead to dead-ends.** The different degrees of expressivity are linked with subtle properties of the pair $\mathfrak{A} \subset \mathfrak{N}$; and there are more open questions than answers in this particular domain of mathematics;
- **It is NOT intuitive AT ALL.** How one can express complexity constraints as choices of subalgebras?

Plan of the Talk

- Linear Logic and Complexity
- Geometry of Interaction and Complexity
- ▶ **Interaction Graphs and Complexity**

  A Danish Winter
- Complexity Theory: the Barriers
- What is a Program?
- Interaction Graphs and Complexity, Revisited

Defining concrete algebras

Principle

Replace pairs $\mathfrak{A} \subset \mathfrak{N}$ by pairs $(\mathbf{X}, \mathfrak{m})$ of a measured space $\mathbf{X}$ and a monoid (group) $\mathfrak{m}$ of measurable maps $\mathbf{X} \to \mathbf{X}$.

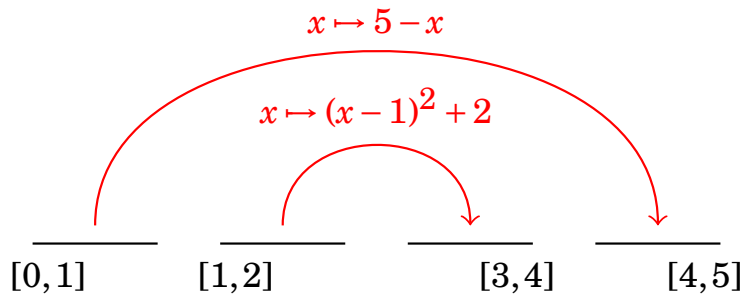
The correspondence with the previous presentation is as follows (when $\mathfrak{m}$ is a group of measure-preserving maps):

- the algebra $\mathfrak{A}$ corresponds to the space $\mathbf{X}$.
  I.e. $\mathfrak{A} := L^\infty(\mathbf{X}) \subset \mathscr{L}(L^2(\mathbf{X}))$.
- the algebra $\mathfrak{N}$ corresponds to the group $\mathfrak{m}$.
  I.e. each measure-preserving map ϕ defines a unitary $u_\phi \in \mathscr{L}(L^2(\mathbf{X}))$ by precomposition, and $\mathfrak{N}$ is the algebra generated by the set $\{u_\phi \mid \phi \in \mathfrak{m}\}$.
- elements of the algebra $\mathfrak{N}$ correspond to *graphings* (generalized graphs).

Graphings and Complexity

What's a graphing?

- Pick a directed graph.
- Replace vertices by measurable sets, e.g. intervals on the real line.
- Decide *how* (i.e. which element of $\mathfrak{m}$) the edges map sources to targets.



The parameters of the construction:

- A measure space $(X, \mathscr{B}, \mu)$;
- A monoid $\mathfrak{m}$ of measurable maps $X \to X$ – called a **microcosm**;
- A monoid Ω;
- A type of graphing (e.g. deterministic, probabilistic);
- A measurable map $g : \Omega \to \mathbf{R}_{\geqslant 0} \cup \{\infty\}$.

Hierarchies of models

Complexity Constraints

Quantitative Aspects
(e.g. probabilities, effects)

Theorem (Seiller, APAL 2017)

For every monoid of measurable maps $\mathfrak{m}$ *(and every monoid* Ω *, and every measurable map* $g : \Omega \to \mathbf{R}_{\geqslant 0} \cup \{\infty\}$ *), the set of* $\mathfrak{m}$*-graphings defines a non-degenerate model of Multiplicative-Additive Linear Logic .*

Geometric Measurement
(Ihara/Ruelle Zeta Functions)

AT LEAST!

