On the Complexity of Numerical Analysis

Eric Allender
Rutgers, the State University of NJ
Department of Computer Science
Piscataway, NJ 08854-8019, USA
allender@cs.rutgers.edu

Peter Bürgisser
Paderborn University
Department of Mathematics
DE-33095 Paderborn, Germany
pbuerg@upb.de

Johan Kjeldgaard-Pedersen
PA Consulting Group
Decision Sciences Practice
Tuborg Blvd. 5, DK 2900 Hellerup, Denmark
johan.kjeldgaard-pedersen@paconsulting.com

Peter Bro Miltersen
University of Aarhus
Department of Computer Science
IT-parken, DK 8200 Aarhus N, Denmark
bromille@daimi.au.dk

Abstract

We study two quite different approaches to understanding the complexity of fundamental problems in numerical analysis. We show that both hinge on the question of understanding the complexity of the following problem, which we call PosSLP*: Given a division-free straight-line program producing an integer N, decide whether $N > 0$. We show that* PosSLP *lies in the counting hierarchy, and combining our results with work of Tiwari, we show that the Euclidean Traveling Salesman Problem lies in the counting hierarchy – the previous best upper bound for this important problem (in terms of classical complexity classes) being* PSPACE.

1 Introduction

The original motivation for this paper comes from a desire to understand the complexity of computation over the reals in the Blum-Shub-Smale model. In Section 1.1 we give a brief introduction to this model and we introduce the problem PosSLP and explain its importance in understanding the Blum-Shub-Smale model.

In Section 1.2 we present yet another reason to be interested in PosSLP. We isolate a computational problem that lies at the root of the task of designing numerically stable algorithms. We show that this task is computationally equivalent to PosSLP. The material in Sections 1.1 and 1.2 provides motivation for studying PosSLP and for attempting to place it within the framework of traditional complexity classes.

In Section 1.3 we discuss our main technical contributions: proving upper and lower bounds on the complexity of PosSLP. In Section 1.4 we present applications of our main result with respect to the Euclidean Traveling Salesman Problem and the Sum-of-Square-Roots problem.

1.1 Polynomial Time Over the Reals

The Blum-Shub-Smale model of computation over the reals provides a very well-studied complexity-theoretic setting in which to study the computational problems of numerical analysis. We refer the reader to Blum, Cucker, Shub and Smale [9] for detailed definitions and background material related to this model; here, we will recall only a few salient facts. In the Blum-Shub-Smale model, each machine computing over the reals has associated with it a finite set of real *machine constants*. The inputs to a machine are elements of $\bigcup_n \mathbb{R}^n = \mathbb{R}^\infty$, and thus each polynomial-time machine over $\mathbb{R}$ accepts a "decision problem" $L \subseteq \mathbb{R}^\infty$. The set of decision problems accepted by polynomial-time machines over $\mathbb{R}$ is denoted $\mathrm{P}_\mathbb{R}$.

There has been considerable interest in relating computation over $\mathbb{R}$ to the classical Boolean complexity classes such as P, NP, PSPACE, etc. This is accomplished by considering the *Boolean part* of decision problems over the reals. That is, given a problem $L \subseteq \mathbb{R}^\infty$, the Boolean part of L is defined as $\mathrm{BP}(L) := L \cap \{0,1\}^\infty$. (Here, we follow the notation of [9]; $\{0,1\}^\infty = \bigcup_n \{0,1\}^n$, which is identical to $\{0,1\}^*$.) The Boolean part of $\mathrm{P}_\mathbb{R}$, denoted $\mathrm{BP}(\mathrm{P}_\mathbb{R})$, is defined as $\{\mathrm{BP}(L) \mid L \in \mathrm{P}_\mathbb{R}\}$.

By encoding the advice function in a single real constant as in Koiran [27], one can show that $\mathsf{P}/\mathrm{poly} \subseteq \mathrm{BP}(\mathrm{P}_\mathbb{R})$.

Dagstuhl Seminar Proceedings 06111
Complexity of Boolean Functions
http://drops.dagstuhl.de/opus/volltexte/2006/613

The best upper bound on the complexity of problems in $\mathrm{BP}(\mathrm{P}_{\mathbb{R}})$ that is currently known was obtained by Cucker and Grigoriev [16]:

$$\mathrm{BP}(\mathrm{P}_{\mathbb{R}}) \subseteq \mathsf{PSPACE}/\mathrm{poly}. \quad (1)$$

There has been *no* work pointing to lower bounds on the complexity of $\mathrm{BP}(\mathrm{P}_{\mathbb{R}})$; nobody has presented any compelling evidence that $\mathrm{BP}(\mathrm{P}_{\mathbb{R}})$ is not equal to P/poly.

There has also been some suggestion that perhaps $\mathrm{BP}(\mathrm{P}_{\mathbb{R}})$ is equal to $\mathsf{PSPACE}/\mathrm{poly}$. For instance, certain variants of the RAM model that provide for unit-cost arithmetic can simulate all of PSPACE in polynomial time [6, 23]. Since the Blum-Shub-Smale model also provides for unit-time multiplication on "large" numbers, Cucker and Grigoriev [16] mention that researchers have raised the possibility that similar arguments might show that polynomial-time computation over $\mathbb{R}$ might be able to simulate PSPACE. Cucker and Grigoriev also observe that certain naïve approaches to provide such a simulation must fail.

One of our goals is to provide evidence that $\mathrm{BP}(\mathrm{P}_{\mathbb{R}})$ lies properly between P/poly and $\mathsf{PSPACE}/\mathrm{poly}$. Towards this goal, it is crucial to understand a certain decision problem PosSLP: *The problem of deciding, for a given straight-line program, whether it represents a positive integer*. (For precise definitions, see the next section.)

The immediate relationship between the Blum-Shub-Smale model and the problem PosSLP is given by the proposition below. Following Bürgisser and Cucker [13], define $\mathrm{P}^0_{\mathbb{R}}$ to be the class of decision problems over the reals decided by polynomial time Blum-Shub-Smale machines using only the constants $0, 1$.

