Lower bounds for algebraic machines, semantically

Luc Pellissier, Thomas Seiller

Lower bounds for algebraic machines, semantically

Luc Pellissier
LACL, Faculté des Sciences et Technologie
61 avenue du Général de Gaulle
94010 Créteil, FRANCE
Email: luc.pellissier@lacl.fr

Thomas Seiller
CNRS, LIPN – UMR 7030
Université Sorbonne Paris Nord
99, Avenue Jean-Baptiste Clément
93430, Villetaneuse, FRANCE
Email: seiller@lipn.fr

***Abstract*—This paper presents a new semantic method for proving lower bounds in computational complexity. We use it to prove that `maxflow`, a PTIME complete problem, is not computable in polylogarithmic time on parallel random access machines (PRAMs) working with integers, showing that $\mathbf{NC_Z} \neq \mathbf{PTIME}$, where $\mathbf{NC_Z}$ is the complexity class defined by such machines, and PTIME is the standard class of polynomial time computable problems (on, say, a Turing machine). On top of showing this new separation result, we show our method captures previous lower bounds results from the literature: Steele and Yao's lower bounds for algebraic decision trees [1], Ben-Or's lower bounds for algebraic computation trees [2], Cucker's proof that $\mathbf{NC_R}$ is not equal to $\mathbf{PTIME_R}$ [3], and Mulmuley's lower bounds for "PRAMs without bit operations" [4].**

I. Introduction

Complexity theory has traditionally been concerned with proving *separation results* between complexity classes. Many problems remain open, such as the much advertised PTIME vs NPTIME question which concerns the difference between feasible sequential computability in deterministic and non-deterministic models, or equivalently the difference between feasible computation and feasible verification. This paper investigates questions related to the NC vs PTIME question, which concerns the difference between efficient sequential computation and (more than) efficient parallel computation.

Proving that two classes $B \subset A$ are not equal can be reduced to finding lower bounds for problems in A: by proving that certain problems cannot be solved with less than certain resources on a specific model of computation, one can show that two classes are not equal. Conversely, proving a separation result $B \subsetneq A$ provides a lower bound for the problems that are A*-complete* [5] – i.e. problems that are in some way *universal* for the class A.

The proven lower bound results are however very rough, and many separation problems remain as generally accepted conjectures. For instance, a proof that the class of non-deterministic exponential problems is not included in what is thought of as a very small class of circuits was not achieved until very recently [6].

The failure of most techniques of proof has been studied in itself, which lead to the proof of the existence of negative results that are commonly called *barriers*. Altogether, these results show that all proof methods we know are ineffective with respect to proving interesting lower bounds. Indeed, there are three barriers: relativisation [7], natural proofs [8] and algebrization [9], and almost every known proof method hits at least one of them: this shows the need for new methods[1]. However, to this day, only one research program aimed at proving new separation results is commonly believed to have the ability to bypass all barriers: Mulmuley and Sohoni's Geometric Complexity Theory (GCT) program [10]. This research program was inspired from an earlier lower bounds result by Mulmuley [4] which we strengthen in this paper.

A. Mulmuley's result

The NC vs PTIME question is one of the foremost open questions in computational complexity. In laymen's terms, it asks whether a problem efficiently computable on a sequential machine can be computed substantially more efficiently on a parallel machine. It is well known that any problem in NC, i.e. that is computable in polylogarithmic time on a parallel machine (with a polynomial number of processors), belongs to PTIME, i.e. is computable in polynomial time on a sequential machine. The converse, however, is expected to be false. Indeed, although many problems in PTIME can be shown to be in NC, some of them seem to resist efficient parallelisation. In particular it is not known whether the `maxflow` problem, known to be PTIME-complete [11], belongs to NC.

As part of the investigations on the NC vs PTIME question, a big step forward is due to K. Mulmuley. In 1999 [4], he showed that a notion of machine introduced under the name "PRAMs without bit operations" does not compute `maxflow` in polylogarithmic time. This notion of machine, quite exotic at first sight, corresponds to an algebraic variant of PRAMs, where registers contain integers and individual processors are allowed to perform sums, subtractions and products of integers. It is argued by Mulmuley that this notion of machine provides an expressive model of computation, able to compute some non trivial problems in NC such as Neff's algorithm for computing approximate roots of polynomials [12]. Although Mulmuley's result has represented a big step forward in the quest for a proof that PTIME and NC are not equal, the result was not strenghtened or reused in the last 20 years, and remained the strongest known lower bound result.

[1] In the words of S. Aaronson and A. Wigderson [9], "We speculate that going beyond this limit [algebrization] will require fundamentally new methods."

B. Contributions.

The main contribution of this work is a strengthening of Mulmuley's lower bounds result. While the latter proves that `maxflow` is not computable in polylogarithmic time in the model of "PRAMs without bit operations", we show here that `maxflow` is not computable in polylogarithmic time in the more expressive model of PRAMs over integers, making an additional step in the direction of a potential proof that NC is different from PTIME. Indeed, our result can be stated as

Theorem 1.

$$\mathrm{NC}_{\mathbf{Z}} \neq \textsc{Ptime},$$

where $\mathrm{NC}_{\mathbf{Z}}$ *is the set of problems decidable in polylogarithmic time by a (not necessarily uniform) family of* PRAMs *over* $\mathbf{Z}$.

The second contribution of the paper is the proof method itself, which is based on *dynamic semantics* for programs by means of *graphings*, a notion introduced in ergodic theory and recently used to define models of linear logic by Seiller [13], [14], [15], [16]. The dual nature of graphings, both continuous and discrete, is essential in the present work, as it enables invariants from continuous mathematics, in particular the notion of *topological entropy* for dynamical systems, while the finite representability of graphings is used in the key lemma (as the number of *edges* appears in the upper bounds of Lemma 5).

In particular, we show how this proof method captures known lower bounds and separation results in algebraic models of computation, namely Steele and Yao's lower bounds for algebraic decision trees [1], Ben-Or's lower bounds on algebraic computation trees [2], Cucker's proof that $\mathrm{NC}_{\mathbf{R}}$ is not equal to $\textsc{Ptime}_{\mathbf{R}}$ (i.e. answering the NC vs PTIME problem for computation over the real numbers).

C. A more detailed view of the proof method

One of the key ingredients in the proof is the representation of programs as graphings, and *quantitative soundness* results. We refer to the next section for a formal statement, and we only provide an intuitive explanation for the moment. Since a program P is represented as a graphing $[\![P]\!]$, which is in some way a dynamical system, the computation $P(a)$ on a given input a is represented as a sequence of values $[\![a]\!], [\![P]\!]([\![a]\!]), [\![P]\!]^2([\![a]\!]), \dots$. Quantitative soundness states that not only $[\![P]\!]$ computes exactly as P, but it does so in the same number of steps, i.e. if $P(a)$ terminates on a value b in time k, then $[\![P]\!]^k([\![a]\!]) = [\![b]\!]$.

The second ingredient is the dual nature of graphings, both continuous and discrete objects. Indeed, a graphing *representative* is a graph-like structure whose edges are represented as continuous maps, i.e. a finite representation of a (partial) continuous dynamical system. Given a graphing, we define its *kth cell decomposition*, which separates the input space into cells such that two inputs in the same cell are indistinguishable in k steps, i.e. the graphing's computational traces on both inputs are equal. We can then use both the finiteness of the graphing representatives and the *topological entropy* of the associated dynamical system to provide upper bounds on the size of a further refinement of this geometric object, namely the *k-th entropic co-tree* of a graphing – a kind of final approximation of the graphing by a computational tree[2]

As we deal with algebraic models of computation, this implies a bound on the representation of the kth cell decomposition as a semi-algebraic variety. In other words, the kth cell decomposition is defined by polynomial in·equalities and we provide bounds on the number and degree of the involved polynomials. The corresponding statement is the main technical result of this paper (Lemma 5).

This lemma can then be used to obtain lower bounds results. Using the Milnor-Thom theorem to bound the number of connected components of the kth cell decomposition, we then recover the lower bounds of Steele and Yao on algebraic decision trees, and the refined result of Ben-Or providing lower bounds for algebraic computation trees. A different argument based on invariant polynomials provides a proof of Cucker's result that $\mathrm{NC}_{\mathbf{R}} \neq \textsc{Ptime}_{\mathbf{R}}$ by showing that a given polynomial that belongs to $\textsc{Ptime}_{\mathbf{R}}$ cannot be computed within $\mathrm{NC}_{\mathbf{R}}$. Lastly, following Mulmuley's geometric representation of the `maxflow` problem, we are able to strenghten his celebrated result to obtain lower bounds on the size (depth) of a PRAM over the integers computing this problem. This proves the following theorem, which has Theorem 1 as a corollary.

Theorem 2. *Let c be a positive integer, M a* PRAM *over* $\mathbf{Z}$ *with $2^{O((\log N)^c)}$ processors, with N the length of the inputs. Then M does not decide* `maxflow` *in $O((\log N)^c)$ steps.*

II. Programs as Dynamical systems

A. Abstract models of computation and graphings

We consider computations as dynamical processes, hence model them as a dynamical system with two main components: a space $\mathbf{X}$ that abstracts the notion of configuration space and a monoid acting on this space that represents the different operations allowed in the model of computation. Although the notion of *space* considered can vary (one could consider e.g. topological spaces, measure spaces, topological vector spaces), we restrict ourselves to topological spaces in this work.

Definition 1. An *abstract model of computation* (AMC) is a monoid action $\alpha : M \curvearrowright \mathbf{X}$, i.e. a monoid morphism from M to the group of endomorphisms of $\mathbf{X}$. The monoid M is often given by a set G of generators and a set of relations R. We denote such an AMC as $\alpha : \langle G, \mathrm{R}\rangle \curvearrowright \mathbf{X}$.

Programs in an AMC $\alpha : \langle G, \mathrm{R}\rangle \curvearrowright \mathbf{X}$ is then defined as *graphings*, i.e. graphs whose vertices are subspaces of the space $\mathbf{X}$ (representing sets of configurations on which the program act in the same way) and edges are labelled by elements of $\mathrm{M}\langle G, \mathrm{R}\rangle$, together with a global control

[2]Intuitively, the k-th entropic co-tree mimicks the behaviour of the graphing for k steps of computation.

state. More precisely, we use here the notion of *topological graphings*[3] [14].

Definition 2. An α-graphing representative G w.r.t. a monoid action $\alpha : M \curvearrowright \mathbf{X}$ is defined as a set of *edges* E^G together with a map that assigns to each element $e \in E^G$ a pair (S_e^G, m_e^G) of a subspace S_e^G of $\mathbf{X}$ – the *source* of e – and an element $m_e^G \in M$ – the *realiser* of e.

While graphing representatives are convenient to manipulate, they do provide too much information about the programs. Indeed, if one is to study programs as dynamical systems, the focus should be on the *dynamics*, i.e. on how the object acts on the underlying space. The following notion of *refinement* captures this idea that the same dynamics may have different graph-like representations.

Definition 3 (Refinement)**.** An α-graphing representative F is a refinement of an α-graphing representative G, noted $F \leqslant G$, if there exists a partition $(E_e^F)_{e \in E^G}$ of E^F such that $\forall e \in E^G$:

$$\left(\cup_{f \in E_e^F} S_f^F\right) \triangle S_e^G = \emptyset; \quad \forall f \neq f' \in E_e^F,\ S_f^F \triangle S_{f'}^F = \emptyset;$$
$$\forall f \in E_e^F,\ m_f^F = m_e^G.$$

This induces an equivalence relation defined as

$$F \sim_{\mathrm{ref}} G \Leftrightarrow \exists H,\ H \leqslant F \wedge H \leqslant G.$$

The notion of *graphing* is therefore obtained by considering the quotient of the set of graphing representatives w.r.t. $\sim_{\mathrm{ref}}$. Intuitively, this corresponds to identifying graphings whose *actions on the underlying space are equal.*

Definition 4. An *α-graphing* is an equivalence class of α-graphing representatives w.r.t. the equivalence relation $\sim_{\mathrm{ref}}$.

We can now define the notion of abstract program. These are defined as graphings

Definition 5. Given an AMC $\alpha : M \curvearrowright \mathbf{X}$, an *$\alpha$-program* A is a $\bar{\alpha}$-graphing G^A w.r.t. the monoid action $\bar{\alpha} = \alpha \times \mathfrak{S}_k \curvearrowright \mathbf{X} \times \mathbf{S^A}$, where $\mathbf{S^A}$ is a finite set of *control states* of cardinality k and $\mathfrak{S}_k$ is the group of permutations of k elements.

Now, as a sanity check, we will show how the notion of graphing do capture the expected dynamics. For this, we restrict to *deterministic graphings*, and show the notion relates to the usual notion of dynamical system.

Definition 6. An α-graphing representative G is deterministic if for all $x \in \mathbf{X}$ there is at most one $e \in E^G$ such that $x \in S_e^G$. An α-graphing is *deterministic* if its representatives are deterministic. An abstract program is *deterministic* if its underlying graphing is deterministic.

Lemma 1. *There is a one-to-one correspondence between the set of deterministic graphings w.r.t. the action $M \curvearrowright \mathbf{X}$ and the set of partial dynamical systems $f : \mathbf{X} \hookrightarrow \mathbf{X}$ whose graph is contained in the preorder*[4] $\{(x, y) \mid \exists m \in M, \alpha(m)(x) = y\}$.