Constraints on Graphings
(e.g. deterministic: (partial) measured dynamical systems,
probabilistic: (discrete time) Markov processes)

Hierarchies of models

Complexity Constraints

Theorem (Seiller, APAL 2017)

For every monoid of measurable maps $\mathfrak{m}$ *(and every monoid* Ω *, and every measurable map* $g : \Omega \to \mathbf{R}_{\geqslant 0} \cup \{\infty\}$ *), the set of* $\mathfrak{m}$*-graphings defines a non-degenerate model of Multiplicative-Additive Linear Logic .*

All Geometry of Interaction constructions are recovered as specific cases

Operators in C* / von Neumann algebras (1989,1990,2011)

Unification/Resolution clauses / Prefix Rewriting (1995,2016)

Microcosms: Geometric Aspect of Complexity

We can define microcosms

$$\mathfrak{m}_1 \subset \mathfrak{m}_2 \subset \cdots \subset \mathfrak{m}_\infty \subset \mathfrak{n} \subset \mathfrak{p}$$

in order to obtain the following characterisations (as the type $\mathtt{nat} \to \mathtt{nbool}$).

Microcosm	$\mathbb{M}_m^{\mathrm{det}}$	$\mathbb{M}_m^{\mathrm{ndet}}$		$\mathbb{M}_m^{\mathrm{prob}}$	Logic	Machines
$\mathfrak{m}_1$	REG	REG	REG	STOC	MALL	2-way Automata (2FA)
⋮	⋮	⋮	⋮	⋮	⋮	⋮
$\mathfrak{m}_k$	D_k	N_k	coN_k	P_k	(…)	k-heads 2FA
⋮	⋮	⋮	⋮	⋮	⋮	⋮
$\mathfrak{m}_\infty$	L	NL	coNL	PL	(…)	multihead-head 2FA (2MHFA)
$\mathfrak{n}$	P	P	P	PP	(…)	2MHFA + Pushdown Stack

Refines and generalises both:

- a series of characterisations of complexity classes (e.g. L, P) with operators (with Aubert) and logic programs (with Aubert, Bagnol and Pistone);
- an independent result where I relate the expressivity of GoI models with a classification of *inclusions of maximal abelian sub-algebras*:

$$\ell^\infty(\mathbf{X}) \subseteq \ell^\infty(\mathbf{X}) \rtimes \mathfrak{m} \quad \left(\subseteq \mathscr{B}(\ell^2(\mathbf{X}))\right) \quad \text{[Feldman-Moore 1977]}$$

Microcosms: Geometric Aspect of Complexity

We can define microcosms

$$\mathfrak{m}_1 \subset \mathfrak{m}_2 \subset \cdots \subset \mathfrak{m}_\infty \subset \mathfrak{n} \subset \mathfrak{p}$$

in order to obtain the following characterisations (as the type $\mathtt{nat} \to \mathtt{nbool}$).

Microcosm	$\mathbb{M}_m^{\mathrm{det}}$	$\mathbb{M}_m^{\mathrm{ndet}}$		$\mathbb{M}_m^{\mathrm{prob}}$	Logic	Machines
$\mathfrak{m}_1$	REG	REG	REG	STOC	MALL	2-way Automata (2FA)
⋮	⋮	⋮	⋮	⋮	⋮	⋮
$\mathfrak{m}_k$	D_k	N_k	coN_k	P_k	(…)	k-heads 2FA
⋮	⋮	⋮	⋮	⋮	⋮	⋮
$\mathfrak{m}_\infty$	L	NL	coNL	PL	(…)	multihead-head 2FA (2MHFA)
$\mathfrak{n}$	P	P	P	PP	(…)	2MHFA + Pushdown Stack

- Only known correspondence between infinite hierarchies of mathematical objects and complexity classes.
- Indicates a strong connection between *geometry* and complexity: cf. microcosms generalise *group actions*, use of (generalised) Zeta functions, (homotopy) equivalence between microcosms implies equality of the classes.

A Geometric Theory of Complexity

Microcosm	$\mathbb{M}_m^{\mathrm{det}}$	$\mathbb{M}_m^{\mathrm{ndet}}$		$\mathbb{M}_m^{\mathrm{prob}}$	Logic	Machines
$\mathfrak{m}_1$	REG	REG	REG	STOC	MALL	2-way Automata (2FA)
⋮	⋮	⋮	⋮	⋮	⋮	⋮
$\mathfrak{m}_k$	D_k	N_k	coN_k	P_k	(...)	k-heads 2FA
⋮	⋮	⋮	⋮	⋮	⋮	⋮
$\mathfrak{m}_\infty$	L	NL	coNL	PL	(...)	multihead-head 2FA (2MHFA)
$\mathfrak{n}$	P	P	P	PP	(...)	2MHFA + Pushdown Stack

Conjecture

(Equivalence classes of) microcosms correspond to complexity constraints.

