Why complexity theorists should care about philosophy

Thomas Seiller
Department of Computer Science, University of Copenhagen

seiller@di.ku.dk

May 25th, 2017

Computational Complexity

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

- What is a computable function?

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).

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)

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:

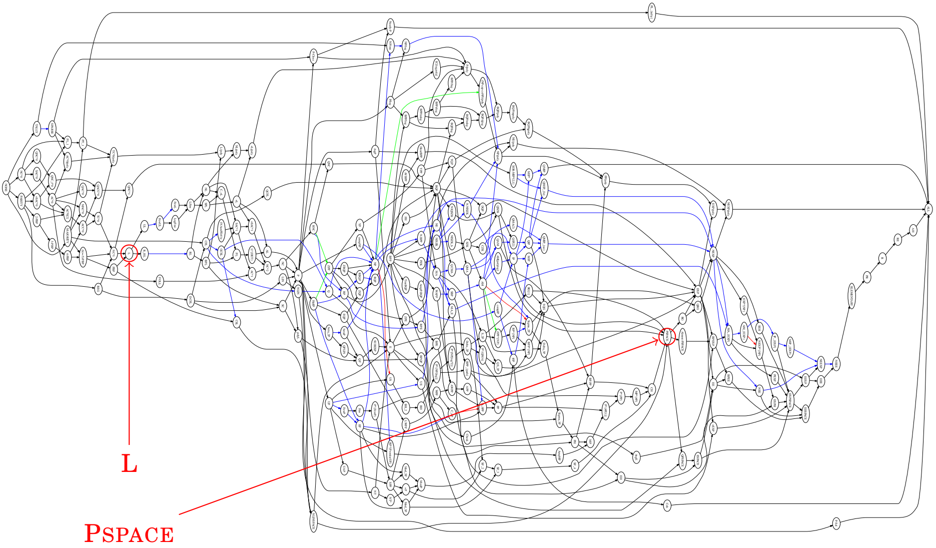
- What is an *efficiently* computable function?

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 $\mathrm{L} \subsetneq \mathrm{PS}\textsc{pace}$, but we don't know which of these inclusions are strict: $\mathrm{L} \subset \mathrm{P} \subset \mathrm{NP} \subset \mathrm{PS}\textsc{pace}$.

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).

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.**

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.

- Natural Proofs: Proof methods expressible as (**Constructible**, Large) predicates on boolean functions are ineffective.

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:

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

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

- 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.

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:

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

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

- 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.

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:

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

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

- 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.

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:

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

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

- 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.
- Natural Proofs: Proof methods expressible as (**Constructible**) predicates on boolean functions are ineffective.

Conclusion: Lack of proof methods for separation.

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**) predicates on boolean functions are ineffective.

Conclusion: Lack of proof methods for separation.

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**) predicates on boolean functions are ineffective.
  - Separation proof methods should not "quotient" the set of programs too much. (by definition, complexity classes are predicates on boolean functions)

Conclusion: Lack of proof methods for separation.

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**) predicates on boolean functions are ineffective.
  - Separation proof methods should not "quotient" the set of programs too much. (by definition, complexity classes are 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)

"*What is a computable function?*" ← Solved (at least for nat → nat)

"*What is a program / algorithm?*" ← Not Solved (Attempts exist though)

Digression: How to overcome barriers

- Mulmuley's program breaks *Largness*, i.e. it aims at developing techniques to prove lower bounds for a specific problem, i.e. the techniques are problem-dependent. (Mainly, GCT works with the determinant vs permanent problem in arbitrary characteristic and advocates for the use of techniques from algebraic geometry.)
- I want to break *constructivity*. (keeping the possibility of breaking largeness as well).

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 [...].*

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 class	**Data 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

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 class	Data 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	$\mathrm{PRIM}^1\mathrm{REC}$	$\mathrm{PRIM}^2\mathrm{REC}$	$\mathrm{PRIM}^3\mathrm{REC}$	$\mathrm{PRIM}^\omega\mathrm{REC}$
RO, unrestricted	PTIME	EXPTIME	$\mathrm{EXP}^2\mathrm{TIME}$	$\mathrm{EXP}^3\mathrm{TIME}$	ELEMENTARY
ROTR tail recursive	LOGSPACE	PSPACE	EXPSPACE	$\mathrm{EXP}^2\mathrm{SPACE}$	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.

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 class	Data 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.

Better foundations

- Hypothesis: Current lack of proof methods for separation is due to a lack of adequate **mathematical** foundations.
- 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** foundations.
- 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** foundations.
- 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.

What is a computation/algorithm?

Several proposals.

- Moschovakis
- Gurevich.

What is a computation/algorithm?

Several proposals.

- 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 $f(s_1,\dots,s_n) := t$, 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.

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.

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 $f(s_1,\dots,s_n) := t$, 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 our project: ad-hoc objects, not based on well-founded mathematical theory. In fact, ASM may be described as generalised pseudo-code.

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 even more actual, with 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.

Where I stand

It seems important to distinguish between several different notions.

- Distinguish between experiments and a computation.
  - You fix a mass to a spring, let go, and write down the oscillations. Is this a computation?

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.

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.

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.

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.).

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.

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.

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, which we will not consider here.
    cf. Blass, Derschowitz, Gurevich *When are two algorithms the same?*.

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, which we will not consider here.
    cf. Blass, Derschowitz, Gurevich *When are two algorithms the same?*.

What is a program?

Informal Definition

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

What is a program?

Informal Definition

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

This principle underlies a number of work.

What is a program?

Informal Definition

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

This principle underlies a number of work.

[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 a formalisation of this idea, Girard's Geometry of interaction, was intended as a proposal for mathematical foundations.