Lastly, we define some restrictions of α-programs that will be important later. First, we will restrict the possible subspaces considered as sources of the edges, as unrestricted α-programs could compute even undecidable problems by, e.g. encoding it into a subspace used as the source of an edge. Given an integer $k \in \omega$, we define the following subspaces of $\mathbf{R}^\omega$, for $\star \in \{>, \geqslant, =, \neq, \leqslant, <\}$:

$$\mathbf{R}^\omega_{k \star 0} = \{(x_1, \dots, x_k, \dots) \in \mathbf{R}^\omega \mid x_k \star 0\}.$$

Definition 7 (Computational graphings)**.** Let $\alpha : \langle G, \mathrm{R}\rangle \curvearrowright \mathbf{X}$ be an AMC. A *computational α-graphing* is an α-graphing T with distinguished states $\top, \bot \in \mathbf{S^A}$ which admits a finite representative such that each edge e has its source equal to one among $\mathbf{R}^\omega$, $\mathbf{R}^\omega_{k \geqslant 0}$, $\mathbf{R}^\omega_{k \leqslant 0}$, $\mathbf{R}^\omega_{k > 0}$, $\mathbf{R}^\omega_{k < 0}$, $\mathbf{R}^\omega_{k = 0}$, and $\mathbf{R}^\omega_{k \neq 0}$.

Definition 8 (treeings)**.** Let $\alpha : \langle G, \mathrm{R}\rangle \curvearrowright \mathbf{X}$ be an AMC. An *α-treeing* is an acyclic and finite α-graphing, i.e. an α-graphing F for which there exists a finite α-graphing representative T whose set of control states $\mathbf{S^T} = \{0, \dots, s\}$ can be endowed with an order $<$ such that every edge of T is state-increasing, i.e. for each edge e of source S_e, for all $x \in S_e$,

$$\pi_{\mathbf{S^T}}(\alpha(m_e)(x)) > \pi_{\mathbf{S^T}}(x),$$

where $\pi_{\mathbf{S^T}}$ denotes the projection onto the control states space.

A *computational α-treeing* is an α-treeing T which is a computational α-graphing with the distinguished states $\top$, $\bot$ being incomparable maximal elements of the state space.

B. Quantitative Soundness

As mentioned in the introduction, we will use the property of *quantitative soundness* of the dynamic semantics just introduced. This result is essential, as it connects the time complexity of programs in the model considered (e.g. PRAMs, algebraic computation trees) with the length of the orbits of the considered dynamical system. We here only state quantitative soundness for *computational graphings*, i.e. graphings that have distinguished states $\top$ and $\bot$ representing acceptance and rejection respectively. In other words, we consider graphings which compute *decision problems.*

Quantitative soundness is expressed with respect to a translation of machines as graphings, together with a translation of inputs as points of the configuration space. In the following section, these operations are defined for each model of computation considered in this paper. In all these cases, the representation of inputs is straightforward.

Definition 9. Let α be an abstract model of computation, and $\mathbb{M}$ a model of computation. A *translation* of $\mathbb{M}$ w.r.t. α is a pair of maps $[\![\cdot]\!]$ which associate to each machine M in

[3]While "measured" graphings were already considered [14], the definition adapts in a straightforward manner to allow for other notions such as graphings over topological vector spaces – which would be objects akin to the notion of quiver used in representation theory.

[4]When α is a group action acting by measure-preserving transformations, this is a *Borel equivalence relation* $\mathcal{R}$, and the condition stated here boils down to requiring that f belongs to the *full group* of α.

$\mathbb{M}$ computing a decision problem a computational α-graphing $[\![M]\!]$ and to each input ι a point $[\![\iota]\!]$ in $\mathbf{X} \times \mathbf{S}$.

Definition 10. Let α be an abstract model of computation, $\mathbb{M}$ a model of computation. The AMC α is *quantitatively sound* for $\mathbb{M}$ w.r.t. a translation $[\![\cdot]\!]$ if for all machine M computing a decision problem and input ι, M accepts ι (resp. rejects ι) in k steps if and only if $[\![M]\!]^k([\![\iota]\!]) = \top$ (resp. $[\![M]\!]^k([\![\iota]\!]) = \bot$).

C. The algebraic AMCs

We now define the actions α_{full} and $\alpha_{\mathbf{R}\text{full}}$. Those will capture all algebraic models of computation considered in this paper, and the main theorem will be stated for this monoid action.

As we intend to consider PRAMs at some point, we consider from the beginning the memory of our machines to be separated in two infinite blocks $\mathbf{Z}^\omega$, intended to represent both *shared* and a *private* memory cells[5].

Definition 11. The underlying space of α_{full} is $\mathbf{X} = \mathbf{Z}^{\mathbf{Z}} \cong \mathbf{Z}^\omega \times \mathbf{Z}^\omega$. The set of generators is defined by their action on the underlying space, writing $k//n$ the floor $\lfloor k/n \rfloor$ of k/n with the convention that $k//n = 0$ when $n = 0$:

- $\texttt{const}_i(c)$ initialises the register i with the constant $c \in \mathbf{Z}$: $\alpha_{\text{full}}(\texttt{const}_i(c))(\vec{x}) = (\vec{x}\{x_i := c\})$;
- $\star_i(j,k)$ ($\star \in \{+,-,\times,//\}$) performs the algebraic operation $\star$ on the values in registers j and k and store the result in register i: $\alpha_{\text{full}}(\star_i(j,k))(\vec{x}) = (\vec{x}\{x_i := x_j \star x_k\})$;
- $\star_i^c(j)$ ($\star \in \{+,-,\times,//\}$) performs the algebraic operation $\star$ on the value in register j and the constant $c \in \mathbf{Z}$ and store the result in register i: $\alpha_{\text{full}}(\star_i^c(j))(\vec{x}) = (\vec{x}\{x_i := c \star x_j\})$;
- $\texttt{copy}(i,j)$ copies the value stored in register j in register i: $\alpha_{\text{full}}(\texttt{copy}(i,j))(\vec{x}) = (\vec{x}\{x_i := x_j\})$;
- $\texttt{copy}(\sharp i, j)$ copies the value stored in register j in the register whose index is the value stored in register i: $\alpha_{\text{full}}(\texttt{copy}(\sharp i,j))(\vec{x}) = (\vec{x}\{x_{x_i} := x_j\})$;
- $\texttt{copy}(i, \sharp j)$ copies the value stored in the register whose index is the value stored in register j in register i: $\alpha_{\text{full}}(\texttt{copy}(i,\sharp j))(\vec{x}) = (\vec{x}\{x_i := x_{x_j}\})$;
- $\sqrt[n]{i}(j)$ computes the floor of the n-th root of the value stored in register j and store the result in register i: $\alpha_{\text{full}}(\sqrt[n]{i}(j))(\vec{x}) = (\vec{x}\{x_i := \sqrt[n]{x_j}\})$.

We also define the real-valued equivalent, which will be essential for the proof of lower bounds. The corresponding AMC $\alpha_{\mathbf{R}\text{full}}$ is defined in the same way than the integer-valued one, but with underlying space $\mathbf{X} = \mathbf{R}^{\mathbf{Z}}$ and with instructions adapted accordingly:

- the division and n-th root operations are the usual operations on the reals;
- the three copy operators are only effective on integers.

Definition 12. The underlying space of $\alpha_{\mathbf{R}\text{full}}$ is $\mathbf{X} = \mathbf{R}^{\mathbf{Z}} \cong \mathbf{R}^\omega \times \mathbf{R}^\omega$. The set of generators is defined by their action on the underlying space, with the convention that $k/n = 0$ when $n = 0$:

- $\texttt{const}_i(c)$ initialises the register i with the constant $c \in \mathbf{R}$: $\alpha_{\mathbf{R}\text{full}}(\texttt{const}_i(c))(\vec{x}) = (\vec{x}\{x_i := c\})$;
- $\star_i(j,k)$ ($\star \in \{+,-,\times,/\}$) performs the algebraic operation $\star$ on the values in registers j and k and store the result in register i: $\alpha_{\mathbf{R}\text{full}}(\star_i(j,k))(\vec{x}) = (\vec{x}\{x_i := x_j \star x_k\})$;
- $\star_i^c(j)$ ($\star \in \{+,-,\times,/\}$) performs the algebraic operation $\star$ on the value in register j and the constant $c \in \mathbf{R}$ and store the result in register i: $\alpha_{\mathbf{R}\text{full}}(\star_i^c(j))(\vec{x}) = (\vec{x}\{x_i := c \star x_j\})$;
- $\texttt{copy}(i,j)$ copies the value stored in register j in register i: $\alpha_{\mathbf{R}\text{full}}(\texttt{copy}(i,j))(\vec{x}) = (\vec{x}\{x_i := x_j\})$;
- $\texttt{copy}(\sharp i, j)$ copies the value stored in register j in the register whose index is the floor of the value stored in register i: $\alpha_{\mathbf{R}\text{full}}(\texttt{copy}(\sharp i,j))(\vec{x}) = (\vec{x}\{x_{\lfloor x_i \rfloor} := x_j\})$;
- $\texttt{copy}(i, \sharp j)$ copies the value stored in the register whose index is the floor of the value stored in register j in register i: $\alpha_{\mathbf{R}\text{full}}(\texttt{copy}(i,\sharp j))(\vec{x}) = (\vec{x}\{x_i := x_{\lfloor x_j \rfloor}\})$;
- $\sqrt[n]{i}(j)$ computes the n-th real root of the value stored in register j and store the result in register i: $\alpha_{\mathbf{R}\text{full}}(\sqrt[n]{i}(j))(\vec{x}) = (\vec{x}\{x_i := \sqrt[n]{x_j}\})$.

III. Algebraic models of computations as AMCs

A. Algebraic computation trees

The first model considered here will be that of *algebraic computation tree* as defined by Ben-Or [2]. Let us note this model refines the *algebraic decision trees* model of Steele and Yao [1], a model of computation consisting in binary trees for which each branching performs a test w.r.t. a polynomial and each leaf is labelled YES or NO. Algebraic computation trees only allow tests w.r.t. 0, while additional vertices corresponding to algebraic operations can be used to construct polynomials.

Definition 13 (algebraic computation trees, [2])**.** An *algebraic computation tree* on $\mathbf{R}^n$ is a binary tree T with a function assigning:

- to any vertex v with only one child (simple vertex) an operational instruction of the form $f_v = f_{v_i} \star f_{v_j}$, $f_v = c \star f_{v_i}$, or $f_v = \sqrt{f_{v_i}}$, where $\star \in \{+,-,\times,/\}$, v_i, v_j are ancestors of v and $c \in \mathbf{R}$ is a constant;
- to any vertex v with two children a test instruction of the form $f_{v_i} \star 0$, where $\star \in \{>, =, \geqslant\}$, and v_i is an ancestor of v or $f_{v_i} \in \{x_1, \ldots, x_n\}$;
- to any leaf an output YES or NO.

Let $W \subseteq \mathbf{R}^n$ be any set and T be an algebraic computation tree. We say that T computes the membership problem for W if for all $x \in \mathbf{R}^n$, the traversal of T following x ends on a leaf labelled YES if and only if $x \in W$.

As algebraic computation trees are *trees*, they will be represented by treeings, i.e. $\alpha_{\mathbf{R}\text{full}}$-programs whose set of control states can be ordered so that any edge in the graphing is strictly increasing on its control states component.

[5]Obviously, this could be done without any explicit separation of the underlying space, but this will ease the constructions of the next section.

Definition 14. Let T be a computational $\alpha_{\mathbf{R}\text{full}}$-treeing. The set of inputs $\mathrm{In}(T)$ (resp. outputs $\mathrm{Out}(T)$) is the set of integers k (resp. i) such that there exists an edge e in T satisfying that:

- either e is realised by one of $+_i(j,k)$, $+_i(k,j)$, $-_i(j,k)$, $-_i(k,j)$, $\times_i(j,k)$, $\times_i(k,j)$, $/_i(j,k)$, $/_i(k,j)$ $+_i^c(k)$, $-_i^c(k)$, $\times_i^c(k)$, $/_i^c(k)$, $\sqrt[n]{i}(k)$;
- or the source of e is one among $\mathbf{R}^\omega_{k\geqslant 0}$, $\mathbf{R}^\omega_{k\leqslant 0}$, $\mathbf{R}^\omega_{k>0}$, $\mathbf{R}^\omega_{k<0}$, $\mathbf{R}^\omega_{k=0}$, and $\mathbf{R}^\omega_{k\neq 0}$.

The *effective input space* $\mathbf{In}^{\mathrm{E}}(T)$ of an α_{act}-treeing T is defined as the set of indices $k \in \omega$ belonging to $\mathrm{In}(T)$ but not to $\mathrm{Out}(T)$. The *implicit input space* $\mathbf{In}^{\mathrm{I}}(T)$ of an α_{act}-treeing T is defined as the set of indices $k \in \omega$ such that $k \notin \mathrm{Out}(T)$.

