PosSLP and Sum of Squares

Markus Bläser[*] Julian Dörfler[†] Gorav Jindal[‡]

Abstract

The problem PosSLP is the problem of determining whether a given straight-line program (SLP) computes a positive integer. PosSLP was introduced by Allender et al. to study the complexity of numerical analysis (Allender et al., 2009). PosSLP can also be reformulated as the problem of deciding whether the integer computed by a given SLP can be expressed as the sum of squares of four integers, based on the well-known result by Lagrange in 1770, which demonstrated that every natural number can be represented as the sum of four non-negative integer squares.

In this paper, we explore several natural extensions of this problem by investigating whether the positive integer computed by a given SLP can be written as the sum of squares of two or three integers. We delve into the complexity of these variations and demonstrate relations between the complexity of the original PosSLP problem and the complexity of these related problems. Additionally, we introduce a new intriguing problem called Div2SLP and illustrate how Div2SLP is connected to DegSLP and the problem of whether an SLP computes an integer expressible as the sum of three squares.

By comprehending the connections between these problems, our results offer a deeper understanding of decision problems associated with SLPs and open avenues for further exciting research.

1 Introduction

1.1 Straight Line Programs and PosSLP

The problem PosSLP was introduced in [All+09] to study the complexity of numerical analysis and relate the computations over the reals (in the so-called Blum-Shub-Smale model, see [Blu+97]) to classical computational complexity. PosSLP asks whether a given integer is positive or not. The problem may seem trivial at first glance but becomes highly non-trivial when the given integer is not explicitly provided but rather represented by an implicit expression which computes it. One way to model the implicit computations of integers and polynomials is through the notion of arithmetic circuits and straight line programs (SLPs).

An arithmetic circuit takes the form of a directed acyclic graph where input nodes are designated with constants 0, 1, or variables $x_1, x_2, \ldots, x_m$. Internal nodes are labeled with mathematical operations such as addition ($+$), subtraction ($-$), multiplication ($\times$), or division ($\div$). Such arithmetic circuits are said to be *constant-free*. In the algebraic complexity theory literature, usually, one studies arithmetic circuits where constants are arbitrary scalars from the underlying field. But in this paper, we are only concerned with arithmetic circuits that are constant-free.

On the other hand, a straight-line program is a series of instructions that corresponds to a sequential evaluation of an arithmetic circuit. If this program does not contain any division operations, it is referred to

[*]Saarland University, Saarland Informatics Campus, Saarbrücken, Germany. Email: `mblaeser@cs.uni-saarland.de`
[†]Saarland University, Saarland Informatics Campus, Saarbrücken, Germany. Email: `jdoerfler@cs.uni-saarland.de`
[‡]Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany. Email: `gorav.jindal@gmail.com`

as "division-free". Unless explicitly specified otherwise, we will exclusively consider division-free straight-line programs. Consequently, straight-line programs can be viewed as a compact representation of polynomials or integers. In many instances, we will be concerned with division-free straight-line programs that do not incorporate variables, representing an integer. Arithmetic circuits and SLPs are used interchangeably in this paper. Now we define the central object of study in this paper.

Problem 1.1 (PosSLP)**.** *Given a straight-line program representing $N \in \mathbb{Z}$, decide whether $N > 0$.*

An SLP P computing an integer is a sequence $(b_0, b_1, b_2, \ldots, b_m)$ of integers such that $b_0 = 1$ and $b_i = b_j \circ_i b_k$ for all $i > 0$, where $j, k < i$ and $\circ_i \in \{+, -, \times\}$. Given such an SLP P, PosSLP is the problem of determining the sign of the integer computed by P, *i.e.*, the sign of b_m. Note that we cannot simply compute b_m from a description of P because the absolute value of b_m can be as large as 2^{2^m}. Therefore, computing b_m exactly might require exponential time. Hence, this brute force approach of determining the sign of b_m is too computationally inefficient. [All+09] also show some evidence that PosSLP might be a hard problem computationally. They do so by showing the polynomial time Turing equivalence of PosSLP to the Boolean part of the problems decidable in polynomial time in the Blum-Shub-Smale (BSS) model and also to the generic task of numerical computation. We briefly survey this relevance of PosSLP to emphasize its importance in numerical analysis. For a more detailed discussion, the interested reader is referred to [All+09, Section 1].

The Blum-Shub-Smale (BSS) computational model deals with computations using real numbers. It is a well-explored area where complexity theory and numerical analysis meet. For a detailed understanding, see [Blu+97]. Here we only dscribe the constant-free BSS model. BSS machines handle inputs from $\mathbb{R}^\infty$, allowing polynomial-time computations over $\mathbb{R}$ to solve "decision problems" $L \subseteq \mathbb{R}^\infty$. The set of problems solvable by polynomial-time BSS machines is denoted by $\mathsf{P}^0_{\mathbb{R}}$, see e.g., [BC06]. To relate the complexity class $\mathsf{P}^0_{\mathbb{R}}$ to classical complexity classes, one considers the *boolean part* of $\mathsf{P}^0_{\mathbb{R}}$, defined as: $\mathrm{BP}(\mathsf{P}^0_{\mathbb{R}}) := \{L \cap \{0,1\}^\infty \mid L \in \mathsf{P}^0_{\mathbb{R}}\}$. To highlight the importance of PosSLP as a bridge between the BSS model and classical complexity classes, [All+09] proved the following Theorem 1.1.

Theorem 1.1 (Proposition 1.1 in [All+09])**.** *We have* $\mathsf{P}^{\mathrm{PosSLP}} = \mathrm{BP}(\mathsf{P}^0_{\mathbb{R}})$.

Another motivation for the complexity of PosSLP comes from its connection to the task of numerical computation. Here we recall this connection from [All+09]. [All+09] defined the following problem to formalize the task of numerical computation:

Problem 1.2 (Generic Task of Numerical Computation (GTNC) [All+09])**.** *Given a straight-line program P with n variables, and given inputs $a_1, a_2, \ldots, a_n$ for P (as floating-point numbers) and an integer k in unary, compute a floating-point approximation of $P(a_1, a_2, \ldots, a_n)$ with k significant bits.*

The following result was also demonstrated in [All+09].

Theorem 1.2 (Proposition 1.2 in [All+09])**.** GTNC *is polynomial-time Turing equivalent to* PosSLP.

1.2 How Hard is PosSLP?

GTNC can be viewed as the task that formalizes what is computationally efficient when we are allowed to compute with arbitrary precision arithmetic. Conversely, the BSS model can be viewed as formalizing computational efficiency where we have infinite precision arithmetic at no cost. Theorem 1.1 and Theorem 1.2 show that both these models are equivalent to PosSLP under polynomial-time Turing reductions. One can also view these results as an indication that PosSLP is computationally intractable. Despite this, no unconditional non-trivial hardness results are known for PosSLP. Still, a lot of important computational problems reduce to PosSLP. We briefly survey some of these problems now. By the n-bit binary representation of an integer N with the condition $|N| < 2^n$, we mean a binary string with a length of $n+1$. This string consists of a sign bit followed by n bits encoding the absolute value of N, with leading zeros added if necessary. A very important problem in complexity theory is the EquSLP problem defined as:

Problem 1.3 (EquSLP, [All+09])**.** *Given a straight-line program representing an integer N, decide whether $N = 0$.*

EquSLP is also known to be equivalent to arithmetic circuit identity testing (ACIT) or polynomial identity testing [All+09]. It is easy to see that EquSLP reduces to PosSLP: $N \in \mathbb{Z}$ is zero if and only if $1 - N^2 > 0$. Recently, a conditional hardness result was proved for PosSLP in [BJ23], formalized below.

Theorem 1.3 (Theorem 1.2 in [BJ23])**.** *If a constructive variant of the radical conjecture of [DSS18] is true and* $\mathsf{PosSLP} \in \mathsf{BPP}$ *then* $\mathsf{NP} \subseteq \mathsf{BPP}$.

