

A realisability approach to complexity theory

Thomas Seiller

Department of Computer Science, University of Copenhagen

`seiller@di.ku.dk`

November 27th, 2015

Introduction

The purpose is to relate two problems:

- **Mathematics.** Are two spaces X, Y homotopy equivalent?
 - ▶ Difficult to answer negatively;
- **C.S.** Are two complexity classes A, B equal?
 - ▶ Difficult to answer negatively;

Introduction

The purpose is to relate two problems:

- **Mathematics.** Are two spaces X, Y homotopy equivalent?
 - ▶ Difficult to answer negatively;
 - ▶ Some proof methods available (e.g. (co)homological invariants).
- **C.S.** Are two complexity classes A, B equal?
 - ▶ Difficult to answer negatively;

Introduction

The purpose is to relate two problems:

- **Mathematics.** Are two spaces X, Y homotopy equivalent?
 - ▶ Difficult to answer negatively;
 - ▶ Some proof methods available (e.g. (co)homological invariants).
- **C.S.** Are two complexity classes A, B equal?
 - ▶ Difficult to answer negatively;
 - ▶ No proof methods available, c.f. *Natural Proofs*.

In a nutshell

Develop a new approach to (implicit) computational complexity theory.

Slogan

There is a correspondence between complexity constraints and algebras.

In a nutshell

Develop a new approach to (implicit) computational complexity theory.

Slogan

There is a correspondence between complexity constraints and (some kinds of) algebras (modulo isomorphisms).

In a nutshell

Develop a new approach to (implicit) computational complexity theory.

Slogan

There is a correspondence between complexity constraints and (some kinds of) algebras (modulo isomorphisms).

- Two important motivations:
 - ▶ obtain a uniform mathematical approach to CT
 - ▶ gain new proof methods from mathematics

In a nutshell

Develop a new approach to (implicit) computational complexity theory.

Slogan

There is a correspondence between complexity constraints and (some kinds of) algebras (modulo isomorphisms).

- Two important motivations:
 - ▶ obtain a uniform mathematical approach to CT
 - ▶ gain new proof methods from mathematics
- At the intersection between two lines of work:
 - ▶ Implicit Computational Complexity, especially approaches using linear logic.
 - ▶ Realisability models for linear logic, and more precisely "interaction graphs".

Algebras as a model of computation

Logic	Computer Science	Algebras
Proof	Program	Operator*
Proof	Data	Operator

*Bounded/Continuous linear map. Think of matrices.

Algebras as a model of computation

Logic	Computer Science	Algebras
Proof	Program	Operator*
Proof	Data	Operator

*Bounded/Continuous linear map. Think of matrices.

Algebras as a model of computation

Logic	Computer Science	Algebras
Proof	Program	Operator*
Proof	Data	Operator
Cut Rule	Application	Functional Equation

*Bounded/Continuous linear map. Think of matrices.

Algebras as a model of computation

Logic	Computer Science	Algebras
Proof	Program	Operator*
Proof	Data	Operator
Cut Rule	Application	Functional Equation
Cut elimination	Computation	Construction of a solution

*Bounded/Continuous linear map. Think of matrices.

Algebras as a model of computation

Logic	Computer Science	Algebras
Proof $\pi \vdash \text{Nat} \Rightarrow \text{Nat}$	Program $f : \text{nat} \rightarrow \text{nat}$	Operator* $F \in \mathcal{L}(\mathbb{H} \oplus \mathbb{H})$
Proof	Data	Operator
Cut Rule	Application	Functional Equation
Cut elimination	Computation	Construction of a solution

*Bounded/Continuous linear map. Think of matrices.

Algebras as a model of computation

Logic	Computer Science	Algebras
Proof $\pi \vdash \mathbf{Nat} \Rightarrow \mathbf{Nat}$	Program $f : \mathbf{nat} \rightarrow \mathbf{nat}$	Operator* $F \in \mathcal{L}(\mathbb{H} \oplus \mathbb{H})$
Proof $\rho \vdash \mathbf{Nat}$	Data $n : \mathbf{nat}$	Operator $N \in \mathcal{L}(\mathbb{H})$
Cut Rule	Application	Functional Equation
Cut elimination	Computation	Construction of a solution

*Bounded/Continuous linear map. Think of matrices.

Algebras as a model of computation

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

*Bounded/Continuous linear map. Think of matrices.

Algebras as a model of computation

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

*Bounded/Continuous linear map. Think of matrices.

Algebras as a model of computation

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

*Bounded/Continuous linear map. Think of matrices.

Algebras as a model of computation

Theorem (Girard '06)

If $a \in \mathcal{L}(\mathbb{H} \oplus \mathbb{K})$, $b \in \mathcal{L}(\mathbb{H})$ 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 $\mathcal{L}(\mathbb{H})$ and $\mathfrak{R} \subset \mathcal{L}(\mathbb{H})$ (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: in fact 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 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.