Definition 15. Let T be an $\alpha_{\mathbf{R}\text{full}}$-treeing, and assume that $1, 2, \ldots, n \in \mathbf{In}^{\mathrm{I}}(T)$. We say that T computes the membership problem for $W \subseteq \mathbf{R}^n$ in k steps if k successive iterations of T restricted to $\{(x_i)_{i\in\omega} \in \mathbf{R}^\omega \mid \forall 1 \leqslant i \leqslant n, x_i = y_i\} \times \{0\}$ reach state $\top$ if and only if $(y_1, y_2, \ldots, y_n) \in W$.

Remark 1. Let $\vec{x} = (x_1, x_2, \ldots, x_n)$ be an element of $\mathbf{R}^n$ and consider two elements a, b in the subspace $\{(y_1, \ldots, y_n, \ldots) \in \mathbf{R}^\omega \mid \forall 1 \geqslant i \geqslant n, y_i = x_i\} \times \{0\}$. One easily checks that $\pi_{\mathbf{S}}(T^k(a)) = \top$ if and only if $\pi_{\mathbf{S}}(T^k(b)) = \top$, where $\pi_{\mathbf{S}}$ is the projection onto the state space and $T^k(a)$ represents the k-th iteration of T on a. It is therefore possible to consider only a standard representative $[\![\vec{x}]\!]$ of $\vec{x} \in \mathbf{R}^n$, for instance $(x_1, \ldots, x_n, 0, 0, \ldots) \in \mathbf{R}^\omega$, to decide whether $\vec{x}$ is accepted by T.

Definition 16. Let T be an algebraic computation tree on $\mathbf{R}^n$, and T° be the associated directed acyclic graph, built from T by merging all the leaves tagged YES in one leaf $\top$ and all the leaves tagged NO in one leaf $\bot$. Suppose the internal vertices are numbered $\{n+1, \ldots, n+\ell\}$; the numbers $1, \ldots, n$ being reserved for the input.

We define $[\![T]\!]$ as the α_{act}-graphing with control states $\{n+1, \ldots, n+\ell, \top, \bot\}$ and where each internal vertex i of T° defines either:

- a single edge of source $\mathbf{R}^\omega$ realized by:
  - $(\star_i(j,k), i \mapsto t)$ ($\star \in \{+,-,\times\}$) if i is associated to $f_{v_i} = f_{v_j} \star f_{v_k}$ and t is the child of i;
  - $(\star_i^c(j), i \mapsto t)$ ($\star \in \{+,-,\times\}$) if i is associated to $f_{v_i} = c \star f_{v_k}$ and t is the child of i;
- a single edge of source $\mathbf{R}^\omega_{k\neq 0}$ realized by:
  - $(/_i(j,k), i \mapsto t)$ if i is associated to $f_{v_i} = f_{v_j}/f_{v_k}$ and t is the child of i;
  - $(/_i^c(k), i \mapsto t)$ if i is associated to $f_{v_i} = c/f_{v_k}$ and t is the child of i;
- a single edge of source $\mathbf{R}^\omega_{k\geqslant 0} \times \{i\}$ realized by $(\sqrt[2]{i}(k), i \mapsto t)$ if i is associated to $f_{v_i} = \sqrt{f_{v_k}}$ and t is the child of i;
- two edges if i is associated to $f_{v_i} \star 0$ (where $\star$ ranges in $>$, $\geqslant$) and its two sons are j and k. Those are of respective sources $\mathbf{R}^\omega_{k\star 0} \times \{i\}$ and $\mathbf{R}^\omega_{k\bar{\star}0} \times \{i\}$ (where $\bar{\star} =' \leqslant'$ if $\star =' >'$, $\bar{\star} =' <'$ if $\star =' \geqslant'$, and $\bar{\star} =' \neq'$ if $\star =' =' .$), respectively realized by $(\mathrm{Id}, i \mapsto j)$ and $(\mathrm{Id}, i \mapsto k)$

Proposition 1. *Any algebraic computation tree T of depth k is faithfully and quantitatively interpreted as the $\alpha_{\mathbf{R}\text{full}}$-program $[\![T]\!]$. I.e. T computes the membership problem for $W \subseteq \mathbf{R}^n$ if and only if $[\![T]\!]$ computes the membership problem for W in k steps – that is $\pi_{\mathbf{S}}([\![T]\!]^k([\![\vec{x}]\!])) = \top$.*

As a corollary of this proposition, we get quantitative soundness.

Theorem 3. *The representation of* ACTs *as $\alpha_{\mathbf{R}\text{full}}$-programs is quantitatively sound.*

B. Algebraic circuits

As we will recover Cucker's proof that $\mathrm{NC}_{\mathbf{R}} \neq \mathrm{PTIME}_{\mathbf{R}}$, we introduce the model of *algebraic circuits* and their representation as $\alpha_{\mathbf{R}\text{full}}$-programs.

Definition 17. An algebraic circuit over the reals with inputs in $\mathbf{R}^n$ is a finite directed graph whose vertices have labels in $\mathbf{N} \times \mathbf{N}$, that satisfies the following conditions:

- There are exactly n vertices $v_{0,1}, v_{0,2}, \ldots, v_{0,n}$ with first index 0, and they have no incoming edges;
- all the other vertices $v_{i,j}$ are of one of the following types:
  1) arithmetic vertex: they have an associated arithmetic operation $\{+,-,\times,/\}$ and there exist natural numbers l, k, r, m with $l, k < i$ such that their two incoming edges are of sources $v_{l,r}$ and $v_{k,m}$;
  2) constant vertex: they have an associated real number y and no incoming edges;
  3) sign vertex: they have a unique incoming edge of source $v_{k,m}$ with $k < i$.

We call *depth* of the circuit the largest m such that there exist a vertex $v_{m,r}$, and *size* of the circuit the total number of vertices. A circuit of depth d is *decisional* if there is only one vertex $v_{d,r}$ at level d, and it is a sign vertex; we call $v_{d,r}$ the *end vertex* of the decisional circuit.

To each vertex v one inductively associates a function f_v of the input variables in the usual way, where a sign node with input x returns 1 if $x > 0$ and 0 otherwise. The accepted set of a decisional circuit C is defined as the set $S \subseteq \mathbf{R}^n$ of points whose image by the associated function is 1, i.e. $S = f_v^{-1}(\{1\})$ where v is the end vertex of C.

We represent algebraic circuit as computational $\alpha_{\mathbf{R}\text{full}}$-treeings as follows. The first index in the pairs $(i,j) \in \mathbf{N} \times \mathbf{N}$ are represented as states, the second index is represented as an index in the infinite product $\mathbf{R}^\omega$, and vertices are represented as edges.

Definition 18. Let C be an algebraic circuit, defined as a finite directed graph (V, E, s, t, ℓ) where $V \subset \mathbf{N} \times \mathbf{N}$, and $\ell : V \to \{\mathrm{init}, +, -, \times, /, \mathrm{sgn}\} \cup \{\mathrm{const}_c \mid c \in \mathbf{R}\}$ is a vertex labelling map. We suppose without loss of generality that for each $j \in \mathbf{N}$, there is at most one $i \in \mathbf{N}$ such that $(i,j) \in V$. We define N as $\max\{j \in \mathbf{N} \mid \exists i \in \mathbf{N}, (i,j) \in V\}$.

We define the $\alpha_{\mathbf{R}\text{full}}$-program $[\![C]\!]$ by choosing as set of control states $\{i \in \mathbf{N} \mid \exists j \in \mathbf{N}, (i,j) \in V\}$ and the collection

of edges $\{e_{(i,j)} \mid i \in \mathbf{N}^*, j \in \mathbf{N}, (i,j) \in V\} \cup \{e^+_{(i,j)} \mid i \in \mathbf{N}^*, j \in \mathbf{N}, (i,j) \in V, \ell(v) = \mathrm{sgn}\}$ realised as follows:

- if $\ell(v) = \mathrm{const}_c$, the edge $e_{(i,j)}$ is realised as $(+_j^{n_v}(c), 0 \mapsto i)$ of source $\mathbf{R}^\omega_{n_v=0} \times \{0\}$;
- if $\ell(v) = \star$ ($\star \in \{+,-,\times\}$) of incoming edges (k,l) and (k',l'), the edge $e_{(i,j)}$ is of source $\mathbf{R}^\omega \times \{\max(k,k')\}$ and realised by $(\star_j(l,l'), \max(k,k') \mapsto i)$;
- if $\ell(v) = /$ of incoming edges (k,l) and (k',l'), the edge $e_{(i,j)}$ is of source $\mathbf{R}^\omega_{l' \neq 0} \times \{\max(k,k')\}$ and realised by $(/_j(l,l'), \max(k,k') \mapsto i)$;
- if $\ell(v) = \mathrm{sgn}$ of incoming edge (k,l), the edges $e_{(i,j)}$ and $e^+_{(i,j)}$ are of respective sources $\mathbf{R}^\omega_{n_v=0 \wedge x_l \leqslant 0} \times \{k\}$ and $\mathbf{R}^\omega_{n_v=0 \wedge x_l > 0} \times \{k\}$ realised by $(\mathrm{Id}, k \mapsto i)$ and $(+_j(n_v, 1), k \mapsto i)$ respectively.

As each step of computation in the algebraic circuit is translated as going through a single edge in the corresponding $\alpha_{\mathbf{R}\mathrm{full}}$-program, the following result is straightforward.

Theorem 4. *The representation of* ALGCIRC *as* $\alpha_{\mathbf{R}\mathrm{full}}$*-programs is quantitatively sound.*

C. Algebraic RAMs

In this paper, we will consider algebraic parallel random access machines, that act not on strings of bits, but on integers. In order to define those properly, we first define the notion of (sequential) random access machine (RAM) before considering their parallelisation.

A RAM *command* is a pair (ℓ, I) of a *line* $\ell \in \mathbf{N}^\star$ and an *instruction* $\mathtt{I}$ among the following, where $i, j \in \mathbf{N}$, $\star \in \{+,-,\times,/\}$, $c \in \mathbf{Z}$ is a constant and $\ell, \ell' \in \mathbf{N}^\star$ are lines:

$$\mathtt{skip};\quad \mathtt{X_i} \coloneqq \mathtt{c};\quad \mathtt{X_i} \coloneqq \mathtt{X_j} \star \mathtt{X_k};\quad \mathtt{X_i} \coloneqq \mathtt{X_j};$$
$$\mathtt{X_i} \coloneqq \sharp\mathtt{X_j};\quad \sharp\mathtt{X_i} \coloneqq \mathtt{X_j};\quad \mathtt{if\ X_i = 0\ goto}\ \ell\ \mathtt{else}\ \ell'.$$

A RAM *machine* M is then a finite set of commands such that the set of lines is $\{1, 2, \ldots, |M|\}$, with $|M|$ the *length* of M. We will denote the commands in M by $(i, \mathrm{Inst}_M(i))$, i.e. $\mathrm{Inst}_M(i)$ denotes the line i instruction.

Following Mulmuley [4], we will here make the assumption that the input in the RAM (and in the PRAM model defined in the next section) is split into *numeric* and *nonumeric* data – e.g. in the `maxflow` problem the nonnumeric data would specify the network and the numeric data would specify the edge-capacities – and that indirect references use pointers depending only on nonnumeric data[6]. We refer the reader to Mulmuley's article for more details.

Machines in the RAM model can be represented as graphings w.r.t. the action α_{full}. Intuitively the encoding works as follows. The notion of *control state* allows to represent the notion of *line* in the program. Then, the action just defined allows for the representation of all commands but the conditionals. The conditionals are represented as follows: depending on the value of X_i one wants to jumps either to the line ℓ or to the line ℓ'; this is easily modelled by two different edges of respective sources $\mathbb{H}(i) = \{\vec{x} \mid x_i = 0\}$ and $\mathbb{H}(i)^{\mathrm{c}} = \{\vec{x} \mid x_i \neq 0\}$.

Definition 19. Let M be a RAM machine. We define the translation $[\![M]\!]$ as the α_{ram}-program with set of control states $\{0, 1, \ldots, L, L{+}1\}$ where each line ℓ defines (in the following, $\star \in \{+,-,\times\}$ and we write $\ell{+}{+}$ the map $\ell \mapsto \ell + 1$):

- a single edge e of source $\mathbf{X} \times \{\ell\}$ and realised by:
  - $(\mathrm{Id}, \ell{+}{+})$ if $\mathrm{Inst}_M(\ell) = \mathtt{skip}$;
  - $(\mathtt{const}_i(c), \ell{+}{+})$ if $\mathrm{Inst}_M(\ell) = \mathtt{X_i} \coloneqq \mathtt{c}$;
  - $(\star_i(j,k), \ell{+}{+})$ if $\mathrm{Inst}_M(\ell) = \mathtt{X_i} \coloneqq \mathtt{X_j} \star \mathtt{X_k}$;
  - $(\mathtt{copy}(i,j), \ell{+}{+})$ if $\mathrm{Inst}_M(\ell) = \mathtt{X_i} \coloneqq \mathtt{X_j}$;
  - $(\mathtt{copy}(i, \sharp j), \ell{+}{+})$ if $\mathrm{Inst}_M(\ell) = \mathtt{X_i} \coloneqq \sharp\mathtt{X_j}$;
  - $(\mathtt{copy}(\sharp i, j), \ell{+}{+})$ if $\mathrm{Inst}_M(\ell) = \sharp\mathtt{X_i} \coloneqq \mathtt{X_j}$.
- an edge e of source $\mathbb{H}(k) \times \{\ell\}$ realised by $(//_i(j,k), \ell{+}{+})$ if $\mathrm{Inst}_M(\ell)$ is $\mathtt{X_i} \coloneqq \mathtt{X_j}/\mathtt{X_k}$;
- a pair of edges e, e^{c} of respective sources $\mathbb{H}(i) \times \{\ell\}$ and $\mathbb{H}(i)^{\mathrm{c}} \times \{\ell\}$ and realised by respectively $(\mathrm{Id}, \ell \mapsto \ell^0)$ and $(\mathrm{Id}, \ell \mapsto \ell^1)$, if the line is a conditional $\mathtt{if\ X_i = 0\ goto}\ \ell^0\ \mathtt{else}\ \ell^1$.