As for upper bounds on PosSLP, PosSLP was shown to be in the counting hierarchy CH in [All+09]. This is still the best-known upper bound on the complexity of PosSLP. Another important problem is the sum of square roots, defined as follows:

Problem 1.4 (Sum of Square Roots (SoSRoot))**.** *Given a list $(a_1, a_2, \ldots, a_n)$ of positive integers and a list $(\delta_1, \delta_2, \ldots, \delta_n) \in \{\pm 1\}^n$ of signs, decide if $\sum_{i=1}^{n} \delta_i \sqrt{a_i}$ is positive.*

SoSRoot is widely recognized and finds applications in computational geometry and various other domains. The Euclidean traveling salesman problem, whose inclusion in NP is not known, is easily seen to be in NP relative to SoSRoot. SoSRoot is conjectured to be in P in [Mal96] but this is far from clear. Still, one can show that SoSRoot reduces to PosSLP [Tiw92; All+09]. There are several other problems related to straight line program which are intimately related to PosSLP. For instance, the following problems were also introduced in [All+09]. These problems would be useful in our discussion later.

Problem 1.5 (BitSLP)**.** *Given a straight-line program representing N, and given $n, i \in \mathbb{N}$ in binary, decide whether the i^{th} bit of the n-bit binary representation of N is 1.*

It was also shown in [All+09] that PosSLP reduces to BitSLP. Although we do not know any unconditional hardness results for PosSLP, BitSLP was shown to be #P-hard in [All+09]. Another important problem related to PosSLP is the following DegSLP problem, which was shown to be reducible to PosSLP in [All+09].

Problem 1.6 (DegSLP)**.** *Given a straight-line program representing a polynomial $f \in \mathbb{Z}[x]$ and a natural number d in binary, decide whether $\deg(f) \leq d$.*

The problem DegSLP was posed in [All+09] for multivariate polynomials, here we have considered its univariate version. But these are seen to be equivalent under polynomial time many one reductions [All+09, Proof of Proposition 2.3], we recall this reduction in Appendix C. We also recall the following new problem from [Dut+21] related to straight line programs, which is important to results in this paper.

Problem 1.7 (OrdSLP)**.** *Given a straight-line program representing a polynomial $f \in \mathbb{Z}[x]$ and a natural number ℓ in binary, decide whether $\operatorname{ord}(f) \geq \ell$. Here, the order of f, denoted as $\operatorname{ord}(f)$, is defined to be the largest k such that $x^k \mid f$.*

1.3 Our Results

Lagrange proved in 1770 that every natural number can be represented as a sum of four non-negative integer squares [NZM91, Theorem 6.26]. Therefore, PosSLP can be reformulated as: Given a straight-line program representing $N \in \mathbb{Z}$, decide if there exist $a, b, c, d \in \mathbb{N}$ such that $N = a^2 + b^2 + c^2 + d^2$. In light of this rephrasing of PosSLP, we study the various sum of squares variants of PosSLP in Section 2 and Section 3. To formally state our results, we define these problems now. For convenience, we say that $n \in \mathbb{N}$ is 3SoS if it can be expressed as the sum of three squares (of integers). We study the following problem.

Problem 1.8 (3SoSSLP)**.** *Given a straight-line program representing $N \in \mathbb{Z}$, decide whether N is a* 3SoS.

One might expect that 3SoSSLP is easier than PosSLP, but we show that PosSLP reduces to 3SoSSLP under polynomial-time Turing reductions. More precisely, we prove the following Theorem 1.4 in Section 2.

Theorem 1.4. PosSLP $\in \mathsf{P}^{\text{3SoSSLP}}$.

Similarly, we say that $n \in \mathbb{N}$ is 2SoS if it can be expressed as the sum of two squares (of integers). We also study the following problem.

Problem 1.9 (2SoSSLP)**.** *Given a straight-line program representing $N \in \mathbb{Z}$, decide whether N is a* 2SoS.

These problems 3SoSSLP and 2SoSSLP can also be seen as special cases of the renowned Waring problem. The Waring problem has an intriguing history in number theory. It asks whether for each $k \in \mathbb{N}$ there exists a positive integer $g(k)$ such that any natural number can be written as the sum of at most $g(k)$ many k^{th} powers of natural numbers. Lagrange's four-square theorem can be seen as the equality $g(2) = 4$. Later, Hilbert settled the Waring problem for integers by proving that $g(k)$ is finite for every k [Hil09]. Therefore problems 2SoSSLP and 3SoSSLP can be seen as computational variants of the Waring problem. These computational variants of the Waring problems are extensively studied in computer algebra and algebraic complexity theory and Shitov actually proved that computing the Waring rank of multivariate polynomials is $\exists\mathbb{R}$-hard [Shi16]. For 2SoSSLP, we prove the following conditional hardness result in Section 3.

Theorem 1.5. *If the generalized Cramér conjecture A (Conjecture 3.1) is true, then* PosSLP $\in \mathsf{NP}^{\text{2SoSSLP}}$.

We also study whether 3SoSSLP can be reduced to PosSLP. Unfortunately, we cannot show this reduction unconditionally. Hence we study and rely on the following problem Div2SLP, which might be of independent interest. One can view Div2SLP as the variant of OrdSLP for numbers in binary.

Problem 1.10 (Div2SLP)**.** *Given a straight-line program representing $N \in \mathbb{Z}$, and a natural number ℓ in binary, decide if 2^ℓ divides $|N|$, i.e., if the ℓ least significant bits of $|N|$ are zero.*

We show that if we are allowed oracle access to both PosSLP and Div2SLP oracles then 3SoSSLP can be decided in polynomial time, formalized below in Theorem 1.6. A proof can be found in Section 2.

Theorem 1.6. 3SoSSLP $\in \mathsf{P}^{\{\text{Div2SLP,PosSLP}\}}$.

We also study how Div2SLP is related to other problems related to straight line programs. To this end, we prove the following Theorem 1.7 in Section 2.

Theorem 1.7. OrdSLP $\equiv_{\mathsf{P}}$ DegSLP $\leq_{\mathsf{P}}$ Div2SLP.

As for the hardness results for 3SoSSLP and 2SoSSLP, we also show that similar to PosSLP, EquSLP reduces to both 3SoSSLP and 2SoSSLP. Analogous to integers, we also study the complexity of deciding the positivity of univariate polynomials computed by a given SLP. In this context, we study the following problem.

Problem 1.11 (PosPolySLP)**.** *Given a straight-line program representing a univariate polynomial $f \in \mathbb{Z}[x]$, decide if f is positive, i.e., $f(x) \geq 0$ for all $x \in \mathbb{R}$.*

We prove that in contrast to PosSLP, hardness of PosPolySLP can be proved unconditionally, formalized below in Theorem 1.8.

Theorem 1.8. PosPolySLP *is* coNP*-hard under polynomial-time many-one reductions.*

In constrast to numbers, every positive polynomial can be written as the sum of two squares (but only over the reals, see Section Section 4 for a detailed discussion). So PosPolySLP is equivalent to the question whether f is the sum of two squares. To conclude, we motivate and study the following problem (see Section 4 for more details).

Problem 1.12 (SqPolySLP)**.** *Given a straight-line program representing a univariate polynomial $f \in \mathbb{Z}[x]$, decide if $\exists g \in \mathbb{Z}[x]$ such that $f = g^2$.*

We show in Section 4 that SqPolySLP is in coRP.

2 SLPs as Sums of Three Squares

This section is concerned with studying the complexity of 3SoSSLP and related problems.

2.1 Lower Bound for 3SoSSLP

In this section, we prove Theorem 1.4. We use the following characterization of integers which can be expressed as the sum of three squares.