Conjecture

$$\mathfrak{m} \equiv \mathfrak{n} \Leftrightarrow \mathtt{Pred}(\mathfrak{m}) = \mathtt{Pred}(\mathfrak{n})$$

A Geometric Theory of Complexity

Microcosm	$\mathbb{M}_m^{\mathrm{det}}$	$\mathbb{M}_m^{\mathrm{ndet}}$		$\mathbb{M}_m^{\mathrm{prob}}$	Logic	Machines
$\mathfrak{m}_1$	REG	REG	REG	STOC	MALL	2-way Automata (2FA)
⋮	⋮	⋮	⋮	⋮	⋮	⋮
$\mathfrak{m}_k$	D_k	N_k	coN_k	P_k	(…)	k-heads 2FA
⋮	⋮	⋮	⋮	⋮	⋮	⋮
$\mathfrak{m}_\infty$	L	NL	coNL	PL	(…)	multihead-head 2FA (2MHFA)
$\mathfrak{n}$	P	P	P	PP	(…)	2MHFA + Pushdown Stack

Conjecture

$$\mathfrak{m} \equiv \mathfrak{n} \Leftrightarrow \mathtt{Pred}(\mathfrak{m}) = \mathtt{Pred}(\mathfrak{n})$$

A Geometric Theory of Complexity

Microcosm	$\mathbb{M}_m^{\mathrm{det}}$	$\mathbb{M}_m^{\mathrm{ndet}}$		$\mathbb{M}_m^{\mathrm{prob}}$	Logic	Machines
$\mathfrak{m}_1$	REG	REG	REG	STOC	MALL	2-way Automata (2FA)
⋮	⋮	⋮	⋮	⋮	⋮	⋮
$\mathfrak{m}_k$	D_k	N_k	coN_k	P_k	(…)	k-heads 2FA
⋮	⋮	⋮	⋮	⋮	⋮	⋮
$\mathfrak{m}_\infty$	L	NL	coNL	PL	(…)	multihead-head 2FA (2MHFA)
$\mathfrak{n}$	P	P	P	PP	(…)	2MHFA + Pushdown Stack

Conjecture

$$\mathfrak{m} \equiv \mathfrak{n} \Leftrightarrow \mathtt{Pred}(\mathfrak{m}) = \mathtt{Pred}(\mathfrak{n})$$

Enable (co)homological invariants to prove separation , e.g. $\ell^{(2)}$-Betti numbers:

$$\mathtt{Pred}(\mathfrak{m}) = \mathtt{Pred}(\mathfrak{n}) \Rightarrow \mathfrak{m} \equiv \mathfrak{n} \Rightarrow \mathscr{P}(\mathfrak{m}) \simeq \mathscr{P}(\mathfrak{n}) \overset{!}{\Rightarrow} \ell^{(2)}(\mathscr{P}(\mathfrak{m})) = \ell^{(2)}(\mathscr{P}(\mathfrak{n}))$$

($\mathscr{P}(\mathfrak{m}) = \{(x,y) \mid \exists h \in \mathfrak{m}, h(x) = y\}$ is a *measurable preorder*)

Plan of the Talk

- Linear Logic and Complexity
- Geometry of Interaction and Complexity
- Interaction Graphs and Complexity
  - **A Danish Winter**
- Complexity Theory: the Barriers
- What is a Program?
- Interaction Graphs and Complexity, Revisited

Plan of the Talk

- Linear Logic and Complexity
- Geometry of Interaction and Complexity
- Interaction Graphs and Complexity

  A Danish Winter
- ▶ **Complexity Theory: the Barriers**
- What is a Program?
- Interaction Graphs and Complexity, Revisited

Computational Complexity

Once upon a time, people asked (and answered) the following question:

- What is a computable function?

