# Revisiting Mulmuley: Simple Proof That Maxflow Is Not in the Algebraic Version of $NC$

Ulysse Léchine
LIPN, Paris, France
IRIF, Paris, France

## Abstract

We give an alternate and simpler proof of the fact that PRAM without bit operations (shortened to iPRAM for integer PRAM) as considered in paper [2] cannot solve the maxflow problem. To do so we consider the model of PRAM working over real number (rPRAM) which is at least as expressive as the iPRAM models when considering integer inputs. We then show that the rPRAM model is as expressive as the algebraic version of NC : algebraic circuits of fan-in 2 and of polylog depth noted $\mathsf{NC}^{\mathsf{alg}}$. We go on to show limitations of the $\mathsf{NC}^{\mathsf{alg}}$ model using basic facts from real analysis : those circuits compute low degree piece wise polynomials. Then, using known results we show that the maxflow function is not a low-degree piece-wise polynomial. Finally we argue that $\mathsf{NC}^{\mathsf{alg}}$ is actually a really limited class which limits our hope of extending our results to the boolean version of $NC$.

**2012 ACM Subject Classification** Theory of computation → Algebraic complexity theory

**Keywords and phrases** Algebraic complexity, P vs NC, algebraic NC, GCT program

**Digital Object Identifier** 10.4230/LIPIcs.FSTTCS.2023.33

## 1 Introduction

### 1.1 Previous work

An important question in complexity theory is determining if $P = NC$, where $P$ is the set of languages solvable by a polynomial time Turing machine and $NC$ the set of languages solvable by uniform family of poly-logarithmic depth, polynomial size circuit. It is thought that $P \neq NC$ but no one has come close to proving it. The maxflow problem is a problem in $P$ where the inputs are directed graphs whose edges are labelled with integers called capacities, a specified node $s$ called the source, a specified node $t$ called the sink, and an integer threshold $k$, the input is in the language if a flow greater than $k$ units may flow from $s$ to $t$. The maxflow problem is known to be $P$ complete with regards to $NC$ reductions, meaning that membership of the maxflow problem to $NC$ is equivalent to $P = NC$. In his celebrated result [2], Mulmuley defines a computation model called integer PRAMs (iPRAM) without bit operation which is akin to the PRAM model (communicating processors working in parallel, see [2] for a formal definition) except that inputs are integers (instead of bits) and the usual bit operations are replaced by multiplication, addition and comparison (all of those take $\Omega(1)$ time regardless of the size of the integers). He then proves that the maxflow problem cannot be computed by iPRAM with polynomially many processors running in polylogarithmic time. The model of iPRAM with polynomially many processors running in polylogarithmic time is basically an algebraic version of $NC$, which we denote by $\mathsf{NC}^{\mathsf{alg}}$, where the inputs are now integers and the gates multiplication, addition and comparison, and the fan-in of each gate is at most 2.[1] Moreover it was also known that the maxflow problem does have algebraic algorithm (working only using multiplication and addition)

[1] making the fan-in to be arbitrary wouldn't change the model, we can replace a large fan-in gate with a low height tree of gates

43rd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2023).
Editors: Patricia Bouyer and Srikanth Srinivasan; Article No. 33; pp. 33:1–33:12
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

running in polynomial time, so in short Mulmuley proved that the maxflow problem is not in the algebraic version of $NC$ yet it is in the algebraic version of $P$. So in the algebraic world $NC \neq P$.

### Size of inputs

For now we have very loosely spoken of “polynomial time”, “poly-logarithmic depth”... it is to be understood that we mean “polynomial time in the size of the input”, but what is the size of an input? It may seem trivial, but we insist upon it to clarify any future misconceptions. In the bit world (where languages are subsets of $\{0,1\}^*$, and inputs bitstrings), the size of the input is its number of bits. In the algebraic world (where languages can be thought of as subsets of $\mathbb{N}^*$ and inputs strings of numbers) the size of an input is its number of numbers. For instance let us consider a specific instance of a maxflow problem, say a weighted directed graph $G$ with 10 nodes given via its matrix representation, two integers to specify which node is the source and which is the sink, and a integer threshold $k$. In the algebraic world the size of this instance is $10^2 + 2 + 1 = 103$. In the bit world we do not have enough information to conclude because we do not know the bitlength of the capactities. This is just to shed light on the difference between the two models, in the algebraic one only the number of integers count, no matter their bitlength. Now when showing impossibility results, for instance that maxflow is not in the algebraic version of $NC$, we will on top of it prove that the impossibility still stand even if we restricts integers to be of bitlength polynomial in the size of the input (i.e. polynomial in the number of numbers). This is interesting because it “levels the playing field” between the algebraic world and the bit world, if the algebraic world comes short of solving the maxflow problem it will not be because of inputs of very large bitlengths[2].

## 1.2 Our result

In this paper we show that maxflow is not in the algebraic version of NC $\mathsf{NC}^{\mathsf{alg}}$ (subsequently also not in iPRAM). We detail things as follows: our technique for showing the result. Contributions and differences between our paper and Mulmuley's. Discussion on the meaning of this result.

## 1.2.1 Technique

### 1.2.1.1 The different PRAM models and algebraic circuits are equivalent

Firstly we consider (in order to use real analysis) real PRAM noted rPRAM (introduced in [3]). rPRAM are akin to iPRAM except inputs are real numbers instead of integers, on which we can perform addition, multiplication, *division* and comparison[3]. We argue efficiently that rPRAM are at least as potent as iPRAM when we restrict the inputs to integers (iPRAM are not defined otherwise). Then we consider the non-uniform[4] family of algebraic circuits

[2] This is how size was handled in [2] hence our choice. Another way of defining the size of inputs in the algebraic world, perhaps more intuitive or familiar to some readers, is to include the bitlength of the inputs in the size. As a side note changing size like this would strengthen the class $\mathsf{NC}^{\mathsf{alg}}$ and our proofs would still work.

