Electronic Colloquium on Computational Complexity, Revision 2 of Report No. 53 (2022)

On the Complexity of Algebraic Numbers, and the Bit-Complexity of Straight-Line Programs[4]

Eric Allender
Department of Computer Science
Rutgers University, USA
allender@cs.rutgers.edu

Nikhil Balaji
Indian Institute of Technology
Delhi, India
nbalaji@cse.iitd.ac.in

Samir Datta
Chennai Mathematical Institute &
UMI-ReLaX, India
sdatta@cmi.ac.in

Rameshwar Pratap
Indian Institute of Technology
Hyderabad, Telangana, India
rameshwar@cse.iith.ac.in

October 4, 2022

Abstract

We investigate the complexity of languages that correspond to algebraic real numbers, and we present improved upper bounds on the complexity of these languages. Our key technical contribution is the presentation of improved uniform TC^0 circuits for division, matrix powering, and related problems, where the improvement is in terms of "majority depth" (initially studied by Maciel and Thérien). As a corollary, we obtain improved bounds on the complexity of certain problems involving arithmetic circuits, which are known to lie in the counting hierarchy, and we answer a question posed by Yap.

1 Introduction

In this article, we consider formal languages that are subsets of $\{0,1\}^* = \{s_0, s_1, \ldots\}$ (where $s_0 = \lambda$ (the empty string), $s_1 = 0, s_2 = 1, s_3 = 00$ etc. in the standard order on $\{0,1\}^*$, where short strings precede long strings, and strings of the same length are placed in lexicographic order. Any such language A can be identified with its characteristic sequence $\chi_A = b_0 b_1 b_2 \ldots$

[1]Preliminary versions of this material appeared in [ABD14] and [DP12].

ISSN 1433-8092

where $b_i = 1$ if $s_i \in A$ and $b_i = 0$ otherwise. The sequence χ_A is also the binary representation of a real number in the interval $[0, 1]$. Thus, it is not uncommon to equate languages with real numbers. (The computability literature contains many investigations of this sort; see, for example, [Mil04, YDD04, DHNS03, NST05].)

Viewed in this light, the finite and co-finite languages correspond exactly to dyadic rational numbers. For instance, the sequences $1000\ldots$ and $0111\ldots$ (corresponding to the languages $\{\lambda\}$ and $\{x : x \neq \lambda\}$, respectively) both denote the number $\frac{1}{2} = \sum_{i=2}^{\infty} 2^{-i}$. Any real number in $[0, 1]$ that is *not* a dyadic rational has exactly one binary representation, and hence corresponds to exactly one language. The literature in computability theory (dating back to Turing's original work [Tur36]) has tended to focus on the placement of various classes of reals in the hierarchy of (non-)computability classes. (See, for example, [Mil04, YDD04, DHNS03, NST05].)

In contrast, the literature in computational complexity theory has tended to focus on the more "practical" questions of either computing approximations of various real numbers to a desired accuracy, or of obtaining *all* of the first n bits of their binary representations. For example, the original investigations of Hartmanis and Stearns [HS65], which can be said to have given birth to the field of computational complexity theory, focused on the question of classifying the complexity of various real numbers, in terms of computing the first n bits of their binary representations. Most other papers that have dealt with the complexity of real numbers (such as [Ko83, YK13, Jeř12, ACLG20, NV18]) have continued in a related vein (focusing more often on approximations), as has most of the significant work formalizing the notions of computability and complexity of real *functions* (e.g., [KF82, Wei00, BH21]). We do not dispute the motivations for defining the "complexity" of a real number in this way. After all, there are many applications that require knowing the bits of $\sqrt{2}$ to desired accuracy, whereas we struggle to conceive of a practical application that requires us to take a large string x as input (such as $x = 10^{100}$) and return the x^{th} bit of $\sqrt{2}$.

Nonetheless, there are important theoretical considerations that argue in favor of precisely this sort of investigation. Freivalds [Fre12] provides a very compelling survey of the history of mathematical considerations that have led investigators over the centuries to study "normal" reals and "automatic" reals. (Briefly, a real number corresponding to a language A is automatic if and only if A is regular. For more on this topic, we refer the reader to the book by Allouche and Shallit [AS03].) It was viewed as a very significant achievement when Adamczewski, Bugeaud, and Luca proved that no irrational algebraic number is automatic [ABL04, AB07]. (It is an easy exercise to see that every rational number corresponds to a regular language, and hence is automatic.)

In this paper, we give an upper bound on the complexity of languages corresponding to algebraic real numbers. We show that all such languages[2] lie in CH_5: the fifth level of the "counting hierarchy", whose levels are defined as follows.

- $\mathsf{CH}_1 = \mathsf{PP}$

[2]It is mistakenly claimed in [DP12] that these languages lie in $\mathsf{PH}^{\mathsf{CH}_2}$. We do not know how to obtain that stronger upper bound.

- $\mathsf{CH}_{k+1} = \mathsf{PP}^{\mathsf{CH}_k}$.

A real number is *algebraic* if it is a root of some (non-zero) univariate polynomial with integer coefficients. Jeřábek [Jeř12] showed that, for any given constant d, there is a uniform family of TC^0 circuits that will take as input a degree d univariate polynomial p (represented by its sequence of integer coefficients), along with 1^n, and produce as output the first n bits of the binary representation of *each* of the roots of p. Because of well-known connections between TC^0 and the counting hierarchy (see Proposition 2), this easily implies that every algebraic number lies in the counting hierarchy. However, Jeřábek's techniques yield threshold circuits whose depth depends on the degree of the polynomial p, and consequently his approach does not provide any constant k such that every algebraic real would lie[3] in CH_k; we show that this does hold, for $k = 5$. (We actually prove a better upper bound: $\mathsf{PH}^{\mathsf{CH}_3}$, which (by Toda's theorem [Tod91]) is contained in $\mathsf{P}^{\mathsf{CH}_4} \subseteq \mathsf{CH}_5$.) It is fair to ask how tight this upper bound is. Thus we also consider lower bounds, although the lower bounds we present are quite weak. For every prime modulus m, we show that there are algebraic numbers (in fact, *rational* numbers) that lie outside of $\mathsf{AC}^0[m]$). For rational numbers, this lower bound is rather tight; the language corresponding to any rational number lies in ACC^0 (and hence lies in $\mathsf{AC}^0[m]$ for some $m \in \mathbb{N}$). Although it seems reasonable to conjecture that irrational algebraic numbers are even more difficult than rational ones, we currently do not know of *any* irrational algebraic number that lies outside of AC^0; nor do we know of any irrational algebraic number that lies *in* AC^0. Our upper bound of $\mathsf{PH}^{\mathsf{CH}_3}$ is the best upper bound that we know of, for *any* irrational algebraic number.

Our techniques also apply to certain transcendental numbers, such as π. Yap [Yap10] showed that there is a logspace-computable function that, on input 1^n, will output the first n bits of π. (This theorem is also discussed by Lipton in [LR13, Chapter 31].) Thus the language corresponding to π lies in PSPACE. We improve this, to show that $\pi \in \mathsf{PH}^{\mathsf{CH}_3}$ (and we show that the first n bits of π can be computed in uniform TC^0). We also answer a question posed by Yap, by showing that, for any base b, the first n digits of π expressed in base b can be produced in TC^0, and hence in logspace.

The main technical contribution of our work consists of improved algorithms for integer division and related problems. The chain that connects integer division and the complexity of algebraic numbers consists of the following links:

1. Our $\mathsf{PH}^{\mathsf{CH}_3}$ upper bound on algebraic numbers relies on an improved upper bound on the problem of computing a given bit of a number represented by an arithmetic circuit (or straight-line program). This problem, known as BitSLP was introduced in [ABKPM09], where it was shown to be hard for $\#\mathsf{P}$ [ABKPM09], and was also shown to lie in the counting hierarchy.[4] The best previously-known upper bound for the

[3] A referee points out that the argument of [Jeř12] requires depth related to the degree d primarily because it is finding *all* of the roots of a polynomial p. The referee asserts that the algorithm in [Jeř12] can be modified, by hardwiring information related to a specific root of p, to show that every algebraic number lies in $\mathsf{PH}^{\mathsf{CH}_4}$, not quite matching the bound that we present.

[4] This connection between numerical computation and arithmetic circuits has also been exploited in other work, such as [COW13, OW14b, OW14a, LOW15].

complexity of BitSLP is the bound mentioned in [ABKPM09] and credited there to [AS05]: $\mathsf{PH}^{\mathsf{CH}_4}$.

2. That bound of $\mathsf{PH}^{\mathsf{CH}_4}$ follows via a straightforward translation of a uniform TC^0 algorithm for division and for conversion to and from Chinese Remainder Representation, which was presented in [HAB02]. The algorithms presented in [HAB02] also play a central role in the TC^0 root-finding algorithm of [Jeř12]. We give improved uniform TC^0 algorithms for these problems, which in turn yield a $\mathsf{PH}^{\mathsf{CH}_3}$ upper bound on BitSLP, and on algebraic reals.

The rest of the paper is organized as follows. In Section 2 we present the necessary background and definitions for our theorems that deal with (arithmetic and Boolean) circuits. In Section 3 we give improved algorithms for a variety of problems, including integer division, iterated product, and matrix powering, and we show how these algorithms yield improved upper bounds in the Counting Hierarchy for some fundamental problems about arithmetic circuits. In Section 4 we review background material about algebraic numbers and root-finding algorithms. Then in Section 4.2 we present our upper bound on the complexity of languages A whose characteristic sequence χ_A is an algebraic number, and we also present related results about certain transcendental numbers (including π). We conclude with a discussion of open problems in Section 5.

2 Arithmetic and Boolean Circuits

The algorithms that we present depend on Chinese Remainder Representation (CRR). Let us fix the notation that we will use. Given a list of primes $\Pi = (p_1, \ldots, p_m)$ and a number $X \in \mathbb{N}$, the CRR_Π representation of X is the list $(X \bmod p_1, \ldots, X \bmod p_m)$. We omit the subscript Π when it is clear from context.

We need to refer (repeatedly) to the binary expansion of a rational number. Furthermore, we want to avoid possible confusion caused by the fact that some numbers have more than one binary expansion (e.g. $1 = \sum_{i=1}^{\infty} 2^{-i}$). Thus the following definition fixes a *unique* binary representation for every rational number.

Definition 1

The binary expansion *of a nonnegative rational number X/Y is the* unique *expression $X/Y = \sum_{i=-\infty}^{\infty} a_i 2^i$, where each $a_i \in \{0, 1\}$, and where the binary expansion of any integer multiple of 2^j has $a_i = 0$ for all $i < j$.*

The binary expansion of X/Y correct to m places *is the sequence of bits representing* $\sum_{i=-m}^{\lfloor \log(X/Y) \rfloor} a_i 2^i$.

A *circuit* is a directed acyclic graph, whose nodes are called *gates* and whose edges are called *wires*. Gates with indegree zero are *input gates*. We usually restrict attention to circuits with exactly one gate of outdegree zero; this gate is called the *output gate*. In this

paper, all input gates will be connected either to constants or to *input variables*, which can be assigned values in $\{0, 1\}$.

In an *arithmetic circuit*, all gates other than input gates are labeled with an operation in $\{+, \times\}$, and the constants that we will allow are $\{0, 1, -1\}$. Thus each wire leading from gate g to gate h will "carry" the integer value that is computed at g to be fed into gate h.

Arithmetic circuits of polynomial size can produce numbers that require exponentially-many bits to represent in binary. The problem[5] known as BitSLP is the problem of determining a given bit of this binary representation. Formally, BitSLP is

$$\mathsf{BitSLP} = \{(C, i, b) : \text{ the } i\text{-th bit of the number represented by arithmetic circuit } C \text{ is } b\}.$$

A related problem,

$$\mathsf{PosSLP} = \{C : \text{the number represented by arithmetic circuit } C \text{ is } > 0\})$$

and a host of other problems on inferring properties of succinctly represented numbers were introduced in an article by Allender, Bürgisser, Kjeldgaard-Pedersen and Miltersen [ABKPM09]. The main goal of [ABKPM09] was to provide a complexity-theoretic framework to study problems arising in numerical analysis. It is shown in [ABKPM09] that BitSLP is hard for #P, and it also conjectured there that $\mathsf{PosSLP} \not\subseteq \mathsf{P/poly}$.

It is important to note that the arithmetic circuits considered in PosSLP and BitSLP do not have any input variables. Let us emphasize this point: **In this paper, we focus on arithmetic circuits *without input variables.*** Thus an arithmetic circuit is a (possibly very compact) representation of an integer.

The *Boolean* circuits that we will consider will have gates (other than input gates) labeled with an operation in {NOT, AND, OR, MAJORITY}. These gates (which have unbounded fan-in, except for NOT gates, which have fan-in 1) produce as output the logical NOT, AND, OR, and MAJORITY of their inputs, respectively (where MAJORITY$(x_1, \ldots, x_n)$ evaluates to 1 iff strictly more than $n/2$ of the input bits are 1).

The *depth* of a circuit is the length of the longest path from an input gate to the output gate. A *circuit family* is a set $\{C_n : n \in \mathbb{N}\}$ where each C_n has n input gates. A circuit family is *dlogtime-uniform* if there is a Turing machine takes an input string (n, g, h) of length m and determines in time $O(m)$ the labels of g and h in C_n and also reports if there is a wire in C_n from g to h. A circuit family is said to be *nonuniform* if no uniformity condition is imposed. TC^0 is the set of languages that are recognized by dlogtime-uniform circuit families of polynomial size and depth $O(1)$. AC^0 is the set of languages that are recognized by such circuit families that have no MAJORITY gates.

We also need to refer to *functions* computable in circuit classes. The function f is said to be in $\mathcal{C}$ (such as $\mathcal{C} = \mathsf{AC}^0$ or TC^0) if the length of $f(x)$ is polynomial in the length of x, and the language $\{(x, i, b) : \text{the } i^{\text{th}} \text{ bit of } f(x) \text{ is } b\}$ is in $\mathcal{C}$.

[5] "SLP" stands for "straight-line program"; which is a model equivalent to arithmetic circuits. Throughout the rest of the paper, we will stick with the arithmetic circuit formalism.

For more on circuit complexity classes such as AC^0 and TC^0, as well as a discussion of dlogtime uniformity, see [Vol99]. For background on other standard complexity classes such as $\mathsf{PP}, \#\mathsf{P}, \mathsf{NP}, \mathsf{P}$ etc., consult a standard text such as [AB09].

There are several possible variants of "depth" that one could choose to study. For instance, several papers have studied circuits consisting *only* of MAJORITY gates, and tight bounds are known for the depth required for several problems, in that model. (See, for instance [GK98, SR94, Weg93, She07] and other work referenced there.) Since our motivation comes largely from the desire to understand the complexity of problems in the counting hierarchy, it turns out that it is much more relevant to consider the notion of *majority depth* that was considered by Maciel and Thérien [MT98]. The class $\widehat{\mathsf{TC}}^0_d$ consists of functions computable by families of threshold circuits of polynomial size and constant depth such that no path from an input to an output gate encounters more than d MAJORITY gates. Thus the class of functions with majority depth zero, $\widehat{\mathsf{TC}}^0_0$, is precisely AC^0. In order to explain the connection between $\widehat{\mathsf{TC}}^0_d$ and the counting hierarchy, recall how the levels of the counting hierarchy are defined:

$$\mathsf{CH}_1 = \mathsf{PP}, \text{ and } \mathsf{CH}_{k+1} = \mathsf{PP}^{\mathsf{CH}_k}.$$

The counting hierarchy is analogous in some ways to the polynomial hierarchy $\mathsf{PH} = \mathsf{NP} \cup \mathsf{NP}^{\mathsf{NP}} \cup \ldots$. The following proposition can be interpreted as saying that $\mathsf{PH}^{\mathsf{CH}_d}$ is an exponential analog of $\widehat{\mathsf{TC}}^0_d$.

