Electronic Colloquium on Computational Complexity, Report No. 42 (2025)

# Polynomial-Time PIT from (Almost) Necessary Assumptions

Robert Andrews* Deepanshu Kush† Roei Tell‡

April 8, 2025

**Abstract**

The celebrated result of Kabanets and Impagliazzo (Computational Complexity, 2004) showed that PIT algorithms imply circuit lower bounds, and vice versa. Since then it has been a major challenge to understand the precise connections between PIT and lower bounds. In particular, a main goal has been to understand which lower bounds suffice to obtain efficient PIT algorithms, and how close are they to lower bounds that are necessary for the conclusion.

We construct polynomial-time PIT algorithms from lower bounds that are, up to relatively minor remaining gaps, necessary for the existence of such algorithms. That is, we prove that these lower bounds are, up to the mentioned minor gaps, both sufficient and necessary for polynomial-time PIT, over fields of characteristic zero. Over sufficiently large finite fields, we show a similar result wherein the PIT algorithm runs in time $n^{\log^{(c)}(n)}$, i.e. a power of $c$-iterated log for an arbitrarily large constant $c > 1$.

The key to these improvements is studying PIT versus lower bounds in the uniform setting, in which we focus on proving lower bounds for uniform arithmetic circuits and their variants (and on deducing algorithms from such lower bounds). Indeed, by working in this setting we obtain results that are significantly tighter than previously known results concerning polynomial-time PIT vs lower bounds, and are in fact also tighter than known hardness-vs-randomness connections in the Boolean setting.

Our results are obtained by combining recent techniques from Boolean hardness vs randomness, and in particular the generator of Chen and Tell (FOCS 2021), with the algebraic hitting-set generator of Guo, Kumar, Saptharishi, and Solomon (SIAM J. Computing 2022) along with the bootstrapping ideas of Agrawal, Ghosh, and Saxena (STOC 2018) and of Kumar, Saptharishi, and Tengse (SODA 2019).

*Cheriton School of Computer Science, University of Waterloo. Email: randrews@uwaterloo.ca.
†Department of Computer Science, University of Toronto. Email: deepkush@cs.toronto.edu.
‡Department of Computer Science, University of Toronto. Email: roei@cs.toronto.edu.

ISSN 1433-8092

# Contents

# 1 Introduction

Polynomial identity testing (PIT) is a central problem in algebraic complexity. In this paper we focus on what is arguably the most well-studied variant: We are given as input a description of an arithmetic circuit $C(x_1, ..., x_n)$ over $\mathbb{F}$ of size $\text{poly}(n)$ computing a nonzero polynomial of degree $\text{poly}(n)$, and our goal is to find $\vec{\alpha}$ such that $C(\vec{\alpha}) \neq 0$.[1] Of course, a trivial randomized algorithm can just pick $\vec{\alpha}$ at random (assuming $\mathbb{F}$ is large enough, or taking an extension field $\mathbb{K} \supseteq \mathbb{F}$), and when we refer to solving PIT we mean solving it by deterministic algorithms (i.e., derandomizing the trivial algorithm).

## 1.1 The question of PIT versus lower bounds

One reason for the importance of PIT is its connection to lower bounds. The study of this connection was initiated by Kabanets and Impagliazzo [KI04], who proved that PIT algorithms imply circuit lower bounds, and vice versa. Specifically, they showed that a PIT algorithm running in polynomial time (or even in sub-exponential time) implies that either there is a problem in $\mathsf{NTIME}[2^{\text{poly}(n)}]$ that is hard for polynomial-sized Boolean circuits, or the permanent is hard for polynomial-sized arithmetic circuits. They also showed partial converses; for example, if there is an exponential-time computable polynomial over $\mathbb{Z}$ that does not have sub-exponential sized arithmetic circuits, we can solve PIT over $\mathbb{Z}$ in quasipolynomial time.

The result of [KI04] revealed a connection between PIT and lower bounds, but it is far from the end of story: The lower bounds that are implied by PIT algorithms in this result are significantly weaker than lower bounds that suffice for PIT algorithms (due to the disjunctive conclusion, and to the fact that one of the lower bounds in the disjunction is only in NEXP). Thus, the result put forward a major open question:

**Problem 1.1.** *What are the connections between* PIT *and lower bounds? Specifically, what lower bounds suffice to obtain efficient* PIT *algorithms, and how close are they to lower bounds that are necessary for the conclusion?*

Let us briefly survey some of the progress that has been made on Problem 1.1 (for a recent survey, see Kumar and Saptharishi [KS19]). One line of works made progress by studying circuit lower bounds in non-standard models of computation. Specifically, Jansen and Santhanam [JS12] defined a hybrid model combining arithmetic complexity and Boolean complexity, dubbed $a \cdot \mathcal{C}$ for a Boolean class $\mathcal{C}$ (see also [KP11]). They showed an equivalence between sub-exponential time PIT algorithms and lower bounds for arithmetic circuits against a problem computable in this model with linearly many non-uniform advice bits. Carmosino, Impagliazzo, Kabanets, and Kolokolova [CIKK15] showed another equivalence, between subexponential-time PIT algorithms that work on average, and average-case circuit lower bounds in the model from [JS12] (without advice), for a notion of average-case complexity called "robustly often containment".

A different line of attack on Problem 1.1 focused on the direction "hardness implies PIT", trying to obtain a stronger conclusion; namely, the goal in these works is to obtain a faster PIT algorithm, compared to the quasipolynomial time (or sub-exponential time) algorithms considered in the results above. For example, Kumar, Saptharishi, and Tengse [KST19] (following Agrawal, Ghosh, and Saxena [AGS18]) showed how to bootstrap relatively slow hitting-set generators (i.e., "black-box" PIT algorithms) for circuits over few variables into hitting-set generators for circuits over many variables that yield PIT in almost polynomial time, i.e. time $n^{2^{2^{O(\log^*(n))}}}$ for $n$-sized circuits. And in another influential work, Guo, Kumar, Saptharishi, and Solomon [GKSS22] deduced a PIT algorithm that indeed runs in polynomial time, from a strong hardness assumption. (Jumping ahead, our results will crucially build on their work.)

This line of work can be viewed as the arithmetic analogue of "hardness vs randomness" results from the Boolean setting. Moreover, many of the underlying techniques for proving the arithmetic results above are inspired by the techniques for proving the analogous Boolean results. However, and in contrast to what one may expect, the hardness-vs-randomness connections in arithmetic complexity so far have been less tight than the Boolean results. (For example, even the equivalences in [JS12; CIKK15] concern sub-exponential time algorithms, need non-uniform advice bits, and/or refer to average-case PIT and circuit lower bounds.) This counters our expectation that progress on arithmetic complexity should be easier.

[1] Indeed, this is the search version of PIT, whereas the decision version calls for deciding whether or not $C$ is indeed nonzero. The distinction between the two will not be crucial in our work (see Remark 1.5).

**Our contributions, in a gist.** In this work, we close the gaps above almost entirely. That is, we prove that certain natural lower bounds are both sufficient and necessary for solving PIT in polynomial time and in the worst case, up to relatively minor remaining gaps. Compared to the results above, we refer to polynomial-time PIT, rather than sub-exponential time; we refer to the standard notion of solving PIT in the worst-case; and the hard problem is computable without non-uniform advice. The lower bounds that we study are for models of computation that are relatively standard and well-studied.

The key to these improvements is studying PIT vs lower bounds in the uniform setting, and in particular focusing on lower bounds for uniform arithmetic circuits (i.e., for circuit families that can be printed by a Turing machine). We stress that our algorithms still solve PIT on any given circuit (i.e., in the standard worst-case sense), and it is only the lower bounds that are different from the past, and need to hold only against uniform circuits. It is easy to show that the relevant lower bounds are necessary for PIT, and the more surprising direction of our results is constructing PIT algorithms from these lower bounds.

This means that proving lower bounds for uniform families is a potential path towards obtaining PIT algorithms. In addition, this is an approach for tackling Problem 1.1 (i.e., connecting PIT to lower bounds for uniform circuits) that yields very tight connections. The connections we show are significantly tighter than previously known connections for the arithmetic setting, and moreover, they are also tighter than the connections known in the Boolean setting (thus closing another long-standing gap, i.e. between hardness-vs-randomness in the arithmetic setting and in the Boolean one).

Technically, analogous to the fact that the results of [KI04; JS12; CIKK15] were inspired by developments in Boolean hardness-vs-randomness machinery in the late 1990's (i.e., by [NW94; IKW02]), our results build on very recent developments in Boolean hardness-vs-randomness (see, e.g., [CT23] for a survey), while also leveraging the state-of-the-art hitting-set generators for arithmetic circuits (i.e., leveraging ideas and techniques from [AGS18; KST19; GKSS22]). We elaborate on our techniques in Section 2.

## 1.2 Lower bounds for uniform arithmetic circuits

Part of our contribution is to carefully delineate the notions that we believe (and demonstrate) are useful for studying PIT vs uniform lower bounds. To set the stage for presenting our results, let us spell out some of these notions in advance, while providing pointers to precise definitions.