The translation $[\![\iota]\!]$ of an input $\iota \in \mathbf{Z}^d$ is the point $(\bar{\iota}, 0)$ where $\bar{\iota}$ is the sequence $(\iota_1, \iota_2, \ldots, \iota_k, 0, 0, \ldots)$.

Now, the main result for the representation of RAMs is the following. The proof is straightforward, as each instruction corresponds to exactly one edge, except for the conditional case (but given a configuration, it lies in the source of at most one of the two edges translating the conditional).

Theorem 5. *The representation of* RAMs *as* α_{full}*-program is quantitatively sound w.r.t. the translation just defined.*

D. The Crew operation and PRAMs

Based on the notion of RAM, we are now able to consider their parallelisation, namely PRAMs. A PRAM M is given as a finite sequence of RAM machines $M_1, \ldots, M_p$, where p is the number of *processors* of M. Each processor M_i has access to its own, private, set of registers $(\mathtt{X}^i_k)_{k \geqslant 0}$ and a *shared memory* represented as a set of registers $(\mathtt{X}^0_k)_{k \geqslant 0}$.

One has to deal with conflicts when several processors try to access the shared memory simultaneously. We here chose to work with the *Concurrent Read, Exclusive Write* (CREW) discipline: at a given step at which several processors try to write in the shared memory, only the processor with the smallest index will be allowed to do so. In order to model such parallel computations, we abstract the CREW at the level of monoids. For this, we suppose that we have two monoid actions $\mathrm{M}\langle G, \mathrm{R}\rangle \curvearrowright \mathbf{X} \times \mathbf{Y}$ and $\mathrm{M}\langle H, \mathrm{Q}\rangle \curvearrowright \mathbf{X} \times \mathbf{Z}$, where $\mathbf{X}$ represents the shared memory. We then consider the subset $\# \subset G \times H$ of pairs of generators that potentially conflict with one another – the conflict relation.

[6] Quoting Mulmuley: "We assume that the pointer involved in an indirect reference is not some numeric argument in the input or a quantity that depends on it. For example, in the max- flow problem the algorithm should not use an edge-capacity as a pointer—which is a reasonable condition. To enforce this restriction, one initially puts an invalid-pointer tag on every numeric argument in the input. During the execution of an arithmetic instruction, the same tag is also propagated to the result if any operand has that tag. Trying to use a memory value with invalid-pointer tag results in error." [4, Page 1468].

Definition 20 (Conflicted sum)**.** Let $\mathrm{M}\langle G, \mathrm{R}\rangle$, $\mathrm{M}\langle G', \mathrm{R}'\rangle$ be two monoids and $\# \subseteq G \times G'$. The *conflicted sum of* $\mathrm{M}\langle G, \mathrm{R}\rangle$ *and* $\mathrm{M}\langle G', \mathrm{R}'\rangle$ *over* $\#$, noted $\mathrm{M}\langle G, \mathrm{R}\rangle *_{\#} \mathrm{M}\langle G', \mathrm{R}'\rangle$, is defined as the monoid with generators $(\{1\}\times G)\cup(\{2\}\times G')$ and relations

$$(\{1\} \times R) \cup (\{2\} \times R') \cup \{(\mathbf{1}, e)\} \cup \{(\mathbf{1}, e')\}$$
$$\cup \{\big((1, g)(2, g'), (2, g')(1, g)\big) \mid (g, g') \notin \#\}$$

where $\mathbf{1}$, e, e' are the units of $\mathrm{M}\langle G, \mathrm{R}\rangle *_{\#} \mathrm{M}\langle G', \mathrm{R}'\rangle$, $\mathrm{M}\langle G, \mathrm{R}\rangle$ and $\mathrm{M}\langle G', \mathrm{R}'\rangle$ respectively.

In the particular case where $\# = (G \times H') \cup (H \times G')$, with H, H' respectively subsets of G and G', we will write the sum $\mathrm{M}\langle G, \mathrm{R}\rangle \ {}_{H}*_{H'} \mathrm{M}\langle G', \mathrm{R}'\rangle$.

Remark 2. When the conflict relation $\#$ is empty, this defines the usual direct product of monoids. This corresponds to the case in which no conflicts can arise w.r.t. the shared memory. In other words, the direct product of monoids corresponds to the parallelisation of processes *without shared memory*.

Dually, when the conflict relation is full ($\# = G \times G'$), this defines the free product of the monoids.

Definition 21. Let $\alpha : M \curvearrowright \mathbf{X} \times \mathbf{Y}$ be a monoid action. We say that an element $m \in M$ is *central relatively to* α (or just *central*) if the action of m commutes with the first projection $\pi_X : \mathbf{X} \times \mathbf{Y} \to \mathbf{X}$, i.e.[7] $\alpha(m); \pi_X = \alpha(m)$; in other words m acts as the identity on $\mathbf{X}$.

Intuitively, central elements are those that will not affect the shared memory. As such, only *non-central elements* require care when putting processes in parallel.

Definition 22. Let $\mathrm{M}\langle G, \mathrm{R}\rangle \curvearrowright \mathbf{X}\times\mathbf{Y}$ be an AMC. We note Z_α the set of central elements and $\bar{Z}_\alpha(G) = \{m \in G \mid n \notin Z_\alpha\}$.

Definition 23 (The CREW of AMCs)**.** Let $\alpha : \mathrm{M}\langle G, \mathrm{R}\rangle \curvearrowright \mathbf{X}\times\mathbf{Y}$ and $\beta : \mathrm{M}\langle H, \mathrm{Q}\rangle \curvearrowright \mathbf{X} \times \mathbf{Z}$ be AMCs. We define the AMC $\textsc{crew}(\alpha, \beta) : \mathrm{M}\langle G, \mathrm{R}\rangle \ {}_{\bar{Z}_\alpha(G)}*_{\bar{Z}_\beta(G')} \mathrm{M}\langle G', \mathrm{R}'\rangle \curvearrowright \mathbf{X}\times\mathbf{Y}\times\mathbf{Z}$ by letting $\textsc{crew}(\alpha, \beta)(m, m') = \alpha(m) * \beta(m')$ on elements of $G \times G'$, where:

$$\alpha(m) * \beta(m') = \begin{cases} \Delta; [\alpha(m); \pi_Y, \beta(m')] & \text{if } m \notin \bar{Z}_\alpha(G), m' \in \bar{Z}_\beta(G'), \\ \Delta; [\alpha(m), \beta(m'); \pi_Z] & \text{otherwise,} \end{cases}$$

with $\Delta : \mathbf{X}\times\mathbf{Y}\times\mathbf{Z} \to \mathbf{X}\times\mathbf{Y}\times\mathbf{X}\times\mathbf{Z}; (x, y, z) \mapsto (x, y, x, z)$.

We can now define AMC of PRAMs and thus the interpretations of PRAMs as abstract programs. For each integer p, we define the AMC $\textsc{crew}^p(\alpha_{\mathrm{full}})$. This allows the consideration of up to p parallel RAMs: the translation of such a RAM with p processors is defined by extending the translation of RAMs by considering a set of states equal to $L_1 \times L_2 \times \cdots \times L_p$ where for all i the set L_i is the set of lines of the i-th processor.

Now, to deal with arbitrary large PRAMs, i.e. with arbitrarily large number of processors, one considers the following AMC defined as a *direct limit*.

[7]Here and in the following, we denote by ; the sequential composition of functions. I.e. $f; g$ denotes what is usually written $g \circ f$.

Definition 24 (The AMC of PRAMs)**.** Let $\alpha : M \curvearrowright \mathbf{X} \times \mathbf{X}$ be the AMC α_{full}. The AMC of PRAMs is defined as $\alpha_{\mathrm{pram}} = \varinjlim \textsc{crew}^k(\alpha)$, where $\textsc{crew}^{k-1}(\alpha)$ is identified with a restriction of $\textsc{crew}^k(\alpha)$ through $\textsc{crew}^{k-1}(\alpha)(m_1, \ldots, m_{k-1}) \mapsto \textsc{crew}^k(\alpha)(m_1, \ldots, m_{k-1}, 1)$.

Remark 3. We notice that the underlying space of the PRAM AMC α_{pram} is defined as the union $\cup_{n\in\omega}\mathbf{Z}^\omega \times (\mathbf{Z}^\omega)^n$ which we will write $\mathbf{Z}^\omega \times (\mathbf{Z}^\omega)^{(\omega)}$. In practise a given α_{pram}-program admitting a finite α_{pram} representative will only use elements in $\textsc{crew}^p(\alpha_{\mathrm{full}})$, and can therefore be understood as a $\textsc{crew}^p(\alpha)$-program.

Theorem 6. *The representation of* PRAMs *as* α_{pram}*-program is quantitatively sound.*

This result, here stated for integer-valued PRAMs, can easily be obtained for *real-valued* PRAMs translated as $\alpha_{\mathbf{R}_{\mathrm{full}}}$-programs.

IV. Entropy and Cells

A. Topological Entropy

Topological Entropy is a standard invariant of dynamical system. It is a value representing the average exponential growth rate of the number of orbit segments distinguishable with a finite (but arbitrarily fine) precision. The definition is based on the notion of open covers.

Definition 25 (Open covers)**.** Given a topological space $\mathbf{X}$, an *open cover* of $\mathbf{X}$ is a family $\mathcal{U} = (U_i)_{i\in I}$ of open subsets of $\mathbf{X}$ such that $\cup_{i\in I} U_i = \mathbf{X}$. A finite cover $\mathcal{U}$ is a cover whose indexing set is finite. A *subcover* of a cover $\mathcal{U} = (U_i)_{i\in I}$ is a sub-family $\mathcal{S} = (U_j)_{j\in J}$ for $J \subseteq I$ such that $\mathcal{S}$ is a cover, i.e. such that $\cup_{j\in J} U_j = \mathbf{X}$. We will denote by $\mathrm{Cov}(\mathbf{X})$ (resp. $\mathrm{FCov}(\mathbf{X})$) the set of all open covers (resp. all finite open covers) of the space $\mathbf{X}$.

Definition 26. An open cover $\mathcal{U} = (U_i)_{i\in I}$, together with a continuous function $f : \mathbf{X} \to \mathbf{X}$, defines the inverse image open cover $f^{-1}(\mathcal{U}) = (f^{-1}(U_i))_{i\in I}$. Given two open covers $\mathcal{U} = (U_i)_{i\in I}$ and $\mathcal{V} = (V_j)_{j\in J}$, we define their join $\mathcal{U} \vee \mathcal{V}$ as the family $(U_i \cap V_j)_{(i,j)\in I\times J}$.

Remark 4. If $\mathcal{U}, \mathcal{V}$ are finite, $f^{-1}(\mathcal{U})$ and $\mathcal{U} \vee \mathcal{V}$ are finite.

Traditionally [17], entropy is defined for continuous maps on a compact set. However, a generalisation of entropy to non-compact sets can easily be defined by restricting the usual definition to *finite* covers[8]. This is the definition we will use here.

Definition 27. Let $\mathbf{X}$ be a topological space, and $\mathcal{U} = (U_i)_{i\in I}$ be a finite cover of $\mathbf{X}$. We define the quantity $H^0_{\mathbf{X}}(\mathcal{U}) = \min\{\log_2(\mathrm{Card}(J)) \mid J \subseteq I, \cup_{j\in J} U_j = \mathbf{X}\}$.

In other words, if k is the cardinality of the smallest subcover of $\mathcal{U}$, $H^0(\mathcal{U}) = \log_2(k)$.

[8]This is discussed by Hofer [18] together with another generalisation based on the Stone-Čech compactification of the underlying space.

Definition 28. Let $\mathbf{X}$ be a topological space and $f : \mathbf{X} \to \mathbf{X}$ be a continuous map. For any finite open cover $\mathcal{U}$ of $\mathbf{X}$, define $H^k_{\mathbf{X}}(f,\mathcal{U}) = \frac{1}{k} H^0_{\mathbf{X}}(\mathcal{U} \vee f^{-1}(\mathcal{U}) \vee \cdots \vee f^{-(k-1)}(\mathcal{U}))$.

One can show that the limit $\lim_{n\to\infty} H^n_{\mathbf{X}}(f,\mathcal{U})$ exists and is finite; it will be noted $h(f,\mathcal{U})$. The topological entropy of f is then defined as the supremum of these values, when $\mathcal{U}$ ranges over the set of all finite covers $\mathrm{FCov}(\mathbf{X})$.