[3] the division is probably superfluous when considering the decision version of a problem (we might be able to simulate it using multiplication and addition) but it makes our model more potent in the general case. In any case if the goal is only to prove the original result of Mulmuley, one may (on second reading) disregard the division operation in the rest of the paper, it would make proofs even simpler

[4] uniformity never plays a role so we may as well not restrict our circuits

whose inputs are (multiple) real numbers, whose gates are multiplication, addition, division, comparison (all of fan-in 2), whose depth is poly-logarithmic in the size of the input and whose largeness is polynomial in the size of the input. We call this class $\mathsf{NC}^{\mathsf{alg}}$, it is to be thought of as the algebraic equivalent of $NC$. We then give an argument to show that under a reasonable time slowdown and largeness blow up, $\mathsf{NC}^{\mathsf{alg}}$ can simulate rPRAM with polynomially many processors running in poly-logarithmic time. Now we can focus our attention on proving lowerbounds for $\mathsf{NC}^{\mathsf{alg}}$ which is much simpler than with rPRAM or iPRAM (but any lower bounds for $\mathsf{NC}^{\mathsf{alg}}$ entail ones for rPRAM and iPRAM). Considering $\mathsf{NC}^{\mathsf{alg}}$ instead of iPRAM is one of the key ingredient to the simplicity of our proof, the other one being restricting our attention to algebraic circuit over one variable.

#### 1.2.1.2 $\mathsf{NC}^{\mathsf{alg}}$ are really limited

First we prove by induction that a family of algebraic circuits whose input is only one number are really simple functions: they are piece-wise polynomials [5] and each of those polynomials is of small degree (we bound the degree and the number of different polynomials by the depth of the circuit, see 2). So for instance any function $f : \mathbb{R} \mapsto \mathbb{R}$ with too many slope changes cannot be computed by an algebraic circuit of small depth.

#### 1.2.1.3 Problem

Then we explain how one can use this result over one input algebraic circuits to limit the complexity of algebraic circuits over multiple inputs: let us say we want to prove that a family of functions $g_n : \mathbb{R}^n \mapsto \mathbb{R}$ is not computable in $\mathsf{NC}^{\mathsf{alg}}$ (i.e. with depth polylog$(n)$) then one just needs to find a family of functions $f_n : \mathbb{R} \mapsto \mathbb{R}^n$ computable with a circuit of depth polylog$(n)$ such that $g_n \circ f_n$ (which is now a function from $\mathbb{R}$ to $\mathbb{R}$ so we can use our theorem) is complex enough not to be computed by algebraic circuits of depth polylog$(n)$. Since $f_n$ is computable by such circuits then it must be $g_n$ which is not. Now for the maxflow problem specifically: call $g_n : \mathbb{R}^{n^2} \mapsto \mathbb{R}$ the function which takes a weighted directed graph given as its matrix representation and returns its maximum flow (the source is node 1 and the sink node $n$). One simply needs to find a simple function $f_n : \mathbb{R} \mapsto \mathbb{R}^n$ such that $g_n \circ f_n$ is too complex to be computed by circuits of depth polylog$(n)$. Such a family of functions $f_n$ has been know for a long time ( [1], Mulmuley even gives his own in [2]), and they are actually very simple since they are affine functions, so definitely computable by algebraic circuits even of depth 1. Then we simply conclude that $g_n$ in not computable by circuits of $\mathsf{NC}^{\mathsf{alg}}$.

### 1.2.2 Contributions and differences

The contributions of this paper are twofold. Firstly the simplicity of the proof when compared with Mulmuley's. We use very basic (univariate) polynomial analysis to limit the complexity of algebraic circuits, and then we use a known result about the parametric complexity of the maxflow problem to immediately conclude. Thanks to our proof this quite famous result may be proven to researchers or student in an hour or so. Secondly, because of the simplicity of the proof we feel it is much easier to understand the true limitations of $\mathsf{NC}^{\mathsf{alg}}$ (therefore iPRAM) and reveal that they are basically humongous and why even basic functions cannot be computed in this model of computation. Mulmuley knew about these limitations but we are more pessimistic than him. Additionally our technique exposes the fundamental and

[5] actually polynomial fractions but this is an unimportant detail

non-trivial link between computing on inputs of large bitlength and computing functions on many inputs of smaller bitlengths : we show that functions over one input cannot be computed even if we restrict the inputs to $x \in [0, 2^{n^d}]$ (where $d$ is an integer) and we use it to conclude that some function $g$ over $e \in \mathbb{N}$ inputs cannot be computed even if we restrict its domain to $[0, 2^n]^e$. The technique that Mulmuley uses is based on much more complicated mathematics and with a model (iPRAM) on which proofs are clumsier that on circuits. He basically uses the same plan and gives the same argument for the non-membership of maxflow to iPRAM: iPRAM may only compute "simple" multivariate-polynomial-like functions but using parametrization we can show that maxflow is not a simple function. To be clear, most of our mathematical arguments may be - from a logically formal standpoint - already contained in Mulmuley's but we do not think it makes this paper redundant at all. Those arguments were not readily apparent in Mulmuley. It is only after careful examination that one realizes that our mathematical arguments are embedded in the ones of Mulmuley's. Our proof, because we limit the expressiveness of univariate circuits to polynomial of low degree, very clearly exposes the shotcomings of the expressivity of the algebraic model. Those shortcomings were was not as apparent in Mulmuley's paper, because he deals with harder concepts (more or less multivariate polynomials): limitations on such objects do not seem as restrictive, and it still seems that we can compute "interesting" functions in iPRAM, so it gives a false sense of how expressive the class iPRAM is. On the other hand the approach used by Mulmuley, although it lacks in simplicity, may allow one to add operations on top of addition and multiplication, for instance he claims his results are still valid if one allows computing the parity of an integer. It is not immediately clear that our proof techniques can be salvaged if we add such an operation (adding the parity function allows us to create a polynomial with exponentially many pieces). Also Mulmuley shows that randomness still does not allow us to compute the maxflow problem. We do not consider randomness in this paper, our educated guess is that our technique can be fitted to include randomness. One last hiccup is that for what seems could be purely technical reasons we were not able to show that the *decision* version of the maxflow problem cannot be computed in $\mathsf{NC}^{\mathsf{alg}}$. We only show that the search version (taking a graph and outputting its maximal flow) is not in $\mathsf{NC}^{\mathsf{alg}}$. At the end of the paper we give promising leads to solve this slight issue.