**Uniform randomized circuits.** A uniform circuit family is a sequence of circuits $\{C_n \in \mathbb{F}[x_1, ..., x_n]\}_{n\in\mathbb{N}}$ such that there is an efficient Turing machine that gets input $1^n$ and prints $C_n$. The notion of "efficient" varies across definitions, as well as the question of printing field elements (see Section 3.3). The study of uniform circuits has a long history in the Boolean setting (see, e.g., the survey by Allender [All89], and recent results such as [CK12; SW13; San23; DPT24]). In arithmetic complexity, suitable definitions were put forward by von zur Gathen [vzGat86a] and by Eberly [Ebe89] in the 1980's, and more recently, lower bounds for uniform arithmetic circuits of bounded depth were proved by Koiran and Perifel [KP09], following Allender [All99] (see also strengthenings in [JS13; CKK14]).

The lower bounds that turn out to be essentially equivalent to PIT are for uniform randomized circuits (in fact, for randomized networks – see below).[2] It is not a-priori obvious how to define randomized arithmetic circuits in general. We bypass this question by focusing on a nice subclass of randomized circuits: namely, deterministic circuits equipped with "PIT gates" (i.e., gates that receive a description of an arithmetic circuit, and solve PIT for that circuit). This subclass represents a limited use of randomness by the circuit – i.e., only for solving PIT – and we prove that lower bounds against this subclass suffice for constructing PIT algorithms.

**Arithmetic networks.** Our results focus on a stronger variant of arithmetic circuits, called arithmetic networks; these were first formally defined by von zur Gathen [vzGat86a]. Networks get arithmetic inputs and produce arithmetic outputs. Their computation consists of standard arithmetic operations $\{+, \times\}$, and they can also "route" the results of the arithmetic computation, using a limited set of Boolean gates. Specifically, the Boolean gates can test whether an arithmetic element (i.e., the value of a gate, computed on the input) is zero, and depending on the outcome of the computation, they choose which arithmetic sub-circuit the

[2] This is analogous to the results of [CT21], who showed bidirectional connections between derandomization (i.e., $pr\mathsf{BPP} = pr\mathsf{P}$) and lower bounds for randomized algorithms. More generally, this follows the blueprint suggested by Kozen [Koz80], wherein simulation of a class turns out to be equivalent to lower bounds for the class (see also [NIR03; NIR06]).

arithmetic computation will proceed on. We stress that the Boolean gates have no access to the content of arithmetic values, except for zero-testing; intuitively, all they can do is "route" arithmetic values around the circuit according to zero-tests. See Definition 3.6 for details.

While the definition may seem unusual at first, arithmetic networks have actually been widely studied (implicitly and explicitly) in arithmetic complexity. For example, many classical "arithmetic algorithms" actually work in this model [BvH82; vzGat84; vzGat86b], including the Euclidean algorithm; and even very recently, the constant-depth GCD algorithm of Andrews and Wigderson [AW24] is, in fact, precisely an arithmetic network.[3] Lower bounds for arithmetic networks have also been studied: the works of [MP93; MMP96; GV17] prove lower bounds for arithmetic networks that decide membership in a (semi-)algebraic set using techniques similar to those used to prove lower bounds for algebraic decision trees [Yao97].

**Lower bounds on all but finitely many inputs.** Lastly, lower bounds for uniform circuits and networks can differ greatly from lower bounds for their non-uniform counterparts. As one important example, for every $\mathbb{F}$, there is (unconditionally!) a $\mathsf{P}$-uniform family of arithmetic circuits $\{C_n\}_{n\in\mathbb{N}}$ of size $n^{O(k)}$ over $\mathbb{F}$ such that for every $n^k$-time-uniform circuit family $\{C'_n\}_{n\in\mathbb{N}}$ of size $n^k$ over $\mathbb{F}$, and for all but at most finitely many $n \in \mathbb{N}$, it holds that $C_n(\vec{x}) \neq C'_n(\vec{x})$ for every $\vec{x} \in \mathbb{F}^n$ (see Fact 3.12). That is, $\{C_n\}_{n\in\mathbb{N}}$ is hard for $n^k$-time-uniform circuits of size $n^k$ on each and every possible input (except, at most, finitely many).

Lower bounds on all but finitely many inputs turn out to be closely related to $\mathsf{PIT}$ algorithms, as we shall show. We stress that the feasibility of this almost-all-inputs hardness relies crucially on the fact that we consider hardness with respect to uniform families of arithmetic circuits and networks. (Indeed, a non-uniform circuit can always have the value of the function at (say) $0^n$ hard-wired.)

## 1.3 Result statements

At a high level, we show that $\mathsf{PIT}$ algorithms are essentially equivalent to lower bounds for uniform randomized arithmetic networks on all but finitely many inputs.

Our first result focuses on fields of characteristic zero. We show that if for some $k$ there is a family of (deterministic) arithmetic circuits of size $n^{k^2}$ that is hard on all inputs for uniform arithmetic networks with $\mathsf{PIT}$ gates of size $n^k$, then $\mathsf{PIT}$ can be solved in polynomial time over $\mathbb{F}$.

**Theorem 1.2** (hardness on all inputs implies $\mathsf{PIT}$; informal, see Theorem 7.3)**.** *Let $\mathbb{F}$ be a field of characteristic zero. Assume that for some sufficiently large $k > 1$ there is a strongly uniform[4] family of arithmetic circuits of size and degree $n^{k^2}$ over $\mathbb{F}$ that is hard on all but finitely many inputs for uniform arithmetic networks with $\mathsf{PIT}$ gates of size $n^k$ over $\mathbb{F}$. Then, there is a uniform family of arithmetic networks of polynomial size solving $\mathsf{PIT}$ over $\mathbb{F}$.*

Our second result is a tight converse for Theorem 1.2. To contextualize this, recall that previously known results reflect a trade-off: When considering suboptimal $\mathsf{PIT}$ algorithms (e.g., running in super-polynomial time, or working only on average), known results rely on hardness assumptions that are necessary or close to being so (see, e.g., [KI04; CIKK15]); whereas, when considering polynomial-time $\mathsf{PIT}$ algorithms, known results rely on hardness assumptions that are natural, but not known to be close to necessary for $\mathsf{PIT}$ (say, as in [GKSS22]). This trade-off between the running time of the $\mathsf{PIT}$ algorithm and the tightness of the hardness assumption may seem reasonable, but our second result shows that it is, essentially, entirely avoidable: The assumptions in Theorem 1.2 are remarkably close to being necessary for the conclusion.

**Theorem 1.3** ($\mathsf{PIT}$ implies hardness on all inputs; informal, see Theorem 7.2)**.** *Let $\mathbb{F}$ be a field of characteristic zero. Assume that there is a uniform family of arithmetic networks of polynomial size solving $\mathsf{PIT}$ over $\mathbb{F}$. Then, for every $k \in \mathbb{N}$, there is a family of uniform arithmetic networks over $\mathbb{F}$ of polynomial size that is hard on all but finitely many inputs for $n^k$-time-uniform arithmetic networks with $\mathsf{PIT}$ gates of size and degree $n^k$ over $\mathbb{F}$.*

[3] There are, in general, good reasons for this. First, networks do not provide significantly more power for computing polynomials (see Section 3.2). Secondly, standard arithmetic circuits can only compute continuous functions of their inputs, whereas many interesting algebraic problems are discontinuous (e.g., computing GCD of polynomials, or solving $\mathsf{PIT}$).

[4] The meaning of "strongly uniform" refers to a stricter notion of uniformity, in which given the indices of gates in $C_n$, we can decide in polynomial time in the input length whether these gates are connected (indeed, the running time is polylogarithmic in the size of $C_n$); see Definition 3.11 for a precise definition. This is analogous to well-studied notions in Boolean complexity.

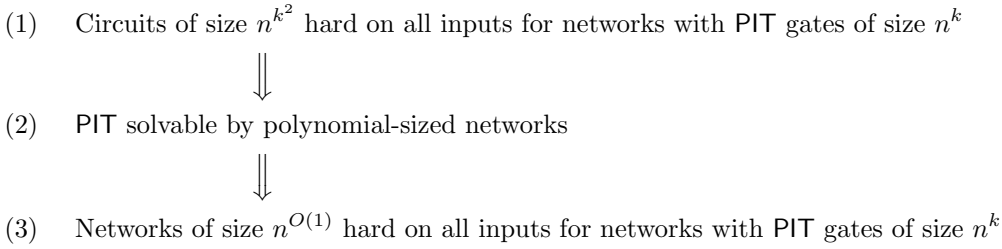


Figure 1: A visual depiction of Theorems 1.2 and 1.3. For simplicity of presentation, we omitted the repeated qualifier "on all *but finitely many* inputs", as well as the precise notions of uniformity and the degree bounds (the latter two cause further gaps between the necessary assumption in (1) and the sufficient one in (3)).

For a visual depiction of the necessary and sufficient assumptions that we prove, focusing for simplicity on the case of characteristic zero, see Figure (1). We stress that neither of the results above refers to circuit lower bounds. Indeed, our algorithms are not hitting-set generators, but rather targeted hitting-set generators: The PIT algorithm makes essential use of the description of the arithmetic circuit given to it.

**A variation for finite fields.** We also show very similar results over sufficiently large finite fields. In this setting, the lower bounds that we focus on are for a more general model of randomized arithmetic networks, wherein the network receives random elements as inputs, and needs to err with low probability whenever these elements are sampled from any large enough set (see Section 3.4.2 for details).