Proposition 2 *(Implicit in [ABKPM09, Theorem 4.1].) Let A be a set such that, for some k, some polynomial-time computable function f and for some dlogtime-uniform $\widehat{\mathsf{TC}}^0_d$ circuit family C_n, it holds that $x \in A$ if and only if $C_{|x|+2^{|x|^k}}(x, f(x,1)f(x,2)\ldots f(x,2^{|x|^k}))$ accepts. Then $A \in \mathsf{PH}^{\mathsf{CH}_d}$.*

(One important part of the proof of Proposition 2 is the fact that, by Toda's theorem [Tod91], for every oracle A, $\mathsf{PP}^{\mathsf{PH}^A} \subseteq \mathsf{P}^{\mathsf{PP}^A}$. Thus all of the AC^0 circuitry inside the $\widehat{\mathsf{TC}}^0_d$ circuit can be swallowed up by the PH part of the simulation.)

Note that the dlogtime-uniformity condition is crucial for Proposition 2. Thus, for the remainder of this paper, all references to $\widehat{\mathsf{TC}}^0_d$ will refer to the dlogtime-uniform version of this class, unless we specifically refer to nonuniform circuits.

3 Overview of the New Algorithmic Results

In this section, we present new uniform TC^0 algorithms for integer division, converting from CRR to binary, computing a power of a given integer, and computing a power of a given matrix (of bounded dimension). Table 1 compares the complexity bounds that Maciel and Thérien obtained in the *nonuniform* setting with the bounds that we are able to obtain in the uniform setting. (Maciel and Thérien also considered several problems for which they gave

Problem	Nonuniform Majority-Depth [MT98]	Uniform Majority-Depth [This Paper]
Iterated multiplication	3	3
Division	2	3
Powering	2	3
CRR-to-binary	1	3
Matrix powering	O(1) [MP00, HAB02]	3

Table 1: Prior bounds and new bounds on Majority-depth for various problems.

uniform circuit bounds; the problems listed in Table 1 were not known to lie in dlogtime-uniform TC^0 until the subsequent work of [HAB02].) All previously-known dlogtime-uniform TC^0 algorithms for these problems rely on the CRR-to-binary algorithm of [HAB02], and thus have at *least* majority-depth 4 (as analyzed by [AS05]); no other depth analysis beyond $O(1)$ was attempted.

In all of the cases where our uniform majority-depth bounds are worse than the nonuniform bounds given by [MT98], our algorithms also give rise to nonuniform algorithms that match the bounds of [MT98] (by hardwiring in some information that depends only on the length), although in all cases the algorithms differ in several respects from those of [MT98].

The most efficient previously-known TC^0 algorithms for the problems considered in this paper all make use of clever decompositions of the problem at hand, in terms of partial evaluations or approximations. The technical innovations in our improved algorithms rely on introducing yet another approximation, as discussed in Lemmas 7 and 10.

Table 1 also lists one problem that was not considered by Maciel and Thérien: the problem of taking as input 1^m and a $k \times k$ matrix A, and producing A^m. For any fixed k, this problem was shown to be in nonuniform TC^0 by Mereghetti and Palano[6] [MP00]; it follows from [HAB02] that their algorithm can be implemented in dlogtime-uniform TC^0. The corresponding problem of computing *large* powers of a $k \times k$ matrix (i.e., when m is given in *binary*) has been discussed recently [OW14b, GOW15]. We show that this version of matrix powering is in $\mathsf{PH}^{\mathsf{CH}_3}$, by making use of the improved algorithm for CRR-to-binary, which plays an important role in our $\mathsf{PH}^{\mathsf{CH}_3}$ algorithm for BitSLP.

In addition to BitSLP, there has also been interest in the related problem PosSLP [EY10, KP07, KS12, KP11]. $\mathsf{PosSLP} \in \mathsf{PH}^{\mathsf{CH}_2}$, and is not known to be in PH [ABKPM09], but in contrast to BitSLP, it is not known (or believed [EY10]) to be NP-hard. Our theorems do not imply any new bounds on the complexity of PosSLP, but we do conjecture that BitSLP and PosSLP both lie in $\mathsf{PH}^{\mathsf{PP}}$. This conjecture is based mainly on the heuristic that says that, for problems of interest, if a nonuniform circuit is known, then corresponding dlogtime-uniform circuits usually also exist. Converting from CRR to binary can be done nonuniformly in majority-depth one, and there is no reason to believe that this is not possible uniformly – although it seems clear that a different approach will be needed, to reach this goal.

[6]The reader should be cautioned that it is stated in [MP00] that iterated matrix product of $k \times k$ integer matrices is computable in NC^1. In fact, the best known upper bound is GapNC^1 [CMTV98].

The well-studied Sum-of-Square-Roots problem reduces to PosSLP [ABKPM09], which in turn reduces to BitSLP. But the relationship between PosSLP and the matrix powering problem (given a matrix A and an n-bit integers (k, j), output the k^{th} bit of a given entry of A^j) is unclear, since matrix powering corresponds to evaluating *very restricted* arithmetic circuits. Note that some types of arithmetic involving large numbers *can* be done in P; see [HKR10, GOW15]. Might matrix powering also lie in PH?

In Section 3.6, we provide a very weak "hardness" result for the problem of computing the bits of large powers of 2-by-2 matrices, to shed some dim light on this question. We show that the Sum-of-Square-Roots problem reduces to matrix powering via $\mathsf{PH}^{\mathsf{PP}}$-Turing reductions.

3.1 Improved Uniform Circuits for Division

Our new algorithm for division makes use of several useful subroutines that are computable in AC^0 and $\widehat{\mathsf{TC}}^0_1$. These are summarized in the following lemma:

Lemma 3 *Let x, y, i, j, k, x_j be numbers in the interval $(0, n^c)$ (where $c \geq 3$ is a constant). Let $X, X_j \in [0, 2^n)$ and let $p < n^c$ be prime. Then the following operations have the indicated complexities:*

1. *$p \mapsto$ first n^c bits of $1/p$ is in $\widehat{\mathsf{TC}}^0_0 = \mathsf{AC}^0$.*
2. *$p, k, X_1 \ldots, X_k \mapsto \sum_{j=1}^{k} X_j \bmod p$ is in $\widehat{\mathsf{TC}}^0_1$.*
3. *$x \mapsto x^i \bmod p$ is in $\widehat{\mathsf{TC}}^0_0 = \mathsf{AC}^0$.*
4. *$p \mapsto g_p$ is in $\widehat{\mathsf{TC}}^0_0 = \mathsf{AC}^0$ where g_p is a generator of the multiplicative group modulo p.*
5. *$X \mapsto X \bmod p$ is in $\widehat{\mathsf{TC}}^0_1$.*
6. *$x, y \mapsto xy \bmod p$ is in $\widehat{\mathsf{TC}}^0_0 = \mathsf{AC}^0$.*
7. *$(x_1, \ldots, x_k) \mapsto \prod_{j=1}^{k} x_j \bmod p$ is in $\widehat{\mathsf{TC}}^0_1$.*

Proof: We list the proofs of items in the Lemma above:

1. Follows from Lemma 4.2 and Corollary 6.2 in [HAB02]. In Section 4.3, we will also need the fact that the base-β representation of $1/p$ is also computable in AC^0. (See also Note 12.) Thus we present the details here.

   Claim 4 *The language $\{(1^n, p, \sigma) : p \leq n^c$ and the n^{th} symbol in the base-β representation of $1/p$ is $\sigma\}$ is in AC^0.*

Proof: Let a and b be such that $\beta^n = ap + b$ with $b = \beta^n \bmod p$. The n^{th} digit of the base-β expansion of the rational number $1/p$ is equal to the low-order digit of a. Since $ap + b$ is congruent to zero modulo β, it follows that the low-order digit of a is equal to $\beta - b \bmod \beta$. The result now follows from the fact that computing $\beta^n \bmod p$ is in AC^0 [HAB02]. □

2. Follows from Corollary 3.4.2 in [MT98].

3. Follows from Corollary 6.2 in [HAB02].

4. Follows from testing each integer $x \in [1, n-1]$ for being a generator by checking if $\forall i \leq \frac{p-1}{2}\ x^i \not\equiv 1 \bmod p$ and reporting the first successful x (implicit in [HAB02, ABKPM09]).

5. Follows from (the proof of) Lemma 4.1 in [HAB02].

6. Follows from Proposition 3.7 in [MT98] and the fact that two $\log n$-bit integers can be multiplied in AC^0.

7. Follows from the reduction of multiplication to addition of discrete logs and the previous parts.

□

3.2 A New Division Algorithm

We now give a construction of efficient threshold circuits for division.

Theorem 5 *The function taking as input $X \in [0, 2^n), Y \in [1, 2^n)$, and 0^m and producing as output the binary expansion of X/Y correct to m places is in $\widehat{\mathsf{TC}}^0_3$.*

Proof: This task is trivial if $Y = 1$; thus in the rest of this argument assume that $Y \geq 2$.

Computing the binary expansion of Z/Y correct to m places is equivalent to computing $\lfloor 2^m Z/Y \rfloor$. Thus we will focus on the task of computing $\lfloor X/Y \rfloor$, given integers X and Y.

The basic structure of all known TC^0 algorithms for division (reducing the problem to iterated product, and computing iterated product via a reduction to iterated addition, via conversion to and from Chinese Remainder Representation) has remained unchanged since the pioneering work of [BCH86]. Subsequent improvements [CDL01, HAB02, MT98, SR94] have focused on finding more efficient implementations of these various tasks.

Our approach will be to compute $\widetilde{V}(X, Y)$, a strict underestimate of X/Y, such that $X/Y - \widetilde{V}(X, Y) < 1/Y$. Since $Y > 1$, we have that $\lfloor X/Y \rfloor \neq \lfloor (X+1)/Y \rfloor$ if and only if $(X+1)/Y = \lfloor X/Y \rfloor + 1$. It follows that in *all* cases $\lfloor X/Y \rfloor = \lfloor \widetilde{V}(X+1, Y) \rfloor$, since

$$\left\lfloor \frac{X}{Y} \right\rfloor \leq \frac{X}{Y} = \frac{X+1}{Y} - \frac{1}{Y} < \widetilde{V}(X+1, Y) < \frac{X+1}{Y}.$$

Note that, in order to compute $\lfloor \frac{X}{Y} \rfloor$, we actually compute an approximation to $(X+1)/Y$.

The approximation $\widetilde{V}(X,Y)$ is actually defined in terms of another rational approximation $W(X,Y)$, which will have the property that $\widetilde{V}(X,Y) \leq W(X,Y) < X/Y$. ($W(X,Y)$ is easier to compute, which is why we introduce it.) We postpone the definition of $\widetilde{V}(X,Y)$, and focus for now on $W(X,Y)$, an under approximation of $\frac{X}{Y}$ with error at most $2^{-(n+1)}$.

Using AC^0 circuitry, we can compute a value $t \geq 2$ such that[7] $2^{t-1} \leq Y < 2^t$.

Let $u = 1 - 2^{-t}Y$. Then $u \in (0, \frac{1}{2}]$. Thus, $Y^{-1} = 2^{-t}(1-u)^{-1} = 2^{-t}(1+u+u^2+\ldots)$. Set $Y' = 2^{-t}(1+u+u^2+\ldots+u^{2n+1})$, then

$$0 < Y^{-1} - Y' \leq 2^{-t} \sum_{j>2n+1} 2^{-j} < 2^{-(2n+1)}$$

Define $W(X,Y)$ to be XY'. Hence, $0 < \frac{X}{Y} - W(X,Y) < 2^{-(n+1)}$.

We find it useful to use this equivalent expression for $W(X,Y)$:

$$W(X,Y) = \frac{X}{2^t} \sum_{j=0}^{2n+1} (1 - \frac{Y}{2^t})^j = \frac{1}{2^{2(n+1)t}} \sum_{j=0}^{2n+1} X(2^t - Y)^j 2^{(2n+1-j)t}.$$

Define $W_j(X,Y)$ to be $X(2^t - Y)^j(2^{(2n+1-j)t})$. Thus $W(X,Y) = \frac{1}{2^{2(n+1)t}} \sum_{j=0}^{2n+1} W_j(X,Y)$.

Lemma 6 *Let Π be any set of primes such that the product M of these primes lies in $(2^{n^c}, 2^{n^d})$ for some $d > c \geq 3$. Then, given X, Y, Π we can compute the CRR_Π representations of the $2(n+1)$ numbers $W_j(X,Y)$ (for $j \in \{0,\ldots,2n+1\}$) in $\widehat{\mathsf{TC}}^0_1$.*

Proof: With the aid of Lemma 3, we see that using AC^0 circuitry, we can compute

- $2^t - Y$
- $2^j \bmod p$ for each prime $p \in \Pi$ and various powers j, and
- a generator mod p for each prime $p \in \Pi$.

In $\widehat{\mathsf{TC}}^0_1$ we can compute $X \bmod p$ and $(2^t - Y) \bmod p$ (each of which has $O(\log n)$ bits). Using those results, with AC^0 circuitry we can compute the powers $(2^t - Y)^j \bmod p$ and then do additional arithmetic on numbers of $O(\log n)$ bits to obtain the product $X(2^t - Y)^j(2^{(2n+1-j)t}) \bmod p$ for each $p \in \Pi$. (The condition that $c \geq 3$ ensures that the numbers that we are representing are all less than M.) $\square$

[7] In Section 4.3, we will need to consider a variant algorithm, where, in this first step, a value t is computed such that $\beta^{t-1} \leq Y < \beta^t$. Observe that this is easy to do, if Y is expressed in base-β notation. Also, in this case, if we set $u = 1 - \beta^{-t}Y$. Then $u \in (0, \frac{\beta-1}{\beta}]$, and $Y^{-1} = \beta^{-t}(1-u)^{-1}$. This will also involve changing the definition of W and W_j, so that $W_j(X,Y)$ is $X(\beta^t - Y)^j(\beta^{(2n+1-j)t})$, and hence $W(X,Y) = \frac{1}{\beta^{2(n+1)t}} \sum_{j=0}^{2n+1} W_j(X,Y)$. We will refer to this variant of the algorithm as the *β-variant*. The analysis of the β-variant differs from that of the binary version in only trivial respects.

Having the CRR_Π representation of the number $W_j(X,Y)$, our goal might be to convert the $W_j(X,Y)$ to binary, and take their sum. In order to do this efficiently, we instead first show how to obtain an approximation (in binary) to $W(X,Y)/M$ where $M = \prod_{p\in\Pi} p$, and then in Lemma 10 we build on this to compute our approximation $\widetilde{V}(X,Y)$ to $W(X,Y)$.