### 1.2.3 Discussion

As explained before we believe our results shed light on the lack of potency of the models considered by Mulmuley and consequently ours. We are still far from proving that $NC \neq P$. To cement that idea we show that trivial problem for bit complexity class, like checking the parity of a bitstring (its last bit is 0) is obviously in $NC$ (its even in $AC^0$), but it is not in $\mathsf{NC}^{\mathsf{alg}}$. This was known of Mulmuley but he somewhat disregards it, we argue later in the paper that it should not be disregarded. We, very humbly, think it somewhat dampens the hope of using such techniques to prove $P \neq NC$, or at the very least that closing the gap using such techniques will be quite involved. Secondly we warn the reader: although the maxflow is special in the bit complexity class $P$, as it is complete, it is as far as we can tell not special in the algebraic world: we could not find general reductions from problems in algebraic-$P$ to the algebraic maxflow problem. The maxflow problem is only special in the sense as it has an intuitive translation from the bit world to the algebraic world: every capacity gets mapped to one number. If we had taken another problem like primality testing for instance, the translation would not make much sense. Indeed in primality testing we have a number $x$ and must determine if it is prime, its intuitive translation in the algebraic world would be similar: we have an integer $x$ and must determine if it is prime. The issue is that in the algebraic world our input size would now be 1 (we have 1 number) and so circuits should now be of constant size.

All of that being said it is still interesting in its own right to know that the maxflow problem cannot be solved in poly-logarithmic time by polynomially many processors. Indeed algorithms used to solve the maxflow problem are, to our knowledge, all algebraic, they do not use the bitwise representation of numbers, only their absolute value. So it gives credible reason to believe $P \neq NC$

## 2 Equivalence between PRAMS and algebraic circuits

We will start with definitions of classes iPRAM, rPRAM and $\mathsf{NC}^{\mathsf{alg}}$. Our definition of iPRAM is taken almost verbatim from Mulmuley's paper [2]. Our definition of rPRAM is taken from [3]

▶ **Definition 1** (iPRAM). *It is like the usual PRAM model, the main difference being that its input are integers (instead of bits) and it does not provide instructions for any bit operations such as $\wedge$, $\vee$, or extract-bit. The model provides usual arithmetic ($+$, $-$, $\times$), comparison ($=$, $\leq$, $<$), store, indirect reference, and branch operations. Each memory location can hold an integer; a rational number is represented by a pair of integers – its numerator and denominator – both of which can be accessed by the processors separately. The processors communicate as in the usual PRAM model. The model does not provide instructions for truncation or integer division with rounding. The lower bounds are proved in the conservative unit-cost model in which each operation is assigned unit cost regardless of the bitlengths of the operands.*

In Mulmuley's setting indirect addressing is not allowed to depend in the input, for instance we may not use a capacity as an address. There need not be such restrictions in our model.

▶ **Definition 2** (rPRAM). *It is like the* iPRAM *model except the inputs are real numbers and we have an additional / (divide) gate. Furthermore each value is now stored in a memory location represented by a real number, i.e. we have an instruction store value $x$ in cell $y$ where both $x$ and $y$ are real numbers. We may also retrieve the content of a memory cell indexed by $y$ a real number.*

Notice that even though our memory cells are labelled by real numbers, at most polynomially many labels can be instantiated over the course of a computation using polynomially many processors and running in poly-logarithmic time.

In order to meaningfully compare the two models (iPRAM and rPRAM) we may restrict our view to integer inputs, in this setting it is clear enough that any computation made by an iPRAM can be made by an rPRAM.

▶ **Definition 3** (Algebraic circuit). *An algebraic circuit is a circuit with gates $+, \times, -, /, =, <, \leq$ of fan-in 2, and constant gates $n$ for all $n \in \mathbb{N}$. Its inputs are real numbers. The comparison gates $=, <, \leq$ return $1$ when true, $0$ otherwise. All the other gates are interpreted in the obvious way.*

▶ **Definition 4** (log-depth algebraic circuits). *$C_n$ is a sequence of* $\mathsf{NC}^{\mathsf{alg}}$ *( polylog-depth algebraic circuits) if each $C_n$ is an algebraic circuit working over $n$ inputs, the size of each circuits grows polynomial in $n$ and the depth polylogarithmically in $n$.*

We restrict the largeness of log-depth algebraic circuits to polynomials. If unrestricted they could be bigger, indeed even if we work with fan in 2 a circuit of depth $\log^d(n)$ could be of largeness as big as $2^{\log^d(n)}$ which is bigger than any polynomial for $d > 1$. Mulmuley considers such circuits in his paper, for simplicity we do not although our results would still stand. We now sketch a proof that $\mathsf{NC}^{\mathsf{alg}}$ (circuits) and $rPRAM$ (prams) with polynomially many processors and of running time polylogarithmic define the same class of functions.