Proposition 1.1 $\mathsf{P}^{\mathrm{PosSLP}} = \mathrm{BP}(\mathrm{P}^0_{\mathbb{R}})$.

Proof. (Sketch) It is clear that PosSLP is in $\mathrm{BP}(\mathrm{P}^0_{\mathbb{R}})$, since we can implement a standard SLP interpreter in the Real Turing Machine framework and evaluate the result in linear time using unit cost instructions. To show the other direction, assume we have a polynomial time machine over $\mathbb{R}$ using only the constants $0, 1$. Given a bit string as input, we simulate the computation by storing the straight-line program representation of the intermediate results instead of their values. Branch instructions can be simulated by using the oracle to determine if the contents of a given register (represented by a straight-line program) is greater than zero. □

It was shown by Chapuis and Koiran [14] that algebraic constants do not help. More specifically, $\mathrm{BP}(\mathrm{P}^0_{\mathbb{R}})$ equals the Boolean part of the class of decision problems over the reals decided by polynomial time Blum-Shub-Smale machines using real algebraic numbers as constants.

As already mentioned, by encoding the advice function in a single real constant, one can show that $\mathsf{P}/\mathrm{poly} \subseteq \mathrm{BP}(\mathrm{P}_{\mathbb{R}})$. The proof in fact shows even $\mathsf{P}^{\mathrm{PosSLP}}/\mathrm{poly} \subseteq \mathrm{BP}(\mathrm{P}_{\mathbb{R}})$. The real constant encoding the advice function, will, of course, in general be transcendental. Thus, there is a strong relationship between non-uniformity in the classical model of computation and the use of transcendental constants in the Blum-Shub-Smale model. We conjecture that this relationship can be further strengthened:

Conjecture 1.2 $\mathsf{P}^{\mathrm{PosSLP}}/\mathrm{poly} = \mathrm{BP}(\mathrm{P}_{\mathbb{R}})$

1.2 The Task of a Numerical Analyst

The Blum-Shub-Smale model is a very elegant one, but it does not take into account the fact that actual numerical computations have to deal with *finitely* represented values. We next observe that even if we take this into account, the PosSLP problem still captures the complexity of numerical computation.

Let $u \neq 0$ be a dyadic rational number. The *floating point* representation of u is obtained by writing $u = v2^m$ where m is an integer and $\frac{1}{2} \leq |v| < 1$. The floating point representation is then given by the sign of v, and the usual binary representations of the numbers $|v|$ and m. The floating point representation of 0 is the string 0 itself. We shall abuse notation and identify the floating point representation of a number with the number itself, using the term "floating point number" for the number as well as its representation.

Let $u \neq 0$ be a real number. We may write u as $u = u'2^m$ where $\frac{1}{2} \leq |u'| < 1$ and m is an integer. Then, we define a *floating point approximation of u with k significant bits* to be a floating point number $v2^m$ so that $|v - u'| \leq 2^{-(k+1)}$.

We will focus on one part of the job that is done by numerical analysts: the design of numerically-stable algorithms. In our scenario, the numerical analyst starts out with a known function f, and the task is to design a "good" algorithm for it. When we say that the function f is "known", we mean that the analyst starts out with some method of computing (or at least approximating) f; we restrict attention to the "easy" case where the method for computing f uses only the arithmetic operations $+, -, *, \div$, and thus the description of f that the analyst is given can be presented as an arithmetic circuit with operations $+, -, *, \div$. Usually, the analyst also has to worry about the problems that are caused by the fact that the inputs to f are not known precisely, but are only given as floating point numbers that are approximations to the "true" inputs – but again we will focus on the "easy" case where the analyst will merely try to compute a good approximation for $f(x_1, \ldots, x_n)$ on the exact floating point numbers $x_1, \ldots, x_n$ that are presented as input:

The generic task of numerical computation: *Given an integer k in unary and a straight-line program (with $\div$) taking as inputs floating point numbers, with a promise that it*

neither evaluates to zero nor does division by zero, compute a floating point approximation of the value of the output with k significant bits.

The traditional approach that numerical analysts have followed in trying to solve problems of this sort is to study the numerical stability of the algorithm represented by the circuit, and in case of instability, to attempt to devise an equivalent computation that is numerically stable. Although stable algorithms have been found for a great many important functions, the task of devising such algorithms frequently involves some highly nontrivial mathematics and algorithmic ingenuity. There seems to be no expectation that there will ever by a purely automatic way to solve this problem, and indeed there seems to be no expectation that a numerically stable algorithm will exist in general. To summarize, there is substantial empirical evidence that the generic task of numerical computation is intractable. It would be of significant practical interest if, contrary to expectation, it should turn out to be very easy to solve (say, solvable in linear time).

We show that the generic task of numerical computation is equivalent in power to PosSLP.

Proposition 1.3 *The generic task of numerical computation is polynomial time Turing equivalent to* PosSLP.

Proof. We first reduce PosSLP to the generic task of numerical computation. Given a straight-line program representing the number N, we construct a straight-line program computing the value $v = 2N - 1$. The only inputs $0, 1$ of this program can be considered to be floating point numbers and this circuit clearly satisfies the promise of the generic task of numerical computation. Then $N > 0$ if $v \geq 1$ and $N \leq 0$ if $v \leq -1$. Determining an approximation of v to one significant bit is enough to distinguish between these cases.

Conversely, suppose we have an oracle solving PosSLP. Given a straight-line program with inputs being floating point numbers, we first convert it to a straight-line program having only input 1; it is easy to see that this can be done in polynomial time. By standard techniques we move all $\div$ gates to the top, so that the program computes a value $v = v_1/v_2$, where v_1, v_2 are given by division-free straight-line programs. We can use the oracle to determine the signs of v_1 and v_2. Without loss of generality assume that v is positive. Next we use the oracle to determine if $v_1 \geq v_2$. Suppose this is indeed the case (the opposite case is handled similarly).