Beyond that, there are two minor differences in the result. On the one hand, we allow the hard family to be computed in arbitrary polynomial size, rather than size $n^{k^2}$ (e.g., it can be computed in size $n^{2^{2^k}}$), so the hardness assumption is slightly more relaxed. On the other hand, the conclusion is slightly weaker: Instead of getting a strictly polynomial-time PIT algorithm, we get an algorithm running in time $n^{\log^{(c)}(n)}$, where $\log^{(c)}$ is the $c$-iterated log function and $c > 1$ can be an arbitrarily large constant.

**Theorem 1.4** (hardness on all inputs implies PIT; informal, see Theorem 7.4)**.** *Let $\mathbb{F} = \{\mathbb{F}_n\}_{n\in\mathbb{N}}$ be a sequence of finite fields, where $n^{\omega(1)} \leq |\mathbb{F}_n| \leq 2^{\mathrm{poly}(n)}$. Assume that for a sufficiently large $k > 1$ there is a strongly uniform family of arithmetic circuits of polynomial size and degree over $\mathbb{F}$ that is hard on all but finitely many inputs for uniform randomized arithmetic networks of size $n^k$ over $\mathbb{F}$. Then, for every $c > 1$ there is a uniform family of arithmetic networks of size $n^{\log^{(c)}(n)}$ solving* PIT *over $\mathbb{F}$.*

Again, Theorem 1.4 is coupled with a converse direction, analogous to Theorem 1.3, but the converse direction only yields lower bounds for networks with PIT gates (rather than for the more general model of randomized networks); see Theorem 7.2 for details. We also show an alternative proof for a special case of Theorem 1.4, wherein the field's characteristic is low, in Appendix B.

**On lower bounds for networks, and the role of degree.** The statements of Theorems 1.2 and 1.4 do not mention any bound on the degree of networks for which we assume lower bounds. We believe that even in this form, both results are interesting. For context, the original results of Kabanets and Impagliazzo [KI04] do not spell out any degree bound in the hardness assumption (and over finite fields, the needed degree in their assumption is indeed high; see [And20] for an improvement). Moreover, the meaning of the notion of "degree" for networks is less straightforward than it is for circuits (see Section 3.2).

Nevertheless, our technical results only rely on lower bounds for networks of relatively low degree. In Theorem 1.2, the degree is quasipolynomial in the input length $n$ (see Theorem 7.3). In Theorem 1.4, the degree is quasipolynomial in $n$ and in the field size $q$; lower bounds for arithmetic circuits or networks of degree more than $q$ are less well studied, and we observe that over fields of small characteristic (say, at most $n$), such lower bounds imply lower bounds for low-depth Boolean circuits (see Appendix B). As far as we are aware, such implication is not known for the general case studied in Theorem 1.4.

**Remark 1.5** (on the formulation of PIT algorithms)**.** *As the reader might have noticed, all of our results refer to solving* PIT *by uniform arithmetic networks; that is, intuitively, we consider arithmetic algorithms for* PIT *(rather than, say, arbitrary Boolean algorithms for* PIT*). Also, recall that we presented the formulation of* PIT *as a search problem (i.e., finding a non-root). This choice is immaterial for our results: Our* PIT *algorithms in Theorems 1.2 and 1.4 solve the search version, whereas the lower bound in Theorem 1.3 follows from an algorithm solving the decision version.*

**Organization of the rest of the paper:** In Section 2, we present a high-level overview of the proofs. Section 3 contains several standard facts in algebraic complexity, as well as the formal definitions and properties of uniform, randomized, and universal arithmetic networks and circuits, which we make use of repeatedly. Section 4 presents an auxiliary technical notion that will be common to our proofs, namely, polynomial decompositions (à la [CT21]). Section 5 and Section 6 then discuss the construction and correctness of our GKSS-based and KI-based targeted hitting set generators, respectively. The proofs of the main results appear in Section 7. We conclude by mentioning several open problems in Section 8.

## 2 Technical overview

Recall that we are given as input the description of an arithmetic circuit $C(x_1, \ldots, x_n)$ of size $\mathrm{poly}(n)$ computing a nonzero polynomial of degree $d \leq \mathrm{poly}(n)$, and we want to find a point $\vec{\alpha} \in \mathbb{F}^n$ such that $C(\vec{\alpha}) \neq 0$. The trivial deterministic algorithm for PIT evaluates the circuit $C$ on an $n$-dimensional grid of side length $d+1$; the Schwartz–Zippel lemma guarantees that $C$ will evaluate to a nonzero value at some point on this grid. This leads to an algorithm for PIT that runs in time $(d+1)^n \cdot \mathrm{poly}(n)$.

A standard approach to obtain a faster algorithm for PIT is to reduce the given $n$-variate instance $C(x_1, \ldots, x_n)$ to an $\ell$-variate instance $C'(y_1, \ldots, y_\ell)$ of degree $d' \leq \mathrm{poly}(n)$. If this reduction is efficient, we improve on the previous running time of $n^{O(n)}$ to $n^{O(\ell)}$. Such a reduction follows from an explicit construction of a hitting-set generator, which is a polynomial map $\mathcal{G} : \mathbb{F}^\ell \to \mathbb{F}^n$ that satisfies $(C \circ \mathcal{G})(y_1, \ldots, y_\ell) \neq 0$ whenever $C$ is sufficiently small circuit. Many prior works on hardness-versus-randomness in algebraic complexity give explicit constructions of hitting set generators to derandomize PIT (see [KS19]).

As mentioned in Section 1.1, our PIT algorithms are inspired by recent developments in Boolean hardness-vs-randomness (see [CT23]). Following Goldreich [Gol11] and Chen and Tell [CT21], our PIT algorithms will use a targeted hitting-set generator, which (in the current arithmetic setting) is a procedure $\mathcal{G}$ mapping a description of $C$ to a polynomial map $\mathcal{G}_C(y_1, ..., y_\ell)$ such that $C'(y_1, ..., y_\ell) = C \circ \mathcal{G}_C$ is nonzero. Note that in contrast to standard hitting-set generators, which are polynomial maps $\mathcal{G}$ that work for all small circuits, here $\mathcal{G}_C$ depends on $C$. Targeted hitting-set generators are weaker objects than standard hitting-set generators, and the advantage in working with them is that we will only need lower bounds against uniform models (whereas standard hitting-set generators necessitate lower bounds for non-uniform circuits).

**The motivating idea.** The starting point of our work is the observation that the arithmetic setting shows promise for a way to bypass what is one of the main current obstacles in the Boolean setting. Specifically, recall that [CT21] proved the following result: If there is $f\colon \{0,1\}^* \to \{0,1\}^*$ computable by logspace-uniform circuits of size $\mathrm{poly}(n)$ and depth $n^2$, but hard on all but finitely many inputs for probabilistic time $n^c$ (for a sufficiently large $c \in \mathbb{N}$), then $pr\mathsf{BPP} = pr\mathsf{P}$. Note the restriction of $n^2$ on the depth of circuits for $f$: This particular restriction is the main gap between the foregoing assumption and a necessary assumption for $pr\mathsf{BPP} = pr\mathsf{P}$, and it is conjectured to be redundant (see, e.g., [CTW23]).

The key point is that in the arithmetic setting we should not have this gap, due to the depth reduction procedure of Valiant, Skyum, Berkowitz, and Rackoff [VSBR83], a foundational result in arithmetic circuit complexity. They showed that any degree-$d$ polynomial computed by a size-$s$ circuit can also be computed by a circuit of size $\mathrm{poly}(s, d)$ and depth $O(\log(s)\log(d))$. In other words, when we restrict attention to low-degree polynomials—as is commonly done in algebraic complexity—small circuits can be assumed to be low-depth without loss of generality.[5] This suggests that by working in the arithmetic setting, we can close the gap

[5] Of course, since we work with uniform circuits, one needs to verify that this depth reduction can be performed in a sufficiently uniform manner. This is indeed the case, as the standard proof of the depth reduction can be carried out in a uniform manner. The same concern regarding uniformity applies any time we invoke a result from arithmetic circuit complexity. For all results we

between necessary and sufficient conditions for derandomization of PIT.

This intuition turns out to be, essentially, correct, and we are able to remove the depth upper bound in the sufficient condition. However, unfortunately, most of the other parts of the Boolean constructions do not migrate cleanly to the arithmetic setting, in which we want a targeted generator implementable as an arithmetic procedure, and we want to base its correctness on hardness for arithmetic networks.

Thus, to prove our results we introduce new constructions of arithmetic targeted hitting-set generators, which rely on specific arithmetic tools (e.g., the hitting-set generators of Kabanets and Impagliazzo [KI04] and Guo, Kumar, Saptharishi, and Solomon [GKSS22] and bootstrapping ideas of [AGS18; KST19]) as well as on refined analyses of these tools.

**Organization.** We first start with the easier case of fields of characteristic zero, described in Section 2.1. This part should help introduce readers who are less familiar with the recent constructions in Boolean hardness vs randomness to the high-level outline of the arguments. Then we move to the more challenging case of finite fields, described in Section 2.2.

## 2.1 Warming up: Fields with characteristic zero

For the sake of simplicity, let us now work over the field $\mathbb{Q}$ of rational numbers.

### 2.1.1 Hardness on all but finitely many inputs is necessary

The proof of Theorem 1.3 is a straightforward diagonalization argument. Recall that we assume that PIT can be solved by a P-uniform family of arithmetic networks of polynomial size, and our goal is to find a P-uniform family of uniform arithmetic networks that is hard on all but finitely many inputs for $n^k$-time-uniform arithmetic networks with PIT gates. To do this, on input length $n$ we diagonalize against the first $n$ families of $n^k$-time-uniform arithmetic networks with PIT gates.