Definition 29. Let $\mathbf{X}$ be a topological space and $f : \mathbf{X} \to \mathbf{X}$ be a continuous map. The *topological entropy* of f is defined as $h(f) = \sup_{\mathcal{U}\in\mathrm{FCov}(\mathbf{X})} h(f,\mathcal{U})$.

B. Graphings and Entropy

We now need to define the entropy of *deterministic graphings*. As mentioned briefly already, deterministic graphings on a space $\mathbf{X}$ are in one-to-one correspondence with partial dynamical systems on $\mathbf{X}$. Thus, we only need to extend the notion of entropy to partial maps, and we can then define the entropy of a graphing G as the entropy of its corresponding map $[G]$.

Given a finite cover $\mathcal{U}$, the only issue with partial continuous maps is that $f^{-1}(\mathcal{U})$ is not in general a cover. Indeed, $\{f^{-1}(U) \mid U \in \mathcal{U}\}$ is a family of open sets by continuity of f but the union $\cup_{U\in\mathcal{U}} f^{-1}(U)$ is a strict subspace of $\mathbf{X}$ (namely, the domain of f). It turns out the solution to this problem is quite simple: we notice that $f^{-1}(\mathcal{U})$ is a cover of $f^{-1}(\mathbf{X})$ and now work with covers of subspaces of $\mathbf{X}$. Indeed, $\mathcal{U} \vee f^{-1}(\mathcal{U})$ is itself a cover of $f^{-1}(\mathbf{X})$ and therefore the quantity $H^2_{\mathbf{X}}(f,\mathcal{U})$ can be defined as $(1/2)H^0_{f^{-1}(\mathbf{X})}(\mathcal{U} \vee f^{-1}(\mathcal{U}))$.

We now generalise this definition to arbitrary iterations of f by extending Definitions 28 and 29 to partial maps as follows.

Definition 30. Let $\mathbf{X}$ be a topological space and $f : \mathbf{X} \to \mathbf{X}$ be a continuous partial map. For any finite open cover $\mathcal{U}$ of $\mathbf{X}$, we define $H^k_{\mathbf{X}}(f,\mathcal{U}) = \frac{1}{k} H^0_{f^{-k+1}(\mathbf{X})}(\mathcal{U} \vee f^{-1}(\mathcal{U}) \vee \cdots \vee f^{-(k-1)}(\mathcal{U}))$. The *entropy* of f is then defined as $h(f) = \sup_{\mathcal{U}\in\mathrm{FCov}(\mathbf{X})} h(f,\mathcal{U})$, where $h(f,\mathcal{U})$ is again defined as the limit $\lim_{n\to\infty} H^n_{\mathbf{X}}(f,\mathcal{U})$.

We now consider the special case of a graphing G with set of control states S^G. For an intuitive understanding, one can think of G as the representation of a PRAM machine. We focus on the specific open cover indexed by the set of control states, i.e. $\mathcal{S} = (\mathbf{X}\times\{s\}_{s\in S^G})$, and call it *the states cover*. We will now show how the partial entropy $H^k(G,\mathcal{S})$ is related to the set of *admissible sequence of states*. Let us define those first.

Definition 31. Let G be a graphing, with set of control states S^G. An admissible sequence of states is a sequence $\mathbf{s} = s_1 s_2 \dots s_n$ of elements of S^G such that for all $i \in \{1,2,\dots,n-1\}$ there exists a subset C of $\mathbf{X}$ – i.e. a set of configurations – such that G contains an edge from $C \times \{s_i\}$ to a subspace of $\mathbf{X} \times \{s_{i+1}\}$.

Example 1. As an example, let us consider the very simple graphing with four control states a, b, c, d and edges from $\mathbf{X}\times\{a\}$ to $\mathbf{X}\times\{b\}$, from $\mathbf{X}\times\{b\}$ to $\mathbf{X}\times\{c\}$, from $\mathbf{X}\times\{c\}$ to $\mathbf{X}\times\{b\}$ and from $\mathbf{X}\times\{c\}$ to $\mathbf{X}\times\{d\}$. Then the sequences *abcd* and *abcbcbc* are admissible, but the sequences *aba*, *abcdd*, and *abcba* are not.

Lemma 2. *Let G be a graphing, and $\mathcal{S}$ its states cover. Then for all integer k, the set $\mathrm{Adm}_k(G)$ of admissible sequences of states of length $k > 1$ is of cardinality $2^{k.H^k(G,\mathcal{S})}$.*

A tractable bound on the number of admissible sequences of states can be obtained by noticing that the sequence $H^k(G,\mathcal{S})$ is *sub-additive*, i.e. $H^{k+k'}(G,\mathcal{S}) \leqslant H^k(G,\mathcal{S}) + H^{k'}(G,\mathcal{S})$. A consequence of this is that $H^k(G,\mathcal{S}) \leqslant kH^1(G,\mathcal{S})$. Thus the number of admissible sequences of states of length k is bounded by $2^{k^2H^1(G,\mathcal{S})}$. We now study how the cardinality of admissible sequences can be related to the entropy of G. This is deduced from Lemma 2 and the following general result (which does not depend on the chosen cover).

Lemma 3. *For all $\epsilon > 0$ and all cover $\mathcal{U}$, there exists a natural number N such that $\forall k \geqslant N$, $H^k(G,\mathcal{U}) < h([G]) + \epsilon$.*

The two previous lemmas combine to give the following.

Lemma 4. *Let G be a graphing. Then* $\mathrm{Card}(\mathrm{Adm}_k(G)) = O(2^{k.h([G])})$ *as $k \to \infty$.*

C. Cells Decomposition

Now, let G be a deterministic graphing with its state cover $\mathcal{S}$. We fix $k > 2$ and consider the partition $(C[\mathbf{s}])_{\mathbf{s}\in\mathrm{Adm}_k(G)}$ of the space $[G]^{-k+1}(\mathbf{X})$, where the sets $C[\mathbf{s}] = C[(s_1 s_2 \dots s_{k-1}, s_k)]$ are defined inductively as follow:

- $C[s_1, s_2]$ is the set $\{x \in \mathbf{X} \mid [G](x, s_1) \in X \times \{s_2\}\}$;
- $C[(s_1 s_2 \dots s_{k-1}, s_k)]$ is the set $\{x \in \mathbf{X} \mid \forall i \in \{2,\dots,k\}, [G]^{i-1}(x, s_1) \in \mathbf{X} \times \{s_i\}\}$.

This decomposition splits the set of initial configurations into cells satisfying the following property: *for any two initial configurations contained in the same cell $C[\mathbf{s}]$, the k-th first iterations of G go through the same admissible sequence of states* $\mathbf{s}$.

Definition 32. Let G be a deterministic graphing, with its state cover $\mathcal{S}$. Given an integer k, we define the k-th cell decomposition of $\mathbf{X}$ along G as the partition $\{C[\mathbf{s}] \mid \mathbf{s} \in \mathrm{Adm}_k(G)\}$.

Then Lemma 2 provides a bound on the cardinality of the k-th cell decomposition. Using the results in the previous section, we can then obtain the following proposition.

Proposition 2. *Let G be a deterministic graphing, with entropy $h(G)$. The cardinality of the k-th cell decomposition of $\mathbf{X}$ w.r.t. G, as a function $c(k)$ of k, is asymptotically bounded by $g(k) = 2^{k.h([G])}$, i.e. $c(k) = O(g(k))$.*

We also state another bound on the number of cells of the k-th cell decomposition, based on the state cover

entropy, i.e. the entropy with respect to the state cover rather than the usual entropy which takes the supremum of cover entropies when the cover ranges over all finite covers of the space. This is a simple consequence of Lemma 2.

Proposition 3. *Let G be a deterministic graphing. We consider the* state cover entropy $h_0([G]) = \lim_{n\to\infty} H^n_{\mathbf{X}}([G], \mathcal{S})$ *where* $\mathbf{S}$ *is the state cover. The cardinality of the k-th cell decomposition of* $\mathbf{X}$ *w.r.t. G, as a function $c(k)$ of k, is asymptotically bounded by $g(k) = 2^{k.h_0([G])}$, i.e. $c(k) = O(g(k))$.*

V. Entropic Cotrees and the Main Lemma

A. Lower Bounds through the Milnor-Thom theorem

The results stated in the last section can be used to prove lower bounds in several models. These results rely on two ingredients: the above bounds on the cardinality of the k-th cell decomposition, and the Milnor-Thom theorem.

The Milnor-Thom theorem, which was proven independently by Milnor [19] and Thom [20], states bounds on the sum of the Betti numbers (i.e. the rank of the homology groups) of an algebraic variety. This theorem provides bounds on the number of connected components (i.e. the 0-th Betti number $\beta_0(V)$) of a semi-algebraic variety V. We here use the statement of the Milnor-Thom theorem as given by Ben-Or [2, Theorem 2].

Theorem 7. *Let $V \subseteq \mathbf{R}^n$ be a set defined by polynomial in·equations ($n, m, h \in \mathbf{N}$):*

$$\begin{aligned}&\{q_i(\vec{x}) = 0 \mid 0 \leqslant i \leqslant m\}\\ &\quad \cup \{p_i(\vec{x}) > 0 \mid 0 \leqslant i \leqslant s\}\\ &\quad\quad \cup \{p_i(\vec{x}) \leqslant 0 \mid s+1 \leqslant i \leqslant h\}.\end{aligned}$$

Then $\beta_0(V)$ is at most $d(2d-1)^{n+h-1}$, where $d = \max\{2, \deg(q_i), \deg(p_j)\}$.

The lower bounds proofs then proceed by the following proof strategy:

1) consider an algebraic model of computation, and define the corresponding AMC;
2) show that the cells in the k-th cell decomposition are semi-algebraic sets defined by systems of equations E with explicit *upper bounds* on the number of equations and the degrees of the polynomials;
3) bound the number of connected components of each cell by the Milnor-Thom theorem;
4) given an algebraic problem (e.g. a subset of $\mathbf{R}^k$), deduce lower bounds on the length of the computations deciding that problem based on its number of connected components.

Among the lower bound proofs using this proof strategy, we point out Steele and Yao lower bounds on algebraic decision trees [1], and Mulmuley's proof of lower bounds on "PRAMs without bit operations" [4]. These results do not use the notion of entropy. Due to space constraints, we do not detail these in this paper.

We will now explain how this method can be refined following Ben-Or's proof of lower bounds for algebraic computational trees. Indeed, while Mulmuley's [4] was not later improved upon, Steele and Yao's lower bounds were extended by Ben-Or [2] to encompass algebraic computational trees with sums, substractions, products, divisions and square roots. The technique of Ben-Or improves on Steele and Yao in that it provides a method to deal with divisions and square roots. We here abstract this method by considering *k-th entropic co-trees* which are a refinement of the k-th cell decomposition. This allows us to recover Ben-Or's result, capture Cucker's proof that $\mathrm{NC}_{\mathbf{R}} \neq \mathrm{PTIME}_{\mathbf{R}}$, and to strengthen Mulmuley's result by allowing the machines considered to use divisions and square roots.

B. Entropic co-trees

The principle underlying the improvement of Ben-Or on Steele and Yao consists in adding additional variables to avoid using the square root or division, obtaining in this way a system of polynomial equations instead of a single equation for a given cell in the k-th cell decomposition. For instance, instead of writing the equation $p/q < 0$, one defines a fresh variable r and considers the system $\{p = qr; r < 0\}$.

To adapt it to graphings, we consider the notion of *entropic co-tree* of a graphing that generalises the k-th cell decomposition to account for the instructions used at each step of the computation.

As we explained in Remark 3, a given PRAM is interpreted as a $\mathrm{CREW}^p(\alpha_{\mathbf{R}\mathrm{full}})$-program for a fixed integer p (the number of processors). It is therefore enough to state the following definitions and results for the AMC $\mathrm{CREW}^p(\alpha_{\mathbf{R}\mathrm{full}})$ to apply them to the interpretations of arbitrary PRAMs.

Definition 33 (k-th entropic co-tree)**.** Consider a deterministic $\mathrm{CREW}^p(\alpha_{\mathbf{R}\mathrm{full}})$-graphing representative T, and fix an element $\top$ of the set of control states. We can define the k-th entropic co-tree of T along $\top$ and the state cover inductively:

- $k = 0$, the co-tree $\mathrm{COT}_0(T)$ is simply the root $n^\epsilon = \mathbf{R}^n \times \{\top\}$;
- $k = 1$, one considers the preimage of n^ϵ through T, i.e. $T^{-1}(\mathbf{R}^n \times \{\top\})$ the set of all non-empty sets $\alpha(m_e)^{-1}(\mathbf{R}^n \times \{\top\})$ and intersects it pairwise with the state cover, leading to a finite family (of cardinality bounded by the number of states multiplied by the number of edges fo T) $(n^i_e)_i$ defined as $n^i = T^{-1}(n^\epsilon) \cap \mathbf{R}^n \times \{i\}$. The first entropic co-tree $\mathrm{COT}_1(T)$ of T is then the tree defined by linking each n^i_e to n^ϵ with an edge labelled by m_e;
- $k+1$, suppose defined the k-th entropic co-tree of T, defined as a family of elements $n^\pi_{\mathbf{e}}$ where π is a finite sequence of states of length at most k and $\mathbf{e}$ a sequence of edges of T of the same length, and where $n^\pi_{\mathbf{e}}$ and $n^{\pi'}_{\mathbf{e}'}$ are linked by an edge labelled f if and only if $\pi' = \pi.s$ and $\mathbf{e}' = f.\mathbf{e}$ where s is a state and f an edge of T. We consider the subset of elements $n^\pi_{\mathbf{e}'}$ where π is exactly

of length k, and for each such element we define new vertices $n^{\pi.s}_{e.\mathbf{e}'}$ defined as $\alpha(m_e)^{-1}(n^{\pi}_{\mathbf{e}'}) \cap \mathbf{R}^n \times \{s\}$ when it is non-empty. The $k+1$-th entropic co-tree $\mathrm{COT}_{k+1}(T)$ is defined by extending the k-th entropic co-tree $\mathrm{COT}_k(T)$, adding the vertices $n^{\pi.s}_{e.\mathbf{e}'}$ and linking them to $n^{\pi}_{\mathbf{e}'}$ with an edge labelled by e.