We then find the least r, so that $2^{r-1} \leq v < 2^r$, by first comparing v_1 with $v_2 2^{2^i}$ for $i = 0, 1, 2, 3, ...$, using the oracle, thus finding the minimum i so that $v < 2^{2^i}$ and afterwards doing a binary search, again using the oracle to compare v_1 to $v_2 2^r$ for various values of r. This takes polynomial time.

The desired output is a floating point number $u = u'2^r$, where $|v - u'| \leq 2^{-(k+1)}$. To obtain u' we first want to find the integer w between 2^k and $2^{k+1} - 1$ so that $w/2^{k+1} \leq v/2^r < (w+1)/2^{k+1}$. Since $w/2^{k+1} \leq v/2^r < (w+1)/2^{k+1}$ iff $w2^r v_2 \leq v_1 2^{k+1} < (w+1)2^r v_2$, we can determine this by another binary search, using $O(k)$ calls to the oracle. We then output the sign of v, the binary representation of the rational $w/2^{k+1}$, and the binary representation of r, together forming the desired floating point approximation of v. □

1.3 The Complexity of PosSLP

We consider Proposition 1.3 to be evidence for the computational intractability of PosSLP. If PosSLP is in P/poly then there is a polynomial-sized "cookbook" that can be used in place of the creative task of devising numerically stable computations. This seems unlikely.

We wish to emphasize that the generic task of numerical computation models the *discrete* computational problem that underlies an important class of computational problems. Thus it differs quite fundamentally from the approach taken in the Blum-Shub-Smale model.

We also wish to emphasize that, in defining the generic task of numerical computation, we are *not* engaging in the debate over which real functions are "efficiently computable". There is by now a large literature comparing and contrasting the relative merits of the Blum-Shub-Smale model with the so-called "bit model" of computing, and there are various competing approaches to defining what it means for a real-valued function to be feasible to compute; see [7, 10, 11, 45, 46] among others. Our concerns here are orthogonal to that debate. We are not trying to determine which real-valued functions are feasible; we are studying a discrete computational problem that is relevant to numerical analysis, with the goal of proving upper and lower bounds on its complexity.

The generic task of numerical computation is one way of formulating the notion of what is feasible to compute in a world where *arbitrary precision* arithmetic is available for free. In contrast, the Blum-Shub-Smale model can be interpreted as formulating the notion of feasibility in a world where *infinite precision* arithmetic is available for free. According to Proposition 1.3, both of these approaches are *equivalent* (and captured by $\mathsf{P}^{\mathrm{PosSLP}}$) when only algebraic constants are allowed in the Blum-Shub-Smale model. Conjecture 1.2 claims that this is also true when allowing arbitrary real constants.

As another demonstration of the computational power of PosSLP, we show in §2 that the problem of determining the total degree of a multivariate polynomial over the integers given as a straight-line program reduces to PosSLP.

The above discussion suggests that PosSLP is not an easy problem. Can more formal evidence of this be given? Although it would be preferable to show that PosSLP is hard for some well-studied complexity class, the best that we can do is observe that a somewhat stronger problem (BitSLP) is hard for #P. This will be done in §2.

The above discussion also suggests that non-trivial upper bounds for PosSLP are of great interest. Prior to this paper, the best upper bound was PSPACE. Our main technical result is an improved upper bound: We show, based on results on the uniform circuit complexity of integer division and the relationship between constant depth circuits and subclasses of PSPACE [3, 24], that PosSLP lies in the counting hierarchy CH, a well-studied subclass of PSPACE that bears more or less the same relationship to #P as the polynomial hierarchy bears to NP [42, 44].

Theorem 1.4 PosSLP *is in* $\mathsf{P}^{\mathsf{PP}^{\mathsf{PP}^{\mathsf{PP}}}}$.

Another interesting upper bound for PosSLP was recently discovered by Tarasov and Vyalyi [39], who give a reduction from PosSLP to the *Semidefinite Feasibility Problem* (SFDP), i.e. the feasibility version of the optimization problem *Semidefinite Programming*. Their result can be seen as a lower bound for SFDP. SFDP is known to reduce to its complement and to lie in $\mathrm{NP}_{\mathbb{R}}$ [36]; also it is easy to see that SFDP reduces to the existential theory of the reals (for instance, see the discussion in [36]), and thus SFPD $\in$ PSPACE.

We suspect that PosSLP lies at an even lower level of CH. We leave as major open problems the question of providing better upper bounds for PosSLP and the question of providing any sort of hardness theorem, reducing a supposedly intractable problem to PosSLP.

We also believe that it would be very interesting to verify Conjecture 1.2, as this would give a characterization of $\mathrm{BP}(\mathrm{P}_{\mathbb{R}})$ in terms of classical complexity classes. But in fact, it would be equally interesting to refute it under some plausible complexity theoretic assumption, as this would give evidence that the power of using transcendental constants in the Blum-Shub-Smale model goes beyond the power of non-uniformity in classical computation.

1.4 Applications

The *Sum-of-square-roots problem* is a well-known problem with many applications to computational geometry and elsewhere. The input to the problem is a list of integers $(d_1, \ldots, d_n)$ and an integer k, and the problem is to decide if $\sum_i \sqrt{d_i} \geq k$. The complexity of this problem is posed as an open question by Garey, Graham and Johnson [22] in connection with the Euclidean traveling salesman problem, which is not known to be in NP, but which is easily seen to be solvable in NP relative to the Sum-of-square-roots problem. See also O'Rourke [34, 35] and Etessami and Yannakakis [21] for additional information. Although it has been conjectured [33] that the problem lies in P, it seems that no classical complexity class smaller than PSPACE has been known to contain this problem. On the other hand, Tiwari [40] showed that the problem can be decided in polynomial time on an "algebraic random-access machine". In fact, it is easy to see that the set of decision problems decided by such machines in polynomial time is exactly $\mathrm{BP}(\mathrm{P}^0_{\mathbb{R}})$. Thus by Proposition 1.1 we see that the Sum-of-square-roots problem reduces to PosSLP. Theorem 1.4 thus yields the following corollary.