Recall that $W(X,Y) = \frac{1}{2^{2(n+1)t}} \sum_{j=0}^{2n+1} W_j(X,Y)$. Thus the number $2^{2(n+1)t}W(X,Y)$ is an integer with the same significant bits as $W(X,Y)$.

Lemma 7 *Let Π be any set of primes such that the product M of these primes lies in $(2^{n^c}, 2^{n^d})$ for a fixed constant $d > c \geq 3$, and let b be any natural number. Then, given X, Y, Π we can compute the binary representation of a* good *approximation to $\frac{2^{2(n+1)t}W(X,Y)}{M}$ in $\widehat{\mathsf{TC}}^0_2$ (where by good we mean that it under-estimates the correct value by at most an additive term of $1/2^{n^b}$).*

Proof: Let $h_p^\Pi = (M/p)^{-1} \bmod p$ for each prime $p \in \Pi$.

If we were to first compute a good approximation $\tilde{A}_\Pi$ to the fractional part of:

$$A_\Pi = \sum_{p\in\Pi} \frac{(2^{2(n+1)t}W(X,Y) \bmod p)h_p^\Pi}{p}$$

i.e. if $\tilde{A}_\Pi$ were a good approximation to $A_\Pi - \lfloor A_\Pi \rfloor$, then $\tilde{A_\Pi}M$ would be a good approximation to $2^{2(n+1)t}W(X,Y)$. This follows from observing that the fractional part of A_Π is exactly $\frac{2^{2(n+1)t}W(X,Y)}{M}$ (as in [HAB02, Lemma 4.3] and [ABKPM09, Theorem 4.2]).

Instead of working with A_Π, we will work with

$$A'_\Pi = \sum_{p\in\Pi} \sum_{j=0}^{2n+1} \frac{(W_j(X,Y) \bmod p)h_p^\Pi}{p}.$$

Note that the exact magnitudes of the two quantities A_Π, A'_Π are not the same but their *fractional parts* will be the same. Thus we will compute $\tilde{A}_\Pi$ as a good approximation to the fractional part of A'_Π. Since we are adding up $2(n+1)|\Pi|$ approximate quantities it suffices to compute each of them to $b_{n,b,\Pi} = 2n^b + 2(n+1)|\Pi|$ bits of accuracy to ensure:

$$0 \leq \frac{2^{2(n+1)t}W(X,Y)}{M} - \tilde{A}_\Pi < \frac{1}{2^{n^b}}.$$

Now we analyze the complexity. By Lemma 6, we obtain in $\widehat{\mathsf{TC}}^0_1$ the CRR_Π representation of $W_j(X,Y) \in [0, 2^n)$ for $j \in \{0, \ldots, O(n)\}$. Also, by Lemma 3, each h_p^Π can be computed in $\widehat{\mathsf{TC}}^0_1$, and polynomially-many bits of the binary expansion of $1/p$ can be obtained in AC^0.

Using AC^0 circuitry we can multiply together the $O(\log n)$-bit numbers $W_j(X,Y) \bmod p$ and h_p^Π, and then obtain the binary expansion of $((W_j(X,Y) \bmod p)h_p^\Pi) \cdot (1/p)$ (since multiplying an n-bit number by a $\log n$ bit number can be done in AC^0).

Thus, with one more layer of majority gates, we can compute a good approximation to

$$A'_{\Pi} = \sum_{p \in \Pi} \sum_{j=0}^{2n+1} \frac{(W_j(X,Y) \bmod p) h_p^{\Pi}}{p}$$

and strip off the integer part, to obtain the desired approximation $\tilde{A}_{\Pi}$. $\square$

Corollary 8 *Let Π be any set of primes such that the product M of these primes lies in $(2^{n^c}, 2^{n^d})$ for a fixed constant $d > c \geq 3$. Then, given Z in CRR_{Π} representation and the numbers h_p^{Π} for each $p \in \Pi$, we can compute the binary representation of a* good *approximation to $\frac{Z}{M}$ in $\widehat{\mathsf{TC}}^0_1$.*

Proof: This follows from the analysis presented in the proof of Lemma 7, in the special case when $W_1 = Z$ and $W_j = 0$ for all $j > 1$. $\square$

Before presenting our approximation $\widetilde{V}(X,Y)$, first we present a claim, which helps motivate the definition. In the claim, and in the subsequent discussion, let Π_i for $i \in \{1, \ldots, n^{\ell}\}$ be n^{ℓ} pairwise disjoint sets of primes such that $M_i = \prod_{p \in \Pi_i} p \in (2^{n^c}, 2^{n^d})$ (for some constants $c, d : 3 \leq \ell \leq c < d$). Let $\Pi = \bigcup_{i=1}^{n^{\ell}} \Pi_i$.

Claim 9 *For any value A, it holds that*

$$A(1 - \frac{n^{\ell}}{2^{n^{\ell}}}) < \frac{A \prod_{i=1}^{n^{\ell}} (M_i - 1)}{\prod_{i=1}^{n^{\ell}} M_i} < A.$$

Proof: It suffices to show that

$$(1 - \frac{n^{\ell}}{2^{n^{\ell}}}) < \frac{\prod_{i=1}^{n^{\ell}} (M_i - 1)}{\prod_{i=1}^{n^{\ell}} M_i} < 1$$

where the final inequality is trivial. Let $m = n^{\ell}$ and let $x = 2^{n^{\ell}}$. Then $(1 - \frac{n^{\ell}}{2^{n^{\ell}}}) = 1 - \frac{m}{x}$ and $(1 - \frac{1}{x})^m \leq \frac{\prod_{i=1}^{n^{\ell}} (M_i - 1)}{\prod_{i=1}^{n^{\ell}} M_i}$. Thus the claim holds if we show that, for all $x > 1$ and for all integers $m > 1$,

$$1 - \frac{m}{x} < \left(1 - \frac{1}{x}\right)^m.$$

This holds by induction. Assume that $1 - m/x \leq (1 - 1/x)^m$. (This holds for $m = 1$.) Then $1 - (m+1)/x = 1 - m/x + m/x - (m+1)/x < (1 - 1/x)^m - 1/x < (1 - 1/x)^m - (1 - 1/x)^m 1/x = (1 - 1/x)^m (1 - 1/x)$. This completes the induction, and the proof of the claim. $\square$

Now, finally, we present our desired approximation. $\widetilde{V}(X,Y)$ is $2^{n^\ell} \cdot V'(X,Y)$, where $V'(X,Y)$ is an approximation (within $1/2^{n^{2\ell}}$) of

$$V(X,Y) = \frac{W(X,Y)\prod_{i=1}^{n^\ell}(M_i-1)/2}{\prod_{i=1}^{n^\ell} M_i}.$$

Note that

$$\begin{aligned} W(X,Y) - 2^{n^\ell}V(X,Y) &= W(X,Y) - 2^{n^\ell}\frac{W(X,Y)\prod_{i=1}^{n^\ell}(M_i-1)/2}{\prod_{i=1}^{n^\ell} M_i} \\ &= W(X,Y) - \frac{W(X,Y)\prod_{i=1}^{n^\ell}(M_i-1)}{\prod_{i=1}^{n^\ell} M_i} \\ &< W(X,Y)\frac{n^\ell}{2^{n^\ell}} < \frac{2^{2n}n^\ell}{2^{n^\ell}} \end{aligned}$$

and

$$\begin{aligned} 2^{n^\ell}V(X,Y) - \widetilde{V}(X,Y) &= 2^{n^\ell}V(X,Y) - 2^{n^\ell}V'(X,Y) \\ &= 2^{n^\ell}(V(X,Y) - V'(X,Y)) \\ &\leq 2^{n^\ell}(\frac{1}{2^{n^{2\ell}}}) \\ &= \frac{2^{n^\ell}}{2^{n^{2\ell}}}. \end{aligned}$$

Thus $X/Y - \widetilde{V}(X,Y)$ is equal to

$$(X/Y - W(X,Y)) + (W(X,Y) - 2^{n^\ell}V(X,Y)) + (2^{n^\ell}V(X,Y) - \widetilde{V}(X,Y))$$

and hence

$$\begin{aligned} X/Y - \widetilde{V}(X,Y) &< 2^{-(n+1)} + \frac{n^\ell 2^{2n}}{2^{n^\ell}} + \frac{2^{n^\ell}}{2^{n^{2\ell}}} \\ &< 1/Y, \end{aligned}$$

for all $n \geq 2$.

Lemma 10 *Let Π_i for $i \in \{1,\ldots,n^\ell\}$ be n^ℓ pairwise disjoint sets of odd primes such that $M_i = \prod_{p\in\Pi_i} p \in (2^{n^c}, 2^{n^d})$ (for some constants $c,d : 3 \leq c < d$). Let $\Pi = \bigcup_{i=1}^{n^\ell}\Pi_i$. Then, given X, Y and the Π_i, we can compute $\widetilde{V}(X,Y)$ in $\widehat{\mathsf{TC}}^0_3$.*

Proof: Via Lemma 3, in $\widehat{\mathsf{TC}}^0_1$ we can compute the CRR_Π representation of each M_i, as well as the numbers $W_j \bmod p$ (using Lemma 6). Also, as in Lemma 7, we can compute the values h_p^Π for each prime p.

Then, via Lemma 3, with one more layer of majority gates we can compute the CRR representation of $\prod_i (M_i - 1)/2$, as well as the CRR representation of $2^{2(n+1)t}W(X,Y) = \sum_{j=0}^{2n+1} W_j(X,Y)$. The CRR representation of the product $2^{2(n+1)t}W(X,Y) \cdot \prod_i (M_i - 1)/2$ can then be computed with AC^0 circuitry to obtain the CRR representation of the numerator of the expression for $V(X,Y)$. (It is important to note that $2^{2(n+1)t}W(X,Y) \cdot \prod_i (M_i - 1)/2 < \prod_i M_i$, so that it is appropriate to talk about this CRR representation. Indeed, that is the reason why we divide each factor $M_i - 1$ by two.)

This value can then be converted to binary with one additional layer of majority gates, via Corollary 8, to obtain $\widetilde{V}(X,Y)$. □

This completes the proof of Theorem 5. □

Corollary 11 *Let Π be any set of primes that is the union of pairwise disjoint sets Π_i, such that, for all i, $M_i = \prod_{p \in \Pi_i} p$ lies in $(2^{n^c}, 2^{n^d})$ for fixed constants $d > c \geq 3$. Then, given Z in CRR_Π representation, the binary representation of Z can be computed in $\widehat{\mathsf{TC}}^0_3$.*

Proof: Recall from the proof of Theorem 5 that, in order to compute the bits of $Z/2$, our circuit actually computes an approximation to $(Z+1)/2$. Although, of course, it is trivial to compute $Z/2$ if Z is given to us in binary, let us consider how to modify the circuit described in the proof of Lemma 10, if we were computing $\widetilde{V}(Z+1,2)$, where we are given Z in CRR representation.

With one layer of majority gates, we can compute the CRR_Π representation of each M_i and the values h_p^Π for each prime p. (We will not need the numbers $W_j \bmod p$.)

Then, with one more layer of majority gates we can compute the CRR representation of $\prod_i (M_i - 1)/2$. In place of the gates that store the value of the CRR representation of $2^{2(n+1)t}W(X,Y)$, we insert the CRR representation of Z (which is given to us as input) and using AC^0 circuitry store the value of $Z+1$. The CRR representation of the product $(Z+1) \cdot \prod_i (M_i - 1)/2$ can then be computed with AC^0 circuitry to obtain the CRR representation of the numerator of the expression for $V(Z+1,2)$.

Then this value can be converted to binary with one additional layer of majority gates, from which the bits of Z can be read off. □

Note 12 *Although we have stated our results in terms of converting from CRR to binary notation, there is nothing special about base 2. As observed in [HAB02, All04], the approach from Lemma 10 (and in Lemma 7) carries over with only trivial adjustments, to convert from CRR to base ten or to representation in any other base. The only "new" ingredient that is required is that the base-β expansion of $1/p$ can be computed to a polynomial number of bits of accuracy in AC^0. (See Claim 4.[8])*

[8] It should be mentioned that the β-variant of the division algorithm is not required, for conversion from CRR to base-β. The β-variant is introduced for other considerations that arise in Section 4.3.

It is rather frustrating to observe that the input values Z are not used until quite late in the $\widehat{\mathsf{TC}}^0_3$ computation (when just one layer of majority gates remains). However, we see no simpler uniform algorithm to convert CRR to binary.

For our application regarding problems in the counting hierarchy, it is useful to consider the analog to Theorem 5 where the values X and Y are presented in CRR notation.

Theorem 13 *The function taking as input $X \in [0, 2^n), Y \in [1, 2^n)$ (in CRR) as well as 0^m, and producing as output the binary expansion of X/Y correct to m places is in $\widehat{\mathsf{TC}}^0_3$.*

Proof: We assume that the CRR basis consists of pairwise disjoint sets of primes M_i, as in Lemma 10.

The algorithm is much the same as in Theorem 5, but there are some important differences that require comment. The first step is to determine if $Y = 1$, which can be done using AC^0 circuitry (since the CRR of 1 is easy to recognize). The next step is to determine a value t such that $2^{t-1} \leq Y < 2^t$. Although this is trivial when the input is presented in binary, when the input is given in CRR it requires the following lemma:

Lemma 14 *(Adapted from [AAD00, DMS94, ABKPM09]) Let X be an integer from $(-2^n, 2^n)$ specified by its residues modulo each $p \in \Pi_n$. Then, the predicate $X > 0$ is in $\widehat{\mathsf{TC}}^0_2$*

Since we are able to determine inequalities in majority-depth two, we will carry out the initial part of the algorithm from Theorem 5 using *all* possible values of t, and then select the correct value between the second and third levels of MAJORITY gates.

Thus, for each t, and for each j, we compute the values $W_{j,t}(X+1, Y) = (X+1)(2^t - Y)^j(2^{(2n+1-j)t})$ in CRR, along with the desired number of bits of accuracy of $1/p$ for each p in our CRR basis.

With this information available, as in Lemma 10, in majority-depth one we can compute h^{Π}_p, as well as the CRR representation of each M_i, and thus with AC^0 circuitry we obtain $(W_{j,t}(X+1, Y)$ and the CRR for each $(M_i - 1)/2$.

Next, with our second layer of majority gates we sum the values $W_{j,t}(X+1, Y)$ (over all j), and at this point we also will have been able to determine which is the correct value of t, so that we can take the correct sum, to obtain $2^{2(n+1)t}W(X+1, Y)$.

Thus, after majority-depth two, we have obtained the same partial results as in the proof of Lemma 10, and the rest of the algorithm is thus identical. □

The following generalization of Theorem 13 will be useful for us in Section 4.3.

Theorem 15 *There is a function computable in $\widehat{\mathsf{TC}}^0_3$ that takes as input the CRR representation of a sequence $X_1, Y_1, X_2, Y_2, \ldots, X_r, Y_r$, as well as as 0^m and 1^t with the property that, for each i, $2^{t-1} \leq Y_i < 2^t$, and produces as output the binary expansion of $\sum_{i=1}^{r} \frac{X_i}{Y_i}$ correct to m places.*

Proof: The naïve approach, of simply taking the sum of the circuits that result from Theorem 13, would be too expensive in terms of majority-depth. Thus we dive deeper into the details of how each quotient is computed.