That's all good in theory, but once first computers were built and in use, people realised there was another important question, namely:

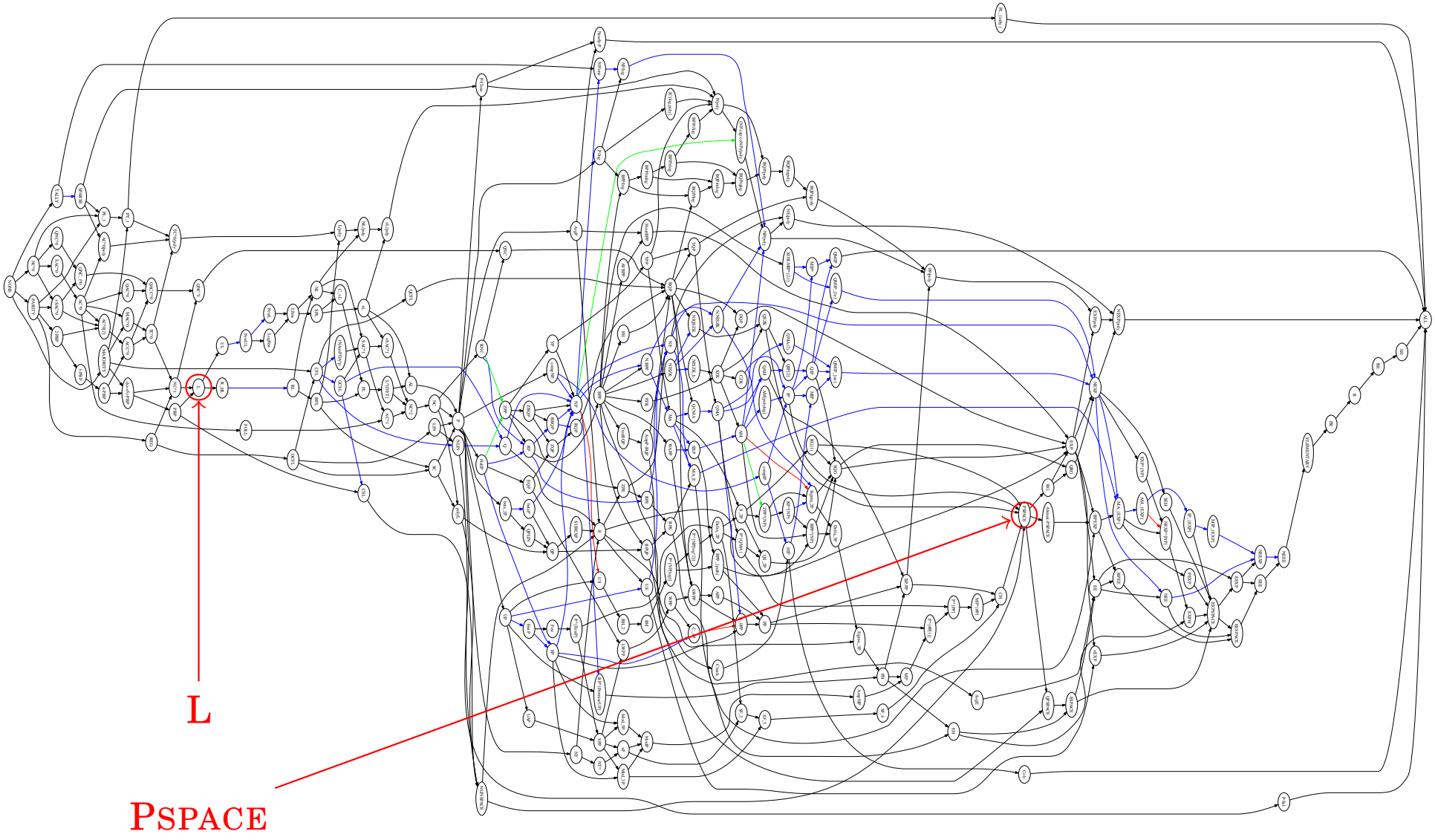
- What is an *efficiently* computable function?

I.e. what if we wanted the answer to be produced within our lifetimes (well, quicker than that really if the result is to be used somehow).