We can easily obtain bounds on the size of the cotrees, refining the bounds on the kth cell decomposition.

Proposition 4. *Let G be a deterministic $\mathrm{CREW}^p(\alpha_{\mathbf{R}\mathrm{full}})$-graphing with a finite set of edges E, and $\mathrm{Seq}_k(E)$ the set of length k sequences of edges in G. We consider the* state cover entropy $h_0([G]) = \lim_{n\to\infty} H^n_{\mathbf{X}}([G], \mathcal{S})$ *where $\mathbf{S}$ is the state cover. The cardinality of the length k vertices of the entropic co-tree of G, as a function $c(k)$ of k, is asymptotically bounded by $g(k) = \mathrm{Card}(\mathrm{Seq}_k(E)).2^{k.h_0([G])}$, which is itself bounded by $2^{\mathrm{Card}(E)}.2^{k.h_0([G])}$.*

C. The main lemma

This definition formalises a notion that appears more or less clearly in the work of Steele and Yao, and of Ben-Or, as well as in the proof by Mulmuley. The vertices for paths of length k in the k-th co-tree corresponds to the k-th cell decomposition, and the corresponding path defines the polynomials describing the semi-algebraic set decided by a computational tree. While in Steele and Yao and Mulmuley's proofs, one obtain directly a polynomial for each cell, we here need to construct a system of equations for each branch of the co-tree.

Given a $\mathrm{CREW}^p(\alpha_{\mathbf{R}\mathrm{full}})$-graphing representative G we will write $\sqrt[\partial]{G}$ the maximal value of n for which an instruction $\sqrt[n]{i}(j)$ appears in the realiser of an edge of G.

Lemma 5. *Let G be a computational graphing representative with edges realised only by generators of the* AMC $\mathrm{CREW}^p(\alpha_{\mathbf{R}\mathrm{full}})$*, and $\mathrm{Seq}_k(E)$ the set of length k sequences of edges in G. Suppose G computes the membership problem for $W \subseteq \mathbf{R}^n$ in k steps, i.e. for each element of $\mathbf{R}^n$, $\pi_{\mathbf{S}}(G^k(x)) = \top$ if and only if $x \in W$. Then W is a semi-algebraic set defined by at most $\mathrm{Card}(\mathrm{Seq}_k(E)).2^{k.h_0([G])}$ systems of pk equations of degree at most $\max(2, \sqrt[\partial]{G})$ and involving at most $pk + n$ variables.*

The proof of this theorem is long but simple to understand. We define, for each vertex of the k-th entropic co-tree, a system of algebraic equations (each of degree at most 2). The system is defined by induction on k, and uses the information of the specific instruction used to extend the sequence indexing the vertex at each step. For instance, the case of division follows Ben-Or's method, introducing a fresh variable and writing down two equations as explained in Section V-B.

VI. Recovering results from the literature

A. Ben-Or's theorem

We now recover Ben-Or result by obtaining a bound on the number of connected components of the subsets $W \subseteq \mathbf{R}^n$ whose membership problem is computed by a graphing in less than a given number of iterations. This theorem is obtained by applying the Milnor-Thom theorem on the obtained systems of equations to bound the number of connected components of each cell. Notice that in this case $p = 1$ and $\sqrt[\partial]{G} = 2$ since the model of algebraic computation trees use only square roots. A mode general result holds for algebraic computation trees extended with arbitrary roots, but we here limit ourselves here to the original model.

Theorem 8. *Let G be a computational $\alpha_{\mathbf{R}\mathrm{full}}$-graphing representative translating an algebraic computational tree, $\mathrm{Seq}_k(E)$ the set of length k sequences of edges in G. Suppose G computes the membership problem for $W \subseteq \mathbf{R}^n$ in k steps. Then W has at most $\mathrm{Card}(\mathrm{Seq}_k(E)).2^{k.h_0([G])+1}3^{2k+n-1}$ connected components.*

Since a subset computed by a tree T of depth k is computed by $[\![T]\!]$ in k steps by Theorem 3, we get as a corollary the original theorem by Ben-Or relating the number of connected components of a set W and the depth of the algebraic computational trees that compute the membership problem for W.

Corollary 1 ([2, Theorem 5])**.** *Let $W \subseteq \mathbf{R}^n$ be any set, and let N be the maximum of the number of connected components of W and $\mathbf{R}^n \setminus W$. An algebraic computation tree computing the membership problem for W has height $\Omega(\log N)$.*

Remark 5. In the case of algebraic PRAMs discussed in the next sections, the k-th entropic co-tree $\mathrm{COT}_k(T)[M]$ of a machine M defines an algebraic computation tree which follows the k-th first steps of computation of M. I.e. the algebraic computation tree $\mathrm{COT}_k(T)[M]$ approximate the computation of M in such a way that M and $\mathrm{COT}_k(T)[M]$ behave in the exact same manner in the first k steps.

B. Cucker's theorem

Cucker's proof considers the problem defined as the following algebraic set.

Definition 34. Define $\mathfrak{Fer}$ to be the set:

$$\{x \in \mathbf{R}^\omega \mid |x| = n \Rightarrow x_1^{2^n} + x_2^{2^n} = 1\},$$

where $|x| = \max\{n \in \omega \mid x_n \neq 0\}$.

It can be shown to lie within $\mathrm{PTIME}_{\mathbf{R}}$, i.e. it is decided by a real Turing machine [21] – i.e. working with real numbers and real operations –, running in polynomial time.

Theorem 9 (Cucker ([3], Proposition 3))**.** *The problem* $\mathfrak{Fer}$ *belongs to* $\mathrm{PTIME}_{\mathbf{R}}$.

We now prove that $\mathfrak{Fer}$ is not computable by an algebraic circuit of polylogarithmic depth. The proof follows Cucker's argument, but uses the lemma proved in the previous section.

Theorem 10 (Cucker ([3], Theorem 3.2))**.** *No algebraic circuit of depth $k = \log^i n$ and size kp compute* $\mathfrak{Fer}$.

Proof. For this, we will use the lower bounds result obtained in the previous section. Indeed, by Theorem 4 and Lemma 5, any problem decided by an algebraic circuit of depth k is a semi-algebraic set defined by at most $\mathrm{Card}(\mathrm{Seq}_k(E)).2^{k.h_0([G])}$ systems of k equations of degree at most $\max(2, \sqrt[\partial]{G}) = 2$ (since only square roots are allowed in the model) and involving at most $k+n$ variables. But the curve $\mathfrak{F}^{\mathbf{R}}_{2^n}$ defined as $\{x_1^{2^n}+x_2^{2^n}-1=0 \mid x_1, x_2 \in \mathbf{R}\}$ is infinite. As a consequence, one of the systems of equations must describe a set containing an infinite number of points of $\mathfrak{F}^{\mathbf{R}}_{2^n}$.

This set S is characterized, up to some transformations on the set of equations obtained from the entropic co-tree, by a finite system of in·equalities of the form

$$\bigwedge_{i=1}^{s} F_i(X_1, X_2) = 0 \wedge \bigwedge_{j=1}^{t} G_j(X_1, X_2) < 0,$$

where t is bounded by kp and the degree of the polynomials F_i and G_i are bounded by 2^k. Moreover, since $\mathfrak{F}^{\mathbf{R}}_{2^n}$ is a curve and no points in S must lie outside of it, we must have $s > 0$.

Finally, the polynomials F_i vanish on that infinite subset of the curve and thus in a 1-dimensional component of the curve. Since the curve is an irreducible one, this implies that every F_i must vanish on the whole curve. Using the fact that the ideal $(X_1^{2^n} + X_2^{2^n} - 1)$ is prime (and thus radical), we conclude that all the F_i are multiples of $X_1^{2^n} + X_2^{2^n} - 1$ which is impossible if their degree is bounded by $2^{\log^i n}$ as it is strictly smaller than 2^n. □

VII. A proof that $\mathrm{NC}_{\mathbf{Z}} \neq$ Ptime

In this section, we provide a new presentation of a result of Mulmuley which is part of his lower bounds for "prams without bit operations". The idea is to encode a specific decision problem and the run of a PRAM as two specific subsets of the same space and show that no short run of the machine can define the set of all instances of the decision problem. More specifically, consider the problem `maxflow`: given a weighted graph, find the maximal flow from a source edge to a target edge. This is an optimization problem. It can be turned into a decision problem by adding a new variable z—a threshold—and asking whether there exists a solution greater than z. This decision problem is known to be Ptime-complete [11].

A. Geometric Interpretation of Optimization Problems

Let $\mathcal{P}_{\mathrm{opt}}$ be an optimization problem on $\mathbf{R}^d$. Solving $\mathcal{P}_{\mathrm{opt}}$ on an instance t amounts to optimizing a function $f_t(\cdot)$ over a space of parameters. We note $\mathrm{Max}\mathcal{P}_{\mathrm{opt}}(t)$ this optimal value. An affine function $\mathsf{Param} : [p;q] \to \mathbf{R}^d$ is called a *parametrization* of $\mathcal{P}_{\mathrm{opt}}$. Such a parametrization defines naturally a decision problem $\mathcal{P}_{\mathrm{dec}}$: for all $(x,y,z) \in \mathbf{Z}^3$, $(x,y,z) \in \mathcal{P}_{\mathrm{dec}}$ iff $z > 0$, $x/z \in [p;q]$ and $y/z \leq \mathrm{Max}\mathcal{P}_{\mathrm{opt}} \circ \mathsf{Param}(x/z)$.

In order to study the geometry of $\mathcal{P}_{\mathrm{dec}}$ in a way that makes its connection with $\mathcal{P}_{\mathrm{opt}}$ clear, we consider the ambient space to be $\mathbf{R}^3$, and we define the *ray* $[p]$ of a point p as the half-line starting at the origin and containing p. The projection $\Pi(p)$ of a point p on a plane is the intersection of $[p]$ and the affine plane $\mathcal{A}_1$ of equation $z = 1$. For any point $p \in \mathcal{A}_1$, and all $p_1 \in [p]$, $\Pi(p_1) = p$. It is clear that for $(p, p', q) \in \mathbf{Z}^2 \times \mathbf{N}^+$, $\Pi((p,p',q)) = (p/q, p'/q, 1)$.

The *cone* $[C]$ of a curve C is the set of rays of points of the curve. The projection $\Pi(C)$ of a surface or a curve C is the set of projections of points in C. We note Front the frontier set $\mathsf{Front} = \{(x,y,1) \in \mathbf{R}^3 \mid y = \mathrm{Max}\mathcal{P}_{\mathrm{opt}} \circ \mathsf{Param}(x)\}$. and we remark that $[\mathsf{Front}] = \{(x,y,z) \in \mathbf{R}^2 \times \mathbf{R}^+ \mid y/z = \mathrm{Max}\mathcal{P}_{\mathrm{opt}} \circ \mathsf{Param}(x/z)\}$.

A machine M decides the problem $\mathcal{P}_{\mathrm{dec}}$ in k steps if the partition of accepting cells in $\mathbf{Z}^3$ induced by the machine – i.e. the k-th cell decomposition – is finer than the one defined by the problem's frontier $[\mathsf{Front}]$ (which is defined by the equation $y/z \leq \mathrm{Max}\mathcal{P}_{\mathrm{opt}} \circ \mathsf{Param}(x/z)$).

Parametric Complexity. We now further restrict the class of problems we are interested in: we will only consider $\mathcal{P}_{\mathrm{opt}}$ such that Front is simple enough.

Definition 35. We say that Param is an *affine parametrization* of $\mathcal{P}_{\mathrm{opt}}$ if $\mathrm{Max}\mathcal{P}_{\mathrm{opt}} \circ \mathsf{Param}$ is convex, piecewise linear with breakpoints $\lambda_1 < \cdots < \lambda_\rho$, and such that all $(\lambda_i)_i$ and $(\mathrm{Max}\mathcal{P}_{\mathrm{opt}} \circ \mathsf{Param}(\lambda_i))_i$ are rational. The *parametric complexity* $\rho(\mathsf{Param})$ is the number of breakpoints ρ. The *bitsize* of the parametrization is the maximum of the bitsizes of the numerators and denominators of the coordinates of the breakpoints of $\mathrm{Max}\mathcal{P}_{\mathrm{opt}} \circ \mathsf{Param}$.