Recall the definition of $W_j(X,Y)$ (immediately before Lemma 6), and recall also that $\frac{X}{Y}$ is approximated by $W(X+1,Y) = \frac{1}{2^{2(n+1)t}} \sum_{j=0}^{2n+1} W_j(X+1,Y)$.

Thus $\sum_{i=1}^{r} \frac{X_i}{Y_i} \approx \sum_{i=1}^{r} \frac{1}{2^{2(n+1)t}} \sum_{j=0}^{2n+1} W_j(X_i+1,Y_i)$, where the approximation is a correct underapproximation to $n^{O(1)}$ bits

As in the proof of Theorem 13, after one level of majority gates, we can have computed the values of $W_j(X_i+1,Y_i)$ (in CRR notation), and with another level of majority gates, we can compute the CRR of $\sum_{i=1}^{r} \frac{1}{2^{2(n+1)t}} \sum_{j=0}^{2n+1} W_j(X_i+1,Y_i)$, and (as in the proof of Theorem 13) with one more level of majority gates, we obtain the binary encoding of the desired result. □

Proposition 16 *Iterated product is in uniform* $\widehat{\mathsf{TC}}^0_3$.

Proof: The overall algorithm is identical to the algorithm outlined in [MT98], although the implementation of the basic building blocks is different. In majority-depth one, we convert the input from binary to CRR. With one more level of majority gates, we compute the CRR of the product.

Simultaneously, in majority-depth two we compute the bottom two levels of our circuit that converts from CRR to binary, as in Corollary 11.

Thus, with one final level of majority gates, we are able to convert the answer from CRR to binary. □

3.3 Consequences for the Counting Hierarchy

Corollary 17 $\mathsf{BitSLP} \in \mathsf{PH}^{\mathsf{CH}_3}$.

Proof: This is immediate from Proposition 2 and Corollary 11.

Let f be the function that takes as input a tuple $(C,(p,j))$ and if p is a prime, evaluates the arithmetic circuit C mod p and outputs the j-th bit of the result. This function f, taken together with the $\widehat{\mathsf{TC}}^0_3$ circuit family promised by Corollary 11, satisfies the hypothesis of Proposition 2. (There is a minor subtlety, regarding how to partition the set of primes into the groupings M_i, but this is easily handled by merely using all of the primes of a given length, at most polynomially-larger than $|C|$.) □

Via essentially identical methods, using Theorem 13, we obtain:

Corollary 18 $\{(C_X, C_Y, N)$ *: the* N^{th} *bit of the quotient* X/Y*, where* X *and* Y *are represented by arithmetic circuits* C_X *and* C_Y*, respectively*$\}$ *is in* $\mathsf{PH}^{\mathsf{CH}_3}$.

Proof: We will appeal to Proposition 2 and Theorem 13. Let f be the polynomial-time-computable function that, on input $(x, y) = ((C_X, C_Y, N), (b, p))$ outputs "p not prime" if p is not prime, and otherwise outputs the value of $C_X \bmod p$ if $b = 0$, and outputs the value of $C_Y \bmod p$ if $b = 1$. Note that the string $(x, f(x,1)f(x,2)\ldots f(x,2^{|x|^k}))$ can be viewed as providing the CRR representation of the numbers represented by C_X and C_Y. (Technically, in order to directly appeal to Proposition 2 and Theorem 13, the definition of f will need to be modified slightly, so that $(x, f(x,1)f(x,2)\ldots f(x,2^{|x|^k}))$ also ends in a sequence of exponentially-many zeros. This is conceptually quite easy – by only considering numbers p whose length, in bits, is polynomially-related to the length of N, and making some minor formatting changes. In order to avoid introducing distracting technicalities, we leave these minor details to the interested reader.)

Now, let A be the language in $\widehat{\mathsf{TC}}^0_3$ that takes the CRR representation of X and Y as input, along with the number 0^N, and outputs the N^{th} bit of X/Y, as follows from Theorem 13. The theorem now follows from Proposition 2. □

3.4 Integer Powering

In this section, we continue presenting our algorithmic results, concentrating on the problem of computing powers of integer matrices.

We begin by presenting an alternative algorithm for *integer* powering, which serves to illustrate our approach for matrix powering. (Note that the upper bound of $\widehat{\mathsf{TC}}^0_3$ already follows from our algorithm for iterated integer multiplication. Here, we are merely presenting an alternative algorithm as a warm-up for matrix powering.)

Theorem 19 *The function taking as input $X \in [0, 2^n)$, 1^m and 1^i (where $i \in [1, nm]$) and producing as output the i-th bit of X^m is in $\widehat{\mathsf{TC}}^0_3$.*

Proof:

Our algorithm is as follows:

1. Convert X to CRR. Let $X \equiv X_j \mod p_j$ for $j \in [k]$. This is implementable in $\widehat{\mathsf{TC}}^0_1$ by item 5 in Lemma 3. In parallel, compute the first two phases of our uniform algorithm to convert CRR to binary (Corollary 11). (Note that these first two stages depend only on the moduli Π, and do not depend on X)

2. Compute X^m by appealing to Fermat's little theorem. More precisely: Since $X^{p-1} \equiv 1 \mod p$ for any prime p, we can compute $X_j^{m_j} \mod p_j$ where $m = q_j(p_j - 1) + m_j$ for $j \in [k]$. This step is in AC^0 via item 3 in Lemma 3.

3. At this stage, we have the answer encoded in CRR, and we invoke the final layer of the circuit from Corollary 11, convert the answer to binary.

Putting these three together, we have integer powering in $\widehat{\mathsf{TC}}^0_3$. □

3.5 Integer Matrix Powering

Theorem 20 *The function MPOW(A, m, p, q, i) taking as input a $(d \times d)$ integer matrix $A \in \{0,1\}^{d^2 n}$, p, q, 1^i, where $p, q \in [d]$, $i \in [O(n)]$ and producing as output the i-th bit of the (p,q)-th entry of A^m is in $\widehat{\mathsf{TC}}^0_3$.*

For a $(d \times d)$ matrix A, the characteristic polynomial $\chi_A(x) : \mathbb{Z} \to \mathbb{Z}$ is a univariate polynomial of degree at most d. Let $q, r : \mathbb{Z} \to \mathbb{Z}$ be univariate polynomials of degree at most $(m-d)$ and $(d-1)$ such that $x^m = q(x)\chi_A(x) + r(x)$. By the Cayley-Hamilton theorem, we have that $\chi_A(A) = 0$. So, in order to compute A^m, it suffices to compute $r(A)$.

Lemma 21 *Given a $(d \times d)$ matrix A with entries that are n-bit integers, the coefficients of the characteristic polynomial of A in CRR can be computed in $\widehat{\mathsf{TC}}^0_1$.*

Proof: We convert the entries of A to CRR and compute the determinant of $(xI - A)$. This involves an iterated sum of $O(2^d d!)$ integers each of which is an iterated product of d n-bit integers. The conversion to CRR is in $\widehat{\mathsf{TC}}^0_1$ by item 5 in Lemma 3. Since addition, multiplication, and powering of $O(1)$ numbers of $O(\log n)$ bits is computable in AC^0 (by Lemma 3, items 3, 4 and 6), it follows that the coefficients of the characteristic polynomial can be computed in $\widehat{\mathsf{TC}}^0_1$. $\square$

Lemma 22 *Given the coefficients of the polynomial r, in CRR, and given A in CRR, we can compute A^m in CRR using AC^0 circuitry.*

Proof: Recall that $A^m = r(A)$. Let $r(x) = r_0 + r_1 x + \ldots + r_{d-1} x^{d-1}$. Computing any entry of $r(A)$ in CRR involves an iterated sum of $O(1)$ many numbers which are themselves an iterated product of $O(1)$ many $O(\log n)$-bit integers. The claim follows by appeal to Lemma 3. $\square$

Lemma 23 *(Adapted from [HV06]) Let p be a prime of magnitude $\mathsf{poly}(m)$. Let $g(x)$ of degree m and $f(x)$ of degree d be monic univariate polynomials over GF_p, such that $g(x) = q(x)f(x) + r(x)$ for some polynomials $q(x)$ of degree $(m-d)$ and $r(x)$ of degree $(d-1)$. Then, given the coefficients of g and f, the coefficients of r can be computed in $\widehat{\mathsf{TC}}^0_1$.*

Proof: Following [HV06], let $f(x) = \sum_{i=0}^{d} a_i x^i$, $g(x) = \sum_{i=0}^{m} b_i x^i$, $r(x) = \sum_{i=0}^{d-1} r_i x^i$ and $q(x) = \sum_{i=0}^{m-d} q_i x^i$. Since f, g are monic, we have $a_d = b_m = 1$. Denote by $f_R(x), g_R(x), r_R(x)$ and $q_R(x)$ respectively the polynomial with the i-th coefficient $a_{d-i}, b_{m-i}, r_{d-i-1}$ and q_{m-d-i} respectively. Then note that $x^d f(1/x) = f_R(x)$, $x^m g(1/x) = g_R(x)$, $x^{m-d} q(1/x) = q_R(x)$ and $x^{d-1} r(1/x) = r_R(x)$.

We use the Kung-Sieveking algorithm (as implemented in [HV06]). The algorithm is as follows:

1. Compute $\tilde{f}_R(x) = \sum_{i=0}^{m-d}(1 - f_R(x))^i$ via interpolation modulo p.

2. Compute $h(x) = \tilde{f}_R(x)g_R(x) = c_0 + c_1 x + \ldots + c_{d(m-d)+m}x^{d(m-d)+m}$. from which the coefficients of $q(x)$ can be obtained as $q_i = c_{d(m-d)+m-i}$.

3. Compute $r(x) = g(x) - q(x)f(x)$.

To prove the correctness of our algorithm, note that we have $g(1/x) = q(1/x)f(1/x) + r(1/x)$. Scaling the whole equation by x^m, we get $g_R(x) = q_R(x)f_R(x) + x^{m-d+1}r_R(x)$. Hence when we compute $h(x) = \tilde{f}_R(x)g_R(x)$ in step 2 of our algorithm, we get

$$h(x) = \tilde{f}_R(x)g_R(x) = \tilde{f}_R(x)q_R(x)f_R(x) + x^{m-d+1}\tilde{f}_R(x)r_R(x).$$

Note that $\tilde{f}_R(x)f_R(x) = \tilde{f}_R(x)(1 - (1 - f_R(x))) = \sum_{i=0}^{m-d}(1 - f_R(x))^i - \sum_{i=0}^{m-d}(1 - f_R(x))^{i+1} = 1 - (1 - f_R(x))^{m-d+1}$ (a telescoping sum). Since f is monic, f_R has a constant term which is 1 and hence $(1 - f_R(x))^{m-d+1}$ does not contain a monomial of degree less than $(m - d + 1)$. This is also the case with $x^{m-d+1}\tilde{f}_R(x)r_R(x)$, and hence all the monomials of degree less than $(m - d + 1)$ belong to $q_R(x)$.

Now we justify why the algorithm above is amenable to a $\widehat{\mathsf{TC}}^0_1$ implementation: Firstly, note that given $f(x)$ and $g(x)$, the coefficients of $f_R(x)$ and $g_R(x)$ can be computed in NC^0. To compute the coefficients of $\tilde{f}_R(x)$, we use interpolation via the discrete Fourier transform (DFT) using arithmetic modulo p. Find a generator w of the multiplicative group modulo p and substitute $x = \{w^1, w^2, \ldots, w^{p-1}\}$ to obtain a system of linear equations in the coefficients F of $\tilde{f}_R(x) : V \cdot F = Y$, where Y is the vector consisting of $\tilde{f}_R(w^i)$ evaluated at the various powers of w. Since the underlying linear transformation $V(w)$ is a DFT, it is invertible; the inverse DFT $V^{-1}(w)$ is equal to $V(w^{-1}) \cdot (p-1)^{-1}$, which is equivalent to $-V(w^{-1}) \bmod p$. We can find each coefficient of $\tilde{f}_R(x)$ by evaluating $V^{-1}Y$, i.e., by an inner product of a row of the inverse DFT-matrix with the vector formed by evaluating $\sum_{i=1}^{(m-d+1)}(1 - f_R(x))^{i-1}$ at various powers of w and dividing by $p - 1$. The terms in this sum can be computed in AC^0, and then the sum can be computed in majority-depth one, to obtain the coefficients of $\tilde{f}_R(x)$. The coefficients of $h(x)$ in step 2 can be obtained by iterated addition of the product of certain coefficients of $\tilde{f}_R$ and g_R, but since the coefficients of $\tilde{f}_R$ are themselves obtained by iterated addition of certain terms t, we roll steps 1 and 2 together by multiplying these terms t by the appropriate coefficients of g_R. Thus steps 1 and 2 can be accomplished in majority-depth 1. Then step 3 can be computed using AC^0 circuitry. □

Proof:(of Theorem 20)

Our $\widehat{\mathsf{TC}}^0_3$ circuit C that implements the ideas above is the following:

0. At the input, we have the d^2 entries A_{ij}, $i, j \in [d]$ of A, a set Π of short primes (such that Π can be partitioned into n^c sets Π_i that are pairwise disjoint, i.e., $\Pi = \cup_{i=1}^{n^c}\Pi_i$), and the numbers $I = \{1, 2, \ldots, (m - d + 1)\}$.

1. In majority-depth one, we obtain (1) $A_{ij} \mod p$ for each prime p in our basis, and (2) $M_i = \prod_{p \in \Pi_i} p$ for each of the n^c sets Π_i that constitute Π, and (3) the CRR of the characteristic polynomial of A (via appeal to Lemma 21).

2. In the next layer of threshold gates, we compute (1) $\prod_i^{n^c}(M_i - 1)/2$ in CRR, and (2) the coefficients of the polynomial r in CRR, by appeal to Lemma 23.

3. At this point, by Lemma 22, AC^0 circuitry can obtain $r(A) = A^m$ in CRR, and with one more layer of MAJORITY gates we can convert to binary, by appeal to Corollary 11.

□

Now we consider the "succinct" versions of the problems of computing powers of integers and matrices (i.e., where the power is given in binary, instead of in unary notation), and show that these problems reduce to BitSLP. (We do not consider taking powers of integers as a separate problem; an integer can be viewed as a 1-by-1 matrix.) Define:

$$\mathsf{Bit-k-MatPow} = \{(A, N, p, q, I) : \text{ the } I^{\text{th}} \text{ bit of entry } (p, q) \text{ of the } (k \times k) \text{ matrix } A^N \text{ is 1 } \}$$

(Here, the matrix A is represented by k^2 arithmetic circuits, with one circuit for each entry.)

Theorem 24 *For every $k \in \mathbb{N}$,* Bit-k-MatPow polynomial time reduces to BitSLP.

Proof: It is sufficient to produce arithmetic circuits computing A^N. This is easily obtained via repeated squaring:

Since M is a $(k \times k)$ matrix with entries that are represented as arithmetic circuits, we can again compute A^N using $n^{O(1)}$ additional arithmetic gates, by repeated squaring (where N is an n-bit number). Again, it is easy to construct this circuit, in polynomial time. □

3.6 Reducing Sum-of-square-roots to Matrix Powering

In this section, we digress slightly from our presentation of efficient algorithms (in TC^0 or in CH), in order to address the question of whether the problems that we have shown to lie in CH might have much more efficient algorithms. Evidence for the intractability of PosSLP and BitSLP is presented in [ABKPM09]. But there is much less evidence that matrix powering is difficult. Here, we show that if one could power 2-by-2 matrices in PH, then it would yield an improved upper bound on the well-known Sum-of-Square-Roots problem. More formally, we present a reduction, showing that the Sum-of-Square-Roots problem is reducible to the problem of computing large powers of 2-by-2 integer matrices, where the power of the reduction lies low in CH.

