Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 177 (2013)

Low-Depth Uniform Threshold Circuits and the Bit-Complexity of Straight Line Programs

Eric Allender[1], Nikhil Balaji[2], and Samir Datta[2]

[1] Department of Computer Science, Rutgers University, USA
allender@cs.rutgers.edu
[2] Chennai Mathematical Institute, India
{nikhil,sdatta}@cmi.ac.in

Abstract. We present 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.

1 Introduction

How hard is it to compute the 10^{100}-th bit of the binary expansion of $\sqrt{2}$? Datta and Pratap [DP12], and Jeřábek [Jeř12] considered the question of computing the m-th bit of an algebraic number. Jeřábek [Jeř12] showed that this problem has uniform TC^0 circuits[3] of size polynomial in m (which is not so useful when $m = 10^{100}$). Earlier, Datta and Pratap showed a related result: when m is expressed in *binary*, this problem lies in the counting hierarchy. More precisely, Datta and Pratap showed that this problem is reducible to the problem of computing certain bits of the quotient of two numbers represented by arithmetic circuits of polynomial size.[4] Thus, we are led to the problem of evaluating arithmetic circuits. **In this paper, we focus on arithmetic circuits *without input variables.*** Thus an arithmetic circuit is a (possibly very compact) representation of a number.

Arithmetic circuits of polynomial size can produce numbers that require exponentially-many bits to represent in binary. The problem[5] known as BitSLP ($= \{(C, i, b) :$ the i-th bit of the number represented by arithmetic circuit C is $b\}$) is known to be hard for $\#\mathsf{P}$ [ABKPM09]. It was known that BitSLP lies in the counting hierarchy [ABKPM09], but the best previously-known bound for this problem is the bound mentioned in [ABKPM09] and credited there to [AS05]: $\mathsf{PH}^{\mathsf{PP}^{\mathsf{PP}^{\mathsf{PP}^{\mathsf{PP}}}}}$. That bound follows via a straightforward translation of a uniform TC^0 algorithm presented in [HAB02].

In this paper, we improve this bound on the complexity of BitSLP to $\mathsf{PH}^{\mathsf{PP}^{\mathsf{PP}^{\mathsf{PP}}}}$. In order to do this, we present improved uniform TC^0 algorithms for a number of problems that were already known to reside in uniform TC^0. The improvements that we provide are related to the *depth* of the TC^0 circuits. 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]. In this model, circuits have unbounded-fan-in AND, OR, and MAJORITY gates (as well as NOT gates). The class $\widehat{\mathsf{TC}}^0_d$ consists of functions computable by families

[3] For somewhat-related TC^0 algorithms on sums of radicals, see [HBM+10].

[4] It is mistakenly claimed in [DP12] that this problem lies in $\mathsf{PH}^{\mathsf{PP}^{\mathsf{PP}}}$. In this paper, we prove the weaker bound that it lies in $\mathsf{PH}^{\mathsf{PP}^{\mathsf{PP}^{\mathsf{PP}}}}$.

[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.

ISSN 1433-8092

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, it is necessary to define the levels of the counting hierarchy.

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

Proposition 1. *(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 $C_{|x|+2^{|x|^k}}(x, f(x,1)f(x,2)\dots f(x,2^{|x|^k}))$ accepts. Then $A \in \mathsf{PH}^{\mathsf{CH}_d}$.*

(One important part of the proof of Proposition 1 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 1. 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. 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 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.

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

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].

All of the TC^0 algorithms that are known for the problems considered in this paper rely on partial evaluations or approximations. The technical innovations in our improved algorithms rely on introducing yet another approximation, as discussed in Lemmas 3 and 4.

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 [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; see the final section of [OW14]. We show that this version of matrix powering is in $\mathsf{PH}^{\mathsf{PP}^{\mathsf{PP}^{\mathsf{PP}}}}$.

In addition to BitSLP, there has also been interest in the related problem PosSLP (= $\{C$: the number represented by arithmetic circuit C is positive$\}$) [EY10,KP07,KS12,KP11]. $\mathsf{PosSLP} \in \mathsf{PH}^{\mathsf{PP}^{\mathsf{PP}}}$, 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.

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 n-bit integer j, output the j^{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]. Might matrix powering also lie in PH?

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 this problem via $\mathsf{PH}^{\mathsf{PP}}$-Turing reductions.