An optimization problem admitting an affine parametrization of complexity ρ is thus represented by a quite simple surface $[\mathsf{Front}]$: the cone of the graph of a piecewise affine function, constituted of ρ segments. We call such a surface is a *ρ-fan* and define its bitsize as β if all its breakpoints are rational and the bitsize of their coordinates is less than β.

The restriction to such optimization problems seems quite dramatic when understood geometrically. Nonetheless, `maxflow` admits such a parametrization.

Theorem 11 (Murty [22], Carstensen [23])**.** *There exists an affine parametrization of bitsize $O(n^2)$ and complexity $2^{\Omega(n)}$ of the* `maxflow` *problem for directed and undirected networks, where n is the number of nodes in the network.*

Surfaces and fans. An algebraic surface in $\mathbf{R}^3$ is a surface defined by an equation of the form $p(x,y,z) = 0$ where p is a polynomial. If S is a set of surfaces S_i, each defined by a polynomial p_i, the *total degree* of S is defined as the sum of the degrees of polynomials p_i.

Let K be a compact of $\mathbf{R}^3$ delimited by algebraic surfaces and S be a finite set of algebraic surfaces of total degree δ. We can assume that K is delimited by two affine planes of equation $z = \mu$ and $z = 2\mu_z$ and the cone of a rectangle $\{(x,y,1) \mid |x|, |y| \leqslant \mu_{x,y}\}$, by taking any such compact containing K and adding the surfaces bounding K to S. S defines a partition of K by considering maximal

compact subspaces of K whose boundaries are included in surfaces of S. Such elements are called the *cells* of the decomposition associated to S.

Definition 36. Let K be a compact of $\mathbf{R}^3$. A finite set of surfaces S on K *separates* a ρ-fan Fan on K if the partition on $\mathbf{Z}^3 \cap K$ induced by S is finer than the one induced by Fan.

A major technical achievement of Mulmuley [4] – not explicitly stated – was to prove the following theorem, of purely geometric nature. We refer to the long version of this work[9] for a detailed proof of this result.

Theorem 12 (Mulmuley)**.** *Let S be a finite set of algebraic surfaces of total degree δ. There exists a polynomial P such that, for all $\rho > P(\delta)$, S does not separate ρ-fans.*

B. Strengthening Mulmuley's result

We will now prove our strengthening of Mulmuley's lower bounds for "PRAMs without bit operations" [4]. For this, we will combine the results from previous sections to establish the following result.

Theorem 2. *Let G be a deterministic graphing interpreting a* PRAM *with $2^{O((\log N)^c)}$ processors, where N is the length of the inputs and c any positive integer.*

Then G does not decide `maxflow` *in $O((\log N)^c)$ steps.*

So, let M be an integer-valued PRAM. We can associate to it a real-valued PRAM $\tilde{M}$ such that M and $\tilde{M}$ accept the same (integer) values, and the ratio between the running time of the two machines is a constant. Indeed:

Proposition 5. *Euclidian division can be computed by a constant time real-valued* PRAM*.*

Proof of Theorem 2. Suppose now that $[\![M]\!]$ has a finite set of edges E. Then $[\![\tilde{M}]\!]$ has too has a finite set of edge of cardinality $O(\mathrm{Card}(E))$. Since the running time of the initial PRAM over integers is equal, up to a constant, to the computation time of the $\mathrm{CREW}^p(\alpha_{\mathbf{R}\mathrm{full}})$-program $[\![\tilde{M}]\!]$, we deduce that if M computes `maxflow` in k steps, then $[\![\tilde{M}]\!]$ computes `maxflow` in at most Ck steps where C is a fixed constant.

By Lemma 5, the problem decided by $[\![\tilde{M}]\!]$ in Ck steps defines a system of equations separating the integral inputs accepted by M from the ones rejected. I.e. if M computes `maxflow` in Ck steps, then this system of equations defines a set of algebraic surfaces that separate the ρ-fan defined by `maxflow`. Moreover, this system of equation has a total degree bounded by $Ck \max(2, \sqrt[\partial]{G}) 2p \times 2^{O(\mathrm{Card}(E))} \times 2^{k.h_0([\![\tilde{M}]\!])}$.

By Theorem 11 and Theorem 12, there exists a polynomial P such that a finite set of algebraic surfaces of total degree δ cannot separate the $2^{\Omega(n)}$-fan defined by `maxflow` as long as $2^{\Omega(n)} > P(\delta)$. But here the entropy of G is $O(p)$, as the entropy of a product $f \times g$ satisfies $h(f \times g) \leqslant h(f) + h(g)$ [24]. Hence $\delta = O(2^p 2^k)$, contradicting the hypotheses that $p = 2^{O((\log N)^c)}$ and $k = 2^{O((\log N)^c)}$. □

This has Theorem 1 as a corollary, which shows that the class $\mathrm{NC}_{\mathbf{Z}}$ does not contain `maxflow`, and hence is distinct from PTIME. The question of how this class relates to NC is open: indeed, while bit extractions cannot be performed in constant time by our machines (a consequence of Theorem 5), they can be simulated in logarithmic time.

[9] For the purpose of double-blind reviews, we do not provide an explicit reference for the moment.

REFERENCES

[1] J. M. Steele and A. Yao, "Lower bounds for algebraic decision trees," *Journal of Algorithms*, vol. 3, pp. 1–8, 1982.

[2] M. Ben-Or, "Lower bounds for algebraic computation trees," in *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, ser. STOC '83. New York, NY, USA: ACM, 1983, pp. 80–86. [Online]. Available: http://doi.acm.org/10.1145/800061.808735

[3] F. Cucker, "$\mathbf{P_R} \neq \mathbf{NC_R}$," *Journal of Complexity*, vol. 8, no. 3, pp. 230 – 238, 1992. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0885064X92900246

[4] K. Mulmuley, "Lower bounds in a parallel model without bit operations," *SIAM J. Comput.*, vol. 28, no. 4, pp. 1460–1509, 1999. [Online]. Available: https://doi.org/10.1137/S0097539794282930

[5] S. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the 3rd ACM Symposium on Theory of Computing*, 1971.

[6] R. Williams, "Nonuniform acc circuit lower bounds," *J. ACM*, vol. 61, no. 1, pp. 2:1–2:32, Jan. 2014. [Online]. Available: http://doi.acm.org/10.1145/2559903

[7] T. Baker, J. Gill, and R. Solovay, "Relativizations of the p = np question," *SIAM Journal on Computing*, vol. 4, no. 4, pp. 431–442, 1975.

[8] A. A. Razborov and S. Rudich, "Natural proofs," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 24 – 35, 1997.

[9] S. Aaronson and A. Wigderson, "Algebrization: A new barrier in complexity theory," *ACM Trans. Comput. Theory*, vol. 1, no. 1, pp. 2:1–2:54, Feb. 2009. [Online]. Available: http://doi.acm.org/10.1145/1490270.1490272

[10] K. D. Mulmuley, "The gct program toward the p vs. np problem," *Commun. ACM*, vol. 55, no. 6, pp. 98–107, Jun. 2012. [Online]. Available: http://doi.acm.org/10.1145/2184319.2184341

[11] L. M. Goldschlager, R. A. Shaw, and J. Staples, "The maximum flow problem is log space complete for p," *Theoretical Computer Science*, vol. 21, p. 105–111, 1982.

[12] C. A. Neff, "Specified precision polynomial root isolation is in NC," *Journal of Computer and System Sciences*, vol. 48, no. 3, pp. 429 – 463, 1994.

[13] T. Seiller, "Interaction graphs: Full linear logic," in *IEEE/ACM Logic in Computer Science (LICS)*, 2016. [Online]. Available: http://arxiv.org/pdf/1504.04152

[14] ——, "Interaction graphs: Graphings," *Annals of Pure and Applied Logic*, vol. 168, no. 2, pp. 278–320, 2017.

[15] ——, "Interaction graphs: Nondeterministic automata," *ACM Transaction in Computational Logic*, vol. 19, no. 3, 2018.

[16] ——, "Interaction Graphs: Exponentials," *Logical Methods in Computer Science*, vol. Volume 15, Issue 3, Aug. 2019. [Online]. Available: https://lmcs.episciences.org/5730

[17] R. L. Adler, A. G. Konheim, and M. H. McAndrew, "Topological entropy," *Transactions of the American Mathematical Society*, vol. 114, no. 2, pp. 309–319, 1965.

[18] J. E. Hofer, "Topological entropy for noncompact spaces." *The Michigan Mathematical Journal*, vol. 21, no. 3, pp. 235–242, 1975.

[19] J. Milnor, "On the Betti numbers of real varieties," in *Proceedings of the American Mathematical Society*, 1964, p. 275.

[20] R. Thom, *Sur l'homologie des variétés algébriques réelles*. Princeton University Press, 1965, pp. 255–265.

[21] L. Blum, M. Shub, and S. Smale, "On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines," *American Mathematical Society. Bulletin. New Series*, vol. 21, no. 1, pp. 1–46, 1989.
[22] K. G. Murty, "Computational complexity of parametric linear programming," *Mathematical programming*, vol. 19, no. 1, pp. 213–219, 1980.
[23] P. J. Carstensen, "The complexity of some problems in parametric linear and combinatorial programming," Ph.D. dissertation, Ann Arbor, MI, USA, 1983.
[24] L. W. Goodwyn, "The product theorem for topological entropy," *Transactions of the American Mathematical Society*, vol. 158, no. 2, pp. 445–452, 1971. [Online]. Available: http://www.jstor.org/stable/1995916

APPENDIX

Proof of Proposition 1. A computation tree defines an α_{act}-graphing $[T]$, and the natural α_{act}-graphing representative obtained from the inductive definition of $[T]$ is clearly an α_{act}-treeing because T is a tree. That this treeing represents faithfully the computational tree T raises no difficulty.

Let us now show that the membership problem of a subset $W \subseteq \mathbf{R}^n$ that can be decided by a computational α_{act}-treeing is also decided by an algebraic computation tree T. We prove the result by induction on the number of states of the computational α_{act}-treeing. The initial case is when T the set of states is exactly $\{1, \top, \bot\}$ with the order defined by $1 < \top$ and $1 < \bot$ and no other relations. This computational α_{act}-treeing has at most 2 edges, since it is deterministic and the source of each edge is a subset among $\mathbf{R}^\omega$, $\mathbf{R}^\omega_{k\geqslant 0}$, $\mathbf{R}^\omega_{k\leqslant 0}$, $\mathbf{R}^\omega_{k>0}$, $\mathbf{R}^\omega_{k<0}$, $\mathbf{R}^\omega_{k=0}$, and $\mathbf{R}^\omega_{k\neq 0}$.

We first treat the case when there is only one edge of source $\mathbf{R}^n$. An element $(x_1, \dots, x_n) \in \mathbf{R}^n$ is decided by T if the main representative $((x_1, \dots, x_n, 0, \dots), 1)$ is mapped to $\top$. Since there is only one edge of source the whole space, either this edge maps into the state $\top$ and the decided subset W is equal to $\mathbf{R}^n$, or it maps into $\bot$ and the subset W is empty. In both cases, there exists an algebraic computation tree deciding W. For the purpose of the proof, we will however construct a specific algebraic computation tree, namely the one that first computes the right expression and then accepts or rejects. I.e. if the only edge mapping into $\top$ (resp. $\bot$) is realised by an element m in the AMC of algebraic computation trees which can be written as a product of generators $g_1, \dots, g_k$, we construct the tree of height $k+1$ that performs (in that order) the operations corresponding to g_1, g_2, etc., and then answers "yes" (resp. "no").

Now, the case where there is one edge of source a strict subspace, e.g. $\mathbf{R}^\omega_{k\geqslant 0}$ (all other cases are treated in a similar manner) and mapping into $\top$ (the other case is treated by symmetry). First, let us remark that if there is no other edge, one could very well add an edge to T mapping into $\bot$ and realised by the identity with source the complementary subspace $\mathbf{R}^\omega_{k<0}$. We build a tree as follows. First, we test whether the variable x_k is greater or equal to zero; this node has two children corresponding to whether the answer to the test is "yes" or "no". We now construct the two subtrees corresponding to these two children. The branch corresponding to "yes" is described by the edge of source $\mathbf{R}^\omega_{k\geqslant 0}$: we construct the tree of height $k+1$ performing the operations corresponding to the generators g_1, g_2, etc. whose product defined the realiser m of e, and then answers "yes" (resp. "no") if the edge e maps into the state $\top$ (resp. $\bot$). Similarly, the other subtree is described by the realiser of the edge of source $\mathbf{R}^\omega_{k<0}$.

The result then follows by induction, plugging small subtrees as described above in place of the leaves of smaller subtrees. □

Proof of Lemma 2. We show that the set $\mathrm{Adm}_k(G)$ of admissible sequences of states of length k has the same cardinality as the smallest subcover of $\mathcal{S} \vee [G]^{-1}(\mathcal{S}) \vee \dots \vee [G]^{-(k-1)}(\mathcal{S}))$. Hence $H^k(G, \mathcal{S}) = \frac{1}{k}\log_2(\mathrm{Card}(\mathrm{Adm}_k(G)))$, which implies the result.