Theorem 2.1 ([Leg97; Gau01; Ank57; Mor58])**.** *An integer n is* 3SoS *if and only if it is not of the form $4^a(8k+7)$, with $a,k \in \mathbb{N}$.*

Theorem 2.1 informally implies that 3SoS integers are "dense" in $\mathbb{N}$ and hence occur very frequently. A useful application of this intuitive high density of 3SoS integers is demonstrated below in Lemma 2.1. More formally, Landau showed that the asymptotic density of 3SoS integers in $\mathbb{N}$ is 5/6 [Lan09]. To reduce PosSLP to 3SoSSLP, we shift the given integer (represented by a given SLP) by a positive number to convert into 3SoS. To this end we prove the following Lemma 2.1.

Lemma 2.1. *For every $n \in \mathbb{N}$, at least one element in the set $\{n, n+2\}$ is* 3SoS.

Proof. If n is 3SoS then we are done. Suppose n is not 3SoS, by using Theorem 2.1 we know that $n = 4^a(8k+7)$ for some $a,k \in \mathbb{N}$. If $a = 0$ then $n = 8k+7$ and hence $n+2 = 8k+9$ is clearly not of the form $4^b(8c+7)$ for any $b,c \in \mathbb{N}$. If $a > 0$ then $n+2 = 4^a(8k+7)+2$ is not divisible by 4. Hence for $n+2$ of the form $4^b(8c+7)$, we have to have $4^a(8k+7)+2 = 8c+7$. This is clearly impossible because the LHS is even whereas RHS is odd. ☐

Lemma 2.2. *If $M \in \mathbb{Z}_+$ then $7M^4$ not a* 3SoS.

Proof. Suppose $M = 4^a(4b+c)$ where a is the largest power of 4 dividing M, $c = \frac{M}{4^a} \pmod 4$ and $b = \lfloor \frac{M}{4^{a+1}} \rfloor$. We prove the claim by analyzing the following cases.

If $c = 0$ then $M = 4^a b$ for some $a,b \in \mathbb{N}$ and 4 does not divide b. Note that here $a > 0$, otherwise c cannot be zero by its definition. Therefore $7M^4 = 4^{4a} \cdot 7b^4$. Now we can apply this Lemma recursively on b (which is smaller than M) to infer that $7b^4$ is of the form $4^\alpha(8\beta+7)$ for some $\alpha, \beta \in \mathbb{N}$. Hence $7M^4$ is also of this form and thus not a 3SoS by using Theorem 2.1.

If $c = 1$ then $7M^4 = 4^{4a} \cdot 7 \cdot (256b^4 + 256b^3 + 96b^2 + 16b + 1) = 4^\alpha(8\beta+7)$ for some $\alpha, \beta \in \mathbb{N}$, hence $7M^4$ is not a 3SoS by using Theorem 2.1.

If $c = 2$ then $7M^4 = 4^{4a+2} \cdot 7 \cdot (16b^4 + 32b^3 + 24b^2 + 8b + 1) = 4^\alpha(8\beta + 7)$ for some $\alpha, \beta \in \mathbb{N}$, hence $7M^4$ is not a 3SoS by using Theorem 2.1.

If $c = 3$ then $7M^4 = 4^{4a} \cdot 7 \cdot (256b^4 + 768b^3 + 964b^2 + 12496b + 81) = 4^\alpha(8\beta + 7)$ for some $\alpha, \beta \in \mathbb{N}$, hence $7M^4$ is not a 3SoS by using Theorem 2.1.

□

Lemma 2.2 implies the following EquSLP hardness of 3SoSSLP.

Lemma 2.3. EquSLP $\le_{\mathsf{P}}$ 3SoSSLP.

Proof. Given a straight-line program representing an integer N, we want to decide whether $N = 0$. Suppose $M = N^2$. We have $M \ge 0$ and $M = 0$ iff $N = 0$. By using Lemma 2.2, we know that $7M^4$ is a 3SoS iff $M = 0$. □

Remark 2.1. Lemma 2.3 illustrates that 3SoSSLP is at least as hard as EquSLP under deterministic polynomial time Turing reductions. This may not appear as a very strong result since EquSLP can be decided in randomized polynomial time anyway. However, unconditionally, even PosSLP is known to be only EquSLP-hard. Moreover, we rely on Lemma 2.3 in the proof of Theorem 1.4 below.

Proof of Theorem 1.4. Given a straight-line program representing an integer N, we want to decide whether $N > 0$. Using an EquSLP oracle, we first check if $N \in \{0, -1, -2\}$. By using Lemma 2.3, these oracle calls to EquSLP can also be simulated by oracle calls to the 3SoSSLP oracle. Hence this task belongs to $\mathsf{P}^{3\text{SoSSLP}}$. If $N \in \{0, -1, -2\}$, then clearly $N > 0$ is false and we answer “No”. Otherwise we check if N is a 3SoS, if it is then clearly $N > 0$ and we answer “Yes”. If it is not a 3SoS then we check if $N + 2$ is a 3SoS. If $N + 2$ is a 3SoS then clearly $N > 0$ because $N \notin \{0, -1, -2\}$. If $N + 2$ is not a 3SoS, then by Lemma 2.1 we can conclude that $N < -2$ and hence we answer “No”. □

2.2 Upper Bound for 3SoSSLP

Now we prove the upper bound for 3SoSSLP, claimed in Theorem 1.6.

Proof of Theorem 1.6. Given an $N \in \mathbb{Z}$ represented by a given SLP, we want to decide if N is a 3SoS. By using the PosSLP oracle, we first check if $N \ge 0$. If $N < 0$ then we answer “No”. Hence we can now assume that $N > 0$. By using Theorem 2.1, it is easy to see that N is not a 3SoS iff the binary representation $\mathrm{Bin}(N)$ of N looks like below:

$$N \text{ is not a 3SoS} \iff \mathrm{Bin}(N) = S1110^t \text{ where } t \text{ is even and } S \in \{0,1\}^*.$$

By using the Div2SLP oracle, we compute the number of trailing zeroes (call it again t) in the binary representation of N. This can be achieved by doing a binary search and repeatedly using the Div2SLP oracle. If t is not even then N is a 3SoS. Next we construct an SLP which computes 2^t, *i.e.*, the number 10^t in the binary representation. Such an SLP can be constructed in time $\mathrm{poly}(\log t)$ and is of size $O(\log^2 t)$. This can be seen by looking at the binary representation of t and then using repeated squaring. We have:

$$\mathrm{Bin}\left(N + 2^t\right) = S'0^{t+3} \iff \mathrm{Bin}(N) = S1110^t \text{ for some } S, S' \in \{0,1\}^*.$$

Hence N is not a 3SoS iff $N + 2^t$ has $t + 3$ trailing zeroes, which again can be decided using the Div2SLP oracle. □

2.3 Complexity of Div2SLP

In this section, we show a DegSLP lower bound for Div2SLP. To this end, we first prove the following equivalence of DegSLP and OrdSLP.

Lemma 2.4. *Given a straight-line program P of length s computing a polynomial $f \in \mathbb{Z}[x]$, we can compute in $\mathsf{poly}(s)$ time:*

1. *A number $m \in \mathbb{N}$ such that $\deg(f) \leq m \leq 2^s$.*
2. *A straight line program Q of length $O(s)$ such that Q computes the polynomial $x^m f\left(\frac{1}{x}\right)$.*

Proof. We generate the desired straight line program Q in an inductive manner. Namely, if a gate g in P computes a polynomial R_g then the corresponding gate in Q computes a number $m_g \geq \deg(R_g)$ (the gate itself does not compute a number, to be precise, but our reduction algorithm does) and the polynomial $x^{m_g} R_g\left(\frac{1}{x}\right) \in \mathbb{Z}[x]$. It is clear how to do it for leaf nodes. Suppose $g = g_1 + g_2$ is a $+$ gate in P. So we have already computed integers m_{g_1}, m_{g_2} and polynomials $x^{m_{g_1}} R_{g_1}\left(\frac{1}{x}\right), x^{m_{g_2}} R_{g_2}\left(\frac{1}{x}\right)$. We consider $m_g := m_{g_1} + m_{g_2}$. We then have:

$$x^{m_g} R_g\left(\frac{1}{x}\right) = x^{m_{g_2}} x^{m_{g_1}} R_{g_1}\left(\frac{1}{x}\right) + x^{m_{g_1}} x^{m_{g_2}} R_{g_2}\left(\frac{1}{x}\right).$$