On input $\vec{\Lambda} \in \mathbb{Q}^n$, we want to compute an output $\vec{\sigma} \in \mathbb{Q}^n$ such that $\sigma_i$ disagrees with the $i^{\text{th}}$ output of the $i^{\text{th}}$ family of uniform arithmetic networks with PIT gates on $\vec{\Lambda}$, for all $i \in [n]$. To do this, we simulate each of the $n$ machines for time $n^k$, and evaluate each of the $n$ corresponding arithmetic networks with PIT gates on $\vec{\Lambda}$, using the assumption that we can solve PIT deterministically in order to simulate the PIT gates. We then obtain $\sigma$ by taking $\sigma_i$ to be some trivial modification of the $i^{\text{th}}$ output element of the $i^{\text{th}}$ network (e.g., adding 1 to it). This diagonalization procedure yields a uniform family of arithmetic networks whose outputs cannot be computed on any input $\vec{\Lambda}$ (except, at most, finitely many) by smaller uniform arithmetic networks with PIT gates.[6]

### 2.1.2 Hardness on all but finitely many inputs is sufficient

The proof of Theorem 1.2 is more intricate and, together with Theorem 1.4, constitutes the technical bulk of our work. By assumption, we have a strongly uniform family of arithmetic circuits $\{C_n\}_{n\in\mathbb{N}}$ of size and degree $n^{k^2}$ that is hard on all but finitely many inputs for uniform arithmetic networks with PIT gates of size $n^k$, and our goal is to use this hard family of circuits to design a uniform family of deterministic arithmetic networks that solve PIT. Thanks to depth reduction for arithmetic circuits [VSBR83], we may assume that the circuits $\{C_n\}$ are of depth $\Delta := O(\log^2 n)$.

Let $D$ be the $n$-variate arithmetic circuit for which we want to solve PIT. By using the universal arithmetic circuits of Raz [Raz10], we can regard the description of $D$ as a point $\vec{\Lambda} \in \mathbb{Q}^n$. Following [CT21], we will construct a generator $\mathcal{G}_{\vec{\Lambda}} : \mathbb{Q}^{1000} \to \mathbb{Q}^n$ so that if $D \circ \mathcal{G}_{\vec{\Lambda}} = 0$, then we can compute the hard function $C_n$ at the point $\vec{\Lambda} \in \mathbb{Q}^n$ with a uniform arithmetic network with PIT gates of bounded polynomial size. Because $C_n$ is hard for such networks on all but finitely many inputs $\vec{\Lambda}$, our generator succeeds in hitting all but finitely many circuits $D$ (i.e., for all but finitely many $\vec{\Lambda}$, the generator $\mathcal{G}_{\vec{\Lambda}}$ hits the circuit represented by $\vec{\Lambda}$).

use, known proofs suffice to obtain sufficiently uniform implementations.

[6] The diagonalization implementation is an arithmetic network (rather than an arithmetic circuit) since the assumed PIT algorithm is a uniform arithmetic network, and not a uniform arithmetic circuit.

**The generator.** The generator $\mathcal{G}_{\vec{\Lambda}}$ is built from a polynomial decomposition of the computation of an arithmetic circuit $C_n$ on input $\vec{\Lambda}$. Evaluating $C_n$ at $\vec{\Lambda}$ assigns each gate of the circuit a rational number. If we view $C_n$ as a layered circuit, then the $i^{\text{th}}$ layer of $C_n$ corresponds to a vector $\vec{v_i} \in \mathbb{Q}^{n^{k^2}}$, which we think of as specifying a "hard" polynomial $f_i$. Indeed, we can simply interpolate $f_i$ from these points, or treat them as the coefficients of $f_i$. However, we would like the resulting sequence of polynomials $\{f_1, ..., f_\Delta\}$ to have the additional property that they are downward self-reducible: We can evaluate $f_i$ quickly (i.e., in time much smaller than $n^{k^2}$) if we have access to $f_{i-1}$. Using an idea inspired by the proof system of Goldwasser, Kalai, and Rothblum [GKR15] (which, in turn, uses the sumcheck protocol), we can encode each layer $\vec{v_i}$ of $C_n(\vec{\Lambda})$ by $\ell \leq \log(n)$ of polynomials $f_{i,1}, \ldots, f_{i,\ell}$ such that the resulting sequence of polynomials $\{f_{i,j}\}_{i\in[\Delta],j\in[\ell]}$ is, indeed, downward reducible: the value of $f_{i,j}$ at any point can be learned by querying $f_{i,j-1}$ (or $f_{i-1,\ell}$, if $j = 0$) at $O(n^\varepsilon)$ many related points, where $\varepsilon > 0$ is a small enough constant. Moreover, in this sequence $\{f_{i,j}\}$ the polynomial computing the input layer is (unconditionally) efficiently computable. The polynomial decomposition relies on $\{C_n\}$ being strongly uniform, and its description appears in Section 4.2.

The targeted generator gets input $\vec{\Lambda}$, and computes the polynomial decomposition of $C_n(\vec{\Lambda})$. It the instantiates the (standard) hitting-set generator $G_{\mathsf{GKSS}}$ of Guo, Kumar, Saptharishi, and Solomon [GKSS22] with each of the $f_{i,j}$'s as the hard polynomial. The final generator construction is

$$\mathcal{G}(\vec{\Lambda}, y_1, y_2, \vec{z}) = \sum_{i,j} L_{i,j}(y_1, y_2) \cdot G_{\mathsf{GKSS}}^{f_{i,j}}(\vec{z}) \quad , \tag{2.1}$$

where the $L_{i,j}$'s are standard Lagrange interpolation polynomials computing the indicator function of $(i, j)$ in $[\Delta] \times [\ell]$. Note that the notation $G_{\mathsf{GKSS}}^{f_{i,j}}(\vec{z})$ suppresses $\vec{\Lambda}$, but $f_{i,j}$ depends on the gate-values of $C_n(\vec{\Lambda})$.

We instantiate the polynomial decomposition so that the hard polynomials $f_{i,j}$ are $m$-variate polynomials $\mathbb{Q}^m \to \mathbb{Q}$ where $m = O(1)$, in which case the seed length of $G_{\mathsf{GKSS}}$ is constant; that is, we have $G_{\mathsf{GKSS}} \colon \mathbb{Q}^{O(1)} \to \mathbb{Q}$. The PIT algorithm evaluates $D$ (i.e., the $n$-variate arithmetic circuit represented by $\vec{\Lambda}$) on a grid of constant dimension and polynomial side length.

**Correctness: An arithmetic reconstruction procedure.** To prove the correctness of $G_{\vec{\Lambda}}$, we need to show that if a small circuit $D$ with description $\vec{\Lambda}$ satisfies $D \circ G_{\vec{\Lambda}} = 0$, then we can compute the mapping $\vec{\Lambda} \mapsto C_n(\vec{\Lambda})$ by a small uniform arithmetic network with PIT gates.

Since $G_{\vec{\Lambda}}$ is the combination of the GKSS generator instantiated with the polynomials $f_{i,j}$, we have that $D \circ G_{\mathsf{GKSS}}^{f_{i,j}} = 0$ for each $f_{i,j}$. The GKSS generator is a reconstructive generator, meaning that using the distinguisher $D$ and a bounded number of queries to $f_{i,j}$, one can reconstruct a circuit for the polynomial $f_{i,j}$ (see more on this below). As shown in [GKSS22], the complexity of this reconstructed circuit largely depends on the complexity of the distinguisher $D$. In particular, if $D$ is a small arithmetic circuit, then we obtain a correspondingly-small arithmetic circuit for $f_{i,j}$. This suggests an obvious roadmap: as we have a small circuit $D$ that satisfies $D \circ G_{\mathsf{GKSS}}^{f_{i,j}} = 0$ for each $f_{i,j}$, apply the GKSS reconstruction algorithm to iteratively reconstruct circuits for each of the polynomials $f_{i,j}$, for $(i, j) = (1, 1), ..., (1, \ell), (2, 1), ..., (\Delta, \ell)$; each time we need to answer queries of the GKSS reconstruction to $f_{i,j}$, use the downward self-reducibility as well as our circuit for $f_{i,j-1}$ (or for $f_{i-1,\ell}$). If the circuit $D$ has size $n'$ and each round of reconstruction can be implemented in time $(n')^c$ for some fixed constant $c$, then by iteratively reconstructing circuits for the polynomials $f_{i,j}$, we obtain in time $(n')^c \cdot \Delta$ a circuit for $f_{\Delta,O(1)}$, whose evaluations (at certain points determined by the polynomial decomposition) corresponds to the output of the circuit $C_n$ on input $\vec{\Lambda}$.

We stress that downward self-reducibility is not used as part of the circuit for $f_{i,j}$, since that would cause an exponential blow-up in the circuit size as we iteratively construct circuits for increasing $(i, j)$'s. Instead, downward self-reducibility is only used to answer queries of the GKSS reconstruction to $f_{i,j}$, and the GKSS reconstruction outputs a circuit of fixed polynomial size (depending on $D$) for $f_{i,j}$.

Technical details for the reconstruction procedures. Now, to be more precise, we need a uniform arithmetic network with PIT gates that iteratively performs the reconstruction argument of GKSS, producing a description of a circuit for $f_{i,j}$ at each step. In particular, we need to implement the reconstruction procedure for the GKSS generator by a uniform arithmetic network with PIT gates (that prints an arithmetic circuit).