Definition 25 [The Sum-of-Square-Roots Problem] *Let $\mathbf{a} = (a_1, \ldots, a_n)$ be a list of n-bit positive integers, and let $\sigma = (\sigma_1, \ldots, \sigma_n) \in \{-1, +1\}^n$. Define* $\mathsf{SSQRT}(\mathbf{a}, \sigma)$ *to be the problem of determining if:*

$$\sum_{i=1}^{n} \sigma_i \sqrt{a_i} > 0$$

Theorem 26 $\mathsf{SSQRT} \in \mathsf{PH}^{\mathsf{PP}^{\mathsf{Bit\text{-}2\text{-}MatPow}}}$.

Our proof makes use of Linear Fractional Transformations (LFTs), which in turn correspond directly to 2-by-2 matrices. We introduce LFTs in the next subsection.

3.7 Linear Fractional Transformations (LFTs)

Here we give a brief introduction to LFTs based on the expositions in [EP97, Pot97, Pot99], concentrating only on the aspects required in this paper.

A linear fractional transformation is a function mapping $y \mapsto \frac{ay+c}{by+d}$ for reals (and preferably integers) a, b, c, d; we associate the matrix $\begin{pmatrix} a & c \\ b & d \end{pmatrix}$ to this mapping. The interesting thing about LFTs is that the matrix corresponding to the composition of two LFTs is the usual product of the matrices corresponding to the two LFTs. In other words, if the matrix corresponding to $\phi_i(y)$ is $\begin{pmatrix} a_i & c_i \\ b_i & d_i \end{pmatrix}$ (for $i = 1, 2$), then a matrix corresponding to $\phi_1\phi_2(y)$ (which abbreviates $\phi_1(\phi_2(y))$) is $\begin{pmatrix} a_1 & c_1 \\ b_1 & d_1 \end{pmatrix}\begin{pmatrix} a_2 & c_2 \\ b_2 & d_2 \end{pmatrix}$, as can be easily verified. In this paper we deal only with nonsingular LFTs (i.e., LFTs whose matrices have non-zero determinant). An LFT is said to be positive if all four entries in its matrix have the same sign.

Let ϕ be an LFT and let $M = \begin{pmatrix} a & c \\ b & d \end{pmatrix}$ be its matrix. Then ϕ acts as a bijection between any interval $[p, q]$ and a subset of the extended reals $\mathbb{R} \cup \{\infty\}$. Further, this subset is also an interval (possibly including ∞): either $[\phi(p), \phi(q)]$ or $[\phi(q), \phi(p)]$. Notice that we do not claim that there is a linear order on the reals augmented with ∞. Instead, we refer to these sets as "intervals" in the same sense that connected subsets of the unit circle can be called intervals.

For a concrete example, $\phi[0, \infty]$ is the interval $[\frac{a}{b}, \frac{c}{d}]$ if $det(M) < 0$ and the interval $[\frac{c}{d}, \frac{a}{b}]$ if $det(M) > 0$. Notice that $\phi(\infty)$ is taken to be $\lim_{y\to\infty} \phi(y) = \frac{a}{b}$. Notice also that $(-1/x)[-1, 1]$ is the interval $[1, -1]$ containing ∞.

An LFT is said to be refining for an interval $[p, q]$ if $\phi[p, q] \subseteq [p, q]$. We will need the following two propositions from [Pot97]:

Proposition 27 *Given two non-trivial intervals $[p, q]$ and $[r, s]$ with $p \neq q$ and $r \neq s$, there exists an LFT ϕ with $\phi[p, q] = [r, s]$.*

Proposition 28 *For LFTs ϕ and ψ we have $\phi[0, \infty] \supseteq \psi[0, \infty]$ iff $\psi = \phi\gamma$ for a positive LFT γ.*

Thus for any sequence of nested intervals $[p_0, q_0] \supseteq [p_1, q_1] \supseteq \ldots \supseteq [p_n, q_n] \supseteq \ldots$ we have $[p_n, q_n] = \phi_0\phi_1 \ldots \phi_n[0, \infty]$ where ϕ_0 is an LFT and all other ϕ_i's are positive LFTs.[9] Thus if

[9] We call $[p_n, q_n]$, the n^{th} convergent of the LFT sequence ϕ.

a sequence of nested intervals converges to a real number r, then the corresponding infinite sequence of LFTs or the corresponding infinite product of matrices represents r; and the initial finite subsequence of LFTs applied to the interval $[0, \infty]$ yields increasingly finer approximations to r.

LFTs are closely related to continued fractions; in fact, the continued fraction

$$a_0 + \cfrac{b_0}{a_1 + \cfrac{b_1}{\ddots}}$$

corresponds to the LFT $\begin{pmatrix} a_0 & b_0 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} a_1 & b_1 \\ 1 & 0 \end{pmatrix} \ldots$.

An LFT for the square root function is:

$$\sqrt{x} \equiv \prod_{n=0}^{\infty} \begin{pmatrix} x & x \\ 1 & x \end{pmatrix}$$

for $x \in [1, \infty]$. This differs slightly from the LFT specified in [Pot97, Pot99]. We establish its correctness below.

To see that this LFT ϕ is an LFT for the square root function, we first establish a bound on the length of the n^{th} convergent. We use the following notation: $\|[p, q]\| = q - p$ denotes the length of the interval $[p, q]$. The following two subsections show that $\|\phi^n[0, \infty]\| \to 0$ as $n \to \infty$.

3.8 Length of the n^{th} convergent

Let $M_i = \begin{pmatrix} a_i & c_i \\ b_i & d_i \end{pmatrix}$ and $P_i = \prod_{j=0}^{i-1} M_j = \begin{pmatrix} A_i & C_i \\ B_i & D_i \end{pmatrix}$. Then the length of the interval $[p_n, q_n] = \prod_{i=0}^{n} M_i[0, \infty] = P_n M_n[0, \infty]$ is given by:

$$\begin{aligned}
\|P_n M_n[0, \infty]\| &= \left\| \begin{pmatrix} A_n & C_n \\ B_n & D_n \end{pmatrix} \begin{pmatrix} a_n & c_n \\ b_n & d_n \end{pmatrix} ([0, \infty]) \right\| \\
&= \left\| \begin{pmatrix} A_n a_n + C_n b_n & A_n c_n + C_n d_n \\ B_n a_n + D_n b_n & B_n c_n + D_n d_n \end{pmatrix} ([0, \infty]) \right\| \\
&= \left| \frac{A_n a_n + C_n b_n}{B_n a_n + D_n b_n} - \frac{A_n c_n + C_n d_n}{B_n c_n + D_n d_n} \right| \\
&= \left| \frac{(A_n D_n - B_n C_n)(a_n d_n - b_n c_n)}{(B_n a_n + D_n b_n)(B_n c_n + D_n d_n)} \right|
\end{aligned}$$

3.9 Length of the n^{th} convergent for the Square Root

Using the notation above with $M_i = \begin{pmatrix} x & x \\ 1 & x \end{pmatrix}$, we get

$$\begin{aligned} \|[p_n, q_n]\| &= \left| \frac{(A_n D_n - B_n C_n)(x^2 - x)}{(xB_n + D_n)(xB_n + xD_n)} \right| \\ &< \left| \frac{(A_n D_n - B_n C_n)(x^2 - x)}{(xB_n)(xD_n)} \right| \\ &= \left| \left(\frac{A_n}{B_n} - \frac{C_n}{D_n} \right) \left(1 - \frac{1}{x} \right) \right| \\ &= \left| (q_{n-1} - p_{n-1}) \left(1 - \frac{1}{x} \right) \right| \end{aligned}$$

Thus, inductively, $q_n - p_n < |(q_0 - p_0)\left(1 - \frac{1}{x}\right)^n|$.

Thus $\phi^n(y) \to y_0$ for some y_0 and all $y \in [0, \infty]$. In particular, $\phi^n(y_0) \to y_0$ and thus $\phi^{n+1}(y) \to \phi(y_0)$ as $n \to \infty$. Thus, $\phi(y_0) = y_0$, so that

$$\frac{xy_0 + x}{y_0 + x} = y_0$$

Hence $x = y_0^2$.

This establishes that ϕ is a LFT for the square root function.

Lemma 29 *If $[p_n(a), q_n(a)]$ denotes the n^{th} convergent for the matrix sequence $M_1, M_2, \ldots$ where each $M_i = L(a) = \begin{pmatrix} a & a \\ 1 & a \end{pmatrix}$, then $q_n(a) - p_n(a) < a\left(1 - \frac{1}{a}\right)^{n+1}$. Thus if $a \in [1, 2]$, then $0 \leq q_n(a) - p_n(a) < 2^{-n}$, and for all $n, \sqrt{a} \in [p_n(a), q_n(a)]$. Furthermore, $p_n(a) = (L(a)^n)_{1,2}/(L(a)^n)_{2,2}$.*

Proof:

From the foregoing, we have that $q_n(x) - p_n(x) < |(q_0(x) - p_0(x))\left(1 - \frac{1}{x}\right)^n|$. But $[p_0(x), q_0(x)] = \begin{pmatrix} x & x \\ 1 & x \end{pmatrix}[0, \infty] = [1, x]$. This yields $q_n(x) - p_n(x) < (x - 1)\left(1 - \frac{1}{x}\right)^n \leq x\left(1 - \frac{1}{x}\right)\left(1 - \frac{1}{x}\right)^n = \left(1 - \frac{1}{x}\right)^{n+1}$.

The other parts of the lemma follow immediately. □

Proof:(of Theorem 26) Let $(\mathbf{a}, \sigma)$ be an input instance for SSQRT. Let α_i be a positive integer satisfying $2^{\alpha_i} < a_i \leq 2^{\alpha_i + 1}$. Further, let $a_i' = a_i / 2^{\alpha_i}$ be a rational in $(1, 2]$. Hence, by an application of Lemma 29, any number, say $p_M(a_i')$ in the M^{th} convergent interval of $L(a_i)$ approximates $\sqrt{a_i'}$ with an error of at most 2^{-M}. To obtain an approximation of $\sqrt{a_i}$ from this we need to multiply $p_M(a_i')$ by $2^{\lfloor \frac{\alpha_i}{2} \rfloor}$ (and if α_i is odd then we must also multiply

this by an approximation to $\sqrt{2}$ – which can also be approximated in this way by setting $a = 2$ and $\alpha = 0$).

How good an approximation is needed? That is, how large must M be? Tiwari has shown [Tiw92] that if a sum of n square roots $\pm\sqrt{a_i}$ is not zero, where each a_i has binary representation of length at most s, then the sum is bounded from below by

$$2^{-(s+1)2^n}$$

Thus taking $M = 2(\log n)(s+1)2^n$ and obtaining an approximation of each $\sqrt{a_i}$ to within 2^{-M} provides enough accuracy to determine the sign of the result. By Lemma 29, a suitable approximation is provided by $(L(a_i)^M)_{1,2}/(L(a_i)^M)_{2,2}$ (or – if α_i is odd – by the expression $(L(a_i)^M)_{1,2}(L(2)^M)_{1,2}/(L(a_i)^M)_{2,2}(L(2)^M)_{2,2})$. Denote this fraction by C_i/D_i. (Note that each $D_i > 0$.)

Note that

$$\begin{aligned}\sum_{i=1}^{n} \sigma_i\sqrt{a_i} > 0 \quad &\Leftrightarrow \quad \sum_{i=1}^{n} \sigma_i \frac{C_i}{D_i} > 0 \\ &\Leftrightarrow \quad \sum_{i=1}^{n} \sigma_i C_i \prod_{j\neq i} D_j > 0\end{aligned}$$

We will need to re-write the expression $\sum_{i=1}^{n} \sigma_i C_i \prod_{j\neq i} D_j$ in order to make it easier to evaluate. First, note that this expression is of the form $\sum_{i=1}^{n}\prod_{j=1}^{n} Z_{i,j}$ for integers $Z_{i,j}$ whose binary representation is of length less than 2^{n^2}. Thus this expression can written in the form $\sum_{i=1}^{n}\prod_{j=1}^{n}\sum_{k=1}^{2^{n^2}} b_{i,j,k}2^k$ where each $b_{i,j,k} \in \{-1, 0, 1\}$ is easily computable from the input and from the oracle Bit-2-MatPow. Via the distributive law, this can be rewritten as $\sum_{i=1}^{n}\sum_{(k_1,k_2,\ldots,k_n)\in[2^{n^2}]^n}\prod_{j=1}^{n} b_{i,j,k_j}2^{k_j}$.

Thus there is a function f computable in polynomial time with an oracle for Bit-2-MatPow that, on input $(\mathbf{a}, \sigma, i, k_1, k_2, \ldots, k_n, j, \ell)$ outputs the ℓ^{th} bit of the number $\prod_{j=1}^{n} b_{i,j,k_j}2^{k_j}$. (Namely, the algorithm queries the n oracle bits corresponding to b_{i,j,k_j} and combines this information with σ to obtain the sign $\in \{-1, 0, 1\}$, and computes the value of the exponent $\sum_j k_j$, and from this easily determines the value of bit ℓ of the binary representation.)

Since addition of m numbers, each consisting of m bits is computable in $\widehat{\mathsf{TC}}^0_1$, it is now immediate that the bits of this expression are computable in $\mathsf{PH}^{\mathsf{PP}}$. Thus in PH, using the bits of this expression as an oracle, one can determine if the number represented in this manner is positive or not. (Namely, is there some bit that is non-zero, and is the sign bit positive?) □

4 Computing Bits of Algebraic Numbers

In this section, we use the tools developed in the previous section to derive upper bounds on the problem of computing bits of algebraic numbers.

4.1 Mathematical Preliminaries

Definition 30 (Algebraic number, Minimal Polynomial, Degree) *A real number α is called* algebraic *if there is a univariate polynomial p with rational (or – equivalently – integer) coefficients such that $p(\alpha) = 0$. There is a unique monic*[10] *univariate polynomial of minimal degree (and rational coefficients) $p_\alpha(x)$ such that $p_\alpha(\alpha) = 0$; p_α is said to be the* defining polynomial *or the* minimal polynomial *of the algebraic number α. The* degree *of α is the degree of p_α. The roots of p_α are called the* Galois conjugates *of α.*

The minimal polynomial p_α is irreducible; all of the Galois conjugates of α have the same degree d. In particular, no root of $p = p_\alpha$ is also a root of the first derivatives of p (denoted p').

The Newton-Raphson method is a well-known algorithm for computing an approximation to a root of a univariate polynomial. For background, the reader can consult a textbook that explains the method, such as [Sta70, FB93].

Definition 31 *(Newton-Raphson) Given a polynomial p and a starting point x_0, recursively define:*

$$x_{i+1} = x_i - \frac{p(x_i)}{p'(x_i)},$$

whenever x_i is defined and $p'(x_i)$ is non-zero.