Corollary 1.5 *The Sum-of-square-roots problem and the Euclidean Traveling Salesman Problem are in* CH.

2 Preliminaries

Our definitions of arithmetic circuits and straight-line programs are standard. An *arithmetic circuit* is a directed acyclic graph with input nodes labeled with the constants $0, 1$ or with indeterminates $X_1, \ldots, X_k$ for some k. Internal nodes are labeled with one of the operations $+, -, *, \div$. A *straight-line program* is a sequence of instructions corresponding to a sequential evaluation of an arithmetic circuit. If it contains no $\div$ operation it is said to be *division free*. Unless otherwise stated, all the straight-line programs considered will be division-free. Thus straight-line programs can be seen as a very compact representation of a polynomial over the integers. In many cases, we will be interested in division-free straight-line programs using no indeterminates, which thus represent an integer.

By the n-bit binary representation of an integer N such that $|N| < 2^n$ we understand a bit string of length $n + 1$ consisting of a *sign bit* followed by n bits encoding $|N|$ (padded with leading zeroes, if needed).

We consider the following problems:

EquSLP Given a straight-line program representing an integer N, decide whether $N = 0$.

ACIT Given a straight-line program representing a polynomial $f \in \mathbb{Z}[X_1, \ldots, X_k]$, decide whether $f = 0$.

DegSLP: Given a straight-line program representing a polynomial $f \in \mathbb{Z}[X_1, \ldots, X_k]$, and given a natural number d in binary, decide whether $\deg f \leq d$.

PosSLP Given a straight-line program representing $N \in \mathbb{Z}$, decide whether $N > 0$.

BitSLP Given a straight-line program representing N, and given $n, i \in \mathbb{N}$ in binary, decide whether the ith bit of the n-bit binary representation of N is 1.

It is not clear that any of these problems is in P, since straight-line program representations of integers can be exponentially smaller than ordinary binary representation.

There is an immediate relationship between the Blum-Shub-Smale model over the complex numbers $\mathbb{C}$ and the problem EquSLP. Let $\mathrm{P}^0_{\mathbb{C}}$ denote the class of decision problems over $\mathbb{C}$ decided by polynomial time Blum-Shub-Smale machines using only the constants $0, 1$. Similarly as for Proposition 1.1 one can show that $\mathsf{P}^{\mathrm{EquSLP}} = \mathrm{BP}(\mathrm{P}^0_{\mathbb{C}})$. On the other hand, it is known that constants can be eliminated in this setting [8, 28], hence $\mathrm{BP}(\mathrm{P}_{\mathbb{C}}) = \mathrm{BP}(\mathrm{P}^0_{\mathbb{C}})$. We therefore have

Proposition 2.1 $\mathsf{P}^{\mathrm{EquSLP}} = \mathrm{BP}(\mathrm{P}_{\mathbb{C}})$.

Clearly, EquSLP is a special case of ACIT. Schönhage [37] showed that EquSLP is in coRP, using computation modulo a randomly chosen prime. Ibarra and Moran [25], building on DeMillo and Lipton [18], Schwartz [38] and Zippel [47], extended this to show that ACIT lies in coRP. The problem ACIT has recently attracted much attention due to the work of Kabanets and Impagliazzo [26] who showed that a deterministic algorithm for ACIT would yield circuit lower bounds. (See [29] for some progress on finding deterministic algorithms for certain versions of the problem.) As far as we know, it has not been pointed out before that ACIT is actually polynomial time equivalent to EquSLP. In other words, disallowing indeterminates in the straight-line program given as input does not make ACIT easier. Or more optimistically: It is enough to find a deterministic algorithm for this special case in order to have circuit lower bounds.

Proposition 2.2 ACIT *is polynomial-time equivalent to* EquSLP.

Proof. We are given a straight-line program of size n with m indeterminates $X_1, \ldots, X_m$, computing the polynomial $p(X_1, \ldots, X_m)$. Define $B_{n,i} = 2^{2^{in^2}}$. Straight-line-programs computing these numbers using iterated squaring can easily be constructed in polynomial time, so given a straight-line-program for p, we can easily construct a straight-line program for $p(B_{n,1}, \ldots, B_{n,m})$. We shall show that for $n \geq 3$, p is identically zero iff $p(B_{n,1}, \ldots, B_{n,m})$ evaluates to zero.

To see this, first note that the "only if" part is trivial, so we only have to show the "if" part. Thus, assume that $p(X_1, \ldots, X_m)$ is not the zero-polynomial. Let $m(X_1, \ldots, X_m)$ be the largest monomial occurring in p with respect to inverse lexicographic order[1] and let k be the number of monomials. We can write $p = \alpha m + \sum_{i=1}^{k-1} \alpha_i m_i$, where $(m_i)_{i=1,\ldots,k-1}$ are the remaining monomials. An easy induction in the size of the straight line program shows that $|\alpha_i| \leq 2^{2^{2n}}$, $k \leq 2^{2^n}$ and that the degree of any variable in any m_i is at most 2^n.

Now, our claim is that the absolute value $|\alpha m(B_{n,1}, \ldots, B_{n,m})|$ is strictly bigger than the absolute value $|\sum_{i=1}^{k-1} \alpha_i m_i(B_{n,1}, \ldots, B_{n,m})|$, and thus we cannot have that $p(B_{n,1}, \ldots, B_{n,m}) = 0$.