▶ **Theorem 1.** $\mathsf{NC}^{\mathsf{alg}}$ *(circuits) and $rPRAM$ (prams) with polynomially many processors and of running time polylogarithmic define the same class of functions.*

**Proof.** It is not hard to see that one can simulate a circuit using a real PRAM. Just have one processor per gate.

The other direction is not so trivial, here is a sketch of the proof. To handle instructions not related to memory manipulation we may use techniques used in the traditional Turing-machine-to-circuit conversions, the tricky part is taking care of the memory addressing using real numbers. Proof of this fact when considering $NC$ and PRAMs (in the boolean setting) can be found in introductory computer science books. Let us consider $p$ processors in parallel running for $k$ steps. Our circuit will consist of $k$ layers $(l_i)_{i=1}^k$, each layers contains $pk$ gates representing the memory state of each processor (each processor modifies at most $k$ internal addresses over the whole run), plus an additional number of gates representing the state of the shared memory. In order to deal with the processor accessing and modifying the shared memory we need to be careful, indeed the index of the cell accessed by a processor is unknown across a range of infinitely many indices, to deal with this issue we use a trick known as dynamic addressing, each memory cell will be represented by a pair of gates one containing the actual content and one containing the index of the cell. Only $kp$ memory cells can be used at most by the processors so we add to each layer a $2kp$ gates representing the state of the share memory (index and content). Moreover all operations on the memory (like checking if a certain memory cell given by its address is empty) can be simulated using a overhead of $O(\log(kp))$. In total our circuit is of size polynomial in $pk$ and of depth $O(k\log(pk))^2$ (the square comes from the Turing machine to circuit conversion). ◀

## 3 Limitations of log-depth algebraic circuits

In this section we give a direct proof that log-depth algebraic circuits over one variable compute piece-wise polynomial fractions of low degree and few pieces (namely less than $2^n$ if the depth is polylog$(n)$). This is detailed in main theorem 2.

▶ **Definition 5** (Interval). *A set $I$ over $\mathbb{R}$ is an interval if there exists $a, b \in (\mathbb{R} \cup \{-\infty; +\infty\})^2$ two real numbers such that $I = [a,b]$ or $I = ]a,b]$ or $I = [a,b[$ or $I = ]a,b[$.*[6]

▶ **Definition 6** (Comparison of intervals). *Let $I$ and $J$ be two non-empty, non-intersecting intervals over $\mathbb{R}$. We write $I < J$ if $\forall x \in I, \forall y \in J, x < y$. Let us have a collection of $r$ intervals $(I_i)_{i=0}^r$, we write $I_1 < I_2 < \ldots < I_r$ to indicate that they are given in order and they are all non-empty and non-intersecting.*

▶ **Definition 7** (Interval cut and pieces). *Given a family $(I_i)_{1\leq i\leq r}$ of $r$ intervals, $I_1 < I_2 < \ldots < I_r$ we say it is an interval cut of size $r$ if $I_1, I_2, \ldots, I_r$ partitions $\mathbb{R}$. Each $I_i$ is called a piece.*

In the following we may use as a shorthand the notation $I$ for a family of $r$ intervals $(I_i)_{1\leq i\leq r}$.

[6] since some reader may no be familiar with these notations: $]a,b]$ is equal to $[a,b] \smallsetminus a$

▶ **Definition 8** (Intertwining interval cuts). *Given an interval cut of size n $I_1 < I_2 < \ldots < I_n$ and an interval cut of size m $J_1 < J_2 < \ldots < J_m$. There is at least one interval cut of k-pieces $K_1, \ldots, K_k$ such that $\forall h \in [k], \exists i, j, K_h \subset I_i \wedge K_h \subset J_j$, if k is minimal then we call $K_1, \ldots, K_k$ an intertwining of I and J.*

Note in the previous definition that when $k$ is minimal it is smaller than $n + m$ (this is not hard to see). As an example of what is an intertwining consider the family $I = (]-\infty; 0], ]0; +\infty[)$ and $J = (]-\infty; 1], ]1; +\infty[)$, then one possible value of an intertwining is $(]-\infty; 0], ]0; 1], ]1; +\infty[)$.

▶ **Definition 9** (Piece-wise polynomial fraction). *A function f over $D \subset \mathbb{R}$ is said to be a piece-wise polynomial fraction if there exists $n \in \mathbb{N} \setminus \{0\}$ and an interval cut $I_1 < I_2 < \ldots < I_n$ and n polynomial fractions $f_1, f_2, \ldots, f_n$ such that $\forall i \in [\![1; n]\!], \forall x \in I_i \cap D, f(x) = f_i(x)$.* [7]

*Given such an interval cut $I_1 < I_2 < \ldots < I_n$ we may say f is a piece-wise polynomial fraction over I. The number of pieces of a piece-wise polynomial fraction is the smallest number n such that there exist $I_1 < I_2 < \ldots < I_n$ as described above. Let us write c the function which takes a piece-wise polynomial fraction f and return its number of pieces (we extend c to also take circuits as inputs when they compute piece-wise polynomial fractions).*

For clarification the poles of a polynomial fraction do not add new pieces, for instance the function $f(x) = \frac{1}{x^2-1}$ defined for all $x \in \mathbb{R} \setminus \{-1; 1\}$ is a one piece polynomial fraction. If one wanted to have functions defined over the whole of $\mathbb{R}$ it would be no issue as whenever performing a division we could add a comparison gate in the circuit to verify that we are not dividing by 0, and if so return an arbitrary value.

▶ **Definition 10** (Augmented degree of a polynomial fraction). *Let $F = \frac{P}{Q}$ be a polynomial fraction over $\mathbb{R}$, meaning that P and Q are coprime polynomials over $\mathbb{R}$, then we define its augmented degree noted $d(F)$ as $d(F) = d(P) + d(Q) + 1$ (where $d(P)$ and $d(Q)$ denote the usual degree of polynomials)*

▶ **Definition 11** (Augmented degree of a piece-wise polynomial fraction). *Let f be a piece-wise polynomial fraction over $\mathbb{R}$ with a minimal interval cut $I_1 < \ldots < I_n$ and $f_1, \ldots, f_n$ the associated polynomial fractions then we define $d(f) = max_{0 \leq i \leq n} d(f_i)$.*
*It is routine to check that this definition does not depend on the chosen interval cut.*

This is the main theorem of the paper on which we derive all impossibility results of section 4.