2 Preliminaries

Given a list of primes $\Pi = (p_1, \ldots, p_m)$ and a number X, the CRR_Π representation of X is the list ($X \bmod p_1, \ldots, X \bmod p_m$). We omit the subscript Π if context makes clear. For more on circuit complexity classes such as $\mathsf{AC}^0, \mathsf{TC}^0, \mathsf{NC}^1$, as well as a discussion of dlogtime uniformity, see [Vol99]. For background on other complexity classes such as $\mathsf{PP}, \#\mathsf{P}, \mathsf{NP}$, etc., consult a standard text such as [AB09].

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 the 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 $\sum_{i=-m}^{\lfloor \log(X/Y) \rfloor} a_i 2^i$.*

The following lemma is a list of useful subroutines of problems that are computable in AC^0 and $\widehat{\mathsf{TC}}_1^0$.

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

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

1. Follows from Lemma 4.2 and Corollary 6.2 in [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 $x^{(p-1)/2} \not\equiv 1 \bmod p$ 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 Uniform Circuits for Division

Theorem 1. *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 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 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 $\widetilde{V}(X, Y) \leq W(X, Y) < X/Y$. 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 such that $2^{t-1} \leq Y < 2^t$.

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 2. *(Adapted from [DP12]) 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 1, 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, as well as finding generators mod p. 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.) □

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 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 4 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 $2^{2(n+1)t} W(X,Y)$ is an integer with the same significant bits as $W(X,Y)$.

Lemma 3. *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 underestimates 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,ABKPM09]).

Instead, we will compute a good approximation $\widetilde{A'}_{\Pi}$ to the fractional part of

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

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. Since we are adding up $2(n+1)|\Pi|$ approximate quantities it suffices to compute each of them to $b_m = 2n^b + 2(n+1)|\Pi|$ bits of accuracy to ensure:

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

Now we analyze the complexity. By Lemma 2, 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 1, 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

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

and strip off the integer part, to obtain the desired approximation. □

Corollary 1. *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 $\widehat{\mathsf{TC}}^0_1$*

Before presenting our approximation $\widetilde{V}(X,Y)$, first we present a claim, which helps motivate the definition.

Claim. Let Π_i for $i \in \{1, \ldots, n^c\}$ be n^c 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 $c, d : 3 \leq c < d$). Let $\Pi = \cup_{i=1}^{n^c} \Pi_i$. Then, for any value A, it holds that

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

The claim follows immediately from Proposition 3, which is provided in the appendix for completeness.

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

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

Note that

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

and

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

Thus $X/Y - \widetilde{V}(X, Y) = (X/Y - W(X, Y) + (W(X, Y) - 2^{n^c} V(X, Y)) + (2^{n^c} V(X, Y) - \widetilde{V}(X, Y)) < 2^{-(n+1)} + n^c 2^{2n} / 2^{n^c} + 2^{n^c} / 2^{n^{2c}} < 1/Y$.

Lemma 4. *Let Π_i for $i \in \{1, \ldots, n^c\}$ be n^c 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 $c, d : 3 \leq c < d$). Let $\Pi = \cup_{i=1}^{n^c} \Pi_i$. Then, given X, Y and the Π_i, we can compute $\widetilde{V}(X, Y)$ in $\widehat{\mathsf{TC}}^0_3$.*

Proof. Via Lemma 1, 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 2). Also, as in Lemma 3, we can compute the values h_p^Π for each prime p.

Then, via Lemma 1, 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 1, to obtain $\widetilde{V}(X, Y)$. □

This completes the proof of Theorem 1. □

Corollary 2. *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, the binary representation of Z can be computed in $\widehat{\mathsf{TC}}^0_3$*

Proof. Recall from the proof of Theorem 1 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 4, 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. □

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 1 where the values X and Y are presented in CRR notation.

Theorem 2. *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 4.

The algorithm is much the same as in Theorem 1, 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 5. *(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 1 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 4, in majority-depth one we can compute h_p^Π, 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 $2^{2(n+1)t}W(X,Y)$.

Thus, after majority-depth two, we have obtained the same partial results as in the proof of Lemma 4, and the rest of the algorithm is thus identical. □

Proposition 2. *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 computes from CRR to binary, as in Corollary 2.