Note that if p is an irreducible polynomial (such as the minimal polynomial p_α for an algebraic number α), then there is an interval $I = [\alpha - \beta, \alpha + \beta]$ around α such that p' has no roots in I, since $p'(\alpha) \neq 0$ and p' has only finitely-many roots. In fact, it is known that if β is small enough, then for any starting point x_0 in I, the sequence $x_0, x_1, \ldots$ not only is infinite (because $p'(x_i) \neq 0$), but it converges "rapidly" to α. Let us make precise the notion of "rapid" convergence.

Let ϵ_i denote the error in the i^{th} iteration of Newton-Raphson: $\epsilon_i = |x_i - \alpha|$. We say that the Newton-Raphson sequence (with starting point x_0) *converges quadratically* (with parameter $M > 0$) if, for all $i \geq 0$, $\epsilon_{i+1} \leq M\epsilon_i^2$. Note that we can assume (pessimistically) that $M \geq 1$.

Much more is known about sufficient conditions on the size of the interval $I = [\alpha-\beta, \alpha+\beta]$ that are sufficient to guarantee quadratic convergence (for *every* choice of $x_0 \in I$, for the same parameter M), but for our purposes it is sufficient to know that this interval exists, and hence there is a rational number x_0 that we can use as the starting point for a Newton-Raphson sequence that converges quadratically to α. The constant x_0 will be hard-wired into our algorithm. We will impose some additional requirements on x_0; in particular, we will pick x_0 so that $\epsilon_0 = |x_0 - \alpha| < \min(\frac{1}{4}, \frac{1}{2M})$. With this restriction on x_0, a simple induction shows that, for all $i > 0$, $\epsilon_i \leq \frac{1}{2^{2^i}M} \leq \frac{1}{2^{2^i}}$.

Since the Newton-Raphson sequence converges quadratically, this means that the number of bits of accuracy is doubling with each iteration (so that, after a polynomial number of

[10] A univariate polynomial $p(x) = \sum_{i=0}^{d} a_i x^i$ is *monic* if $a_d = 1$.

iterations, the approximation is accurate to an exponential number of bits). But if the i^{th} bit of x_{n^c} is 0 (for example), do we really know that the i^{th} bit of α is 0? If the next 100 bits of x_{n^c} are all 1, it could still be possible that in our next underestimate x_{n^c+1}, the i-th bit would be 1, followed by 100 0's and then a 1. How far do we need to "look ahead", in order to be confident about the value of the i^{th} bit of α?

The answer is provided by Liouville's Theorem, which provides useful bounds on approximating algebraic numbers by rational numbers (see e.g. Shidlovskii[Shi89] or Yap [Yap00]).

Fact 32 *(Liouville's Theorem) If α is a real algebraic number of degree $d \geq 1$, then there exists a constant $c = c(\alpha) > 0$ such that the following inequality holds for any $\gamma \in \mathbb{Z}$ and $\beta \in \mathbb{N}$, $\gamma/\beta \neq \alpha$:*

$$\left|\alpha - \frac{\gamma}{\beta}\right| > \frac{c}{\beta^d}$$

Roth's theorem sharpens the inequality in Liouville's theorem:

Fact 33 *(Roth's Theorem [Rot55]) If α is a real irrational algebraic number then there exists a constant $c = c(\alpha, \epsilon) > 0$ such that the following inequality holds for any $\gamma \in \mathbb{Z}$ and $\beta \in \mathbb{N}$,*

$$\left|\alpha - \frac{\gamma}{\beta}\right| > \frac{c(\alpha, \epsilon)}{\beta^{2+\epsilon}}$$

Roth's theorem is optimal, in some ways: the number 2 in the exponent cannot be decreased.

The rest of this subsection is an adaptation of the corresponding material in [Yap10], which gives a logspace algorithm to compute the digits of π. In contrast to the development in [Yap10], we choose to utilize Liouville's Theorem for algebraic numbers, instead of the advanced arguments required for bounding the irrationality measure of π. We could throughout replace the use of Liouville's theorem by the much stronger and deeper Roth's theorem, but we prefer not to do so, in order to retain the elementary nature of the arguments. We now introduce some terminology and notation that we will be using (some of which is standard, and some of which was introduced in [Yap10]).

Definition 34 *Let α be a real number. Let $\{\alpha\} = \alpha - \lfloor\alpha\rfloor$ be the fractional part of α. Further, let $\{\alpha\}_n = \{2^n\alpha\}$ and let α_n be the n-th bit after the binary point.*

It is clear that $\alpha_n = 1$ iff $\{\alpha\}_{n-1} \geq \frac{1}{2}$. For algebraic numbers we can sharpen this:

Lemma 35 *(Adapted from [Yap10]) Let α be an irrational algebraic number of degree d, and let $c = c(\alpha)$ be the constant guaranteed by Liouville's theorem. Let $\delta_n = \frac{c}{2^{(d-1)n+1}}$. Then we have:*

- $\alpha_n = 1$ *iff* $\{\alpha\}_{n-1} > \frac{1}{2} + \delta_n$.
- $\alpha_n = 0$ *iff* $\{\alpha\}_{n-1} < \frac{1}{2} - \delta_n$.

Proof: Taking $\beta = 2^n$, and letting $\gamma \in \mathbb{N}$ be the number whose binary representation corresponds to the first n bits in the binary representation of α, Liouville's theorem implies that $|\alpha - 2^{-n}\gamma| > \frac{c}{2^{dn}}$, so $|2^n\alpha - \gamma| > \frac{c}{2^{(d-1)n}} = 2\delta_n$. Also, we have that $\gamma = 2\gamma' + \alpha_n$, where γ' is the first $n-1$ bits of the binary representation of α.

Thus $\{\alpha\}_{n-1} = \{2^{n-1}\alpha\} = 2^{n-1}\alpha - \gamma' = \frac{1}{2}(2^n\alpha - \gamma) + \frac{\alpha_n}{2}$. This is greater than $\frac{1}{2} + \delta_n$ if $\alpha_n = 1$. A similar calculation establishes the claim also in the case when $\alpha_n = 0$. $\square$

As a consequence of Lemma 35, in order to be confident that our approximation to α is giving us the correct value for the i^{th} bit of α, it is sufficient to have an approximation with error at most $\frac{1}{2^{O(i)}}$. In other words, there is a constant b such that, if the i^{th} bit of α is 1, then the binary representation of α will have another 1 appearing no later than position number $di + b$; there is a bound on how many consecutive 1's can appear in the binary representation of α. (And similarly, if the i^{th} bit of α is 0, there will be another 0 that appears not too much later.) More specifically, recalling that our starting point x_0 is in the interval I where Newton-Raphson converges quadratically to α, and where $\epsilon_0 = |x_0 - \alpha| < \min(\frac{1}{4}, \frac{1}{2M})$, this means that there is a constant $c \in \mathbb{N}$ such that, given an n-bit number N, the N^{th} bit of α is equal to the N^{th} bit of x_{cn}.

4.2 Algebraic Numbers in CH_5

In this section, we prove our main result concerning algebraic numbers.

Theorem 36 *Let $\alpha = \sum_{i=0}^{\infty} a_i 2^{-(i+1)}$ be an algebraic number, where each $a_i \in \{0,1\}$. Then the language $A_\alpha = \{j : a_j = 1\}$ is in $\mathsf{PH}^{\mathsf{CH}_3}$.*

Proof: Let x_0 be the starting point for the Newton-Raphson method as described in Section 4.1. Since x_0 is rational, let $x_0 = \frac{a}{b}$ for integers a and b. Let p be the minimal polynomial for α. Recall that the Newton-Raphson sequence is defined as $x_{i+1} = f(x_i) = x_i - \frac{p(x_i)}{p'(x_i)} = \frac{x_i p'(x_i) - p(x_i)}{p'(x_i)}$ for all i. Thus $x_2 = f(f(x_0))$ and more generally $x_i = f^{[i]}(x_0)$, where $f^{[i]}$ denotes the i-fold composition of f.

Consider an input string j of length n. As discussed in Section 4, $j \in A_\alpha$ if and only bit j of x_{cn} is equal to 1 (for some constant c). Our algorithm will create arithmetic circuits N and D (in polynomial time) so that the rational number x_{cn} is equal to $\frac{N(a,b)}{D(a,b)}$. Then we will use our $\mathsf{PH}^{\mathsf{CH}_3}$ algorithm from Corollary 18, to obtain the j^{th} bit of $x_{cn} = f^{[cn]}(a/b)$.

First, note that any polynomial $q(x) = \sum_{i=0}^{d} a_i x^i$, with *rational* coefficients, when applied to a rational input $x = \frac{y}{z}$, (where $z \neq 0$) can be written as the quotient of two bivariate *integer* polynomials $n_q(y,z)$ and $d_q(z)$:

$$q\left(\frac{y}{z}\right) = \sum_{i=0}^{d} a_i \frac{y^i}{z^i} \cdot \frac{z^{d-i}}{z^{d-i}} = \frac{\sum_{i=0}^{d} L a_i y^i z^{d-i}}{L z^d} = \frac{n_q(y,z)}{d_q(z)},$$

where L is the least common multiple of the denominators of the rational coefficients a_i.

We are especially interested in the polynomials $n_p, d_p, n_{p'}$ and $d_{p'}$. Thus

$$\begin{aligned} f(y/z) &= \frac{(y/z)p'(y/z) - p(y/z)}{p'(y/z)} \\ &= \frac{\frac{yn_{p'}(y,z)}{zd_{p'}(z)} - \frac{n_p(y,z)}{d_p(z)}}{\frac{n_{p'}(y,z)}{d_{p'}(z)}} \\ &= \frac{\frac{yn_{p'}(y,z)d_p(z) - zd_{p'}(z)n_p(y,z)}{zd_{p'}(z)d_p(z)}}{\frac{n_{p'}(y,z)}{d_{p'}(z)}} \\ &= \frac{(yn_{p'}(y,z)d_p(z) - zd_{p'}(z)n_p(y,z))d_{p'}(z)}{zd_{p'}(z)d_p(z)n_{p'}(y,z)} \\ &= \frac{F(y,z)}{G(y,z)}, \end{aligned}$$

for integer bivariate polynomials F and G.

Definition 37 *For a positive integer t, let the t-bicomposition of (F, G) be the pair of bivariate polynomials $(F^{[t]}, G^{[t]})$ defined as follows:*

$$F^{[1]}(y,z) = F(y,z), G^{[1]}(y,z) = G(y,z),$$

and,

$$\begin{aligned} F^{[t+1]}(y,z) &= F(F^{[t]}(y,z), G^{[t]}(y,z)), \\ G^{[t+1]}(y,z) &= G(F^{[t]}(y,z), G^{[t]}(y,z)). \end{aligned}$$

A straightforward induction shows that $f^{[t]}(y/z) = \frac{F^{[t]}(y,z)}{G^{[t]}(y,z)}$.

Recall that p is the minimal polynomial for α, and it does not depend on the input j to A_α. Similarly, the starting point $x_0 = \frac{a}{b}$ is a fixed constant. Thus there are arithmetic circuits $N_1(a,b)$ and $D_1(a,b)$ of size $O(1)$ computing $F(a,b)$, and $G(a,b)$, respectively. For each $t > 1$, the circuit $N_t(a,b)$ computing $F^{[t]}(a,b)$ can be constructed by running wires from the output gates of N_{t-1} and D_{t-1} to the a, b input gates, respectively, of a copy of the circuit for $N_1(a,b)$. The circuit for $D_t(a,b)$ is constructed similarly.

In this way, we construct in polynomial time the circuits $N(a,b) = N_{cn}(a,b)$ and $D(a,b) = D_{cn}(a,b)$ so that the rational number x_{cn} is equal to $\frac{N(a,b)}{D(a,b)}$, as desired. This completes the proof. □

4.3 Certain Transcendentals in CH_5

Our results on algebraic numbers made use of Liouville's Theorem, which shows that algebraic numbers can not be "too close" to rational numbers with small denominators. A similar property holds for several important transcendental numbers. This motivates the following definition:

Definition 38 *The* irrationality measure *of a transcendental number α is the infimum of*

$$\left\{ k : \textit{for all but finitely many } (\gamma, \beta) \in \mathbb{Z} \times \mathbb{N} \ \left| \alpha - \frac{\gamma}{\beta} \right| > \frac{1}{\beta^k} \right\}$$

The irrationality measure of π is no more than 7.10321 [ZZ20]. The irrationality measure of e is equal to 2 (see [BB87]).

Yap took as his starting point a remarkable expression for π, discovered by Borwein, Borwein, and Plouffe [BBP97]:

$$\pi = \sum_{k=0}^{\infty} \frac{120k^2 + 151k + 47}{(16)^k(2^9k^4 + 2^{10}k^3 + 712k^2 + 194k + 15)}$$

Borwein, Borwein, and Plouffe exhibited similar series for other transcendental numbers, such as $\log_{10} 2$ and π^2. Yap defined a series $\sum_{k=0}^{\infty} t_k$ to be *BBP-like* if there are integer polynomials p and q and an integer c so that $t_k = \frac{p(k)}{2^{ck}q(k)}$.

With these definitions in hand, we can state Yap's theorem:

Theorem 39 *[Yap10] If $\alpha = a_0a_1a_2\ldots$ is a transcendental real number that*

- *has finite irrationality measure, and*
- *can be expressed as a BBP-like series,*

then there is a logspace-computable function f_α such that $f_\alpha(1^n) = a_o a_1 \ldots a_n$.

We note that some of the motivation for the algorithms presented in [BBP97], as well as some of the exposition in [Yap10], comes from the ability to compute individual bits, without having to compute all of the earlier bits. In spite of this, the algorithm presented in [Yap10] does yield a logspace algorithm computing *all* of the first n bits.

We improve on Theorem 39, by placing the function f_α in the (seemingly) smaller complexity class TC^0. (We discuss additional improvements to Theorem 39 later in this section.)

Theorem 40 *If $\alpha = a_0a_1a_2\ldots$ is a transcendental real number that*

- *has finite irrationality measure, and*
- *can be expressed as a BBP-like series,*

then there is a TC^0-computable function f_α such that $f_\alpha(1^n) = a_o a_1 \ldots a_n$.

Proof: Yap showed in [Yap10] that the finite irrationality measure condition on α implies that, in order to compute $a_0a_1\ldots a_n$, it is sufficient to compute $\sum_{k=0}^{n^{c'}} t_k = \sum_{k=0}^{n^{c'}} \frac{p(k)}{2^{ck}q(k)}$ (for some constant c') to $n^{c'}$ bits of accuracy, and then output the first $n+1$ bits.

Let $T_k = \prod_{\{i \le n, i \neq k\}} 2^{ck}q(k)$. Then $\sum_{k=0}^{n^{c'}} t_k = \frac{\sum_{k=0}^{n^{c'}} T_k p(k)}{\prod_{k=0}^{n^{c'}} 2^{ck}q(k)}$.

The numerator and denominator are both computable in TC^0. Thus the theorem follows immediately by an application of Theorem 5. $\square$

Both [Yap10] and [BBP97] explicitly ask if the algorithms that they present for π and similar numbers hold only for the *binary* representation, or if they can be modified to yield algorithms for the base-b representations for other bases b. But, as observed in Note 12, our techniques work equally well for base 10 or any other base.