- This somehow marked the birth of *computational complexity*: three papers addressed this question within a year.
  (Cobham 1965; Hartmanis and Stearns 1965; Edmonds 1965)

And now let's fastforward.

Complexity Theory, Today (well, in 2006)



Complexity Theory, Today

- A number of separation results were obtained, most of them in the 70s. But a lot of questions remain open. For instance: we know $\mathrm{L} \subsetneq \mathrm{PSPACE}$, but we don't know which of these inclusions are strict: $\mathrm{L} \subset \mathrm{P} \subset \mathrm{NP} \subset \mathrm{PSPACE}$.
- In fact, the three more important results are *negative results* (called *barriers*) showing that known proof methods for separation of complexity classes are inefficient w.r.t. currently open problems. They are: relativisation (1975), natural proofs (1995), and algebrization (2008).
- **Thus: no proof methods for (new) separation results exist today.**
- (Proviso) One research program (but one only) is considered as viable for obtaining new results: Mulmuley's *Geometric Complexity Theory* (GCT). However, according to Mulmuley, **if** GCT produces results, it will not be during our lifetimes (and maybe not our childrens' lifetime either*), since it requires the development of much involved new techniques in algebraic geometry.

Barriers in Computational Complexity.

Morally, there are two barriers (here for P vs. NP):

- Relativization/Algebrization: Proof methods that are oblivious to the use/disuse of oracles are ineffective.
  - There exists oracles $\mathscr{A}, \mathscr{B}$ such that:

$$\textsc{Ptime}^{\mathscr{A}^{\sim}} \neq \textsc{NPtime}^{\mathscr{A}}$$

$$\textsc{Ptime}^{\mathscr{B}} = \textsc{NPtime}^{\mathscr{B}^{\sim}}$$

- Natural Proofs: Proof methods expressible as (**Constructible**, Large) predicates on boolean functions are ineffective.
  - A natural proof of $\textsc{Ptime} \neq \textsc{NPtime}$ implies that no pseudo-random generators (in P) have exponential hardness.

Conclusion: Lack of proof methods for separation. But why?

Barriers as Guidelines

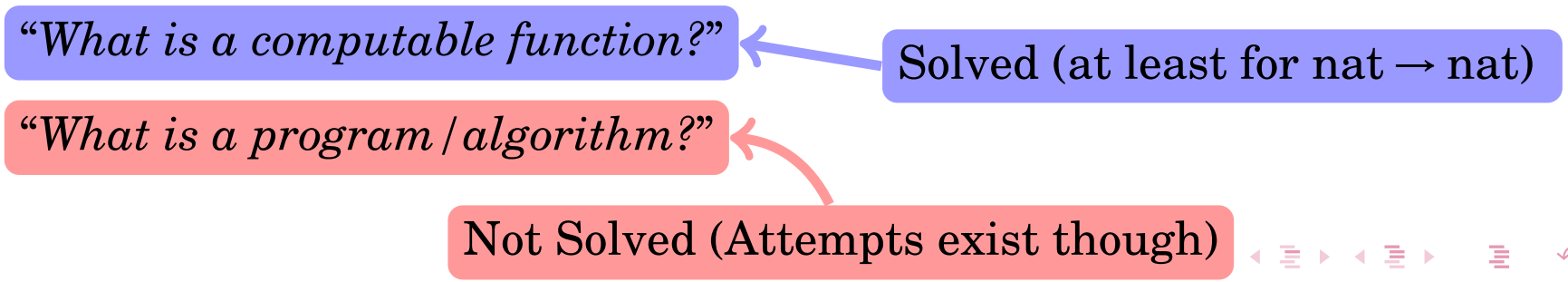
State of the Art in Complexity (Separation Problem): Barriers.

- Relativization/Algebrization: Proof methods that are oblivious to the use/disuse of oracles are ineffective.
  - Separation proof methods should depend on the computational principles allowed in the model.
- Natural Proofs: Proof methods expressible as (**Constructible**, Large) predicates on boolean functions are ineffective.
  - Separation proof methods should not "quotient" the set of programs too much. (by definition, complexity classes are non-decidable predicates on boolean functions)

Conclusion: Lack of proof methods for separation.