We also construct a straight-line program of length $O(s)$ that simultaneously computes x^{m_h} for all gates h in P. With this, we can compute $x^{m_g} R_g\left(\frac{1}{x}\right)$ using 3 additional gates. This implies the straight-line program for Q can be implemented using only $O(s)$ gates. Similarly for a $\times$ gate $g = g_1 \times g_2$, we can simply use $x^{m_g} R_g\left(\frac{1}{x}\right) = x^{m_{g_1}+m_{g_2}} R_{g_1}\left(\frac{1}{x}\right) R_{g_2}\left(\frac{1}{x}\right)$ with $m_g = m_{g_1} + m_{g_2}$. By induction it is also clear that at the top gate g, we have $m_g \leq 2^s$. It remains to describe a straight-line program of length $O(s)$ which computes x^{m_h} for all gates h in P. Consider the straight-line program P' obtained from P by changing every addition gate into a multiplication gate. If g' is a gate in P' corresponding to the gate g in P, then one can show via induction that $R_{g'}(x) = x^{m_g}$. This gives the desired straight-line program. ☐

Lemma 2.5. DegSLP $\leq_{\mathsf{P}}$ OrdSLP.

Proof. Suppose we are given a straight line program P of length s computing a polynomial $f \in \mathbb{Z}[x]$. By using Lemma 2.4, we compute:

1. A number $m \in \mathbb{N}$ such that $\deg(f) \leq m \leq 2^s$.
2. A straight line program Q of length $O(s^2)$ such that Q computes the polynomial $x^m f\left(\frac{1}{x}\right) \in \mathbb{Z}[x]$.

Now it clear that:

$$\deg(f) \leq d \iff \operatorname{ord}\left(x^m f\left(\frac{1}{x}\right)\right) \geq (m-d).$$

Hence the claim follows. ☐

Proof of the following Lemma 2.6 is almost same to that of Lemma 2.5, hence we omit it.

Lemma 2.6. OrdSLP $\leq_{\mathsf{P}}$ DegSLP.

Theorem 2.2. OrdSLP $\equiv_{\mathsf{P}}$ DegSLP.

Proof. Follows immediately from Lemma 2.5 and Lemma 2.6. ☐

Theorem 2.3. OrdSLP $\equiv_{\mathsf{P}}$ DegSLP $\leq_{\mathsf{P}}$ Div2SLP.

Proof. We only need to show that OrdSLP $\leq_{\mathsf{P}}$ Div2SLP. Suppose we are given a straight line program P of length s computing a polynomial $f \in \mathbb{Z}[x]$ and $\ell \in \mathbb{N}$ in binary, we want to decide if $\operatorname{ord}(f) \geq \ell$. We know that $\|f\|_\infty \leq 2^{2^s}$, where $\|f\|_\infty$ is the maximum absolute value of coefficients of $f(x)$. We now construct an SLP which computes $f(B)$ where B is a suitably chosen large integer which we will specify in a moment. If $\operatorname{ord}(f) \geq \ell$ then clearly B^ℓ divides $f(B)$. Now consider the case when $\operatorname{ord}(f) < \ell$. So we have $f = x^m(f_0 + xg)$ for some $f_0 \in \mathbb{Z}, g \in \mathbb{Z}[x]$ and $m < \ell$. In this case we have:

$$f(B) = B^m(f_0 + Bg(B)).$$

If B is chosen large enough then B does not divide $f_0 + Bg(B)$ and hence B^ℓ does not divide $f(B)$. It can be verified that choosing $B = 2^{2^{3s}}$ suffices for this argument. It is also not hard to see that an SLP for $f(B)$ can be constructed in polynomial time. Hence we conclude:

$$\operatorname{ord}(f) \geq \ell \iff 2^{\ell 2^{3s}} \text{ divides } f(2^{2^{3s}}).$$

This completes the reduction. □

Problem 2.1. *What is the exact complexity of* Div2SLP*?*

Now we show that Div2SLP is in CH, this claim follows by employing ideas from [All+09].

Lemma 2.7. Div2SLP *is in* CH.

Proof. Given a straight-line program representing $N \in \mathbb{Z}$, and a natural number ℓ in binary, we want to decide if 2^ℓ divides $|N|$, *i.e.*, if the ℓ least significant bits of $|N|$ are zero. We show that this can be done in $\mathsf{coNP}^{\mathrm{BitSLP}}$. The condition $2^\ell \nmid |N|$ is equivalent to the statement that at least one bit in ℓ least significant bits of $|N|$ is one. Hence there is a witness of this statement, *i.e.*, the index $i \leq \ell$ such that i^{th} bit of $|N|$ is one. By using the BitSLP oracle, we can verify the existence of such a witness in polynomial time. Therefore Div2SLP $\in \mathsf{coNP}^{\mathrm{BitSLP}}$. By using [All+09, Theorem 4.1], we get that Div2SLP $\in \mathsf{coNP}^{\mathsf{CH}} \subseteq \mathsf{CH}$. □

In Appendix B, we give a more general proof that shows that "SLP versions" of problems in dlogtime uniform TC_0 are in CH.

3 SLPs as Sum of Two and Fewer Squares

This section is primarily concerned with studying the complexity of 2SoSSLP. To this end, we first recall the following renowned Theorem 3.1 which characterizes when a natural number is a sum of two squares.

Theorem 3.1 ([Dud12, Section 18])**.** *An integer $n > 1$ is not* 2SoS *if and only if the prime-power decomposition of n contains a prime of the form $4k+3$ with an odd power.*

When the input integer n is given explicitly as a bit string, Theorem 3.1 illustrates that a factorization oracle suffices to determine whether n is a 2SoS. In fact, we are not aware of any algorithm that bypasses the need for factorization. For $x \in \mathbb{Z}_+$, let $B(x)$ denote the number of 2SoS integers in $[x]$. Landau's Theorem [Lan08] gives the following asymptotic formula for $B(x)$.

Theorem 3.2 ([Lan08])**.** $B(x) = K\frac{x}{\sqrt{\ln x}} + O\left(\frac{x}{\ln^{3/2} x}\right)$ *as $x \to \infty$, where K is the Landau-Ramanujan constant with $K \approx 0.764$.*

Ideally, we want to use the above Theorem 3.2 on the density of 2SoS to show that PosSLP reduces to 2SoSSLP, as we did for 3SoSSLP. There are two issues with this approach:

1. The density of 2SoS integers is not as high as 3SoS integers, hence to find the next 2SoS integer after a given $N \in \mathbb{N}$ might require a larger shift (as compared to the shift of 2 for 3SoS). This issue is overcome below by using NP oracle reductions instead of P reductions.

2. A more serious issue is that Theorem 3.2 says something about the density of 2SoS integers only asymptotically, as $x \to \infty$. But for this idea of finding the next 2SoS integer after a given integer only works if this density bound is true for all intervals of naturals. This issue is side stepped by relying on the Conjecture 3.1 below.

Let q and r be positive integers such that $1 \le r < q$ and $\gcd(q,r) = 1$. We use $G_{q,r}(x)$ to denote maximum gap between primes in the arithmetic progression $\{qn + r \mid n \in \mathbb{N}, qn + r \le x\}$. We use $\varphi(n)$ to denote the Euler's totient function, *i.e.*, the number of positive $m \le n$ with $\gcd(m,n) = 1$.

Conjecture 3.1 (Generalized Cramér conjecture A, [Kou18])**.** For any $q > r \ge 1$ with $\gcd(q,r) = 1$, we have

$$G_{q,r}(p) = O(\varphi(q)\log^2 p).$$

3.1 Lower Bounds for 2SoSSLP

Lemma 3.1. EquSLP $\le_{\mathsf{P}}$ 2SoSSLP.