Indeed, since the monomial m was the biggest in the inverse lexicographic ordering, we have that for any other monomial m_i there is an index j so that

$$\frac{m(B_{n,1}, \ldots, B_{n,m})}{m_i(B_{n,1}, \ldots, B_{n,m})} \geq \frac{2^{2^{jn^2}}}{\prod_{l=1}^{j-1} 2^{2^{ln^2} \cdot 2^n}} > 2^{2^{n^2-1}},$$

so we can bound

$$\begin{aligned} & |\sum_{i=1}^{k-1} \alpha_i m_i(B_{n,1}, \ldots, B_{n,m})| \\ \leq \; & 2^{2^n} 2^{2^{2n}} |\max_{i=1}^{k-1} m_i(B_{n,1}, \ldots, B_{n,m})| \\ \leq \; & 2^{2^n} 2^{2^{2n}} 2^{-2^{n^2-1}} |m(B_{n,1}, \ldots, B_{n,m})| \\ < \; & m(B_{n,1}, \ldots, B_{n,m}) \leq |\alpha m(B_{n,1}, \ldots, B_{n,m})|, \end{aligned}$$

which proves the claim. □

The problem DegSLP is not known to lie in BPP, even for the special case of univariate polynomials. Here, we show that it reduces to PosSLP.

Proposition 2.3 DegSLP *polynomial time many-one reduces to* PosSLP.

Proof. We first show the reduction for the case of univariate polynomials (i.e., straight-line-programs with a single indeterminate) and afterwards we reduce the multivariate case to the univariate case.

Let $f \in \mathbb{Z}[X]$ be given by a straight-line program of length n. To avoid having to deal with the zero polynomial of degree $-\infty$ and to ensure that the image of the polynomial is a subset of the non-negative integers, we first change the straight-line program computing f into a straight-line program computing $f_1(X) = (Xf(X) + 1)^2$ by adding a few extra lines. We can check if the degree of f is at most d by checking if the degree of f_1 is at most $D = 2(d + 1)$ (except for $d = -\infty$ in which case we check if the degree of f_1 is at most $D = 0$).

Let B_n be the integer $2^{2^{n^2}}$. As in the proof of Proposition 2.2, we can easily construct a straight-line program computing B_n and from this a straight-line program computing $f_1(B_n)$.

[1] $X_1^{\alpha_1} \cdots X_m^{\alpha_m}$ is greater than $X_1^{\beta_1} \cdots X_m^{\beta_m}$ in this order iff the right-most nonzero component of $\alpha - \beta$ is positive, cf. Cox, Little and O'Shea [15, p. 59].

Now, suppose that $\deg f_1 \leq D$. Using the same bounds on sizes of the coefficients as in the proof of Proposition 2.2 and assuming without loss of generality that $n \geq 3$, we then have

$$f_1(B_n) \leq \sum_{i=0}^{D} 2^{2^{2n}} B_n^i < (2^n+1)2^{2^{2n}} B_n^D$$
$$\leq (2^{2^n}+1)2^{2^{2n}-2^{n^2}} B_n^{D+1} < B_n^{D+1}/2.$$

On the other hand suppose that $\deg f_1 \geq D+1$. Then we have

$$f_1(B_n) \geq (B_n)^{D+1} - \sum_{i=0}^{D} 2^{2^{2n}} B_n^i \geq$$
$$B_n^{D+1} - 2^{2^n} 2^{2^{2n}} 2^{-2^{n^2}} B_n^{D+1} > B_n^{D+1}/2.$$

Thus, to check whether $\deg f_1 \leq D$, we just need to construct a straight-line-program for $2f_1(B_n) - B_n^{D+1}$ and check whether it computes a positive integer. This completes the reduction for the univariate case.

We next reduce the multivariate case to the univariate case. Thus, let $f \in \mathbb{Z}[X_1, \ldots, X_m]$ be given by a straight-line program of length n. Let $f^* \in \mathbb{Z}[X_1, \ldots, X_m, Y]$ be defined by $f^*(X_1, \ldots, X_m, Y) = f(X_1 Y, \ldots, X_m Y)$. We claim that if we let $B_{n,i} = 2^{2^{in^2}}$ as in the proof of Proposition 2.2, then, for $n \geq 3$, the degree of the univariate polynomial $f^*(B_{n,1}, \ldots, B_{n,m}, Y)$ is equal to the total degree of f. Indeed, we can write f^* as a polynomial in Y with coefficients in $\mathbb{Z}[X_1, \ldots, X_m]$:

$$f^*(X_1, \ldots, X_m, Y) = \sum_{j=0}^{d^*} g_j(X_1, \ldots, X_m) Y^j$$

where d^* is the degree of variable Y in the polynomial f^*. Note that this is also the total degree of the polynomial f. Now, the same argument as used in the proof of Proportion 2.2 shows that since g_{d^*} is not the zero-polynomial, $g_{d^*}(B_{n,1}, B_{n,2}, \ldots, B_{n,m})$ is different from 0. □

As PosSLP easily reduces to BitSLP, we obtain the chain of reductions

$$\text{ACIT} \leq_{\text{m}}^{\text{p}} \text{DegSLP} \leq_{\text{m}}^{\text{p}} \text{PosSLP} \leq_{\text{m}}^{\text{p}} \text{BitSLP}.$$

In §3 we will show that all the above problems in fact lie in the counting hierarchy CH.

The complexity of BitSLP contrasts sharply with that of EquSLP.

Proposition 2.4 BitSLP *is hard for* #P.

Proof. The proof is quite similar to that of Bürgisser [12, Prop. 5.3], which in turn is based on ideas of Valiant [43]. We show that computing the permanent of matrices with entries from {0,1} is reducible to BitSLP.

Given a matrix X with entries $x_{i,j} \in \{0,1\}$, consider the univariate polynomial

$$f_n = \sum_i f_{n,i} Y^i = \prod_{i=1}^{n} \left(\sum_{j=1}^{n} x_{i,j} Y^{2^{j-1}} \right)$$

which can be represented by a straight-line program of size $O(n^2)$. Then $f_{n,2^n-1}$ equals the permanent of X. Let N be the number that is represented by the straight-line program that results by replacing the indeterminate Y with 2^{n^3}. It is easy to see that the binary representation of $f_{n,2^n-1}$ appears as a sequence of consecutive bits in the binary representation of N. □

3 PosSLP lies in CH

The counting hierarchy CH was defined by Wagner [44] and was studied further by Toran [42]; see also [5, 3]. A problem lies in CH if it lies in one of the classes in the sequence $\mathsf{PP}, \mathsf{PP}^{\mathsf{PP}}$, etc.