Thus arguably due to the following:

(Note Moschovakis already argues along these lines, but does not discuss barriers)



A more general issue

P.J. Denning, in *The Field of Programmers Myth*, Comm. ACM 47 (7), 2004:

> *We are captured by a historic tradition that sees programs as mathematical functions [...].*

- The notion of computable functions is a very bad measure of the expressivity of a model of computation. E.g. Neil Jones' *Life without cons*.

Program	Data				
class	order 0	Order 1	Order 2	Order 3	Limit
RW, unrestricted	REC.ENUM	REC.ENUM	REC.ENUM	REC.ENUM	REC.ENUM
RWPR, fold only	PRIM.REC	PRIM^1REC	PRIM^2REC	PRIM^3REC	$\text{PRIM}^\omega\text{REC}$
RO, unrestricted	PTIME	EXPTIME	EXP^2TIME	EXP^3TIME	ELEMENTARY
ROTR tail recursive	LOGSPACE	PSPACE	EXPSPACE	EXP^2SPACE	ELEMENTARY
ROPR, fold only	LOGSPACE	PTIME	PSPACE	EXPTIME	ELEMENTARY

- More generally, complexity is a bad measure of the expressivity. Somehow, it is erroneous to think that characterising a specific class of functions, e.g. Ptime, means we understood something about complexity.
- This functional point of view can explain why we are not able to generalise the notion of complexity to higher-order functions / concurrent computation.

Plan of the Talk

- Linear Logic and Complexity
- Geometry of Interaction and Complexity
- Interaction Graphs and Complexity

  A Danish Winter
- Complexity Theory: the Barriers
- ▶ **What is a Program?**
- Interaction Graphs and Complexity, Revisited

Better foundations

- Hypothesis: Current lack of proof methods for separation is due to a lack of adequate **mathematical** answer to the question "What is a program?".
- Suppose there exists adequate mathematical foundations.
  I.e. (this is objectively very fuzzy) for every computation $\mathscr{C}$ there exists a mathematical object $\|\mathscr{C}\|$ with $\|\cdot\|$ injective.

Claim

There are no barriers for the set of proof techniques based on such foundations.

- The argument is simple. Injectivity implies that if all such proof methods can be shown ineffective, it amounts to prove that separation is undecidable.

Better foundations

- Hypothesis: Current lack of proof methods for separation is due to a lack of adequate **mathematical** answer to the question "What is a program?".
- Suppose there exists adequate mathematical foundations.
  I.e. (this is objectively very fuzzy) for every computation $\mathscr{C}$ there exists a mathematical object $\|\mathscr{C}\|$ with $\|\cdot\|$ injective, up to some trivial equivalences (e.g. renaming of control states).

Claim

There are no barriers for the set of proof techniques based on such foundations.

- The argument is simple. (Quasi-)injectivity implies that if all such proof methods can be shown ineffective, it amounts to prove that separation is undecidable.

Better foundations

- Hypothesis: Current lack of proof methods for separation is due to a lack of adequate **mathematical** answer to the question "What is a program?".
- Suppose there exists adequate mathematical foundations.
  I.e. (this is objectively very fuzzy) for every computation $\mathscr{C}$ there exists a mathematical object $\|\mathscr{C}\|$ with $\|\cdot\|$ injective, up to some trivial equivalences (e.g. renaming of control states).

Claim

There are no barriers for the set of proof techniques based on such foundations.

- The argument is simple. (Quasi-)injectivity implies that if all such proof methods can be shown ineffective, it amounts to prove that separation is undecidable.

What is a computation/algorithm?

Several proposals.

- Turing
- Kolmogorov
- Gandy
- Moschovakis
- Gurevich.

  An ASM is a sequence of "updates" to be applied on a model of first-order logic over a fixed signature. An update is defined as either (1) a generalised assignment $f(s_1,\ldots,s_n) := t$, where f is any function symbol and the s_i and t are arbitrary terms, or (2) a conditional **if** C **then** P or **if** C **then** P **else** Q, where C is a propositional combination of equalities between terms and P, Q are sequences of updates, or (3) a parallel composition of sequences of update.