Proof. Given a straight-line program representing an integer N, we want to decide whether $N = 0$. Suppose $M = N^2$. We have $M \ge 0$ and $M = 0$ iff $N = 0$. If $M \ne 0$ then by employing Theorem 3.1, $3M^2$ cannot be a 2SoS. Hence $3M^2$ is a 2SoS iff $M = 0$.

□

Proof of Theorem 1.5. Given a straight-line program of size s representing an integer N, we want to decide whether $N > 0$. Choose $M = 2^{3s}$. We compute the value of $N \bmod T$, where $T := 2M + 1$. This can be done in $\mathsf{poly}(s)$ time by simulating the computation of the given SLP (which computes N) modulo T. If $|N| \le M$, then we can even recover the exact value of N by knowing $N \bmod T$. So by assuming $|N| \le M$, we guess the exact value of N. Let us call this guessed value (computed by knowing $N \bmod T$) to be N'. Then by using EquSLP oracle (which can be simulated by 2SoSSLP oracle using Lemma 3.1), we check if the equality $N = N'$ is actually true. If $N = N'$ then we can easily determine the sign of N. Otherwise our assumption $|N| \le M$ is false and hence we can assume that $|N| > M$.

Suppose $p \ge |N|$ is the smallest prime of the form $4k + 1$. By using the results in [Bre32], we know that $p \le 2|N|$ for $|N| \ge 7$. By using Conjecture 3.1 with $q = 4, r = 1$, we get that $p \le |N| + O(\varphi(4)\log^2 p) \le |N| + c\log^2|N|$ for some absolute constant c. For large enough $|N|$, it implies that $p \le |N| + \log^3|N| \le |N| + 2^{3s}$. Therefore $p - |N| \le M$. By using Theorem 3.1, we know that p is a 2SoS. Now we guess the witness $S := p - |N|$, clearly this has a binary description of size at most $O(s)$. Now we use 2SoSSLP oracle to check if $N + S$ is a 2SoS. If $N > 0$ then clearly such a witness exists. On the other hand if $N < 0$ then we know that $N < -M$ and $N + S$ cannot be a 2SoS. Therefore if Conjecture 3.1is true then PosSLP $\in \mathsf{NP}^{\text{2SoSSLP}}$. □

Similarly to 2SoSSLP and 3SoSSLP, one can also study the complexity of the following problem SquSLP.

Problem 3.1 (SquSLP, Problem 7 in [JG23])**.** *Given a straight-line program representing $N \in \mathbb{Z}$, decide whether $N = a^2$ for some $a \in \mathbb{Z}$.*

SquSLP was shown to be decidable in randomized polynomial time in [JG23, Sec 4.2], assuming GRH. The complexity of 2SoSSLP remains an intriguing open problem. If 2SoSSLP were to be in P then this would disprove Conjecture 3.1 or prove that PosSLP $\in$ NP, neither of which is currently known.

4 Polynomials as Sum of Squares

4.1 Positivity of Polynomials

Analogous to PosSLP, we also study the positivity problem for polynomials represented by straight line programs. In particular, we study the following problem, called PosPolySLP.

Problem 4.1 (PosPolySLP)**.** *Given a straight-line program representing a univariate polynomial $f \in \mathbb{Z}[x]$, decide if f is positive, i.e., $f(x) \geq 0$ for all $x \in \mathsf{R}$.*

It is known that every positive univariate polynomial f can be written as sum of two squares. The formal statement (Lemma A.1) and its folklore proof can be found in the appendix.

Now we look at the rational variant of the Lemma A.1. Suppose $f \in \mathbb{Z}[x] \subset \mathbb{Q}[x]$ is a positive polynomial. We know that it can be written as sum of squares of two real polynomials. Can it also be written as sum of squares of rational polynomials? In this direction, Landau proved that each positive polynomial in $\mathbb{Q}[x]$ can be expressed as a sum of at most eight polynomial squares in $\mathbb{Q}[x]$ [Edm06]. Pourchet improved this result and proved that only five or fewer squares are needed [Y71].

We now show that PosPolySLP is coNP-hard, this result follows from an application of results proved in [PS07]. Suppose W is a 3-SAT formula on n literals $x_1, x_2, \ldots, x_n$ with $W = C_1 \wedge C_2 \wedge \cdots \wedge C_\ell$, here C_i is a clause composed of 3 literals. We choose any n distinct odd primes $p_1 < p_2 < \cdots < p_n$. So x_i is associated with the prime p_i. Thereafter, we define $M := \prod_{i \in [n]} p_i$. The following Theorem 4.1 was proved in [PS07].

Theorem 4.1 ([PS07])**.** *One can construct a SLP C of size $\mathsf{poly}(p_n, \ell)$ which computes a polynomial $P_M(W)$ of the form:*

$$P_M(W) := \sum_{i \in [\ell]} (F_M(C_i))^2$$

such that $P_M(W)$ has a real root iff W is satisfiable. Here, $F_M(C_i)$ is a univariate polynomial that depends on C_i (see [PS07] for more details).

Theorem 4.2 (Theorem 1.2 in [BLR09])**.** *Let $f \in \mathbb{Z}[x]$ be a univariate polynomial of degree d taking only positive values on the interval $[0,1]$. Let τ be an upper bound on the bit size of the coefficients of f. Let m denote the minimum of f over $[0,1]$. Then*

$$m > \frac{3^{d/2}}{2^{(2d-1)\tau}(d+1)^{2d-1/2}}.$$

The theorem above proves the lower bound for the interval $[0,1]$. Next, we extend it to the whole real line.

Lemma 4.1. *Let $f \in \mathbb{Z}[x]$ be a positive univariate polynomial of degree d. Let τ be an upper bound on the bit size of the coefficients of f. Let m denote the minimum of f over $\mathbb{R}$. If $m \neq 0$ then*

$$m > \frac{3^{d/2}}{2^{(2d-1)\tau}(d+1)^{2d-1/2}}.$$

Proof. We assume $m \neq 0$. Consider the reverse polynomial $f_{\text{rev}} := x^d f\left(\frac{1}{x}\right)$. It is is clear that f_{rev} is positive on $[0,\infty)$. Moreover, f_{rev} has degree d and τ is an upper bound on the bit size of its coefficients. By employing Theorem 4.2 on f_{rev}, we infer that

$$\min_{a \in [0,1]} f_{\text{rev}}(a) > \frac{3^{d/2}}{2^{(2d-1)\tau}(d+1)^{2d-1/2}}.$$

Theorem 4.2 implies that:

$$\min_{a\in[0,1]} f(a) > \frac{3^{d/2}}{2^{(2d-1)\tau}(d+1)^{2d-1/2}}. \tag{1}$$

Now consider a $\lambda \in [0,1]$, we have:

$$f\left(\frac{1}{\lambda}\right) = \frac{f_{\mathrm{rev}}(\lambda)}{\lambda^d} \geq f_{\mathrm{rev}}(\lambda) > \frac{3^{d/2}}{2^{(2d-1)\tau}(d+1)^{2d-1/2}}. \tag{2}$$

By combining Equation (1) and Equation (2), we obtain that:

$$\min_{a\in[0,\infty)} f(a) > \frac{3^{d/2}}{2^{(2d-1)\tau}(d+1)^{2d-1/2}}.$$

By repeating the above argument on $f(-x)$ instead of $f(x)$, we obtain:

$$m = \min_{a\in(-\infty,\infty)} f(a) > \frac{3^{d/2}}{2^{(2d-1)\tau}(d+1)^{2d-1/2}}.$$

□

Proof of Theorem 1.8. Suppose W is 3-SAT formula on n literals $x_1, x_2, \ldots, x_n$ with $W = C_1 \wedge C_2 \wedge \cdots \wedge C_\ell$, here C_i is a clause composed of 3 literals. By using Theorem 4.1, we can construct a SLP of size $\mathsf{poly}(p_n, \ell)$ which computes a polynomial $P(W) \in \mathbb{Z}[x]$ such that $P(W)$ has a real root iff W is satisfiable. (Recall that $p_1 < \cdots < p_n$ was a sequence of odd primes.) Since $P(W)$ is a sum of squares, $P(W)$ is positive. Suppose m is the minimum value of $P(W)$ over $\mathbb{R}$. We know that $m \geq 0$.