Theorem 3.1 BitSLP *is in* CH.

Proof. It was shown by Hesse et al. [24] that there are Dlogtime-uniform threshold circuits of polynomial size and constant depth that compute the following function:

Input A number X in Chinese Remainder Representation. That is, a sequence of values indexed (p, j) giving the j-th bit of X mod p, for each prime $p < n^2$, where $0 \leq X \leq 2^n$ (thus we view n as an appropriate "size" measure of the input).

Output The binary representation of the unique natural number $X < \prod_{p \text{ prime}, p<n^2} p$ whose value modulo each small prime is encoded in the input.

Let this circuit family be denoted $\{D_n\}$.

Now, as in the proof of [3, Lemma 5], we consider the following exponentially-big circuit family $\{E_n\}$, that computes BitSLP.

Given as input an encoding of a straight-line program representing integer W, we first build a new program computing the positive integer $X = W + 2^{2^n}$. Note that the bits of the binary representation of W (including the sign bit) can easily be obtained from the bits of X.

Level 1 of the circuit E_n consists of gates labeled (p, j) for each prime p such that $p < 2^{2n}$ and for each $j : 1 \leq j \leq \lceil \log p \rceil$. The output of this gate records the jth bit of $X \bmod p$. (Observe that there are exponentially many gates on level 1, and also note that the output of each gate (p, j)

can be computed in time polynomial in the size of the binary encoding of p and the size of the given straight-line program representing X. Note also that the gates on Level 1 correspond to the gates on the input level of the circuit $D_{2^{2n}}$.

The higher levels of the circuit are simply the gates of $D_{2^{2n}}$.

Now, similar to the proof of [3, Lemma 5], we claim that for each constant d, the following language is in the counting hierarchy: $L_d = \{(F, P, b) : F$ is the name of a gate on level d of E_n and F evaluates to b when given straight-line program P as input$\}$.

We have already observed that this is true when $d = 1$. For the inductive step, assume that $L_d \in \mathsf{CH}$. Here is an algorithm to solve L_{d+1} using oracle access to L_d. On input (F, P, b), we need to determine if the gate F is a gate of E_n, and if so, we need to determine if it evaluates to b on input P. F is a gate of E_n iff it is connected to some gate G such that, for some b', $(G, P, b') \in L_d$. This can be determined in $\mathsf{NP}^{L_d} \subseteq \mathsf{PP}^{L_d}$, since D_n is Dlogtime-uniform. That is, we can guess a gate G, check that G is connected to F (this takes only linear time because of the uniformity condition) and then use our oracle for L_d. If F is a gate of E_n, we need to determine if the majority of the gates that feed into it evaluate to 1. (Note that all of the gates in D_n are MAJORITY gates.) That is, we need to determine if it is the case that for most bit strings G such that G is the name of a gate that is connected to F, $(G, P, 1)$ is in L_d. This is clearly computable in PP^{L_d}.

Thus in order to compute BitSLP, given program P and index i, compute the name F of the output bit of E_n that produces the ith bit of N (which is easy because of the uniformity of the circuits $D_{2^{2n}}$) and determine if $(F, P, 1) \in L_d$, where d is determined by the depth of the constant-depth family of circuits presented in [24]. □

Theorem 3.1 shows that $\mathrm{BP}(\mathrm{P}^0_{\mathbb{R}})$ lies in CH. A similar argument can be applied to an analogous restriction of "digital" $\mathrm{NP}_{\mathbb{R}}$ (i.e., where nondeterministic machines over the reals can guess "bits" but cannot guess arbitrary real numbers). Bürgisser and Cucker [13] present some problems in PSPACE that are related to *counting* problems over $\mathbb{R}$. It would be interesting to know if these problems lie in CH.

Although Theorem 3.1 shows that BitSLP and PosSLP both lie in CH, some additional effort is required in order to determine the level of CH where these problems reside. We present a more detailed analysis for PosSLP, since it is our main concern in this paper. (A similar analysis can be carried out for BitSLP, showing that it lies in $\mathsf{PH}^{\mathsf{PP}^{\mathsf{PP}^{\mathsf{PP}^{\mathsf{PP}}}}}$ [4].)

The following result implies Theorem 1.4, since Toda's Theorem [41] shows that $\mathsf{PP}^{\mathsf{PH}^A} \subseteq \mathsf{P}^{\mathsf{PP}^A}$ for every oracle A.

Theorem 3.2 $\mathrm{PosSLP} \in \mathsf{PH}^{\mathsf{PP}^{\mathsf{PP}}}$.

Proof. We will use the Chinese remaindering algorithm of [24] to obtain our upper bound on PosSLP. (Related algorithms, which do not lead directly to the bound reported here, have been used on several occasions [1, 17, 20, 30, 31].) Let us introduce some notation relating to Chinese remaindering.

For $n \in \mathbb{N}$ let M_n be the product of all odd primes p less than 2^{n^2}. By the prime number theorem, $2^{2^n} < M_n < 2^{2^{n^2+1}}$ for n sufficiently large. For such primes p let $h_{p,n}$ denote the inverse of $M_n/p \bmod p$.

Any integer $0 \leq X < M_n$ can be represented uniquely as a list (x_p), where p runs over the odd primes $p < 2^{n^2}$ and $x_p = X \bmod p$. Moreover, X is congruent to $\sum_p x_p h_{p,n} M_n/p$ modulo M_n. Hence X/M_n is the fractional part of $\sum_p x_p h_{p,n}/p$.

Define the family of approximation functions $app_n(X)$ to be $\sum_p B_p$, where $B_p = x_p h_{p,n} \sigma_{p,n}$ and $\sigma_{p,n}$ is the result of truncating the binary expansion of $1/p$ after 2^{n^4} bits. Note that for n sufficiently large and $X < M_n$, $app_n(X)$ is within $2^{-2^{n^3}}$ of X/M_n.