Critic of Gurevich approach

- Gurevich.

  An ASM is a sequence of "updates" to be applied on a model of first-order logic over a fixed signature. An update is defined as either (1) a generalised assignment $f(s_1,\ldots,s_n) := t$, where f is any function symbol and the s_i and t are arbitrary terms, or (2) a conditional **if** C **then** P or **if** C **then** P **else** Q, where C is a propositional combination of equalities between terms and P,Q are sequences of updates, or (3) a parallel composition of sequences of update.

- From a point of view of capturing the notion of computation: arguably satisfying for sequential, probabilistic computation, but it is unclear if it generalises well to, e.g., cellular automata, continuous time.
- From the point of view of complexity: ad-hoc objects, not based on well-founded mathematical theory. (In fact, ASM may be described as generalised pseudo-code.) It does not enable new *proof methods*.

What is a computation/program/algorithm?

From a philosophical point of view, very few work tackle this question (at least, I could not find many). It is quite surprising, given the development of new models of computation (e.g. quantum, biological).
As a starting point for the reflexion, let us consider the following questions:

- Is the universe just a big computation?
- If I let a rock fall from the top of a tower, is this a computation? If not, why?
- What about if I let a rock fall from the same tower, but depending on the initial height it activates a number n of mechanical apparatus that release a number m of balls? (e.g. the rock activates levers every meter, with the lever at height k releasing 2k+1 balls)
- What about a similar experiment where flowing water activates some mill equipped with a similar apparatus? (Is this a computation on streams?)

Where I stand

It seems important to distinguish between several different notions.

- Distinguish between experiments and a computation.
  - Differ in their intention: test the theory vs. use the theory to (locally) predict the outcome.
- Distinguish between a computation and a program.
  - Differ in their abstraction: mechanical processes / Electric signals vs. some flow of information.
  - A program is somehow distinguished from its physical realisation – the computation. I.e. one can *run* a program several times, producing several computations. However, it is bound to a model of computation (i.e. turing machines, automata, etc.).
- Distinguish between a program and an algorithm.
  - An algorithm is an abstraction of programs, free of models of computation. E.g. Sieve of Eratosthenes.
  - Very difficult task to formalise this notion.
    cf. Blass, Derschowitz, Gurevich *When are two algorithms the same?*.

Plan of the Talk

- Linear Logic and Complexity
- Geometry of Interaction and Complexity
- Interaction Graphs and Complexity

  A Danish Winter
- Complexity Theory: the Barriers
- What is a Program?
- ▶ **Interaction Graphs and Complexity, Revisited**

What is a program?

Informal Definition

A program is a dynamical process possibly involving exchange/duplication/erasure/modification of information.

[Complexity] Implicit Computational Complexity.
Size-change termination (Lee, Jones, Ben-Amram), mwp-polynomials (Jones, Kristiansen), Loop peeling (Moyen, Rubiano, Seiller).

[Semantics] Dynamic Semantics
Geometry of Interaction (Girard), Game Semantics (Abramsky/Jagadeesan/Malacaria, Hyland/Ong), Interaction Graphs (Seiller).

[Compilation] Compilation techniques.
Work by U. Schöpp (cf. Habilitation thesis), Loop peeling (Moyen, Rubiano, Seiller)

[VLSI design] Synthesis methods for VLSI design.
Geometry of Synthesis programme (Ghica).

What is a program?

Informal Definition

A program is a dynamical process possibly involving exchange/duplication/erasure/modification of information.

In fact Girard's Geometry of interaction was intended as a proposal for mathematical foundations.

> *This paper is the main piece in a general program of mathematisation of algorithmics, called geometry of interaction. We would like to define independently of any concrete machine, any extant language, the mathematical notion of an algorithm (maybe with some proviso, e.g. deterministic algorithms), so that it would be possible to establish general results which hold once for all.*
>
> *Girard, Geometry of Interaction II (1988)*

Abstract Programs

Definition

An *abstract model of computation* (AMC) is defined as a monoid action $\alpha : M \curvearrowright \mathbf{X}$ of a monoid M on a space $\mathbf{X}$. I.e. $\alpha : M \to \mathscr{M}(\mathbf{X} \to \mathbf{X})$.