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

3.1 Consequences for the Counting Hierarchy

Corollary 3. $\mathsf{BitSLP} \in \mathsf{PH}^{\mathsf{PP}^{\mathsf{PP}^{\mathsf{PP}}}}$.

Proof. This is immediate from Proposition 1 and Corollary 2.

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 2, satisfies the hypothesis of Proposition 1. (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 2, we obtain:

Corollary 4. *$\{(C_X, C_Y, i)$: the i-th bit of the quotient X/Y, where X and Y are represented by arithmetic circuits C_X and C_Y, respectively, is in $\mathsf{PH}^{\mathsf{PP}^{\mathsf{PP}^{\mathsf{PP}}}}$.*

4 Integer Powering

In this section, we present an alternative algorithm for integer powering, which serves to illustrate the approach that we take to matrix powering.

Theorem 3. *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 1.
2. Compute X^m by reducing via Fermat's little theorem. 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 1. In parallel, compute the first two phases of our uniform algorithm to convert CRR to binary (Corollary 2).
3. At this stage, we have the answer encoded in CRR, and we invoke the final layer of the circuit from Corollary 2, convert the answer to binary.

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

5 Powering

We investigate the complexity of integer powering and powering constant size matrices from the perspective of optimizing the majority depth. We present TC^0 circuits with majority depth three for both these problems.

Since iterated integer product is in uniform $\widehat{\mathsf{TC}}^0_3$, by Proposition 2, it is immediate that integer powering is also in this class.

5.1 Integer Matrix Powering

Theorem 4. *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, 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, we just have to compute $r(A)$.

Lemma 6. *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 1. Since addition, multiplication, and powering of $O(1)$ numbers of $O(\log n)$ bits is computable in AC^0 (by Lemma 1, items 3, 4 and 6), it follows that the coefficients of the characteristic polynomial can be computed in $\widehat{\mathsf{TC}}^0_1$.

Lemma 7. *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 1. □

Lemma 8. *(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 $\tilde{f}_R(x)$ 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 could 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 4)

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 in to n^c sets Π_i that are pairwise disjoint, i.e., $\Pi = \cup_{i=1}^{n^c}\Pi_i$), 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 all the n^c sets that constitute Π, and (3) the CRR of the characteristic polynomial of A (via appeal to Lemma 6).
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 8.
3. At this point, by Lemma 7, 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 2.

□

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

In this section, we present a reduction, showing that the Sum-of-Square-Roots problem is reducible, in some sense, to the problem of computing large powers of 2-by-2 integer matrices. First, we define the problems of interest:

Definition 2. [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:*

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

Definition 3. $\mathsf{Bit\text{-}2\text{-}MatPow}(A, N, B, i, j)$ *(where* A *is a* 2×2 *matrix of* n*-bit integers and* N, B *are* n*-bit positive integers) is the problem of determining the* B^{th} *bit of* $(A^N)_{i,j}$*.*

Theorem 5. $\mathsf{SSQRT} \in \mathsf{PH}^{\mathsf{PP}^{\mathsf{Bit\text{-}2\text{-}MatPow}}}$.

Our proof makes use of Linear Fractional Transformations (LFTs), which in turn correspond directly to 2-by-2 matrices. Appendix B contains the necessary background concerning LFTs, including the proof of the following lemma:

Lemma 9. *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. (of Theorem 5) 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 9, 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 9, 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

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

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?) □

7 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 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 $\mathsf{BitSLP} \in (\Sigma_k^p)^A$ 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].

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.

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

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.

AS05. E. Allender and H. Schnorr. The complexity of the BitSLP problem. Unpublished Manuscript, 2005.

BCH86. P. W. Beame, S. A. Cook, and H. J. Hoover. Log depth circuits for division and related problems. *SIAM Journal on Computing*, 15:994–1003, 1986.

CDL01. A. Chiu, G. I. Davida, and B. Litow. Division in logspace-uniform NC^1. *ITA*, 35(3):259–275, 2001.

DMS94. P. Dietz, I Macarie, and J. Seiferas. Bits and relative order from residues, space efficiently. *Information Processing Letters*, 50(3):123–127, 1994.

DP12. S. Datta and R. Pratap. Computing bits of algebraic numbers. In *TAMC*, pages 189–201, 2012.