> *This paper is the main piece in a general program of mathematatisation 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)*

What is a program?

Informal Definition

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

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

- At first (technically) limited to sequential, deterministic, computation (may explain why it was somehow forgotten/discarded);

What is a program?

Informal Definition

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

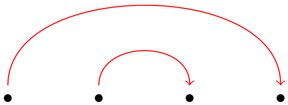
In fact a formalisation of this idea, Girard's Geometry of interaction, was intended as a proposal for mathematical foundations.

- At first (technically) limited to sequential, deterministic, computation (may explain why it was somehow forgotten/discarded);
- New approach – Interaction Graphs – bypasses these limitations and allows for modelling many aspects of computation. Technically, we replace operators (i.e. bounded linear operators acting on Hilbert spaces) by *graphings*, obtaining a model which is both more general and more tractable.

What's a graphing?

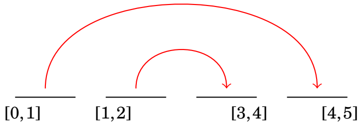
What's a graphing?

- Pick a directed graph.



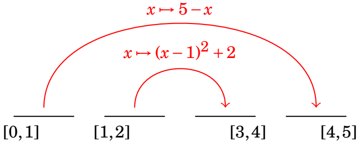
What's a graphing?

- Pick a directed graph.
- Replace vertices by measurable sets, e.g. intervals on the real line.



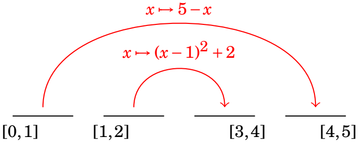
What's a graphing?

- Pick a directed graph.
- Replace vertices by measurable sets, e.g. intervals on the real line.
- Decide *how* the edges map sources to targets.



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 $g : \Omega \to \mathbf{R}_{\geqslant 0} \cup \{\infty\}$.

Models of computation and logic

Logic	Lambda-calculus	Interaction Graphs

Proofs "Pararoofs"	Terms "Paraterms"	Winning Graphings Graphings
Cut rule	Application	Feedback
Normalisation cut-elimination	Reduction β-rule	Execution Compute paths

"Proofness" Correctness criterion	Orthogonality $t \perp E(\cdot)$ iff $E(t)$ SN	Orthogonality Complicated measurement
Formulas $\text{Proofs}(A^{\perp}) = \text{Tests}(A)$	Types Realisability constr.	"Conducts" $C = T^{\perp}$, (iff $C = C^{\perp\perp}$)

Hierarchies of models

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.*

Hierarchies of models

Theorem (Seiller, APAL 2017)

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

AT LEAST!

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* $g : \Omega \rightarrow \mathbf{R}_{\geqslant 0} \cup \{\infty\}$*), the set of* $\mathfrak{m}$*-graphings defines a non-degenerate model of Multiplicative-Additive Linear Logic.*

Geometric Measurement
(Ihara/Ruelle Zeta Functions)

AT LEAST!

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

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)

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 $\texttt{nat} \rightarrow \texttt{nbool}$).

Microcosm	$\mathbb{M}_m^{\text{det}}$	$\mathbb{M}_m^{\text{ndet}}$		$\mathbb{M}_m^{\text{prob}}$	Logic	Machines
$\mathfrak{m}_1$	REG	REG	REG	STOC	MALL	2-way Automata (2FA)
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\mathfrak{m}_k$	D_k	N_k	coN_k	P_k	$(\dots)$	k-heads 2FA
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\mathfrak{m}_\infty$	L	NL	coNL	PL	$(\dots)$	multihead-head 2FA (2MHFA)
$\mathfrak{n}$	P	P	P	PP	$(\dots)$	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 `nat` → `nbool`).

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

Conjecture

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

A Geometric Theory of Complexity

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

Conjecture

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

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

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

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

A Geometric Theory of Complexity

Microcosm	$\mathbb{M}_m^{\text{det}}$	$\mathbb{M}_m^{\text{ndet}}$		$\mathbb{M}_m^{\text{prob}}$	Logic	Machines
$\mathfrak{m}_1$	Reg	Reg	Reg	Stoc	MALL	2-way Automata (2FA)
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\mathfrak{m}_k$	D_k	N_k	coN_k	P_k	(...)	k-heads 2FA
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\mathfrak{m}_\infty$	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:

$$\mathtt{Pred}(\mathfrak{m}) = \mathtt{Pred}(\mathfrak{n}) \Rightarrow \mathfrak{m} \equiv \mathfrak{n} \Rightarrow \mathscr{P}(\mathfrak{m}) \simeq \mathscr{P}(\mathfrak{n}) \overset{!}{\Rightarrow} \ell^{(2)}(\mathscr{P}(\mathfrak{m})) = \ell^{(2)}(\mathscr{P}(\mathfrak{n}))$$
$$(\mathscr{P}(\mathfrak{m}) = \{(x,y) \mid \exists h \in \mathfrak{m}, h(x) = y\} \text{ is a } \textit{measurable preorder})$$

Summary

- Understand the first part as a *manifesto* to start a collaborative reflexion on the question: "What is a program", in the same way researchers once tackled the question "What is a computable function?".
- The second part is my own proposition for an answer. While I believe it is a (good starting point for finding a) satisfying solution, I expect it to be challenged.
- The last part shows (well, very quickly mentions) how this proposition reveals some geometric nature of computation/complexity which could be exploited for developing separation methods.
- In particular, the approach defines the complexity of a *program* 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 lead to an automatic optimisation tool (prototype) in the LLVM compiler.

Why complexity theorists should care about philosophy

Thomas Seiller
Department of Computer Science, University of Copenhagen

seiller@di.ku.dk

May 25th, 2017