By the prime number theorem we can assume $p_n = O(n \log n)$. Moreover, it is easy to see that $\ell \leq 8n^3$. Hence the constructed SLP is of size $s = \mathsf{poly}(n)$. Suppose τ is an upper bound on the bit size of the coefficients of $P(W)$. It is easy to see that $\deg(P(W)) \leq 2^s$ and $\tau \leq 2^s$. If W is not satisfiable then we know that $m \neq 0$ and therefore Lemma 4.1 implies that

$$\log(m) > 2^{s-1}\log 3 - (2^{s+1}-1)2^s - (2^{s+1}-1/2)\log(2^s+1) > -2^{2s+2}.$$

Hence

$$m > \frac{1}{2^{2^{2s+2}}}.$$

Suppose $B = 2^{2^{2s+2}}$. Then $B \cdot P(W) - 1$ is positive iff $m > 0$. Hence we have:

$$B \cdot P(W) - 1 \text{ is positive iff } W \text{ is unsatisfiable.}$$

Moreover $B \cdot P(W) - 1$ has a SLP of size $O(s) = \mathsf{poly}(n)$ and this SLP can be constructed in time $\mathsf{poly}(n)$. Since determining the unsatisfiability of W is coNP-complete, it follows that PosPolySLP is coNP-hard. □

4.2 Checking if a Polynomial is a Square

In light of [Y71]'s result and Theorem 1.8, we also study the following related problem SqPolySLP. Another motivation to study this problem also comes from the quest for studying the complexity of factors of polynomials. In this context, one wants to prove that if a polynomial can be computed a small arithmetic circuit then so can be its factors. In this direction, Kaltofen showed that if a polynomial $f = g^e h$ can be computed an arithmetic circuit of size s and g, h are coprime, then g can also be computed by a circuit of size $\mathsf{poly}(e, \deg(g), s)$ [Kal87]. When $f = g^e$, Kaltofen also showed that g can be computed an arithmetic circuit of size $\mathsf{poly}(\deg(g), s)$ [Kal87]. This question for finite fields is asked as an open question in [KSS14]. What if we do not want to find a small circuit for polynomial g in case $f = g^e$ but only want to determine if f is e^{th} power of some polynomial. And in this decision problem, we want to avoid the dependency on $\deg(g)$ in running time, which can be exponential in s. We study this problem for $e = 2$ in SqPolySLP, but our results work for any arbitrary constant e.

Problem 4.2 (SqPolySLP)**.** *Given a straight-line program representing a univariate polynomial $f \in \mathbb{Z}[x]$, decide if $\exists g \in \mathbb{Z}[x]$ such that $f = g^2$.*

One can also study the complexity of determining if the given univariate polynomial can be written as a sum of two, three or four squares but in this section, we only focus on the problem SqPolySLP. The following Theorem 4.3 hints to an approach that SqPolySLP can be reduced to SquSLP.

Theorem 4.3 (Theorem 4 in [Mur08])**.** *For $f \in \mathbb{Z}[x]$, $\exists g \in \mathbb{Z}[x]$ with $f = g^2$ iff $\forall t \in \mathbb{Z}$, $f(t)$ is a perfect square.*

We shall use an effective variant of Theorem 4.3 which follows the following effective variant of the Hilbert's irreducibility theorem. For an integer polynomial f, $H(f)$ is the height of f, *i.e.*, the maximum of the absolute values of the coefficients of f.

Theorem 4.4 ([Wal05; DW08])**.** *Suppose $P(T,Y)$ is an irreducible polynomial in $\mathbb{Q}[T,Y]$ with $\deg_Y(P) \geq 2$ and with coefficients in $\mathbb{Z}$ assumed to be relatively prime. Suppose B is a positive integer such that $B \geq 2$. We define:*

$$\begin{aligned} m &:= \deg_T(P) \\ n &:= \deg_Y(P) \\ H &:= \max(H(P), e^e) \\ S(P,B) &:= |\{1 \leq t \leq B \mid P(t,Y) \text{ is reducible in } \mathbb{Q}[Y]\}| \end{aligned}$$

Then we have:

$$S(P,B) \leq 2^{165} m^{64} 2^{296n} \log^{19}(H) B^{\frac{1}{2}} \log^5(B).$$

Corollary 4.1. *Suppose $f(x) \in \mathbb{Z}[x]$ is an integer polynomial computed by a SLP of size s. Define $S(f) := \left|\{1 \leq t \leq 2^{200s} \mid f(t) \text{ is a square }\}\right|$. If f is not a square then we have:*

$$S(f) < 2^{800} s^5 2^{183s}.$$

Proof. Consider the polynomial $P(T,Y) := Y^2 - f(T)$. Since $f(x)$ is not a square, we infer that $P(T,Y)$ is an irreducible polynomial in $\mathbb{Q}[T,Y]$. Now we employ Theorem 4.4 on $P(T,Y)$ with $B = 2^{200s}$, we have $m \leq 2^s, n = 2$ and $H \leq 2^{2^s}$. In this case, we have $S(P,B) = S(f)$. By using Theorem 4.4 , we have:

$$S(f) \leq 2^{165} 2^{64s} 2^{592} 2^{19s} 2^{100s} (200s)^5 < 2^{800} s^5 2^{183s}.$$

□

Corollary 4.1 implies a randomized polynomial time algorithm for SqPolySLP, as demonstrated below in Theorem 4.5.

Theorem 4.5. SqPolySLP *is in* coRP.

Proof. Given an integer polynomial $f(x)$ computed by a SLP of size s, we want to decide if $f = g^2$ for some $g \in \mathbb{Z}[x]$. We sample a positive integer uniformly at random from the set $\{1 \leq t \leq 2^{200s} \mid t \in \mathbb{N}\}$. Using the algorithm in [JG23, Sec 4.2], we test if $f(t)$ is a square. We output "Yes" if $f(t)$ is a square. If $f = g^2$ for some $g \in \mathbb{Z}[x]$, then we always output "Yes". Suppose $f \neq g^2$ for any $g \in \mathbb{Z}[x]$. By using Corollary 4.1, we obtain that:

$$\Pr[f(t) \text{ is a square}] < \frac{2^{800}s^5 2^{183s}}{2^{200s}} < \frac{1}{100} \text{ for } s > 100.$$

Hence with probability at least 0.99 we sample a t such that $f(t)$ is not a square. The algorithm for SquSLP verifies that $f(t)$ is not a square with probability at least $\frac{1}{3}$ [JG23, Sec 4.2]. Hence we output "No" with probability at least 0.33. This implies SqPolySLP $\in$ coRP. $\square$

5 Conclusion and Open Problems

We studied the connection between PosSLP and problems related to the representation of integers as sums of squares, drawing on Lagrange's four-square theorem from 1770. We investigated variants of the problem, considering whether the positive integer computed by a given SLP can be represented as the sum of squares of two or three integers. We analyzed the complexity of these variations and established relationships between them and the original PosSLP problem. Additionally, we introduced the Div2SLP problem, which involves determining if a given SLP computes an integer divisible by a given power of 2. We showed that Div2SLP is at least as hard as DegSLP. We also showed the relevance of Div2SLP in connecting the 3SoSSLP to PosSLP. In contrast to PosSLP, we also showed that the polynomial variant of the PosSLP problem is unconditionally coNP-hard. Overall, this paper contributes to a deeper understanding of decision problems associated with SLPs and provides insights into the computational complexity of problems related to the representation of integers as sums of squares. A visual representation illustrating the problems discussed in this paper and their interrelations is available in Figure 1. Our results open avenues for further research in this area; in particular, we highlight the following research avenues:

1. What is the complexity of Div2SLP? We showed it is DegSLP hard. Is it NP-hard too? How does it relate to PosSLP?
2. Can we prove Theorem 1.5 without relying on Conjecture 3.1?
3. One can also study the problems of deciding whether a given SLP computes an integer univariate polynomial, which can be written as the sum of two, three, or four squares. We studied these questions for integers in this paper. But it makes for an interesting research to study these questions for polynomials.
4. And finally, can we prove unconditional hardness results for PosSLP?