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

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

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

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

HAB02. W. Hesse, E. Allender, and D.A.M. Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *Journal of Computer and System Sciences*, 65:695–716, 2002.

HBM+10. P. Hunter, P. Bouyer, N. Markey, J. Ouaknine, and J. Worrell. Computing rational radical sums in uniform TC^0. In *FSTTCS*, pages 308–316, 2010.

HKR10. M. Hirvensalo, J. Karhumäki, and A. Rabinovich. Computing partial information out of intractable: Powers of algebraic numbers as an example. *Journal of Number Theory*, 130:232–253, 2010.

HV06. A. Healy and E. Viola. Constant-depth circuits for arithmetic in finite fields of characteristic two. In *STACS 2006*, pages 672–683. Springer, 2006.

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

JS12. G. Jindal and T. Saranurak. Subtraction makes computing integers faster. *CoRR*, abs/1212.2549, 2012.

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

KP11. P. Koiran and S. Perifel. Interpolation in Valiant's theory. *Computational Complexity*, 20(1):1–20, 2011.

KS12. N. Kayal and C. Saha. On the sum of square roots of polynomials and related problems. *TOCT*, 4(4):9, 2012.

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

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

OW14. J. Ouaknine and J. Worrell. Positivity problems for low-order linear recurrence sequences. In *SODA*, pages 366–379, 2014.

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

Pot99. P. J. Potts. *Exact Real Arithmetic using Möbius Transformations.* PhD thesis, Imperial College, University of London, 1999.
She07. A. A. Sherstov. Powering requires threshold depth 3. *Inf. Process. Lett.*, 102(2-3):104–107, 2007.
SR94. K.-Y. Siu and V. P. Roychowdhury. On optimal depth threshold circuits for multiplication and related problems. *SIAM J. Discrete Math.*, 7(2):284–292, 1994.
Tiw92. P. Tiwari. A problem that is easier to solve on the unit-cost algebraic ram. *J. Complexity*, 8(4):393–397, 1992.
Tod91. S. Toda. PP is as hard as the polynomial time hierarchy. *SIAM J. Comput.*, 20:865–877, 1991.
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.

Acknowledgments

The first author acknowledges the support of NSF grants CCF-0832787 and CCF-1064785. We would like to thank anonymous referees for help in improving the presentation of the paper.

A Proof of Claim 3

The following simple proposition is used in Claim 3, and is included for completeness.

Proposition 3. *Let $x > 1$ be given. Then for all $n > 1$,*

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

Proof. By induction. Assume that $1 - n/x \leq (1-1/x)^n$. Then $1-(n+1)/x = 1 - n/x + n/x - (n+1)/x < (1-1/x)^n - 1/x < (1-1/x)^n - (1-1/x)^n 1/x = (1-1/x)^n(1-1/x)$. □

B 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 and the associated matrix is $\begin{pmatrix} a & c \\ b & d \end{pmatrix}$. 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 matrix has a 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. ϕ acts as a bijection between any interval $[p, q]$ and a subset of the extended reals i.e. the usual reals augmented with ∞. 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 4. *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 5. *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.[6] Thus if 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 $[0, \infty]$ yield 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, and conclude by proving Lemma 9.

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 $n \to \infty$.

B.1 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:

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

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

B.2 Length of the n^{th} convergent for the Square Root, and the Proof of Lemma 9

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.

Now recall Lemma 9, which states that if $[p_n(x), q_n(x)]$ denotes the n^{th} convergent for the matrix sequence $M_1, M_2, \ldots$ where each $M_i = L(x) = \begin{pmatrix} x & x \\ 1 & x \end{pmatrix}$, then $q_n(x) - p_n(x) < x\left(1 - \frac{1}{x}\right)^{n+1}$. Thus if $x \in [1, 2]$, then $0 \le q_n(x) - p_n(x) < 2^{-n}$, and for all n, $\sqrt{x} \in [p_n(x), q_n(x)]$. Furthermore, $p_n(x) = (L(x)^n)_{1,2}/(L(x)^n)_{2,2}$.

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 $q_n(x) - p_n(x) < (x - 1)\left(1 - \frac{1}{x}\right)^n \le x\left(1 - \frac{1}{x}\right)\left(1 - \frac{1}{x}\right)^n = \left(1 - \frac{1}{x}\right)^{n+1}$.

The other parts of Lemma 9 follow immediately.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de