Let us point out key parts of this implementation, deferring the full details to Section 5.2.1. In the original proof of [GKSS22], the reconstruction algorithm for the generator $G^f_{\mathsf{GKSS}}$ first constructs a circuit for the partial derivatives of order up to $n$ of the homogeneous components of $f$ up to degree $n$. Because $f$ is a polynomial on a constant number of variables, there are $n^{O(1)}$ such polynomials to compute, each of which is a sum of $n^{O(1)}$ monomials. After this base case, the reconstruction algorithm of [GKSS22] uses the distinguisher circuit $D$ to compute all partial derivatives of order up to $n$ of the homogeneous components of $f$, proceeding iteratively one degree at a time.

To obtain the (partial derivatives of) the degree-$d$ homogeneous component of $f$, the reconstruction algorithm of [GKSS22] first obtains evaluations of the order-$n$ partial derivatives of this component. The algorithm then interpolates the order-$n$ derivatives from these evaluations, and then repeatedly applies Euler's formula (Fact 5.5) to reconstruct the degree-$d$ homogeneous component of $f$. Crucially, we cannot evaluate the order-$n$ partials of the degree-$d$ component of $f$ on any points of our choosing, but only on points where a certain polynomial does not vanish. Very roughly, these evaluations are obtained by solving a univariate polynomial equation to first order. If $g(x)$ is a univariate polynomial, then we can Taylor expand $g$ at $a$ as

$$g(x) = g(a) + (x-a)g'(a) + O((x-a)^2).$$

If we want to find a point where $g(x) = 0$, then we can obtain an approximate root by solving

$$0 = g(x) = g(a) + (x-a)g'(a) + O((x-a)^2).$$

To first order, we have

$$x = a - \frac{g(a)}{g'(a)}.$$

For the solution $x$ to be well-defined, we need to ensure that $a$ is chosen so that $g'(a) \neq 0$.

We obtain the order-$n$ partials of the degree-$d$ component of $f$ in a similar manner. To ensure that the evaluations we obtain can be used to reconstruct the partial derivatives themselves, we need to find a set of points where (i) a first-order partial derivative of some related polynomial does not vanish, and (ii) these points are an interpolating set for degree-$d$ polynomials in $n$ variables. We can write down an explicit arithmetic circuit that computes a nonzero polynomial that vanishes on sets of points that do not satisfy both of these conditions. By using $\mathsf{PIT}$ gates within the reconstruction, we can explicitly construct an interpolating set that allows us to recover the partial derivatives of higher-degree homogeneous components of $f$. With a good interpolating set in hand, this iterative part of the reconstruction can be implemented by a uniform network in a fairly straightforward manner.

Perhaps surprisingly, the starting point of each execution of the GKSS reconstruction is more difficult to implement than the iterative part sketched above. In the setting of [GKSS22], they simply hard-wire the partial derivatives up to order $n$ of the relevant homogeneous components to the reconstructed circuit (since they do not care about uniformity). In our setting, we could compute these by polynomial interpolation (i.e., we could query $f_{i,j}$ to learn the coefficients of its low-degree homogeneous components, using downward self-reducibility), but interpolating the entire polynomial $f_{i,j}$ would be too expensive. The key idea to avoid this is to construct a circuit $C^{\mathsf{temp}}_{i,j}$ for $f_{i,j}$, which uses downward-self-reducibility and the circuit for $f_{i,j-1}$, and then homogenize $C^{\mathsf{temp}}_{i,j}$ up to degree $n$. Indeed, the circuit $C^{\mathsf{temp}}_{i,j}$ is too big for us to simply use it as the circuit for $f_{i,j}$ in the next iteration (recall that, as explained above, doing so in each iteration would cause an exponential blow-up), but we will only be using $C^{\mathsf{temp}}_{i,j}$ temporarily to extract information—the coefficients of its low-degree homogeneous components—when implementing the GKSS reconstruction, which outputs a circuit of fixed polynomial size for $f_{i,j}$.

Finally, the number of partial derivatives is bounded, but it is unfortunately not independent of the number of points used to define $f_{i,j}$; in other words, their number does depend on the size of the circuit $C_n$. This is why in our GKSS-based reconstruction, we reconstruct circuits of size $n^K$ by arithmetic networks with $\mathsf{PIT}$ gates of size $n^{\sqrt{K}}$ (indeed, the result was stated in Section 1 using $K = k^2$). For further details, see Section 5.

## 2.2 PIT over finite fields

Turning to the case of finite fields of super-polynomial size, we now use the $\mathsf{KI}$ generator [KI04] instead of the GKSS generator. That is, for each polynomial $f_{i,j}$ encoding a row in the polynomial decomposition,

we instantiate the KI generator with $f_{i,j}$. The final generator will combine the $f_i$'s similarly to Eq. 2.1, and we will implement a reconstruction procedure by repeatedly invoking a uniform implementation of the reconstruction procedure of KI (i.e., analogously to Section 2.1, but working with a uniform reconstruction of KI instead of GKSS, and making the necessary adjustments).

However, there is a problem with this idea. For now, let us focus even just on KI with a single polynomial $f = f_{i,j}$. In fact, let us even ignore the fact that we added $\Delta \cdot \ell$ auxiliary polynomials, and just pretend that each polynomial $f$ corresponds to a row of gate-values in $C_n(\vec{\Lambda})$.

An inherent limitation of KI is that it only yields PIT algorithms running in quasipolynomial time, whereas we are trying to get in PIT algorithms running in polynomial time (or close to it). One reason why this limitation arises is because the KI generator needs to output $n$ elements (for the $n$-input circuit on which we want to solve PIT). Since it uses combinatorial designs à la [NW94], it must rely on a hard polynomial over $\ell = O(\log n)$ variables, in which case the number of input elements to the generator KI is at least $O(\ell)$. Evaluating the given circuit of degree $d = \text{poly}(n)$ on a grid of side length $d+1$ and dimension $O(\ell)$, we get a quasipolynomial-time PIT. (In fact, there is another reason that using KI only yields a quasipolynomial-time PIT algorithm, which we explain below; for now, let us focus on the current challenge.)

Bootstrapping techniques from [AGS18; KST19] suggest a way out of this conundrum, but at a cost of a stronger hardness assumption. Specifically, recall that we started with a circuit $D_0$ that has $n_0 = n$ inputs, and used an $(\ell_0 = \ell)$-variate polynomial $f^{(0)} = f$ to reduce the effective number of inputs of $D_0$ to be $O(\ell_0)$; that is, we constructed a nonzero $O(\ell_0)$-variate circuit $D_1 = D_0 \circ \mathsf{KI}^{f^{(0)}}$ of polynomial size and degree. Of course, if we would have more hard polynomials with different parameters, we would be able to repeat this procedure: At each iteration $i \geq 1$ we use a hard polynomial over $\ell_i = O(\log(\ell_{i-1}))$ variables, and obtain $D_i = D_{i-1} \circ \mathsf{KI}^{f^{(i)}}$ over $O(\ell_i)$ variables, terminating after the number of variables is sufficiently small. The problem is that this bootstrapping procedure requires a sequence of hard polynomials, each over fewer and fewer variables; we do not wish to rely on this seemingly stronger assumption.

**A free lunch setting for bootstrapping.** Our key idea for avoiding this involves a change of perspective. Instead of thinking of $f$ as a hard polynomial, let us just think of the gate-values of $C_n(\vec{\Lambda})$ at layer $i$ as a hard sequence of field elements. That is, this sequence should be hard to compute quickly, otherwise (i.e., if all rows in the polynomial decomposition are "easy") a reconstruction procedure as in Section 2.1 computes $C_n(\vec{\Lambda})$ too quickly. The advantage in this perspective is that now we are allowed to define a hard polynomial from this sequence using an arithmetization of our choosing; that is, the string specifies a sequence of values, and we are allowed to interpolate an $\ell$-variate polynomial from this sequence for a value of $\ell$ of our choosing.

We stress that arithmetizing each layer differently does not affect the hardness assumption. The assumption is still that $C_n(\vec{\Lambda})$ is hard to compute quickly. It just so happens that when we arithmetize a layer with a value of $\ell$ that we choose, it specifies an $\ell$-variate polynomial. The ability to choose $\ell$ is thus, effectively, free of cost in terms of our hardness assumption (in contrast to the setting of [AGS18; KST19]). (One subtlety is that the way we arithmetize must conform to the way the polynomial decomposition is defined. Let us ignore this complication for simplicity of presentation; it does not affect the argument.)

This suggests a natural approach. We compute the hard circuit $C_n(\vec{\Lambda})$ and use the KI generator to obtain a description $\vec{\Lambda}_1$ of $D_1$, which is defined as above, with a standard arithmetization of the layers (i.e., as $(O(\ell_0 = \log n))$-variate polynomials). Now, at each iteration $i$, we arithmetize each layer as an $\ell_i$-variate polynomial to obtain $D_i$, which is the composition of $D_{i-1}$ with the resulting targeted hitting-set generator (obtained by using KI with the $\ell_i$-variate polynomials given by the layers). Thus, the PIT algorithm involves a recursive sequence of computations of $C_n$, each time on a new circuit $\vec{\Lambda}_i$ representing $D_i$, and after each computation $i$ of $C_n$, we arithmetize the layers differently to obtain the targeted hitting-set generator. Since $C_n$ is hard to compute on all inputs, each of the applications of the generator (i.e., on each $\vec{\Lambda}_i$) will result in a nonzero polynomial. After $c = O(1)$ iterations we obtain a polynomial with $\log^{(c)}(n)$ variables, which we can evaluate over a grid of the appropriate side length.