The proof is done by induction. As a base case, we consider the set of $\mathrm{Adm}_2(G)$ of length 2 admissible sequences of states and the cover $\mathcal{V} = \mathcal{S} \vee [G]^{-1}(\mathcal{S})$ of $D = [G]^{-1}(\mathbf{X})$. An element of $\mathcal{V}$ is an intersection $\mathbf{X} \times \{s_1\} \cap [G]^{-1}(\mathbf{X} \times \{s_2\})$, and is therefore equal to $C[s_1, s_2] \times \{s_1\}$ where $C[s_1, s_2] \subset \mathbf{X}$ is the set $\{x \in \mathbf{X} \mid [G](x, s_1) \in \mathbf{X} \times \{s_2\}\}$. This set is empty if and only if the sequence $s_1 s_2$ belongs to $\mathrm{Adm}_2(G)$. Moreover, given another sequence of states $s_1' s_2'$, the sets $C[s_1, s_2]$ and $C[s_1, s_2]$ are disjoint. Hence a set $C[s_1, s_2]$ is *removable from the cover* $\mathcal{V}$ if and only if $s_1 s_2$ is not admissible. This proves the case $k = 2$.

The step for the induction is similar. One considers the partition $\mathcal{S}_k = \bigvee_{i=0}^{-(k-1)} [G]^i(\mathcal{S})$ as $\mathcal{S}_{k-1} \vee [G]^{-(k-1)}(\mathcal{S})$. By the same argument, one shows elements of $\mathcal{S}_{k-1} \vee [G]^{-(k-1)}(\mathcal{S})$ are of the form $C[\mathbf{s} = (s_0 s_1 \dots s_{k-1}), s_k] \times \{s_1\}$ where $C[\mathbf{s}, s_k]$ is the set $\{x \in \mathbf{X} \mid \forall i = 2, \dots, k, [G]^{i-1}(x, s_1) \in \mathbf{X} \times \{s_i\}\}$. Again, these sets $C[\mathbf{s}, s_k]$ are pairwise disjoint and empty if and only if the sequence $s_0 s_1 \dots s_{k-1}, s_k$ is not admissible. □

Proof of Lemma 3. Let us fix some $\epsilon > 0$. Notice that if we let $H_k(G, \mathcal{U}) = H^0(\mathcal{U} \vee [G]^{-1}(\mathcal{U}) \vee \dots \vee [G]^{-(k-1)}(\mathcal{U})))$, the sequence $H_k(U)$ satisfies $H_{k+l}(\mathcal{U}) \leqslant H_k(\mathcal{U}) + H_l(\mathcal{U})$. By Fekete's lemma on subadditive sequences, this implies that $\lim_{k\to\infty} H_k/k$ exists and is equal to $\inf_k H_k/k$. Thus $h([G], \mathcal{U}) = \inf_k H_k/k$.

Now, the entropy $h([G])$ is defined as $\sup_{\mathcal{U}} \lim_{k\to\infty} H_k(\mathcal{U})/k$. This then rewrites as $\sup_{\mathcal{U}} \inf_k H_k(\mathcal{U})/k$. We can conclude that $h([G]) \geqslant \inf_k H_k(\mathcal{U})/k$ for all finite open cover $\mathcal{U}$.

Since $\inf_k H_k(\mathcal{U})/k$ is the limit of the sequence H_k/k, there exists an integer N such that for all $k \geqslant N$ the following inequality holds: $|H_k(\mathcal{U})/k - \inf_k H_k(\mathcal{U})/k| < \epsilon$, which rewrites as $H_k(\mathcal{U})/k - \inf_k H_k(\mathcal{U})/k < \epsilon$. From this we deduce $H_k(\mathcal{U})/k < h([G]) + \epsilon$, hence $H^k(G, \mathcal{U}) < h([G]) + \epsilon$ since $H^k(G, \mathcal{U}) = H_k(G, \mathcal{U})$. □

Proof of Proposition 4. For a fixed sequence $\vec{e}$, the number of elements $n^\pi_{\vec{e}}$ of length m in $\mathrm{coT}_k(T)$ is bounded by the number of elements in the m-th cell decomposition of T, and is therefore bounded by $g(m) = 2^{m.h_0([T])}$ by 3. The number of sequences $\vec{e}$ is bounded by $\mathrm{Card}(\mathrm{Seq}_k(E))$

and therefore the size of $\mathrm{coT}_k(T)$ is thus bounded by $\mathrm{Card}(\mathrm{Seq}_k(E)).2^{(k+1).h_0([T])}$. □

Proof of Lemma 5. If G computes the membership problem for W in k steps, it means W can be described as the union of the subspaces corresponding to the nodes $n^\pi_{\mathbf{e}}$ with π of length k in $\mathrm{coT}_k(T)$. Now, each such subspace is an algebraic set, as it can be described by a set of polynomials as follows.

Finally let us note that, as in Mulmuley's work [4], since in our model the memory pointers are allowed to depend only on the nonnumeric parameters, indirect memory instructions can be treated as standard – direct – memory instructions. In other words, whenever an instruction involving a memory pointer is encountered during the course of execution, the value of the pointer is completely determined by nonnumerical data, and the index of the involved registers is completely determined, independently of the numerical inputs.

We define a system of equations $(E^{\mathbf{e}}_i)_i$ for each node $n^\pi_{\mathbf{e}}$ of the entropic co-tree $\mathrm{coT}_k(T)$. We explicit the construction for the case $p = 1$, i.e. for the AMC $\mathrm{CREW}^1(\alpha_{\mathbf{R}\mathrm{full}}) = \alpha_{\mathbf{R}\mathrm{full}}$; the case for arbitrary p is then dealt with by following the construction and introducing p equations at each step (one for each of the p instructions in $\alpha_{\mathbf{R}\mathrm{full}}$ corresponding to an element of $\mathrm{CREW}^p(\alpha_{\mathbf{R}\mathrm{full}})$). This is done inductively on the size of the path $\vec{e}$, keeping track of the last modifications of each register. I.e. we define both the system of equations $(E^{\mathbf{e}}_i)_i$ and a function $\mathfrak{h}(\mathbf{e}) : \mathbf{R}^\omega + \bot \to \omega$ (which is almost everywhere null)[10]. For an empty sequence, the system of equations is empty, and the function $\mathfrak{h}(\epsilon)$ is constant, equal to 0.

Suppose now that $\vec{e'} = (e_1, \ldots, e_m, e_m + 1)$, with $\vec{e} = (e_1, \ldots, e_m)$, and that one already computed $(E^{\mathbf{e}}_i)_{i \geqslant m}$ and the function $\mathfrak{h}(\mathbf{e})$. We now consider the edge e_{m+1} and let (r, r') be its realizer. We extend the system of equations $(E^{\mathbf{e}}_i)_{i \geqslant m}$ by a new equation E_{m+1} and define the function $\mathfrak{h}(\mathbf{e}')$ as follows:

- if $r = +_i(j,k)$, $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x) + 1$ if $x = i$, and $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x)$ otherwise; then E_{m+1} is $x_i^{\mathfrak{h}(\mathbf{e}')(i)} = x_j^{\mathfrak{h}(\mathbf{e}')(j)} + x_k^{\mathfrak{h}(\mathbf{e}')(k)}$;
- if $r = -_i(j,k)$, $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x) + 1$ if $x = i$, and $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x)$ otherwise; then E_{m+1} is $x_i^{\mathfrak{h}(\mathbf{e}')(i)} = x_j^{\mathfrak{h}(\mathbf{e}')(j)} - x_k^{\mathfrak{h}(\mathbf{e}')(k)}$;
- if $r = \times_i(j,k)$, $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x) + 1$ if $x = i$, and $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x)$ otherwise; then E_{m+1} is $x_i^{\mathfrak{h}(\mathbf{e}')(i)} = x_j^{\mathfrak{h}(\mathbf{e}')(j)} \times x_k^{\mathfrak{h}(\mathbf{e}')(k)}$;
- if $r = /_i(j,k)$, $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x) + 1$ if $x = i$, and $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x)$ otherwise; then E_{m+1} is $x_i^{\mathfrak{h}(\mathbf{e}')(i)} = x_j^{\mathfrak{h}(\mathbf{e}')(j)} / x_k^{\mathfrak{h}(\mathbf{e}')(k)}$;
- if $r = +^c_i(k)$, $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x) + 1$ if $x = i$, and $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x)$ otherwise; then E_{m+1} is $x_i^{\mathfrak{h}(\mathbf{e}')(i)} = c + x_k^{\mathfrak{h}(\mathbf{e}')(k)}$;
- if $r = -^c_i(k)$, $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x) + 1$ if $x = i$, and $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x)$ otherwise; then E_{m+1} is $x_i^{\mathfrak{h}(\mathbf{e}')(i)} = c - x_k^{\mathfrak{h}(\mathbf{e}')(k)}$;
- if $r = \times^c_i(k)$, $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x) + 1$ if $x = i$, and $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x)$ otherwise; then E_{m+1} is $x_i^{\mathfrak{h}(\mathbf{e}')(i)} = c \times x_k^{\mathfrak{h}(\mathbf{e}')(k)}$;
- if $r = /^c_i(k)$, $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x) + 1$ if $x = i$, and $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x)$ otherwise; then E_{m+1} is $x_i^{\mathfrak{h}(\mathbf{e}')(i)} = c / x_k^{\mathfrak{h}(\mathbf{e}')(k)}$;
- if $r = \sqrt[n]{i}(k)$, $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x) + 1$ if $x = i$, and $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x)$ otherwise; then E_{m+1} is $x_i^{\mathfrak{h}(\mathbf{e}')(i)} = \sqrt[n]{x_k^{\mathfrak{h}(\mathbf{e}')(k)}}$;
- if $r = \mathrm{Id}$, the source of the edge e_q is of the form $\{(x_1, \ldots, x_{n+\ell}) \in \mathbf{R}^{n+\ell} \mid P(x_k)\} \times \{i\}$ where P compares the variable x_k with 0:
  - if $P(x_k)$ is $x_k \neq 0$, $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x) + 1$ if $x = \bot$, and $\mathfrak{h}(\mathbf{e}')(x) = \mathfrak{h}(\mathbf{e})(x)$ otherwise then E_{m+1} is $x_\bot^{\mathfrak{h}(\mathbf{e}')(\bot)} x_k^{\mathfrak{h}(\mathbf{e}')(k)} - 1 = 0$;
  - otherwise we set $\mathfrak{h}(\mathbf{e}') = \mathfrak{h}(\mathbf{e})$ and E_{m+1} equal to P.

We now consider the system of equations $(E_i)_{i=1}^k$ defined from the path $\mathbf{e}$ of length k corresponding to a node $n^\pi_{\mathbf{e}}$ of the k-th entropic co-tree of G. This system consists in k equations of degree at most $\max(2, \sqrt[\partial]{G})$ and containing at most $k + n$ variables, counting the variables $x_1^0, \ldots, x_n^0$ corresponding to the initial registers, and adding at most k additional variables since an edge of $\vec{e}$ introduces at most one fresh variable. Since the number of vertices $n^\pi_{\mathbf{e}}$ is bounded by $\mathrm{Card}(\mathrm{Seq}_k(E)).2^{k.h_0([G])}$ by 4, we obtained the stated result in the case $p = 1$.

The case for arbitrary p is then deduced by noticing that each step in the induction would introduce at most p new equations and p new variables. The resulting system thus contains at most pk equations of degree at most $\max(2, \sqrt[\partial]{G})$ and containing at most $pk + n$ variables. □

Proof of Theorem 8. By Lemma 5 (using the fact that $p = 1$ and $\sqrt[\partial]{G} = 2$), the problem W decided by G in k steps is described by at most $\mathrm{Card}(\mathrm{Seq}_k(E)).2^{k.h_0([G])}$ systems of k equations of degree 2 involving at most $k + n$ variables. Applying Theorem 7, we deduce that each such system of in·equations (of k equations of degree 2 in $\mathbf{R}^{k+n}$) describes a semi-algebraic variety S such that $\beta_0(S) < 2.3^{(n+k)+k-1}$. This begin true for each of the $\mathrm{Card}(\mathrm{Seq}_k(E)).2^{k.h_0([G])}$ cells, we have that $\beta_0(W) < \mathrm{Card}(\mathrm{Seq}_k(E)).2^{k.h_0([G])+1}3^{2k+n-1}$. □

Proof of Corollary 1. Let T be an algebraic computation tree computing the membership problem for W, and consider the computational treeing $[T]$. Let d be the height of T; by definition of $[T]$ the membership problem for W is computed in exactly d steps. Thus, by the previous theorem, W has at most $\mathrm{Card}(\mathrm{Seq}_k(E)).2^{d.h_0([T])+1}3^{2d+n-1}$ connected components. As the interpretation of an algebraic computational tree, $h_0([T])$ is at most equal to 2, and $\mathrm{Card}(\mathrm{Seq}_k(E))$ is bounded by 2^d. Hence $N \leqslant 2^d.2^{2d+1}3^{n-1}3^{2d}$, i.e. $d = \Omega(\log N)$. □

[10] The use of $\bot$ is to allow for the creation of fresh variables not related to a register.

Proof of Proposition 5. To compute $p//q$, where $p, q \in \mathbf{Z}$, consider the real-valued machine such that the i^{th} processor computes $x = p/q - i$ and if $0 < x \leqslant 1$, writes i in the shared memory. This operation generalizes euclidian division and is computed in constant time. Moreover, this only uses a number of processor linear in the bitsize of the inputs if they are integers. □