Definition

One can define the following partial order on AMCs. If $\alpha : M \curvearrowright \mathbf{X}$ and $\beta : N \curvearrowright \mathbf{Y}$ are AMCs, $\alpha \leqslant \beta$ iff there exists an automorphism $\phi : \mathbf{X} \to \mathbf{Y}$ s.t. for all $m \in M_\alpha$, $\phi(\alpha(m))$ is a glueing of a finite number of restrictions of elements $\beta(n_1), \beta(n_2), \ldots, \beta(n_k)$.
This induces an equivalence relation on AMCs (which is finer than homotopy equivalence).

Definition

An *abstract program* is a $\alpha(M)$-graphing.

Back to the results

AMC	det. model	non-det. model		prob. model
α_1	REG	REG	REG	STOC
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
α_k	D_k	N_k	co-N_k	P_k
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
α_∞	L	NL	coNL	PL
β	P	P	P	P?
γ	P	NP	coNP	PP?

Conjecture

If $\alpha : M \curvearrowright \mathbf{X}$ and $\beta : N \curvearrowright \mathbf{Y}$ are *separable* (by e.g. ℓ^2-Betti numbers), they characterise different complexity classes.

▶ Can be checked on classical separation results and the computation of the invariants on the AMCs $\alpha_1, \ldots, \alpha_k, \ldots, \alpha_\infty$.

Intrinsic/Mathematical complexity

Definition

An *abstract model of computation* (AMC) is defined as a monoid action $\alpha : M \curvearrowright \mathbf{X}$ of a monoid M on a space $\mathbf{X}$. I.e. $\alpha : M \to \mathscr{M}(\mathbf{X} \to \mathbf{X})$.

Definition

An *abstract program* is a $\alpha(M)$-graphing.

Definition

Define(!) the *intrinsic complexity* of a program as the *smallest* monoid action $\alpha : M \curvearrowright \mathbf{X}$ needed to interpret it as an abstract program.

Conjecture

Intrinsic complexity of programs coincide with computational complexity.

What about barriers?

Why barriers do not apply to this approach:

- (Relativisation/Algebrization)
  - How to describe oracles in this setting?
  - It has to be defined *explicitly*, i.e. extend the AMC by adding a new computational principle as a measurable map $o : \mathbf{O} \to \mathbf{O}$;
  - Impact the invariants: if $\alpha : M \curvearrowright \mathbf{X}$ and $\beta : N \curvearrowright \mathbf{Y}$ are separable, there are no reasons to believe that $\alpha + o : M \curvearrowright \mathbf{X}$ and $\beta + o : N \curvearrowright \mathbf{Y}$ are separable.
- (Natural Proofs)
  - As explained above, the approach should violate the constructivity axiom of the Natural Proof barrier.
  - More importantly, we can argue that if barriers exists in this setting then the separation problem is undecidable.

Summary

- We probably need to start a long overdue collaborative reflexion with philosophers (to guarantee some degree of “informal rigour” – cf. Kreisel) on the question: "What is a program?".
- The end of the talk proposes a tentative technical answer. While I believe it is a (good starting point for finding a) satisfying solution, I expect it to be challenged.
- The last part how this proposition could lead to a geometric theory of computation/complexity which could be exploited for developing separation methods.
- In particular, the approach defines a notion of complexity of *programs* intrinsically (i.e. as an equivalence class of group/monoid actions/acts), i.e. a definition which is not based on an arbitrary input/output behaviour.
- While I insisted on complexity issues, the whole framework comes from logic, and raises numerous questions as to which logical systems arise from these abstract models of computation.
- Although very abstract, this relates to ICC techniques that yielded an automatic optimisation tool (prototype) in the LLVM compiler.

A Geometric Theory of Computational Complexity

Thomas Seiller
Department of Computer Science, University of Copenhagen
(soon) CNRS – LIPN (Paris 13 Univ.)

seiller@di.ku.dk seiller@lipn.fr

Thirty Years of Linear Logic, Rome
October 25th, 2017