Acknowledgments: We would like to thank Robert Andrews for providing a simpler proof of Lemma 2.4. We had a proof of it that was a bit longer. Robert Andrews simplified the proof after reviewing our proof in a personal communication.

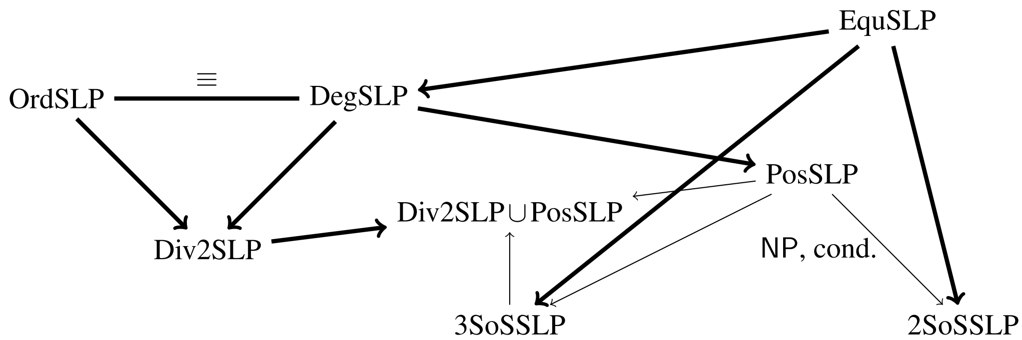

Figure 1: A visualization of the relations between the problems studied in this work. An arrow means that there is a Turing reduction. A thicker arrow indicates a polynomial time many one reduction. The reduction from PosSLP to 2SoSSLP is nondeterministic and depends on Conjecture 3.1.

References

[All+01] Eric Allender et al. "Time-Space Tradeoffs in the Counting Hierarchy". In: *Proceedings of the 16th Annual IEEE Conference on Computational Complexity, Chicago, Illinois, USA, June 18-21, 2001*. IEEE Computer Society, 2001, pp. 295–302. DOI: 10.1109/CCC.2001.933896. URL: https://doi.org/10.1109/CCC.2001.933896.

[All+09] Eric Allender et al. "On the Complexity of Numerical Analysis". In: *SIAM Journal on Computing* 38.5 (2009), pp. 1987–2006. DOI: 10.1137/070697926. eprint: https://doi.org/10.1137/070697926. URL: https://doi.org/10.1137/070697926.

[Ank57] N. C. Ankeny. "Sums of three squares". In: *Proceedings of the American Mathematical Society* 8.2 (1957), pp. 316–319. DOI: 10.1090/s0002-9939-1957-0085275-8. URL: https://doi.org/10.1090/s0002-9939-1957-0085275-8.

[BC06] Peter Bürgisser and Felipe Cucker. "Counting complexity classes for numeric computations II: Algebraic and semialgebraic sets". In: *Journal of Complexity* 22.2 (2006), pp. 147–191. ISSN: 0885-064X. DOI: https://doi.org/10.1016/j.jco.2005.11.001. URL: https://www.sciencedirect.com/science/article/pii/S0885064X05001081.

[BJ23] Peter Bürgisser and Gorav Jindal. *On the Hardness of PosSLP*. 2023. arXiv: 2307.08008 [cs.CC].

[BLR09] Saugata Basu, Richard Leroy, and Marie-Francoise Roy. *A bound on the minimum of a real positive polynomial over the standard simplex*. 2009. arXiv: 0902.3304 [cs.SC].

[Blu+97] Lenore Blum et al. *Complexity and Real Computation*. Berlin, Heidelberg: Springer-Verlag, 1997. ISBN: 0387982817.

[Bre32] R. Breusch. "Zur Verallgemeinerung des Bertrandschen Postulates, dass zwischen x und 2x stets Primzahlen liegen". In: *Mathematische Zeitschrift* 34 (1932), pp. 505–526. URL: http://eudml.org/doc/168326.

[DSS18] Pranjal Dutta, Nitin Saxena, and Amit Sinhababu. "Discovering the Roots: Uniform Closure Results for Algebraic Classes under Factoring". In: *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2018. Los Angeles, CA, USA: Association for Computing Machinery, 2018, pp. 1152–1165. ISBN: 9781450355599. DOI: 10.1145/3188745.3188760. URL: https://doi.org/10.1145/3188745.3188760.

[Dud12] U. Dudley. *Elementary Number Theory: Second Edition*. Dover Books on Mathematics. Dover Publications, 2012. ISBN: 9780486134871.

[Dut+21] Pranjal Dutta et al. "Arithmetic Circuit Complexity of Division and Truncation". In: *36th Computational Complexity Conference (CCC 2021)*. Ed. by Valentine Kabanets. Vol. 200. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 25:1–25:36. ISBN: 978-3-95977-193-1. DOI: 10.4230/LIPIcs.CCC.2021.25. URL: https://drops.dagstuhl.de/opus/volltexte/2021/14299.

[DW08] Pierre Debes and Yann Walkowiak. "Bounds for Hilbert's Irreducibility Theorem". In: *Pure and Applied Mathematics Quarterly* 4.4 (2008), pp. 1059–1083. ISSN: 1558-8602. DOI: 10.4310/pamq.2008.v4.n4.a4. URL: http://dx.doi.org/10.4310/PAMQ.2008.v4.n4.a4.

[Edm06] Landau Edmund. "Über die Darstellung definiter Funktionen durch Quadrate". ger. In: *Mathematische Annalen* 62 (1906), pp. 272–285. URL: http://eudml.org/doc/158257.

[Gau01] C.F. Gauss. *Disquisitiones arithmeticae*. Apud G. Fleischer, 1801. URL: https://books.google.de/books?id=OwX6GwAACAAJ.

[Hil09] David R. Hilbert. "Beweis für die Darstellbarkeit der ganzen Zahlen durch eine feste Anzahlnter Potenzen (Waringsches Problem)". In: *Mathematische Annalen* 67 (1909), pp. 281–300.

[JG23] Gorav Jindal and Louis Gaillard. "On the Order of Power Series and the Sum of Square Roots Problem". In: *Proceedings of the 2023 International Symposium on Symbolic and Algebraic Computation*. ISSAC '23. , Tromsø, Norway, Association for Computing Machinery, 2023, pp. 354–362. DOI: 10.1145/3597066.3597079. URL: https://doi.org/10.1145/3597066.3597079.

[Kal87] E. Kaltofen. "Single-factor Hensel lifting and its application to the straight-line complexity of certain polynomials". In: *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*. STOC '87. New York, New York, USA: Association for Computing Machinery, 1987, pp. 443–452. ISBN: 0897912217. DOI: 10.1145/28395.28443. URL: https://doi.org/10.1145/28395.28443.

[Kou18] Alexei Kourbatov. *On the distribution of maximal gaps between primes in residue classes*. 2018. arXiv: 1610.03340 [math.NT].

[KSS14] Swastik Kopparty, Shubhangi Saraf, and Amir Shpilka. "Equivalence of polynomial identity testing and deterministic multivariate polynomial factorization". In: *2014 IEEE 29th Conference on Computational Complexity (CCC)*. IEEE. 2014, pp. 169–180.

[Lan08] Edmund Landau. *Über die Einteilung der positiven ganzen Zahlen in vier Klassen nach der Mindestzahl der zu ihrer additiven Zusammensetzung erforderliche Quadrate*. ger. [S.l.]: [s.n.] [S.l.], 1908. URL: https://books.google.de/books?id=e3XBnQAACAAJ.