Let the input to PosSLP be a program P of size n representing the integer W and put $Y_n = 2^{2^n}$. Since $|W| \leq Y_n$, the number $X := W + Y_n$ is nonnegative and we can easily transform P into a program of size $2n + 2$ representing X. Clearly, $W > 0$ iff $X > Y_n$. Note that if $X > Y_n$, then X/M_n and Y_n/M_n differ by at least $1/M_n > 2^{-2^{n^2+1}}$, which implies that it is enough to compare the binary expansions of $app_n(X)$ and $app_n(Y_n)$. (Interestingly, this seems to be somewhat easier than computing the bits of X directly.)

We can determine if $X > Y_n$ in PH relative to the following oracle: $A = \{(P, j, b, 1^n) :$ the j-th bit of the binary expansion of $app_n(X)$ is b, where X is the number represented by straight-line program P and j is given in binary$\}$. Lemma 3.3 completes the proof by showing that $A \in \mathsf{PH}^{\mathsf{PP}^{\mathsf{PP}}}$. □

Lemma 3.3 $A \in \mathsf{PH}^{\mathsf{PP}^{\mathsf{PP}}}$.

Proof. Assume for the moment that we can show that $B \in \mathsf{PH}^{\mathsf{PP}}$, where $B := \{(P, j, b, p, 1^n) :$ the j-th bit of the binary expansion of B_p $(= x_p h_{p,n} \sigma_{p,n})$ is b, where $p < 2^{n^2}$ is an odd prime, $x_p = X \bmod p$, X is the number represented by the straight-line program P, and j is given in binary$\}$. In order to recognize the set A, it clearly suffices to compute 2^{n^4} bits of the binary representation of the sum of the numbers B_p. A uniform circuit family for iterated sum is presented by Maciel and Thérien in [32, Corollary 3.4.2] consisting of MAJORITY gates on the bottom (input) level, with three levels of AND and OR gates above.

As in the proof of Theorem 3.1, the construction of Maciel and Thérien immediately yields a $\mathsf{PH}^{\mathsf{PP}^B}$ algorithm for A, by simulating the MAJORITY gates by PP^B computation, simulating the OR gates above the MAJORITY gates by $\mathsf{NP}^{\mathsf{PP}^B}$ computation, etc. The claim follows, since by Toda's Theorem [41] $\mathsf{PH}^{\mathsf{PP}^B} \subseteq \mathsf{PH}^{\mathsf{PP}^{\mathsf{PH}^{\mathsf{PP}}}} = \mathsf{PH}^{\mathsf{PP}^{\mathsf{PP}}}$. It remains only to show that $B \in \mathsf{PH}^{\mathsf{PP}}$. □

Lemma 3.4 $B \in \mathsf{PH}^{\mathsf{PP}}$.

Proof. Observe that given (P, j, b, p) we can determine in polynomial time if p is prime [2], and we can compute x_p.

In $\mathsf{PH} \subseteq \mathsf{P}^{\mathsf{PP}}$ we can find the least generator g_p of the multiplicative group of the integers mod p. The set $C = \{(q, g_p, i, p) : p \neq q$ are primes and i is the least number for which $g_p^i \equiv q \mod p\}$ is easily seen to lie in PH. We can compute the discrete log base g_p of the number $M_n/p \bmod p$ in $\#\mathsf{P}^C \subseteq \mathsf{P}^{\mathsf{PP}}$, by the algorithm that nondeterministically guesses q and i, verifies that $(q, g_p, i, p) \in C$, and if so generates i accepting paths. Thus we can compute the number $M_n/p \bmod p$ itself in P^{PP} by first computing its discrete log, and then computing g_p to that power, mod p. The inverse $h_{p,n}$ is now easy to compute in P^{PP}, by finding the inverse of M_n/p mod p.

Our goal is to compute the j-th bit of the binary expansion of $x_p h_{p,n} \sigma_{p,n}$. We have already computed x_p and $h_{p,n}$ in P^{PP}, so it is easy to compute $x_p h_{p,n}$. The jth bit of $1/p$ is 1 iff $2^j \bmod p$ is odd, so bits of $\sigma_{p,n}$ are easy to compute in polynomial time. (Note that j is exponentially large.)

Thus our task is to obtain the j-th bit of the product of $x_p h_{p,n}$ and $\sigma_{p,n}$, or (equivalently) adding $\sigma_{p,n}$ to itself $x_p h_{p,n}$ times. The problem of adding $\log^{O(1)} n$ many n-bit numbers lies in uniform AC^0 [19]. Simulating these AC^0 circuits leads to the desired $\mathsf{PH}^{\mathsf{PP}}$ algorithm for B. □

Acknowledgments

We acknowledge helpful conversations with Kousha Etessami, Sambuddha Roy, Felipe Cucker, Lenore Blum, Richard Lipton, Parikshit Gopalan, Mark Braverman, Madhu Sudan, Klaus Meer, Pascal Koiran, Qi Cheng, and Kristoffer Arnsfelt Hansen. The first author acknowledges the support of NSF Grant CCF-0514155. The second author acknowledges the support of DFG Grant BU 1371.

References

[1] M. Agrawal, E. Allender, and S. Datta. On TC^0, AC^0, and arithmetic circuits. *J. Comp. Syst. Sci.*, 60:395–421, 2000.

[2] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *Annals of Mathematics*, 160:781–793, 2004.

[3] E. Allender, M. Koucký, D. Ronneburger, S. Roy, and V. Vinay. Time-space tradeoffs in the counting hierarchy. In *Proc. 16th Ann. IEEE Conf. on Computational Complexity (CCC '01)*, pages 295–302, 2001. Revised version to appear in *Theory of Computing Systems*.

[4] E. Allender and H. Schnorr. The complexity of the BitSLP problem. manuscript, 2005.

[5] E. Allender and K. W. Wagner. Counting hierarchies: polynomial time and constant depth circuits. In G. Rozenberg and A. Salomaa, editors, *Current Trends in Theoretical Computer Science*, pages 469–483. World Scientific, 1993.