▶ **Theorem 2.** *A circuit P of depth k taking only one input, computes a function which is a piece-wise polynomial fraction. Let us call with $c(P)$ its number of pieces and $d(P)$ its augmented degree, then $d(P) \leq 2^k$ and $c(P) \leq 2^{\frac{k^2+3k}{2}}$*

**Proof.** We prove the result for each gate by induction on the depth of the gate, both that we do indeed compute a piece-wise polynomial fraction and that the bounds are correct. Let us denote $c_k$ the maximal number of pieces of a function computed by a gate at depth $k$. Let us denote $d_k$ the maximal augmented degree of a function computed by a gate at depth $k$. We are going to prove that $d_k \leq 2^k$ and $c_k \leq 2^{\frac{k^2+3k}{2}}$ by induction.

For depth $k = 1$ the function computed is either $f = (x \mapsto x)$ or $g = (x \mapsto 1)$. Notice that $d(f) = 2$, $c(f) = 1$, $d(g) = 1$ and $c(g) = 1$. Therefore $d_1 \leq 2^1$ and $c_1 \leq 2^2$.

[7] $[\![1; n]\!]$ denotes the integers in the interval $[1; n]$

Going from $k$ to $k+1$: Let us consider a gate called $p$ at depth $k+1$. Our induction hypothesis is that $d_k \leq 2^k$ and $c_k \leq 2^{\frac{k(k+1)}{2}}$.

Multiplication gate: $p = p_1 * p_2$. Let $r = c(p_1)$ and $m = c(p_2)$ we consider the interval cuts of $p_1$ and $p_2$ which we denote $I_1 < \ldots < I_r$ and $J_1, < \ldots < J_m$, let us consider $K_1, \ldots, K_k$ an intertwining of $I$ and $J$ such that $k < r + m$, note that $p$ is a piece-wise polynomial fractions over $K$. Therefore $c(p) \leq c(p_1) + c(p_2) \leq 2 * c_k \Rightarrow c_(p) \leq 2 * c_k \leq 2^{\frac{(k+1)(k+2)}{2}}$. By reasoning over each piece of the interval cut $K$ we have that $d(p) \leq d(p_1) + d(p_2) \leq 2d_k$.

Addition and division gate work in a similar fashion.

Comparison gate: $p = (p_1 \leq p_2)$. Let us define $n = c(p_1)$ and $m = c(p_2)$ and let $A_0 \leq A_1 \leq \ldots \leq A_n$ be the pieces of $p_1$, $B_0 \leq B_1 \leq \ldots \leq B_m$ the pieces of $p_2$. Let $I_0 \leq I_1 \leq \ldots \leq I_r$ be an intertwining of $A$ and $B$. For all $i$, $p_1$ and $p_2$ are both polynomial fraction over $I_i$ and we can canonically write $p_{i,1} = \frac{q_{i,1}}{t_{i,1}}$ and $p_{i,2} = \frac{q_{i,2}}{t_{i,2}}$ . We want to count the number of pieces of $p$, to that end we focus on each $I_i$ one by one. Notice that over $I_i$ any new piece of $p$ contained in $I_i$ may only occur on places where $p_{i,1} - p_{i,2}$ changes sign, moreover $p_{i,1} = p_{i,2} \Leftrightarrow q_{i,1} * t_{i,2} - t_{i,1} * q_{i,1} = 0$. But the polynomial $q_{i,1} * t_{i,2} - t_{i,1} * q_{i,1}$ has at most $d(p_1) + d(p_2) - 1$ roots, therefore over $I_i$ we "add" at most $d(p_1) + d(p_2)$ pieces. Since we have at most $r \leq n + m = c(p_1) + c(p_2)$ pieces $I_i$, we have that $c_p$ is at most $(d(p_1) + d(p_2)) * (c(p_1) + c(p_2)) \leq 2d_k * 2c_k \leq 2 * 2^k * 2 * 2^{\frac{k^2+3k}{2}} \leq 2^{\frac{(k+1)^2+3(k+1)}{2}}$. And $d(p)$ is equal to $1 \leq 2^{k+1}$.

So no matter the last computation gate the induction hypothesis still holds at depth $k+1$ therefore $d_k \leq 2^{k+1}$ and $c_k \leq 2^{\frac{(k+1)^2+3(k+1)}{2}}$. This concludes the proof. ◀

## 4 Problems not in $\mathsf{NC}^{\mathsf{alg}}$

In this section using theorem 2 we will present different problems not in $\mathsf{NC}^{\mathsf{alg}}$, including the maxflow problem. As a warm up we will begin by showing that the the mod 2 function (or bit extraction as Mulmuley calls it in his paper) cannot be computed by algebraic circuits of small depth. But first let us properly define what we mean by computing a function.

In the beginning of this paper we have proved that $\mathsf{NC}^{\mathsf{alg}}$ is at least as expressive as iPRAM when only considering integer inputs. Therefore given a family of functions $f_n$ and a family of intervals $D_n$ such that $f_n : D_n \mapsto \mathbb{R}$ there are two ways to meaningfully analyze its computability in $\mathsf{NC}^{\mathsf{alg}}$. The first one is the obvious one: is there a sequence of circuits $C_n \in \mathsf{NC}^{\mathsf{alg}}$ such that $\forall \lambda \in D_n, C_n(\lambda) = f_n(\lambda)$. The second one focuses only on integers inputs: is there a sequence of circuits $C_n \in \mathsf{NC}^{\mathsf{alg}}$ such that $\forall k \in \mathbb{N} \bigcap D_n, C_n(k) = f_n(k)$. If a sequence of function cannot be computed on the integers by circuits of $\mathsf{NC}^{\mathsf{alg}}$ then it cannot be computed by iPRAM and it surely cannot be computed on the whole of $D_n$. When tackling problems we will often consider both version of computability.