(We intentionally leave the precise details of how polynomials are arithmetized vague, since they depend on the implementation details of the polynomial decomposition. For full details, see Sections 4 and 6.)

**A subtle challenge when recursing: The targeted generator is not an arithmetic circuit.** The reader might have noticed one subtle point in the description above: At each iteration $i$, we need

$D_i = D_{i-1} \circ \mathcal{G}_{\mathsf{KI}}$ to remain an arithmetic circuit of polynomial size and degree, where $\mathcal{G}_{\mathsf{KI}}$ is our KI-based targeted generator. This is because we will be feeding $D_i$ to $\mathcal{G}_{\mathsf{KI}}$ again as the starting point of iteration $i+1$, and we want the next iteration maintain the property that $D_{i+1}$ is nonzero.[7] This might seem like a triviality, given that we expect our generator to be computable arithmetically. However, unfortunately, our targeted generator is not an arithmetic circuit; it is inherently an arithmetic network.

Surprisingly, this issue is manageable. To see how, recall that in iteration $i$ we are composing $D_{i-1}$ with $\mathcal{G}_{\mathsf{KI}}$, when the latter targeted generator is applied to input $\vec{\Lambda}_{i-1}$ that is the description of $D_{i-1}$. The key observation is that we are not actually composing $D_{i-1}$ with a procedure that maps $\vec{\Lambda}_{i-1}$ to $\mathcal{G}_{\mathsf{KI}}$ (this would be the arithmetic network); we are composing $D_{i-1}$ with a procedure $\mathcal{G}_{\mathsf{KI},\vec{\Lambda}_{i-1}}$ that has $\vec{\Lambda}_{i-1}$ fixed and "hard-wired", and computes the outputs of $\mathcal{G}_{\mathsf{KI},\vec{\Lambda}_{i-1}}(\vec{s}) = \mathcal{G}_{\mathsf{KI}}(\vec{\Lambda}_{i-1}, \vec{s})$ as a function of the new input variables $\vec{s}$ (i.e., of the seed of the targeted generator). In particular, we can construct a circuit for $\mathcal{G}_{\mathsf{KI},\vec{\Lambda}_{i-1}}$ that has the hard-to-compute values hard-wired, and only computes the (easy-to-compute) mapping of seed $\vec{s}$ to output $\mathcal{G}_{\mathsf{KI}}(\vec{\Lambda}_{i-1}, \vec{s})$. To see the implementation details, see Section 7.2.2.

Lastly, since we are interested in a PIT algorithm implementable by an arithmetic network, we show how to implement all of the above as an arithmetic network. This network gets $\vec{\Lambda}$, and iteratively produces descriptions of the circuits $D_i = D_{i-1} \circ \mathcal{G}_{\vec{\Lambda}_{i-1}}$, each time using $C_n$ to compute the values of $\mathcal{G}_{\mathsf{KI}}$ that it needs. The construction for a single recursive iteration (which is the crux of the proof) appears in Section 6.1, and the recursive application appears in Section 7.2.2.

**The problematic first iteration.** Finally, we return to the second obstacle involved with using the KI generator, which was mentioned above. The problem is that the reconstruction procedure of KI requires quasipolynomial circuit size when it works with an $O(\log n)$-variate polynomial (as it does in the first iteration). This is a problem, since if the reconstruction is of quasipolynomial size, our hard circuit must also be of (larger) quasipolynomial size, in which case we only obtain a quasipolynomial PIT algorithm.

To see where the inefficiency comes from, recall that the reconstruction procedure involves constructing partial truth-tables of the hard polynomial $f$, by interpolating $f$ on a grid of bounded side length $d_f$ and dimension $\varepsilon \cdot \log(n)$, where $d_f$ is the individual degree of $f$.[8] The issue is that the individual degree of $f$ is not small enough, and in particular, in the first iteration we have $d_f \approx \text{polylog}(n)$ and hence $d_f^{\varepsilon \cdot \log(n)} = n^{O(\log\log n)}$. (In subsequent iterations we can handle this using an appropriate arithmetization, but in the first iteration we must use an $O(\log n)$-variate polynomial.)

To resolve this issue we change the polynomial decomposition, only for the first iteration of the KI-based targeted generator, so that $f$ has constant individual degree (and thus $d_f^{\varepsilon \cdot \log(n)} = n^{O(\varepsilon)}$). In a gist, recall that the polynomials are defined based on ideas from the sumcheck protocol, and we note that they involve computing a certain low-degree formula corresponding to the circuit-structure function of $C_n$. We use an idea of Kalai, Lombardi, and Vaikuntanathan [KLV23], who apply a Cook-Levin-style trick to reduce the individual degree of this formula to be constant, while increasing the round complexity of the relevant sumcheck protocol. In our setting this will result in having a KI generator with $\text{polylog}(n)$ input variables in the first iteration (rather than $O(\log n)$), but this overhead becomes immaterial after subsequent recursions.

# 3 Preliminaries

We use $\mathbb{F}$ to denote a field. Often, we will need to work with a countably-infinite family of fields, such as the family $\{\mathbb{F}_{2^n} : n \in \mathbb{N}\}$ of finite fields of characteristic 2. To simplify notation, we will write such a family of fields as $\{\mathbb{F}_n : n \in \mathbb{N}\}$ with the understanding that $\mathbb{F}_n$ is not necessarily a finite field of order $n$. In some

[7] Specifically, if $D_i$ is not an arithmetic circuit of polynomial size, then our reconstruction procedure will not be an arithmetic network of polynomial size computing the hard problem. Thus, we will not be guaranteed that $\mathcal{G}_{\mathsf{KI}}$ will work in iteration $i+1$ (i.e., that $D_{i+1} = D_i \circ \mathcal{G}_{\mathsf{KI}}$ will be nonzero). This problem is not just syntactic: Recall that if $D_i$ is an arithmetic network, it may not even compute a polynomial, and thus designing a hitting-set generator for it seems to be a problem of a completely different nature.

[8] That is, the reconstruction is an arbitrarily small constant fraction of the size of sets in the combinatorial design (corresponding to the size of pairwise intersections of sets in the design). When working with an $\ell$-variate polynomial, the designs sets are of size $\ell$, and hence the pairwise intersections can be made of size $\varepsilon \cdot \ell$.

instances, we will only need to work over a single field, such as $\mathbb{Q}$; for notational consistency, we still write this as a family of fields $\{\mathbb{F}_n : n \in \mathbb{N}\}$ where $\mathbb{F}_n = \mathbb{Q}$ for all $n \in \mathbb{N}$.

Whenever considering a field sequence $\mathbb{F} = \{\mathbb{F}_n\}$, we implicitly also associate with it a way of representing field elements as bit-strings.[9] We say that $\mathbb{F} = \{\mathbb{F}_n\}$ is feasible if there is a uniform Turing machine that gets as input $(1^n, 1^m)$ where $m \leq |\mathbb{F}_n|$, runs in time $\mathrm{poly}(n+m)$, and outputs $m$ distinct elements in $\mathbb{F}_n$. Most natural field sequences are feasible (when considering the straightforward way of representing field elements as bit-strings). For example, if $\mathbb{F}_n = \mathbb{F}_{p(n)}$ is a finite field of prime order, a uniform machine can simply print the first $m$ integers (i.e., without having to know the prime $p(n)$); and similarly, if $\mathbb{F}_n = \mathbb{F}_{p^r}$, then a uniform machine can print $m$ distinct vectors in $[p]^r$ (i.e., without having to know an irreducible of degree $r$ over $\mathbb{F}_p$). When $\mathbb{F}_n$ is of characteristic zero, the machine can also just print $m$ integers.

We abbreviate a tuple $(x_1, \ldots, x_n)$ as $\vec{x} \coloneqq (x_1, \ldots, x_n)$; in all instances, the length of the tuple will either be explicitly stated or will be clear from context. For an exponent vector $\vec{e} = (e_1, \ldots, e_n) \in \mathbb{Z}^n_{\geq 0}$, we write $|\vec{e}| \coloneqq e_1 + \cdots + e_n$ for the 1-norm of $\vec{e}$ and use $\vec{x}^{\vec{e}}$ to denote the monomial $x_1^{e_1} \cdots x_n^{e_n}$. We write $\mathbb{F}[x_1, \ldots, x_n]$ and $\mathbb{F}(x_1, \ldots, x_n)$ for the polynomial ring and field of rational functions, respectively, over the variables $x_1, \ldots, x_n$ with coefficients in $\mathbb{F}$. Given a polynomial $P(\vec{x}) \in \mathbb{F}[x_1, \ldots, x_n]$, we write $\deg(P)$ and i-deg$(P)$ for the degree and individual degree of $P$, respectively. We use $\partial_{\vec{x}^{\vec{e}}}(P(\vec{x}))$ to abbreviate the partial derivative $\frac{\partial^{\vec{e}}}{\partial \vec{x}^{\vec{e}}}(P(x_1, \ldots, x_n))$. We use $\langle \vec{x} \rangle^i$ to denote the ideal in the polynomial ring $\mathbb{F}[x_1, \ldots, x_n]$ that is generated by all degree-$i$ monomials in the variables $\vec{x}$.