[6] A. Bertoni, G. Mauri, and N. Sabadini. Simulations among classes of random access machines and equivalence among numbers succinctly represented. *Ann. Discrete Math.*, 25:65–90, 1985.

[7] L. Blum. Computing over the reals: Where Turing meets Newton. *Notices of the American Mathematical Society*, 51:1024–1034, 2004.

[8] L. Blum, F. Cucker, M. Shub, and S. Smale. Algebraic Settings for the Problem "$P \neq NP$?". In *The mathematics of numerical analysis*, number 32 in Lectures in Applied Mathematics, pages 125–144. Amer. Math. Soc., 1996.

[9] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation*. Springer, 1998.

[10] M. Braverman. On the complexity of real functions. In *FOCS*, pages 155–164, 2005.

[11] M. Braverman and S. Cook. Computing over the reals: Foundations for scientific computing. *Notices of the AMS*, 55:318–329, 2006.

[12] P. Bürgisser. The complexity of factors of multivariate polynomials. *Foundations of Computational Mathematics*, 4:369–396, 2004.

[13] P. Bürgisser and F. Cucker. Counting complexity classes for numeric computations II: Algebraic and semialgebraic sets. *J. Compl.* To appear. Extended abstract in *Proc. 36th Ann. ACM STOC*, pages 475–485, 2004.

[14] O. Chapuis and P. Koiran. Saturation and stability in the theory of computation over the reals. *Annals of Pure and Applied Logic*, 99:1–49, 1999.

[15] D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms*. Springer, 1991.

[16] F. Cucker and D.Yu. Grigoriev. On the power of real Turing machines over binary inputs. *SIAM J. Comp.*, 26:243–254, 1997.

[17] G.I. Davida and B. Litow. Fast parallel arithmetic via modular representation. *SIAM J. Comp.*, 20(4):756–765, August 1991.

[18] R. DeMillo and R. Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7:193–195, 1978.

[19] L. Denenberg, Y. Gurevich, and S. Shelah. Definability by constant-depth polynomial-size circuits. *Information and Control*, 70:216–240, 1986.

[20] P.F. Dietz, I.I. Macarie, and J.I. Seiferas. Bits and relative order from residues, space efficiently. *Inf. Proc. Letters*, 50(3):123–127, 9 May 1994.

[21] K. Etessami and M. Yannakakis. Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. In V. Diekert and B. Durand, editors, *22nd Ann. Symposium on Theoretical Aspects of Computer Science (STACS'05)*, number 3404 in LNCS, pages 340–352, 2005.

[22] M. Garey, R.L. Graham, and D.S. Johnson. Some NP-complete geometric problems. In *Proc. ACM Symp. Theory Comp.*, pages 10–22, 1976.

[23] J. Hartmanis and J. Simon. On the power of multiplication in random-access machines. In *Proc. 15th Ann.. IEEE Sympos. Switching Automata Theory*, pages 13–23, 1974.

[24] W. Hesse, E. Allender, and D. A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *J. Comp. Syst. Sci.*, 65:695–716, 2002.

[25] O.H. Ibarra and S. Moran. Equivalence of straight-line programs. *J. ACM*, 30:217–228, 1983.

[26] V. Kabanets and R. Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. In *Proc. ACM Symp. Theory Comp.*, pages 355–364, 2003.

[27] P. Koiran. Computing over the reals with addition and order. *Theoret. Comp. Sci.*, 133:35–47, 1994.

[28] P. Koiran. Elimination of constants from machines over algebraically closed fields. *J. Compl.*, 13:65–82, 1997.

[29] R. Lipton and N. Vishnoi. Deterministic identity testing for multivariate polynomials. In *Proc. 14th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 756–760, 2003.

[30] B. Litow. On iterated integer product. *Inf. Proc. Letters*, 42(5):269–272, 03 July 1992.

[31] I. Macarie. Space-efficient deterministic simulation of probabilistic automata. *SIAM J. Comp.*, 27:448–465, 1998.

[32] A. Maciel and D. Thérien. Threshold circuits of small majority-depth. *Information and Computing*, 146:55–83, 1998.

[33] G. Malajovich. An effective version of Kronecker's theorem on simultaneous Diophantine approximation. Technical report, City University of Hong Kong, 1996.

[34] J. O'Rourke. http://maven.smith.edu/˜ orourke/TOPP. Webpage.

[35] J. O'Rourke. Advanced problem 6369. *Amer. Math. Monthly*, 88:769, 1981.

[36] M. Ramana. An exact duality theory for semidefinite programming and its complexity implications. *Mathematical Programming*, 77:129–162, 1997.

[37] A. Schönhage. On the power of random access machines. In H.A. Maurer, editor, *Automata, languages and programming ICALP'79*, number 71 in LNCS, pages 520–529, 1979.

[38] J.T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27:701–717, 1980.

[39] S.P. Tarasov and M.N. Vyalyi. Semidefinite programming and arithmetic circuit evaluation. arXiv manuscript cs.CC/0512035. Submitted to *Special issue of DAM in memory of L.Khachiyan*, 2005.

[40] P. Tiwari. A problem that is easier to solve on the unit-cost algebraic RAM. *Journal of Complexity*, 8:393–397, 1992.

[41] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comp.*, 21(2):865–877, 1991.

[42] J. Torán. Complexity classes defined by counting quantifiers. *J. ACM*, 38:753–774, 1991.

[43] L.G. Valiant. Reducibility by algebraic projections. In *Logic and Algorithmic: an International Symposium held in honor of Ernst Specker*, volume 30, pages 365–380. Monogr. No. 30 de l'Enseign. Math., 1982.

[44] K. W. Wagner. The complexity of combinatorial problems with succinct input representation. *Acta Informatica*, 23:325–356, 1986.

[45] K. Weihrauch. *Computable Analysis*. Springer Verlag, 2000.

[46] H. Wozniakowski. Why does information-based complexity use the real number model? *Theor. Comput. Sci.*, 219:451–465, 1999.

[47] R.E.B. Zippel. Simplification of radicals with applications to solving polynomial equations. Master's thesis, M.I.T., 1977.