Here is the general scheme for proving that a decision problem (the function returns either 0 or 1 depending on membership in a specified language) is not computed by an algebraic circuit of low depth. Take a subset $L$ of $\mathbb{R}^n$ which will be the decision problem and restrict your view to $[0, 2^m]^n$, $n$ is to be thought as the number of inputs and $m$ as the bitlength of the integers we work with. Often in problems we only consider $n$ and take $m$ to be polynomial in $n$, it makes sense as in the bit world we may in polynomial time, only work with at most polynomial bit length integers. If $L$ is complex enough the hope is that there will a a lot of alternations inside the hypercube $[0, 2^m]^n$ between areas belonging to $L$ and areas not belonging to $L$. For instance if $n = 1$ (which is the only case studied by theorem 2), $L$ is complex if $[0, 2^m] \bigcap L$ is equal to the union of exponentially many disjoint intervals (exponential in $m$)[8]. Indeed in that case $L$ cannot be recognized (meaning returning 1 when the output is in $L$, 0 otherwise) by an algebraic circuit of depth polylogarithmic in $m$ because those cannot compute piece-wise polynomial fraction with exponentially many pieces. When $n$ is not equal to 1 we need not worry, one must simply find an integer $c$ and a one dimensional curve $f : [0; 2^{m^c}] \mapsto [0; 2^m]^n$ called parametrization, such that as $\lambda$ ranges across $[0; 2^{m^c}]$, $f$ goes through many alternations of $L$ and $L^C$ (the complementary of $L$), and such that $f$ is expressible using a circuit of $\mathsf{NC}^{\mathsf{alg}}$. Then no circuit of $\mathsf{NC}^{\mathsf{alg}}$ may recognize $L$, otherwise a circuit for $L$ composed with a circuit for $f$ would yield a piece-wise polynomial function with exponentially many pieces. In our proof for maxflow we use a linear parametrization, but in general it does not need be linear. This parametrization trick may not work in some instances, meaning there probably exists problems over $\mathbb{R}^n$ not in $\mathsf{NC}^{\mathsf{alg}}$ but such that no parametrization goes through exponentially many alternation of $L$ and $L^C$. For instance, in the decision version of the maxflow problem we fail to come up with a proper parametrization to show that it does not belong to $\mathsf{NC}^{\mathsf{alg}}$. Is it because such parametrization does not exist or because we could not find it ? We do not know. Our guess is that proving that the decision version of maxflow is not in $\mathsf{NC}^{\mathsf{alg}}$, is most likely provable, if not using parametrization, using tech.

## 4.1 Bit extraction

We start with proving that the mod 2 function cannot be computed by a circuit of small depth. Although the mod 2 function can be defined on $\mathbb{R}$ we only look at the mod 2 function on $[\![0; 2^m]\!] (= [0; 2^m] \cap \mathbb{N})$ and we prove that on integer inputs no circuit of depth polylog$(m)$ may always agree with the mod 2 function.

The next lemma states that no piece-wise polynomial fraction with few pieces or low augmented degree can agree with the mod 2 function on many consecutive integers.

▶ **Lemma 1.** *Let $f$ be a piece-wise polynomial fraction over $\mathbb{R}$, if $\forall k \in [\![0; N]\!], f(k) = k\%2$ then $d(f) * c(f) \geq \frac{N}{10}$*

**Proof.** Let $f$ be a piece-wise polynomial fraction as described in the theorem and call $c$ its number of pieces, $I_1, \dots, I_c$ its pieces. Assume $c$ is less than $N/10$ (otherwise the theorem is proved), there must be a piece $I_j$ containing at least $N/c$ integers of $[\![0; N]\!]$. We remind that over $I_j$ $f$ is a polynomial fraction which we note $\frac{P}{Q}$ , since $k \mapsto k\%2$ has $N/2c$ zeroes over $I_j$, so must $P$, $P$ also cannot be the zero function because $k \mapsto k\%2$ is equal to 1 for some integers in $I_j$, therefore P is of degree at least $N/2c$. Therefore $d(f) * c \geq (N/2c) * c \geq N/10$. ◀

The next theorem states that no circuit of polylog$(n)$ depth may compute the mod 2 function (we only consider the first $2^n$ integer inputs).

▶ **Theorem 3.** *Let $c$ and $d$ be two positive real numbers, for any function family $f_n$ computed by a circuit family of depth $c \log^d(n)$, there exists $N \in \mathbb{N}$ such that $\forall n > N, \exists k \in [\![0; 2^n]\!], f_n(k) \neq k\%2$.*

**Proof.** Let $(C_n)_n$ be a sequence of circuits of depth $k_n = O(c \log^d(n))$. Since $n \mapsto n\%2$ is a function with one input, we may consider $(C_n)_n$ to be a sequence of circuits with one input. For any $n$, the function computed by $C_n$ is a piece-wise polynomial fraction $f$. By theorem 2 we have that $c(f) \leq 2^{\frac{k_n^2+3k_n}{2}}$ and $d(f) \leq 2^{k_n}$ therefore $c(f)d(f) = o(2^n)$. Therefore for large enough $n$, using Lemma 1, $f$ cannot coincide with $n \mapsto n\%2$ for all integers in $[\![0; 2^n]\!]$. ◀

[8] actually any interval is the union of exponentially many disjoint ones, so more precisely we should require that $L$ is not equal to the union of less than exponentially many intervals

It was already known to Mulmuley that the problem of computing the last bit was a hard problem for $\mathsf{NC}^{\mathsf{alg}}$, from our understanding and if we recap his views informally : he dismisses it because there is no polynomial-depth circuit for this problem either[9], so it is not so interesting. Indeed he considers that the size of this problem is 1 : we are given 1 number and are asked to return its last bit. Ìn theorem 4 prove that a very similar set has no circuits in $\mathsf{NC}^{\mathsf{alg}}$ yet it has polynomial-depth circuits.