Similarly, our proof of Theorem 40 applies not only to series that satisfy Yap's BBP-like criterion, but for *any* series $\sum_{k=0}^{\infty} t_k$ that converges rapidly to a transcendental number with bounded irrationality measure, if $t_k = \frac{n(k)}{d(k)}$ and the functions n and d are computable by TC^0 circuits of size polynomial in n. They do not need to be polynomials (or polynomials multiplied by b^{ck}). For example, to the best of our knowledge, no BBP-like series is known to converge to Euler's constant e, but since $e = \sum_{k=0}^{\infty} \frac{1}{k!}$ where the error term $\epsilon_n = e - \sum_{k=0}^{n} \frac{1}{k!} < \frac{1}{n!}$, and where the numerator $n(k) = 1$ and denominator $d(k) = k!$ are easily computable by TC^0 circuits of size polynomial in n if $k = n^{O(1)}$, this shows that the bits of e can also be computed in TC^0. (This can also be derived from the earlier work of Reif and Tate [RT92], who showed that various numerical computations can be performed in P-uniform TC^0; their algorithms can be made dlogtime-uniform by appeal to [HAB02].)

Much of the motivation in [BBP97] (and, to a lesser extent, in [Yap10]) comes from efficient implementations of programs to compute various constants. We do not claim that the algorithms presented here lend themselves to efficient implementation. They merely show that certain computations can be performed in TC^0.

We now wish to prove a result that is analogous to Theorem 36, for various important transcendental numbers. However, in Theorem 36 we equate the characteristic sequence of a language with a real number in $[0, 1]$, whereas the constants such as π and e that we consider lie outside of this interval. Thus, when we say "$\pi \in \mathsf{CH}_k$" and "$e \in \mathsf{CH}_k$", we mean that the languages corresponding to the fractional parts of π and e ($\{\pi\}$ and $\{e\}$, respectively, using the notation introduced above) lie in CH_k.

First, we show that the transcendental numbers that Yap studied in [Yap10] all lie in $\mathsf{PH}^{\mathsf{CH}_3}$.

Theorem 41 *Let $\alpha = \sum_{k=0}^{\infty} t_k$ be a transcendental real number having finite irrationality measure and a BBP-like series. Then $\alpha \in \mathsf{PH}^{\mathsf{CH}_3}$.*

Proof: We follow the same basic strategy as in the proof of Theorem 36, but we make use of the TC^0 circuits constructed in Theorem 15.

Given an input string j of length n, our goal is to compute the j^{th} bit of the fractional part of α. As in Theorem 40, because of bounded irrationality measure and the rapid convergence of our series

$$\alpha = \sum_{k=0}^{\infty} t_k = \sum_{k=0}^{\infty} \frac{p(k)}{2^{ck} q(k)},$$

(for integer polynomials p and q), it suffices to compute the j^{th} bit of $\sum_{k=0}^{2^{c'n}} t_k$, for some constant c'. As in the proof of Corollary 18, we first make use of some polynomial-time

computation (in this case, it will be PH computation), to prepare the CRR-representation of an exponentially-large input instance for a TC^0 circuit.

First, let us assume that $t_k > 0$ for all k. (This is true for the series for π.) Then we will modify the algorithm for more general BBP-like series.

The input instances for Theorem 15 need to be of the form $\sum_{i=1}^{r} \frac{X_i}{Y_i}$, where there is a natural number t such that $2^{t-1} \leq Y_i < 2^t$ for all i. Using PH computation, we can compute $B = \max\{2^{ck}q(k) : 1 \leq k \leq 2^{c'n}\}$. To see this, note that the number $2^{ck}q(k)$ will require an exponential number of bits to write in binary for k at the high end of this range. However, the number ck requires only $n^{O(1)}$ bits, and similarly $q(k)$ requires only $n^{O(1)}$ bits. Thus each such number $2^{ck}q(k)$ can be expressed exactly as $2^{ck}q(k) = x_k \cdot 2^{e_k}$, for values (x_k, e_k) that are easy to compute, given k, and furthermore, it is easy to determine, given (x_k, e_k) and $(x_{k'}, e_{k'})$, if $2^{ck}q(k) \geq 2^{ck'}q(k')$. Thus, with a PH oracle, a polynomial-time machine can find k such that, for all k', it holds that $2^{ck}q(k) \geq 2^{ck'}q(k')$. This determines B. Then, with B in hand, we can compute t such that $2^{t-1} \leq B < 2^t$.

Given k, we can compute $p(k)$ and $q(k)$ such that $t_k = \frac{p(k)}{2^{ck}q(k)}$, and can compute a number b_k such that $2^{b_k-1} \leq 2^{ck}q(k) < 2^{b_k}$. Let $e_k = t - b_k$. Note that $t_k = \frac{2^{e_k}p(k)}{2^{e_k}2^{ck}q(k)}$, and for every $k, 2^{t-1} \leq 2^{e_k}2^{ck}q(k) < 2^t$.

Now, similar to the proof of Corollary 18, we can let f be the function that, on input $(x, y) = ((j, t), (k, b, p))$ outputs "p is not prime" if p is not prime, and otherwise outputs $2^{e_k}p(k) \bmod p$ if $b = 0$ and outputs $2^{e_k}2^{ck}q(k) \bmod p$ if $b = 1$. Thus, we now have the CRR representation that satisfies the input requirements of Theorem 15, and we can appeal to Proposition 2.

This completes the argument, in the case when $t_k > 0$ for all k. Now we consider the more general case.

Since p and q are polynomials, they each tend to either ∞ or $-\infty$ for large k. If they are both positive or both negative for large k, then $t_k > 0$ for all large k, and the analysis is very similar to what appears above. Namely, let $t_k > 0$ for all $k > K$ Let $Q = \sum_{k=0}^{K} t_k$. Then we can compute an approximation to $\sum_{k=K+1}^{\infty} t_k$ as above, and then simply add Q to the result.

In the remaining case, $t_k < 0$ for all large k. In that case, we can compute Q as above, and compute our underestimate A to $\sum_{k=K+1}^{\infty} |t_k|$ as above, and then the desired answer is $Q - \sum_{k=K+1}^{\infty} |t_k|$ – but note that $Q - A$ is an *overestimate* to α, which means that, if the j^{th} bit of $Q - A$ is 1, it is still possible that the j^{th} bit of α is 0. However, here we can once again appeal to the bounded irrationality measure of α. If we increase the accuracy of our approximation A (by summing up to $k = 2^{n^{c''}}$ for some $c'' > c$), we are guaranteed to obtain an approximation that yields the correct bit for j. □

Corollary 42 $\pi \in \mathsf{PH}^{\mathsf{CH}_3}$.

There has been considerable progress using the BBP framework since the original paper [BBP97] appeared. An extensive list can be found in [Bai17]. Many of the "BBP-like" series presented in [Bai17] do not actually fit Yap's definition of "BBP-like"; the definition

in [Yap10] requires that t_k be of the form $\frac{p(k)}{2^{ck}q(k)}$, whereas many of the series presented in [Bai17] have t_k of the form $\frac{p(k)}{\beta^k q(k)}$ for some integer β where $\beta \neq 2$. With some minor adjustments, we can accommodate this broader definition, too.

Corollary 43 *Let $\alpha = \sum_{k=0}^{\infty} t_k$ be a transcendental real number having finite irrationality measure, where there are integer polynomials p and q and integer β with $|\beta| \geq 2$, and $t_k = \frac{p(k)}{\beta^k q(k)}$. Then $\alpha \in \mathsf{PH}^{\mathsf{CH}_3}$.*

Proof: First we deal with the case when $\beta > 0$.

As above, given an input string j of length n, our goal is to compute the j^{th} bit of the fractional part of α, and once again it suffices to compute the j^{th} bit of $\sum_{k=0}^{2^{n^{c'}}} t_k$, for some constant c', and once again we begin with a PH computation, to prepare the CRR-representation of an exponentially-large input instance for a TC^0 circuit.

Now, however, we will compute $p(k)$ and $q(k)$ in β-ary notation, and begin by computing $B = \max\{\beta^k q(k) : 1 \leq k \leq 2^{n^{c'}}\}$, and then find t such that $\beta^{t-1} \leq B < \beta^t$, and let f be the function that, on input $(x, y) = ((j, t), (k, b, p))$ outputs "p is not prime" if p is not prime, and otherwise outputs $\beta^{e_k} p(k) \bmod p$ if $b = 0$ and outputs $\beta^{e_k} q(k) \bmod p$ if $b = 1$. Instead of the circuits that were presented in the proof of Theorem 15, we make use of the β-variant of those circuits (as defined in the footnotes to the proof of Theorem 5). This is because determining if a number B is less than β^t is easy in base β, but less easy in base 2. Note that these β-variant circuits still produce the final answer in binary; it is merely the case that some of the intermediate computations take place using powers of β rather than powers of 2.

The analysis proceeds precisely as in the preceding theorem.

Now, we consider the case where $\beta < 0$. In this case, note that $\alpha = \sum_{k=0}^{\infty} t_k = \sum_{\ell=0}^{\infty}(t_{2\ell} + t_{2\ell+1}) = \sum_{\ell=0}^{\infty}(\frac{p(2\ell)}{\beta^{2\ell} q(2\ell)} + \frac{p(2\ell+1)}{\beta^{2\ell+1} q(2\ell+1)}) = \sum_{\ell=0}^{\infty}(\frac{p(2\ell)q(2\ell+1)+p(2\ell+1)\beta q(2\ell)}{\beta^{2\ell+1} q(2\ell)q(2\ell+1)}) = \sum_{\ell=0} \frac{r(\ell)}{\gamma^\ell s(\ell)}$ for $\gamma = \beta^2 > 0$ and integer polynomials r and s. Thus this case reduces to the previous case.

□

We can, in principle, handle even more general terms t_k. Our proof requires only that $t_k = \frac{n(k)}{d(k)}$ for numerator and denominator functions n and d, such that $\beta^t n(k) \bmod p$ and $\beta^t d(k) \bmod p$ be "easy" to compute (in PH), and also that it be possible (in PH) to find t such that $\beta^{t-1} \leq d(k) < \beta^t$. We have no examples at hand, to show that this generalization is useful for transcendental reals of interest. In particular, we do not see how to give a $\mathsf{PH}^{\mathsf{CH}_3}$ upper bound for e.

It is clear from the discussion after Theorem 40 that $e \in \mathsf{CH}$. The "naïve approach" mentioned at the start of the proof of Theorem 15 leads to a proof that $e \in \mathsf{PH}^{\mathsf{CH}_4}$.

4.4 Lower Bound

In this section, we complement our upper bounds on algebraic numbers by giving (rather weak) lower bounds for certain algebraic numbers. We emphasize that our lower bounds are

only for *rational* numbers; we do not have any lower bounds at all for irrational algebraic numbers. Our lower bounds for rational numbers are fairly tight; we observe that all such numbers lie in $\mathsf{ACC}^0 = \bigcup_m \mathsf{AC}^0[m]$, and for any prime m there is a rational number that lies outside $\mathsf{AC}^0[m]$.

First, we show membership in ACC^0. The binary expansion of any rational number $\alpha \in [0,1]$ is ultimately periodic, meaning that it has the form $\alpha = \sum_{i=0}^{\infty} a_i 2^{-(i+1)}$ where the sequence $a_0, a_1, a_2, \ldots$ has the property that there is some k and some N such that, for all $j \geq N, a_j = a_{j+k}$. Let $S = \{\ell : N \leq \ell < N+k, a_\ell = 1\} = \{\ell_1, \ell_2, \ldots \ell_r\}$ for some $r < k$. Thus the language $A_\alpha = \{x : a_x = 1\}$ is equal to $F \cup L_1 \cup \ldots \cup L_r$, where F is a finite set (of some strings that lexicographically precede N) and $L_i = \{\ell_i + ck : c \in \mathbb{N}\}$. Each L_i is therefore a *linear set*, and A is a *semi-linear set* (defined as the union of linear sets). Barrington and Corbett [BC91] showed that all semi-linear sets lie in ACC^0. Thus we have shown:

Theorem 44 *[BC91] Let α be a rational number in $[0,1]$. Then the language $A_\alpha \in \mathsf{ACC}^0$.*

Now we turn to a lower bound.

Lemma 45 *For a given odd modulus m, there is a rational number α_m whose language A_{α_m} is hard for $\mathsf{AC}^0[m]$ under AC^0-Turing reductions. More precisely, the MOD$_m$ function reduces to A_{α_m} under Dlogtime-uniform projections.*

Proof: Let m be an odd number, $m > 1$, and let α_m be the rational corresponding to the language $A_{\alpha_m} = \{cm : c \in \mathbb{N}\}$.

The well-known Carmichael function $\lambda(m)$ from number theory has the property that, for any odd $m > 1$, $\lambda(m) > 1$ and $2^{\lambda(m)} \equiv 1 \pmod{m}$.

The MOD$_m$ function is defined so that MOD$_m(x) = 1$ if the number of 1's in x is a multiple of m, and MOD$_m(x) = 0$ otherwise. Thus our goal is to take x as input and produce as output a number $f(x)$ that is a multiple of m if and only if the number of 1's in x is a multiple of m. We make use of a construction that was used earlier in [ASS01, BL87]. If $x = x_0 x_1 \ldots x_n$, let $f(x)$ be the number with binary representation

$$x_n 0^d x_{n-1} 0^d x_{n-2} \ldots x_2 0^d x_1 0^d x_0$$

where $d + 1 = \lambda(m)$. Thus $f(x) = \sum_{i=0}^{n} 2^{i\lambda(m)} x_i$, which is equivalent to $\sum_{i=0}^{n} x_i \bmod m$. It is clear that f is easy to compute via a Dlogtime-uniform projection. The lemma follows, since the MOD$_m$ language is complete for $\mathsf{AC}^0[m]$ under AC^0-Turing reductions. □

Corollary 46 *For every prime m, there is a rational number β_m such that $A_{\beta_m} \notin \mathsf{AC}^0[m]$.*

Proof: Let p be any prime other than m. If the language A_{α_p} from Lemma 45 were in $\mathsf{AC}^0[m]$, then the MOD$_p$ language would also be in $\mathsf{AC}^0[m]$, contrary to the lower bounds of [Raz87, Smo87]. □

The restriction to prime moduli in Corollary 46 is essential, given the current state of lower bound techniques. It is still not known whether $\mathsf{NP} = \mathsf{AC}^0[6]$. The best lower bounds against $\mathsf{AC}^0[m]$ for composite m are those of [All99, MW20].

It should perhaps be mentioned that there are rationals in AC^0, other than dyadic rationals. For instance, $\frac{1}{3} = 010101\ldots$ corresponds to the set given by the regular expression $(0 \cup 1)^*1$, which is clearly in AC^0. Since every rational number corresponds to a regular language, and since there are nice characterizations of the regular languages in AC^0 [BCST92, CS01], there is probably a very crisp characterization of the rational numbers in AC^0.

5 Open Questions and Discussion

Is conversion from CRR to binary in dlogtime-uniform $\widehat{\mathsf{TC}}^0_1$*?* This problem has been known to be in P-uniform $\widehat{\mathsf{TC}}^0_1$ starting with the seminal work of Beame, Cook, and Hoover [BCH86], but the subsequent improvements on the uniformity condition [CDL01, HAB02] introduced additional complexity that translated into increased depth. We have been able to reduce the majority-depth (relative to the construction in [HAB02]) by rearranging the algorithmic components introduced in this line of research, but it appears to us that a fresh approach will be needed, in order to decrease the depth further.