[Lan09] E. Landau. *Über die Einteilung der positiven ganzen Zahlen in vier Klassen nach der Mindestzahl der zu ihrer additiven Zusammensetzung erforderlichen Quadrate*. 1909.

[Leg97] Adrien Marie Legendre. *Essai Sur La Théorie Des Nombres*. fre. Paris: Duprat, 1797. URL: http://eudml.org/doc/204253.

[Mal96] Gregorio Malajovich. *An effective version of Kronecker's theorem on simultaneous Diophantine approximation*. Tech. rep. Citeseer, 1996.

[Mor58] LJ Mordell. "On the representation of a number as a sum of three squares". In: *Rev. Math. Pures Appl* 3 (1958), pp. 25–27.

[Mur08] M Ram Murty. "Polynomials assuming square values". In: *Number theory and discrete geometry* (2008), pp. 155–163.

[NZM91] Ivan Niven, Herbert S. Zuckerman, and Hugh L. Montgomery. *An Introduction to the Theory of Numbers*. English. Hardcover. Wiley, Jan. 1991, p. 544. ISBN: 978-0471625469. URL: https://lead.to/amazon/com/?op=bt&la=en&cu=usd&key=0471625469.

[PS07] Daniel Perrucci and Juan Sabia. "Real roots of univariate polynomials and straight line programs". In: *Journal of Discrete Algorithms* 5.3 (2007). Selected papers from Ad Hoc Now 2005, pp. 471–478. ISSN: 1570-8667. DOI: https://doi.org/10.1016/j.jda.2006.10.002. URL: https://www.sciencedirect.com/science/article/pii/S1570866706000840.

[Ruz81] Walter L. Ruzzo. "On Uniform Circuit Complexity". In: *J. Comput. Syst. Sci.* 22.3 (1981), pp. 365–383. DOI: 10.1016/0022-0000(81)90038-6. URL: https://doi.org/10.1016/0022-0000(81)90038-6.

[Shi16] Yaroslav Shitov. *How hard is the tensor rank?* 2016. arXiv: 1611.01559 [math.CO].

[Tiw92] Prasoon Tiwari. "A problem that is easier to solve on the unit-cost algebraic RAM". In: *Journal of Complexity* 8.4 (1992), pp. 393–397. ISSN: 0885-064X. DOI: https://doi.org/10.1016/0885-064X(92)90003-T. URL: https://www.sciencedirect.com/science/article/pii/0885064X9290003T.

[Wal05] Yann Walkowiak. "Théorème d'irréductibilité de Hilbert effectif". fre. In: *Acta Arithmetica* 116.4 (2005), pp. 343–362. URL: http://eudml.org/doc/277977.

[Y71] Pourchet Y. "Sur la représentation en somme de carrés des polynômes à une indéterminée sur un corps de nombres algébriques". fre. In: *Acta Arithmetica* 19.1 (1971), pp. 89–104. URL: http://eudml.org/doc/205020.

A Missing Proofs

Lemma A.1. *For every positive polynomial $f \in \mathbb{R}[x]$, there exist $g,h \in \mathbb{R}[x]$ such that $f = g^2 + h^2$*

Proof. If $f(x) \geq 0$ for $x \in \mathsf{R}$ and α is a real root of f, then it must have even multiplicity. We have $(x-\alpha)^2 = (x-\alpha)^2 + 0^2$. We use $\imath$ to denote $\sqrt{-1}$. If $\beta = s + \imath t$ and $\overline{\beta} = s - \imath t$ are a complex-conjugate pair of roots of f then, $(x-\beta)(x-\overline{\beta}) = (x-s-\imath t)(x-s+\imath t) = (x-s)^2 + t^2$ is a sum of two squares. Now the claim follows by using the identity $(a^2+b^2)(c^2+d^2) = (ac-bd)^2 + (ad+bc)^2$. □

B Alternative Proof of Lemma 2.7

We prove a general theorem on how to show that problems involving SLPs are in CH. It is similar to the proof of [All+01, Lem 5]. Let C be a Boolean circuit in TC_0. C consists of unbounded AND, unbounded OR, and unbounded majority gates (MAJ). According to [Ruz81], when a family (C_n) is in dlogtime-uniform TC_0, this means that there is a deterministic Turing machine (DTM) that decides in time $O(\log n)$ whether given (n, f, g) the gate f is connected to the gate g and whether given (n, f, t) the gate f has type t. All numbers are given in binary. For a language $B \subseteq \{0,1\}^*$, let $\mathrm{SLP}(B)$ be the language:

$$\mathrm{SLP}(B) := \{P \mid P \text{ is an SLP computing a number } N \text{ such that } \mathrm{Bin}(N) \in B \text{ (as a binary string)}\}$$

This can be viewed as the "SLP-version" of B.

Lemma B.1. *Let B be in dlogtime-uniform* TC_0. *Then* $\mathrm{SLP}(B) \in \mathsf{CH}$.

Proof. The proof is by induction on the depth. We prove the more general statement: Let M be a DTM from the definition of dlogtime and (C_n) be the sequence of circuits for B. Let P be the given SLP encoding a number N. Given (P, g, b) we can decide in CH_{t+c} whether the value of the gate g on input N given by P is b. t is the depth of g. If $t = 0$, then g is an input gate. Thus this problem is BitSLP which is in CH_c for some c. If $t > 0$, then we have to decide whether the majority of the gates that are children of g are 1. This can be done using a PP-machine with oracle to CH_{c+t-1}. We guess a gate f and check using the DTM M whether f is a predecessor of g. If not we add an accepting and rejecting path. If yes, we use the oracle to check whether f has value 1. If yes we accept and otherwise, we reject. □

It is easy to see that checking whether the ℓ least significant bits of a number given in binary are 0 can be done in dlogtime-uniform TC_0. Thus Div2SLP is in CH by the lemma above.

C Reduction from multivariate DegSLP to univariate DegSLP

We use mDegSLP to denote the multivariate variant of the DegSLP problem, which we define formally below.

Problem C.1 (mDegSLP). *Given a straight line program representing a polynomial $f \in \mathbb{Z}[x_1, x_2, \ldots, x_n]$, and given a natural number d in binary, decide whether* $\deg(f) \le d$.

mDegSLP was simply called DegSLP in [All+09]. Now we recall the proof in [All+09], to show that to study the hardness of mDegSLP, it is enough to focus on its univariate variant DegSLP. To this end, we note the following Observation C.1.

Observation C.1 ([All+09])**.** mDegSLP is equivalent to DegSLP under deterministic polynomial time many one reductions.

Proof. We only need to show that mDegSLP reduces o DegSLP under deterministic polynomial time many one reductions, other direction is trivial. Suppose we are given an SLP of size s which computes $f \in \mathbb{Z}[x_1, x_2, \ldots, x_n]$, and we want to decide whether $\deg(f) \le d$ for a given $d \in \mathbb{N}$. Suppose $D = \deg(f)$. For all $i \in \{0, 1, \ldots, D\}$, we use f_i to denote the homogeneous degree i part of f. Now notice that for any $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n) \in \mathbb{Z}^d$, we have:

$$f(y\alpha) = f(y\alpha_1, y\alpha_2, \ldots, y\alpha_n) = \sum_{i=0}^{D} y^i f_i(\alpha),$$

where y is a fresh variable. So if α is chosen such that $f_D(\alpha)$ is non-zero then $\deg(f(y\alpha)) = \deg(f) = D$. If we choose $\alpha_i = 2^{2^{is^2}}$ then it can seen that $f_D(\alpha)$ is non-zero, see e.g. [All+09, Proof of Proposition 2.2]. SLPs computing α_i can be constructed using iterated squaring in polynomial time. Hence we can construct an SLP for $f(y\alpha)$ in polynomial time. By this argument, we know that $\deg(f(y\alpha)) \le d$ if and only if $\deg(f) \le d$. Therefore mDegSLP reduces to DegSLP under polynomial time many one reductions. □