▶ **Theorem 4.** *The set*

$$\{(x_1, \ldots, x_n) \in \mathbb{N}^n ; x_n \% 2 = 0 \wedge x_n < 2^n\} .$$

*has polynomially bounded arithmetic circuits which agree with it over* $[[0; 2^n]]$*, but none in* $\mathsf{NC}^{\mathsf{alg}}$*.*

**Proof.** Let us call $A$ the set of the theorem. One can check that $x_n$ is less than $2^n$ and if so retrieve with a circuit of size linear in $n$ the last bit of $x_n$ (using standard dichotomic search). So set $A$ has polynomially bounded circuits.

On the other hand setting all $x_i$ but $x_n$ to 0 we already know that the set

$$\{(0, \ldots, 0, x_n) \in \mathbb{N}^n ; x_n \% 2 = 0 \wedge x_n < 2^n\} .$$

is not in $\mathsf{NC}^{\mathsf{alg}}$ by a proof similar to the one of theorem 3. Therefore the set A is not in $\mathsf{NC}^{\mathsf{alg}}$ either. ◀

Now we would like to take a moment to ponder over what the last theorem means with regards to the algebraic approach. When presented with Mulmuley's original result we were told this three-part story: One wants to prove that $P \neq NC$. Maxflow is $P$-complete, so we just need to prove that Maxflow is not in $NC$. It is too hard, but fortunately we can prove that the algebraic version of Maxflow is not in the algebraic version of $NC$. Surely this has drawn us closer to proving $P \neq NC$? Well it seems to us that the last theorem dampens that assumption, we have an example of a trivial problem for bit complexity classes which is not in $\mathsf{NC}^{\mathsf{alg}}$ (yet has polynomial size circuits !). So then arises the question why bother with maxflow when trivial problems already are not in $\mathsf{NC}^{\mathsf{alg}}$ ? In our opinion, this remark is valid in the context of assessing the computational power of $\mathsf{NC}^{\mathsf{alg}}$ and thus the extent to which impossibility results are meaningful. However there are still good reasons to care about the maxflow problem : first of all, even though maxflow is a P-complete problem in the bit world, there is no obvious guarantee (at least to us) that maxflow is complete in the algebraic world, so a proof that a trivial problem is not in $\mathsf{NC}^{\mathsf{alg}}$ could not entail anything regarding maxflow, secondly it is interesting in its own right to know if maxflow can be solved fast in parallel using only algebraic operations.

## 4.2 Bijections from $\mathbb{N}$ to $\mathbb{N}^2$

Before going on to the maxflow problem we focus on the problem of computing a bijection between $\mathbb{N}$ and $\mathbb{N}^2$. If our model could do this we could reduce problems over $n$ inputs to problems over 1 input by repeatedly applying the bijection (the same way in the bit world problems over $\mathbb{N}^2$ might as well be problems over $\mathbb{N}$ by associating a number to each pair of number). We do not directly prove that no bijection exists however we prove that we cannot inverse the usual Cantor pairing function. Using similar arguments one may be able to prove that no bijection between $\mathbb{N}$ and $\mathbb{N}^2$ may be computed by a circuit of low depth.

[9] we note : unless the bitlength is part of the size

▶ **Theorem 5.** *Let $f : \mathbb{N} \times \mathbb{N} \mapsto \mathbb{N}$ be the usual Cantor pairing function $f(n, m) = \frac{1}{2}(m + n)(m + n + 1) + m$. This function is bijective and its reciprocal $f^{-1}$ cannot be computed over $[\![0; 2^n]\!]$ by an algebraic circuit of depth polylogarithmic in $n$ .*

**Proof.** Consider the function $g(x, y) = (x < y)$. (it returns either 0 or 1) The function $g(f^{-1})$ has $2^{\frac{n}{2}}$ alternations between 0 and 1 over $[\![0; 2^n]\!]$ therefore it cannot be computed by a circuit of $\mathsf{NC}^{\mathsf{alg}}$ (by arguments used to prove theorem 1 and 3) . But $g$ can be computed by a circuit of $\mathsf{NC}^{\mathsf{alg}}$ ($<$ is a gate of $\mathsf{NC}^{\mathsf{alg}}$) therefore it must be $f^{-1}$ which cannot be computed by a circuit of $\mathsf{NC}^{\mathsf{alg}}$. ◀

## 4.3 Maxflow

Now we focus our attention to the maxflow problem. In the maxflow problem we are given a network (an directed graph) of edges with specified capacities, a threshold $k$, a two specified nodes $s$ and $t$, the question then asked is can a flow of size $k$ flow from $s$ to $t$.

▶ **Definition 12** (Maxflow problem). *An instance of an* ***algebraic maxflow problem*** *of size $n^2$ is a directed graph with $n$ nodes with each edge (as much as $n^2$) labelled with a real number called capacity. It is represented by its adjacency matrix representation. We say that a circuit $A$ solves the* ***algebraic maxflow problem*** *on size $n^2$ if on any instance of size $n^2$ it returns the maximal flow from node 1 to node $n$. We say that a circuit $A$ solves the* ***integer maxflow problem*** *on size $n^2$ if on any instance containing only integers it returns the maximal flow from node 1 to node $n$*

▶ **Definition 13** (Decision maxflow problem). *An instance of a* ***decision maxflow problem*** *of size $n^2$ is an undirected graph with $n$ nodes with each edge (as much as $n^2$) labelled with a real number called capacity and a threshold $k$. The graph is represented by its adjacency matrix representation. We say that a circuit $A$ solves the* ***decision maxlow problem*** *on size $n^2$ if on any instance of size $n^2$ it returns 1 if the maximal flow from node 1 to node $n$ is greater than $k$, 0 otherwise. We say that a circuit $A$ solves the* ***integer decision maxflow problem*** *on size $n^2$ if on any instance containing only integers it returns whether the maximal flow from node 1 to node $n$ is greater than $k$.*