We say that a function $f \colon \mathcal{D} \to \mathcal{R}$ represents a string $x_f \in \mathcal{R}^{\mathcal{D}}$ if $f(i) = (x_f)_i$ for all $i \in \mathcal{D}$.

## 3.1 Standard results

Here, we quote the Schwartz–Zippel Lemma and Gauss's Lemma, two standard results that we make use of throughout our work. We begin with the Schwartz–Zippel Lemma.

**Lemma 3.1** (Schwartz–Zippel Lemma, total degree [Sch80])**.** *Let $\mathbb{F}$ be a field and let $f \in \mathbb{F}[\vec{x}]$ be a nonzero polynomial. Then for any finite set $S \subseteq \mathbb{F}$, we have*

$$\mathop{\mathbb{P}}_{\vec{\alpha} \leftarrow S^n}[f(\vec{\alpha}) = 0] \leq \frac{\deg(f)}{|S|}.$$

**Lemma 3.2** (Schwartz–Zippel Lemma, individual degree [Zip79])**.** *Let $\mathbb{F}$ be a field and let $f \in \mathbb{F}[x_1, \ldots, x_n]$ be a nonzero polynomial. Then for any finite set $S \subseteq \mathbb{F}$, we have*

$$\mathop{\mathbb{P}}_{\vec{\alpha} \leftarrow S^n}[f(\vec{\alpha}) = 0] \leq 1 - \left(1 - \frac{\text{i-deg}(f)}{|S|}\right)^n.$$

*Let $f$ be a nonzero $n$-variate polynomial of individual degree at most $d$ over a field $\mathbb{F}$. Then for any set $S \subseteq \mathbb{F}$ with $|S| > d$, there is a point $\vec{a} \in S^n$ such that $f(\vec{a}) \neq 0$.*

Next, we recall the statement of Gauss's Lemma, which allows us to relate the factorizations of a polynomial $f \in \mathbb{F}[\vec{x}, y]$ over the two rings $\mathbb{F}(\vec{x})[y]$ and $\mathbb{F}[\vec{x}][y]$.

**Lemma 3.3** (Gauss's Lemma [vzGG13, Corollary 6.10])**.** *Let $f \in \mathbb{F}[\vec{x}, y]$ be monic in $y$. Then $f$ is irreducible in $\mathbb{F}(\vec{x})[y]$ if and only if $f$ is irreducible in $\mathbb{F}[\vec{x}][y] \cong \mathbb{F}[\vec{x}, y]$.*

Gauss's Lemma implies the following corollary, which we frequently use.

**Corollary 3.4.** *Let $f \in \mathbb{F}[\vec{x}, y]$ and let $g \in \mathbb{F}[\vec{x}]$. Suppose that $f(\vec{x}, g(\vec{x})) = 0$. Then $y - g(\vec{x})$ is an irreducible factor of $f(\vec{x}, y)$ in $\mathbb{F}[\vec{x}, y]$.*

[9] For fields such as $\mathbb{C}$, we can either represent only a subset of field elements as bit-strings (e.g., the rational ones), or work in a more generalized real-word-RAM-like model. Our results are not sensitive to this choice: see Remarks 3.9 and 3.10.

## 3.2 Arithmetic circuits and networks

In this subsection, we recall the definitions of arithmetic circuits and arithmetic networks. We start with arithmetic circuits.

**Definition 3.5.** *Let $\mathbb{F}$ be a field. An* arithmetic circuit *over $\mathbb{F}$ is a directed acyclic graph whose internal gates are labeled as addition or multiplication gates, whose input gates are labeled with variables $x_i$ or constants from $\mathbb{F}$, and which computes a polynomial in the natural way. We say that an arithmetic circuit is* homogeneous *if every gate of the circuit computes a homogeneous polynomial. Additionally, we assume that gates labeled by field constants appear only in the bottom-most layer of an arithmetic circuit.*[10]

We assume throughout the paper that all arithmetic circuits are alternating and have fan-in two. In addition to arithmetic circuits, we also need the notion of an arithmetic network, first defined by von zur Gathen [vzGat86a]. Arithmetic networks endow arithmetic circuits with the ability to test if a computed value is zero and branch accordingly.

**Definition 3.6** (arithmetic networks)**.** *Let $\mathbb{F}$ be a field. An* arithmetic network *over $\mathbb{F}$ is a directed acyclic graph together with the following additional data.*

- *Internal gates of the network are labeled with either an arithmetic operation from $\{+, -, \times, \div\}$, a Boolean operation from $\{\wedge, \vee, \neg\}$, a test operation $\overset{?}{=} 0$, or a selection operation* select*.*
- *The input gates are labeled by variables $x_i$ that take values in $\mathbb{F}$, variables $y_j$ that take Boolean values in $\{0, 1\}$, and by field constants $\alpha \in \mathbb{F}$.*
- *The gates of the circuit compute as follows.*
  - *Arithmetic gates $\{+, -, \times, \div\}$ receive two arithmetic values as input and output the corresponding function of the inputs. If division by zero occurs, the output of the gate is undefined.*
  - *Boolean gates $\{\wedge, \vee, \neg\}$ receive two Boolean values as input and output the corresponding function of the inputs.*
  - *The test gate $\overset{?}{=} 0$ receives an arithmetic value $\alpha \in \mathbb{F}$ as input and outputs a Boolean value that indicates whether $\alpha$ is zero or nonzero.*
  - *The selection gate* select *receives one Boolean input $b$ and two arithmetic inputs $\alpha, \beta \in \mathbb{F}$ and computes the function*

$$\mathsf{select}(b, \alpha, \beta) := \begin{cases} \alpha & \text{if } b = 0 \\ \beta & \text{if } b = 1. \end{cases}$$

*An arithmetic network computes a function $\mathbb{F}^n \times \{0,1\}^m \to \mathbb{F}^\nu \times \{0,1\}^\mu$. The* size *of a network is the number of gates in the network. The* depth *of a network is the length of the longest path from an input gate to the output gate.*

*When the Boolean inputs to the network and the outputs of the* select *gates are fixed, the arithmetic gates of the network compute a rational function $f/g \in \mathbb{F}(x_1, \ldots, x_n)$ of the arithmetic inputs. The degree of this rational function $f/g$ is given by $\deg(f/g) := \deg(f) + \deg(g)$. The* degree *of an arithmetic network is the maximum degree of any rational function computed by an arithmetic gate, where the maximum is taken over all fixings of the Boolean inputs and the outputs of the* select *gates in the network.*

As is typical in arithmetic circuit complexity, we will restrict our attention to networks of low degree, typically degree $\mathrm{poly}(n)$ where $n$ is the number of arithmetic inputs to the network.

Arithmetic networks clearly generalize arithmetic circuits, as networks can compute strictly more functions than circuits. For polynomial functions, which are computable by both circuits and networks, the following remark shows that networks in fact provide no advantage over circuits, at least in the non-uniform setting.

[10] This simplifying assumption does not meaningfully lose generality, since constants in the bottom layer can be used in upper layers by propagating a constant element $\sigma$ using the operations $\sigma \times 1$ and $\sigma + 0$.

**Remark 3.7.** *Let $\mathbb{F}$ be an algebraically closed field and let $f(\vec{x}) \in \mathbb{F}[\vec{x}]$ be a polynomial. If there is an arithmetic network of size $s$ that computes the function $\mathbb{F}^n \to \mathbb{F}$ given by $\vec{\alpha} \mapsto f(\vec{\alpha})$, then there is an arithmetic circuit of size $s$ that computes $f(\vec{x})$. To see this, take the generic path through the network: assume all comparisons with zero return "not equal to zero," unless the value being compared is identically zero. This fixes all Boolean values appearing in the network; under this fixing, the network behaves as an arithmetic circuit instead. The resulting arithmetic circuit computes a polynomial $g(\vec{x})$ that agrees with $f$ on a nonempty Zariski-open subset $U \subseteq \mathbb{F}^n$. It is a basic fact of algebraic geometry (see, e.g., [Sha13, Chapter 1, Section 3.2]) that if two polynomials agree as functions on a nonempty Zariski-open subset of $\mathbb{F}^n$, then they are the same polynomial. This implies that the circuit obtained from the generic path in the network correctly computes the polynomial $f(\vec{x})$ on all inputs.*

Unfortunately, this method of eliminating the Boolean part of an arithmetic network will not apply to our reconstruction algorithms. In that setting, we compute a polynomial using a uniform randomized arithmetic network (a model of computation whose definition appears in Section 3.4), and it is less clear if a randomized arithmetic network can be efficiently converted into an equivalent arithmetic circuit.

## 3.3 Uniform arithmetic circuits and networks

In this section we define our notions of uniform arithmetic circuits and networks. We first explain some general considerations, then formally define two notions of uniformity, and then show that there are functions that are hard on all but finitely many inputs for uniform circuits.

### 3.3.1 Background

The data describing an arithmetic network can be divided into a Boolean part, which encompasses the structure and labeling (without the field constants) of the directed acyclic graph, and an arithmetic part, corresponding to the particular field constants used in the arithmetic network. To impose a uniformity constraint on arithmetic networks, we must specify a notion of uniformity for both the Boolean part and the arithmetic part of the network.

It is fairly clear how this impacts the Boolean part of an arithmetic network: the underlying dag and its labeling should be computable by a Turing machine within some resource bound. It is less clear how uniformity interacts with the arithmetic part of the network description. There are several notions of uniformity for the arithmetic part of the network considered in the literature.