(False) Conjecture

Different von Neumann algebras = different degrees of expressivity.

Sub-algebras and complexity classes

Reality is a bit more subtle: in fact 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 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.

(Corrected) Conjecture

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

Limits of the algebraic approach

The approach as described above is not very useful. In particular:

- **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 the h... complexity constraints are expressed in the algebraic framework?

This can be solved by using another approach of these models I developed. It has the advantage of being both more combinatorial and more general than the approach taken above.

Defining concrete algebras

Principle

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

The correspondence with the previous presentation is as follows:

- the algebra $\mathfrak{A}$ corresponds to the space $\mathbf{X}$;
- the algebra $\mathfrak{N}$ corresponds to the monoid m ;
- elements of the algebra $\mathfrak{N}$ correspond to *graphings* (generalized graphs).

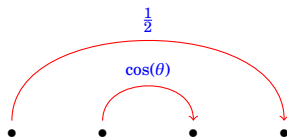
Digression: programs as graphings

What's a graphing?

Digression: programs as graphings

What's a graphing?

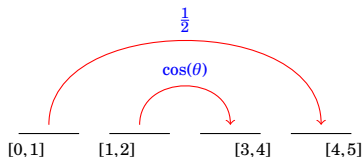
- Pick a directed graph, add weights (from a monoid Ω) on the edges.



Digression: programs as graphings

What's a graphing?

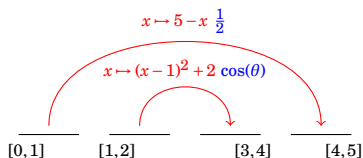
- Pick a directed graph, add weights (from a monoid Ω) on the edges.
- Consider that vertices are measurable sets, e.g. intervals.



Digression: programs as graphings

What's a graphing?

- Pick a directed graph, add weights (from a monoid Ω) on the edges.
- Consider that vertices are measurable sets, e.g. intervals.
- Decide *how* the edges map sources to targets.



Digression: programs as graphings

What's a graphing?

- Pick a directed graph, add weights (from a monoid Ω) on the edges.
- Consider that vertices are measurable sets, e.g. intervals.
- Decide *how* (i.e. **which element of m**) the edges map sources to targets.

The parameters of the interpretations:

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

Digression: programs as graphings

What's a graphing?

- Pick a directed graph, add weights (from a monoid Ω) on the edges.
- Consider that vertices are measurable sets, e.g. intervals.
- Decide *how* (i.e. **which element of m**) the edges map sources to targets.

The parameters of the interpretations:

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

Digression: programs as graphings

What's a graphing?

- Pick a directed graph, add weights (from a monoid Ω) on the edges.
- Consider that vertices are measurable sets, e.g. intervals.
- Decide *how* (i.e. **which element of m**) the edges map sources to targets.

The parameters of the interpretations:

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

First results

We can define monoids of measurable maps:

$$m_1 \subset m_2 \subset \dots \subset m_\infty \subset n \subset p$$

in order to obtain the following characterisations (which predicates can be computed by an element of the model).

Microcosm	$\mathbb{M}_m^{\det}$	$\mathbb{M}_m^{\text{ndet}}$	$\mathbb{M}_m^{\text{ndet}}$	$\mathbb{M}_m^{\text{prob}}$	Logic	Machines
m_1	REG	REG	REG	STOC	MALL	2-way Automata (2FA)
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
m_k	D_k	N_k	CON_k	P_k	(...)	k -heads 2FA
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
m_∞	L	NL	CONL	PL	(...)	multihead-head 2FA (2MHFA)
n	P	P	P	P?	(...)	2MHFA + Pushdown Stack
p	P	NP	CONP	PP?	ELL	Ptime Turing Machines

The conjecture, formally

- We define an equivalence relation on monoid of measurable maps.
- Notation: we fix every parameter but the monoid m , and write $\text{Pred}(m)$ the complexity class (predicates) thus characterised.

Theorem

If $m \equiv n$, then $\text{Pred}(m) = \text{Pred}(n)$.

Conjecture

The converse holds, i.e. $\text{Pred}(m) = \text{Pred}(n)$ implies $m \equiv n$.

If this conjecture holds, it would provide new proof techniques for separation through (co)homological invariants, e.g. $\ell^{(2)}$ -Betti numbers:

$$\text{Pred}(m) = \text{Pred}(n) \Rightarrow m \equiv n \Rightarrow \mathcal{P}(m) \simeq \mathcal{P}(n) \stackrel{!}{\Rightarrow} \ell^{(2)}(\mathcal{P}(m)) = \ell^{(2)}(\mathcal{P}(n))$$

where $\mathcal{P}(m) = \{(x, y) \mid \exists m \in m, m(x) = y\}$ is a “measurable preorder”.