This gives us 4 variants of the maxflow problem. We can of course further limit the complexity of these problem by only requiring algorithms solving them to work on an interval $D_n$ instead of the whole of $\mathbb{R}$. In his paper [2] Mulmuley proves that all 4 cannot be solved by circuits of $\mathsf{iPRAM}$ even if we only consider polynomial size bit lengths integer, i.e $D_n = [0, 2^{n^c}]$. We will now give a simple proof that the algebraic maxflow problem cannot be solved by circuits of $\mathsf{NC}^{\mathsf{alg}}$. We introduce for that the linear parametrization version of the maxflow problem. For a given size $n^2$, each capacity $c$ is now an affine function of parameter $\lambda$ where the coefficients $a$ and $b$ are integers of bitlength polynomial in $n$ ($c = a + b * \lambda$). Given $p, q \in \mathbb{Z}^2$ this defines for all $\lambda \in [p, q]$ an instance $G(\lambda)$ of the maxflow problem. Let $I$ be the function such that $I(G(\lambda))$ returns the maximal flow from node 1 to node $n$. It has been proven in [1] and [2] that there is a family $G_n(\lambda)$ of linearly parametrized graph with polynomial-bit-length parameters and $O(n^2)$ edges, such that $I(G_n(\lambda))$ has $O(2^n)$ breakpoints (slopes changes) over $[0, T]$ with $T = 2^{n+1}$.

▶ **Theorem 6.** *The algebraic maxflow problem cannot be computed by circuits of* $\mathsf{NC}^{\mathsf{alg}}$

**Proof.** Let us suppose there is a family of circuits $C_n \in \mathsf{NC}^{\mathsf{alg}}$ such that $C_n$ computes the algebraic maxflow problem for size $n^2$ graphs. Then $C_n$ must return the correct value on $G_n(\lambda)$ for all $\lambda$. Consider $B_n$ a circuit of depth 1 and $O(n^2)$ largeness taking $\lambda$ as input and outputting $G_n(\lambda)$ (represented via its adjacency matrix). Then the circuit $C_n(B_n(\lambda))$ is a circuit of $\mathsf{NC}^{\mathsf{alg}}$ with one input computing $I(G_n(\lambda))$ which is a linear function with $2^n$ breakpoints, but such a function cannot be computed by a circuit of $\mathsf{NC}^{\mathsf{alg}}$ by theorem 2, contradiction. ◀

▶ **Theorem 7.** *The integer algebraic maxflow problem cannot be computed by circuits of* $\mathsf{NC}^{\mathsf{alg}}$

**Proof.** We sketch an idea of proof only. Consider the construction of Mulmuley, notice that the breakpoints of $I(G_n(\lambda))$ are more than distance 2 away from each other, scale everything up by a factor of $2^n$ (one may create the integer $2^n$ in $log(n)$ depth circuits using repeated squaring) so that now the breakpoints are $2^n$ away from each other, this gives you a function $I'(\lambda)$ which is piece wise linear with each piece containing $2^n$ integers, argue that any piece-wise polynomial fraction which agrees with $I'$ must have a large number of pieces or a large augmented degree (like in Lemma 1), then conclude. ◀

The last remaining piece remaining is the decision version, but we come across a slight hiccup: solvability of the decision version of a problem means being able to compute its epigraph, indeed the decision version asks us to compute $(x, y) \mapsto (y < f(x))$. Intuitively speaking it would be very strange if we could in our algebraic setting compute the epigraph of function but not its graph. The solvability of the decision version should entail the solvability of the search version (up to a given precision) by way of dichotomic search: if we can compute $(x, y) \mapsto (y < f(x))$ then we can approximatively compute $x \mapsto f(x)$ in a similar time (for instance the time is multiplied by $\log(f(x))$ if we want to be precise up to the closest integer). Unfortunately in $\mathsf{NC}^{\mathsf{alg}}$ for functions outputting values exponential in $n$, we cannot reduce the complexity of the decision version to that of the search version via dichotomic search (we would need a depth $\log(2^n) = n$ but we only have $\text{polylog}(n)$ depth). Looking closer at the construction of Mulmuley in [2], the function $I(G_n(\lambda))$ does output for most $\lambda \in [0, 2^n]$ values bigger than $O(2^n)$.

We now give leads to tackle what looks to us to be only a technical issue. The most promising idea would be to find a family of function $f_n$ computable with low depth circuits and close enough to $I(G_n(\lambda))$, i.e. such that the difference $f_n(\lambda) - I(G_n(\lambda)) = O(n^d)$ for many inputs, we could then use dichotomic search to find the value of $I(G_n(\lambda))$. This might actually be possible because looking closer at Mulmuley's construction $I(G_n(\lambda))$ vaguely grows in $\lambda^{\log(n)}$ which is a polynomial in $\lambda$ (we can hardcode $\log(n)$ for every $n$), so an algebraic circuit of low depth may compute it. The technical part is properly analyzing the growth rate of the function $I(G_n(\lambda))$.

In case this fails another idea would be to fiddle around a bit with Mulmuley's construction to directly deal with the decision version. Yet another approach would be to analyse circuits of $\mathsf{NC}^{\mathsf{alg}}$ with 2 inputs but it would certainly complicate the mathematics. Finally there is hope that for non pathological functions $f$ we may prove in general that uncomputability of $f$ entails the uncomputability of $(y, x) \mapsto y < f(x)$.

## References

1 Patricia June Carstensen. *The Complexity of Some Problems in Parametric Linear and Combinatorial Programming.* PhD thesis, University of Michigan, USA, 1983. AAI8314249.

2 Ketan Mulmuley. Lower bounds in a parallel model without bit operations. *SIAM Journal on Computing*, 28(4):1460–1509, 1999. doi:10.1137/S0097539794282930.

3 Thomas Seiller, Luc Pellissier, and Ulysse Léchine. Unifying lower bounds for algebraic machines, semantically. quoicoubeh, November 2022. URL: https://hal.science/hal-01921942.