Is BitSLP *in* $\mathsf{PH}^{\mathsf{PP}}$*?* An affirmative answer to the first question implies an affirmative answer to the second, and this would pin down the complexity of BitSLP between $\mathsf{P}^{\#\mathsf{P}}$ and $\mathsf{PH}^{\mathsf{PP}}$. We have not attempted to determine a small value of k such that $\mathsf{BitSLP} \in (\Sigma^p_k)^A$ for some set $A \in \mathsf{CH}_3$, because we suspect that BitSLP does reside lower in CH, and any improvement in majority-depth will be more significant than optimizing the depth of AC^0 circuitry, since $\mathsf{PH} \subseteq \mathsf{P}^{\mathsf{PP}}$.

Is PosSLP *in* PH*?* Some interesting observations related to this problem were announced recently [Ete13, JS12].

Is it easy to compute bits of large powers of small matrices? We remark in this regard, that there are some surprising things that one can compute, regarding large powers of integers [HKR10] and the most significant bits of 2×2 matrices [GOW15].

Does every transcendental real with a BBP-like series have finite irrationality measure?

Is $e \in \mathsf{PH}^{\mathsf{CH}_3}$*?* Is there any fundamental reason why e should be more complex than π in this sense?

Is any irrational algebraic number in PH*? Is every irrational algebraic number in* AC^0*?* We strongly conjecture that the answer to the second question is "no", and we suspect that the answer to the first question is also "no", although there is absolutely no evidence to support either conjecture. We think that it would be very instructive see an example of an irrational algebraic number that is outside of AC^0. It would be remarkable and important, if it should turn out that irrational algebraic numbers reside in $\mathsf{PH}^{\mathsf{CH}_3} - \mathsf{PH}$.

Acknowledgments

The first author acknowledges the support of NSF grants CCF-1909216 and CCF-1909683. The second and the third authors were partially funded by a grant from Infosys Foundation. We would like to thank anonymous referees for help in improving the presentation of the paper, and we thank Howard Straubing and Bruno Grenet for helpful comments.

References

[AAD00] Manindra Agrawal, Eric Allender, and Samir Datta. On TC^0, AC^0, and Arithmetic circuits. *Journal of Computer and System Sciences*, 60(2):395–421, 2000.

[AB07] Boris Adamczewski and Yann Bugeaud. On the complexity of algebraic numbers I. expansions in integer bases. *Annals of Mathematics*, pages 547–565, 2007.

[AB09] S. Arora and B. Barak. *Computational complexity: a modern approach*, volume 1. Cambridge University Press, 2009.

[ABD14] Eric Allender, Nikhil Balaji, and Samir Datta. Low-depth uniform threshold circuits and the bit-complexity of straight line programs. In *Mathematical Foundations of Computer Science (MFCS)*, volume 8635, pages 13–24. Springer, 2014.

[ABKPM09] Eric Allender, Peter Bürgisser, Johan Kjeldgaard-Pedersen, and Peter Bro Miltersen. On the complexity of numerical analysis. *SIAM J. Comput.*, 38(5):1987–2006, 2009.

[ABL04] Boris Adamczewski, Yann Bugeaud, and Florian Luca. Sur la complexité des nombres algébriques. *Comptes Rendus Mathematique*, 339(1):11–14, 2004.

[ACLG20] Boris Adamczewski, Julien Cassaigne, and Marion Le Gonidec. On the computational complexity of algebraic numbers: the Hartmanis–Stearns problem revisited. *Transactions of the American Mathematical Society*, 373(5):3085–3115, 2020.

[All99] Eric Allender. The permanent requires large uniform threshold circuits. *Chicago J. Theor. Comput. Sci.*, 1999, 1999.

[All04] Eric Allender. The division breakthroughs. In Gheorghe Paun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Current Trends in Theoretical Computer Science, The Challenge of the New Century, Vol. 1: Algorithms and Complexity*, pages 147–164. World Scientific, 2004.

[AS03] Jean-Paul Allouche and Jeffrey Shallit. *Automatic sequences : theory, applications, generalizations*. Cambridge University Press, 2003.

[AS05] Eric Allender and Henning Schnorr. The complexity of the BitSLP problem. Unpublished Manuscript, 2005.

[ASS01] Eric Allender, Michael E. Saks, and Igor E. Shparlinski. A lower bound for primality. *Journal of Computer and System Sciences*, 62(2):356–366, 2001.

[Bai17] David H. Bailey. A compendium of BBP-type formulas for mathematical constants. Report, Lawrence Berkeley National Laboratory, Berkeley, CA, USA, August 2017.

[BB87] Jonathan M. Borwein and Peter B. Borwein. *Pi and the AGM: a study in the analytic number theory and computational complexity*. Wiley-Interscience, 1987.

[BBP97] David Bailey, Peter Borwein, and Simon Plouffe. On the rapid computation of various polylogarithmic constants. *Mathematics of Computation*, 66(218):903–913, 1997.

[BC91] David A. Mix Barrington and James C. Corbett. A note on some languages in uniform ACC^0. *Theor. Comput. Sci.*, 78(2):357–362, 1991.

[BCH86] Paul W. Beame, Stephen A. Cook, and H. James Hoover. Log depth circuits for division and related problems. *SIAM Journal on Computing*, 15:994–1003, 1986.

[BCST92] David A. Mix Barrington, Kevin J. Compton, Howard Straubing, and Denis Thérien. Regular languages in NC1. *Journal of Computer and System Sciences*, 44(3):478–499, 1992.

[BH21] Vasco Brattka and Peter Hertling. *Handbook of Computability and Complexity in Analysis*. Springer, 2021.

[BL87] Ravi B. Boppana and J. C. Lagarias. One-way functions and circuit complexity. *Information and Computation*, 74(3):226–240, 1987.

[CDL01] Andrew Chiu, George I. Davida, and Bruce E. Litow. Division in logspace-uniform NC^1. *RAIRO Theoretical Informatics and Applications*, 35(3):259–275, 2001.

[CMTV98] Hervé Caussinus, Pierre McKenzie, Denis Thérien, and Heribert Vollmer. Nondeterministic NC^1 computation. *Journal of Computer and System Sciences*, 57:200–212, 1998.

[COW13] Ventsislav Chonev, Joël Ouaknine, and James Worrell. The orbit problem in higher dimensions. In *Proc. Symposium on Theory of Computing Conference (STOC)*, pages 941–950. ACM, 2013.

[CS01] Kevin J. Compton and Howard Straubing. Characterizations of regular languages in low level complexity classes. In Gheorghe Paun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Current Trends in Theoretical Computer Science, Entering the 21th Century*, pages 235–246. World Scientific, 2001.

[DHNS03] Rod G Downey, Denis R Hirschfeldt, André Nies, and Frank Stephan. Trivial reals. In *Proceedings of the 7th and 8th Asian Logic Conferences*, pages 103–131. World Scientific, 2003.

[DMS94] Paul Dietz, Ioan Macarie, and Joel Seiferas. Bits and relative order from residues, space efficiently. *Information Processing Letters*, 50(3):123–127, 1994.

[DP12] Samir Datta and Rameshwar Pratap. Computing bits of algebraic numbers. In *Proc. 9th Theory and Applications of Models of Computation (TAMC)*, volume 7287 of *Lecture Notes in Computer Science*, pages 189–201. Springer, 2012.

[EP97] A. Edalat and Peter M. Potts. A new representation for exact real numbers. *Electronic Notes in Theoretical Computer Science*, 6:119–132, 1997.

[Ete13] Kousha Etessami. Probability, recursion, games, and fixed points. Talk presented at Horizons in TCS: A Celebration of Mihalis Yannakakis' 60th Birthday, 2013.

[EY10] Kousha Etessami and Mihalis Yannakakis. On the complexity of Nash equilibria and other fixed points. *SIAM J. Comput.*, 39(6):2531–2597, 2010.

[FB93] J. Douglas Faires and Richard L. Burden. *Numerical Methods*. PWS Publishing, Boston, 1993.

[Fre12] Rusins Freivalds. Hartmanis-Stearns conjecture on real time and transcendence. In Michael J. Dinneen, Bakhadyr Khoussainov, and André Nies, editors, *Computation, Physics and Beyond - International Workshop on Theoretical Computer Science, WTCS 2012, Dedicated to Cristian S. Calude on the Occasion of His 60th Birthday, Revised Selected and Invited Papers*, volume 7160 of *Lecture Notes in Computer Science*, pages 105–119. Springer, 2012.

[GK98] Mikael Goldmann and Marek Karpinski. Simulating threshold circuits by majority circuits. *SIAM J. Comput.*, 27(1):230–246, 1998.

[GOW15] Esther Galby, Joël Ouaknine, and James Worrell. On matrix powering in low dimensions. In *Proc. 32nd International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 30 of *LIPIcs*, pages 329–340. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.

[HAB02] William Hesse, Eric Allender, and David A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *Journal of Computer and System Sciences*, 65:695–716, 2002.

[HKR10] Mika Hirvensalo, Juhani Karhumäki, and Alexander Rabinovich. Computing partial information out of intractable: Powers of algebraic numbers as an example. *Journal of Number Theory*, 130:232–253, 2010.

[HS65] Juris Hartmanis and Richard E Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.

[HV06] Alexander Healy and Emanuele Viola. Constant-depth circuits for arithmetic in finite fields of characteristic two. In *Proc. 23rd International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 3884 of *Lecture Notes in Computer Science*, pages 672–683. Springer, 2006.

[Jeř12] Emil Jeřábek. Root finding with threshold circuits. *Theoretical Computer Science*, 462:59–69, 2012.

[JS12] Gorav Jindal and Thatchaphol Saranurak. Subtraction makes computing integers faster. *CoRR*, abs/1212.2549, 2012.

[KF82] Ker-I Ko and Harvey Friedman. Computational complexity of real functions. *Theor. Comput. Sci.*, 20:323–352, 1982.

[Ko83] Ker-I Ko. On the definitions of some complexity classes of real numbers. *Mathematical Systems Theory*, 16(2):95–109, 1983.

[KP07] Pascal Koiran and Sylvain Perifel. The complexity of two problems on arithmetic circuits. *Theor. Comput. Sci.*, 389(1-2):172–181, 2007.

[KP11] Pascal Koiran and Sylvain Perifel. Interpolation in Valiant's theory. *Computational Complexity*, 20(1):1–20, 2011.

[KS12] Neeraj Kayal and Chandan Saha. On the sum of square roots of polynomials and related problems. *ACM Transactions on Computation Theory*, 4(4):9:1–9:15, 2012.

[LOW15] Antonia Lechner, Joël Ouaknine, and James Worrell. On the complexity of linear arithmetic with divisibility. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 667–676. IEEE Computer Society, 2015.

[LR13] Richard J. Lipton and Kenneth W. Regan. *People, Problems, and Proofs - Essays from Gödel's Lost Letter: 2010*. Springer, 2013.

[Mil04] Joseph S. Miller. Every 2-random real is Kolmogorov random. *Journal of Symbolic Logic*, 69(3):907–913, 2004.

[MP00] Carlo Mereghetti and Beatrice Palano. Threshold circuits for iterated matrix product and powering. *ITA*, 34(1):39–46, 2000.

[MT98] Alexis Maciel and Denis Thérien. Threshold circuits of small majority-depth. *Inf. Comput.*, 146(1):55–83, 1998.

[MW20] Cody D. Murray and R. Ryan Williams. Circuit lower bounds for nondeterministic quasi-polytime from a new easy witness lemma. *SIAM Journal on Computing*, 49(5), 2020.

[NST05] André Nies, Frank Stephan, and Sebastiaan A Terwijn. Randomness, relativization and Turing degrees. *The Journal of Symbolic Logic*, 70(2):515–535, 2005.

[NV18] Masaki Nakanishi and Marcos Villagra. Computational complexity of space-bounded real numbers. *CoRR*, abs/1805.02572, 2018.

[OW14a] Joël Ouaknine and James Worrell. On the positivity problem for simple linear recurrence sequences,. In *Proc. 41st International Colloquium on Automata, Languages, and Programming (ICALP), Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 318–329. Springer, 2014.

[OW14b] Joël Ouaknine and James Worrell. Positivity problems for low-order linear recurrence sequences. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 366–379. Society for Industrial and Applied Mathematics, 2014.

[Pot97] Peter M. Potts. Efficient on-line computation of real functions using exact floating point. *Manuscript, Dept. of Computing, Imperial College, London*, 1997.

[Pot99] Peter M. Potts. *Exact Real Arithmetic using Möbius Transformations*. PhD thesis, Imperial College, University of London, 1999.

[Raz87] Alexander A. Razborov. Lower bounds on the size of bounded depth networks over a complete basis with logical addition. *Matematicheskie Zametki*, 41:598–607, 1987. In Russian. English translation in *Mathematical Notes of the Academy of Sciences of the USSR* 41:333–338, 1987.

[Rot55] Klaus Friedrich Roth. Rational approximations to algebraic numbers. *Mathematika. A Journal of Pure and Applied Mathematics*, 2:1–20, 1955.

[RT92] John H. Reif and Stephen R. Tate. On threshold circuits and polynomial computation. *SIAM J. Comput.*, 21(5):896–908, 1992.

[She07] Alexander A. Sherstov. Powering requires threshold depth 3. *Inf. Process. Lett.*, 102(2-3):104–107, 2007.

[Shi89] Andrei B. Shidlovskii. *Transcendental Numbers*. de Gruyter, New York, 1989.

[Smo87] R. Smolensky. Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In *Proceedings, 19th ACM Symposium on Theory of Computing*, pages 77–82, 1987.

[SR94] Kai-Yeung Siu and Vwani P. Roychowdhury. On optimal depth threshold circuits for multiplication and related problems. *SIAM J. Discrete Math.*, 7(2):284–292, 1994.

[Sta70] Peter Stark. *Introduction to Numerical Methods*. Macmillan, 1970.

[Tiw92] Prasoon Tiwari. A problem that is easier to solve on the unit-cost algebraic RAM. *J. Complexity*, 8(4):393–397, 1992.

[Tod91] Seinosuke Toda. PP is as hard as the polynomial time hierarchy. *SIAM J. Comput.*, 20:865–877, 1991.

[Tur36] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Soc.*, 2(42):230–265, 1936.

[Vol99] H. Vollmer. *Introduction to Circuit Complexity*. Springer-Verlag, 1999.

[Weg93] Ingo Wegener. Optimal lower bounds on the depth of polynomial-size threshold circuits for some arithmetic functions. *Inf. Process. Lett.*, 46(2):85–87, 1993.

[Wei00] Klaus Weihrauch. *Computable Analysis - An Introduction*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2000.

[Yap00] Chee Yap. *Fundamental Problems in Algorithmic Algebra*. Oxford University Press, 2000.

[Yap10] Chee Yap. Pi is in log space. manuscript available at https://cs.nyu.edu/exact/doc/pi-log.pdf, June 2010.

[YDD04] Liang Yu, Decheng Ding, and Rodney Downey. The Kolmogorov complexity of random reals. *Annals of Pure and Applied Logic*, 129(1-3):163–180, 2004.

[YK13] Fuxiang Yu and Ker-I Ko. On logarithmic-space computable real numbers. *Theoretical Computer Science*, 469:127–133, 2013.

[ZZ20] Doron Zeilberger and Wadim Zudilin. The irrationality measure of π is at most 7.103205334137.... *Moscow Journal of Combinatorics and Number Theory*, 9(4):407–419, 2020.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il