1. The most restrictive form of uniformity requires the network to be constant-free, i.e., only the constant $1 \in \mathbb{F}$ is used by the network.

2. A more relaxed notion is what von zur Gathen [vzGat86a] calls $\neq$-uniformity: the network can make use of a list of pairwise distinct field elements $(\alpha_1 = 1, \alpha_2, \alpha_3, \ldots)$, but the network is required to compute the desired function regardless of the precise values of $\alpha_2, \alpha_3, \ldots$. For example, one can implement polynomial interpolation in a $\neq$-uniform manner (since many distinct field elements are needed, but their precise values are not important).

3. For feasible fields, a more general notion is the simplest one: Constants are allowed in the circuit, but the Turing machine is required to print them in the circuit's description. Indeed, this calls for considering some natural notion of representing field elements.

4. A different generalization of $\neq$-uniformity suggested by von zur Gathen [vzGat86a] allows one to specify polynomial equations or inequations that must be satisfied by the constants appearing in the network, where the (in)equations specifying the $i$-th constant $\beta_i$ are computed by a uniform arithmetic circuit that is allowed to use the first $i-1$ constants $\beta_1, \ldots, \beta_{i-1}$ in its description.[11]

The notion that our definitions below will use is essentially the third one above (i.e., the machine needs to print the constants labeling the circuit), but many of our results also hold if we allow the more relaxed notion of $\neq$-uniformity (see Remarks 3.9 and 3.10).

[11] Formally, the numerical values of the constants $\beta_1, \ldots, \beta_{i-1}$ are not printed as part of the circuit's description. Instead, they are treated as symbolic variables, with the intended meaning that the variable $\beta_j$ corresponds to the $j$-th constant specified thus far.

### 3.3.2 P-uniform circuits and networks

The first and more relaxed notion of uniformity that we consider is P-uniformity, which essentially says that an arithmetic circuit or network can be printed in time that is polynomial in its size.

**Definition 3.8** (P-uniform circuits and networks)**.** *Let $\mathbb{F} = \{\mathbb{F}_n\}_{n\in\mathbb{N}}$ be a sequence of fields and let $\mathcal{N} = \{\mathcal{N}_n\}_{n\in\mathbb{N}}$ be a family of arithmetic networks where $\mathcal{N}_n$ is defined over $\mathbb{F}_n$. We say that $\mathcal{N}$ is* P-uniform *if there is Turing machine $M$ that on input $1^n$ runs in time polynomial in the size of $\mathcal{N}_n$ and outputs a description of $\mathcal{N}_n$, including the graph structure of the network and the constants labeling gates in the bottom layer.*

If the network family $\mathcal{N} = \{\mathcal{N}_n\}_{n\in\mathbb{N}}$ in Definition 3.8 is in fact a family of arithmetic circuits, then we refer to the family $\mathcal{N}$ as a P-uniform family of arithmetic circuits. We also extend Definition 3.8 in the straightforward way to circuit families that can be printed in some fixed polynomial time. That is, a circuit family is $n^c$-time-uniform if it satisfies Definition 3.8 with a machine $M$ that runs in time at most $n^c$.

**Remark 3.9.** *Our results are not sensitive to the choice of how to represent field elements in Definition 3.8. For example, over fields of characteristic zero, our results hold if we only allow constants that can be represented as bit-strings; but they also hold if we consider a real-RAM-like model in which machines can store field elements in registers, perform operations on them, and print them. The crucial point in Definition 3.8 is uniformity, i.e. that the networks $\{\mathcal{N}_n\}$ are printable by a uniform machine of constant description size (in whichever model of descriptions we are working with).*

**Remark 3.10.** *All of our results in the "hardness ⇒ randomness" direction hold even if the required hardness is only for the more restrictive notion of $\neq$-uniform circuits (recall that in this notion a machine does not need to print field elements, so we can avoid the question of representing field elements). Specifically, recall that when the field sequence $\mathbb{F} = \{\mathbb{F}_n\}$ is feasible,* P*-uniformity (as in Definition 3.8) is a more relaxed notion than $\neq$-uniformity. In Theorems 7.3 and 7.4 we only use the notion of* P*-uniformity to model the networks against which we need hardness, and we can indeed replace this notion by $\neq$-uniformity in these results, yielding a weaker hardness assumption. (We can do this because the reconstruction procedures in Theorems 5.1 and 6.1 can be implemented by $\neq$-uniform networks.) The reason we define* P*-uniformity as our model of choice for the paper, rather than $\neq$-uniformity, is only since the upper bound obtained via diagonalization in Theorem 7.2 (and Fact 3.12) does not seem to be $\neq$-uniform, but rather only* P*-uniform.*

### 3.3.3 $\log^c$-uniform circuits and networks

Our generators will rely on hard functions that satisfy a stricter notion of uniformity. Loosely speaking, the circuit-structure function, which receives names of gates and computes whether or not these gates are connected in the circuit, should be computable in time that is polynomial in the size of the gates' names (i.e., polylogarithmic in the circuit size).

**Definition 3.11** ($\log^c$-uniform circuit families)**.** *Let $\mathbb{F} = \{\mathbb{F}_n\}_{n\in\mathbb{N}}$ be a sequence of fields and let $c \in \mathbb{N}$. We say that an arithmetic circuit family $\{C_n\}$ over $\mathbb{F}$ of size $s(n)$ and depth $\Delta(n)$ is* $\log^c$-uniform *if the following holds.*

1. *(Circuit-structure function.) For each $n \in \mathbb{N}$, let $\Phi_n$ be the function that gets as input $i \in [\Delta(n)]$ and $u, v, w \in \{0,1\}^{\log(s(n))}$, returns 1 if gate $u$ in layer $i$ of $C_n$ is fed by gates $v$ and $w$ in layer $i-1$, and returns 0 otherwise. Then the family of functions $\Phi = \{\Phi_n\}_{n\in\mathbb{N}}$ is computable by a* P*-uniform family of Boolean formulas of size $(\log(s(n)))^c$.*[12]

2. *(Circuit-constants function.) For each $n \in \mathbb{N}$, let $\Psi_n$ be the function that gets as input $\vec{\Lambda} \in \mathbb{F}_n^n$ and $w \in \{0,1\}^{\log(s(n))}$ and outputs the value of the $w^{th}$ gate in the input layer of $C_n(\vec{\Lambda})$.*[13] *Then, there is a* P*-uniform family of arithmetic circuits $\{\Psi'_n \colon \mathbb{F}_n^n \times \mathbb{F}_n^{\log(s)} \to \mathbb{F}_n\}_{n\in\mathbb{N}}$ of size $n \cdot (\log(s(n)))^c$ and*

[12] For concreteness, we consider Boolean formulas with NAND gates of fan-in two (analogously to closely related definitions in [GKR15; Gol18; CT21]), and the precise choice of gate-basis does not matter for our results.

[13] Recall that arithmetic circuits have gates labeled with constant field elements only in their bottom layer (see Definition 3.5). That is, for a size-$s$ circuit, the bottom layer consists of $n$ input gates and at most $s-n$ gates labeled with various constants in the field. Accordingly, if $w \leq n$ then $\Psi_n(\vec{\Lambda}, w) = \vec{\Lambda}_w$, and otherwise $\Psi_n(\vec{\Lambda}, w)$ is the constant that labels gate $w$ in the input layer.

*degree* $n \cdot (\log(s(n)))^c$ *that computes* $\Psi = \{\Psi_n\}$. *That is,* $\Psi'_n$ *agrees with* $\Psi_n$ *on all inputs in the set* $\mathbb{F}_n^n \times \{0,1\}^{\log(s)}$.

We note that the assumption on the computability of the circuit-constants function above is quite ad hoc. We allowed arithmetic circuits for convenience, and since these can handle arbitrary field elements, but we could also replace this by requiring Boolean formulas akin to the circuit-structure function. The crucial points are only that the circuits/formulas will be of size polylog($s$) and that they are arithmetizable as low-degree polynomials (i.e., if these are Boolean models, we need them to be formulas).

### 3.3.4 Lower bounds for uniform circuits on all but finitely many inputs

As explained in Section 1, when considering uniform families of circuits with multiple output elements, there are functions that are hard for such circuits on all but finitely many inputs. This fact follows from a simple diagonalization argument.

**Fact 3.12** (almost-all-inputs hierarchy for uniform arithmetic circuits)**.** *For every integer* $k \in \mathbb{N}$ *and every sequence* $\mathbb{F} = \{\mathbb{F}_n\}_{n\in\mathbb{N}}$ *of fields, where the fields may be finite or infinite (e.g., we may have* $\mathbb{F}_n = \mathbb{C}$ *for all* $n \in \mathbb{N}$*), there is a* P*-uniform family of arithmetic circuits* $\{C_n\}_{n\in\mathbb{N}}$ *with* $n$ *input gates and* $n$ *output gates such that the following holds. For every* $n^k$*-time-uniform family* $\{C'_n\}$ *of arithmetic circuits of size* $n^k$ *and all but finitely many* $n \in \mathbb{N}$*, for every* $\vec{\alpha} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{F}_n^n$ *it holds that* $C_n(\vec{\alpha}) \neq C'_n(\vec{\alpha})$.

**Proof.** The proof follows by simple diagonalization. We define the Turing machine that prints $C = \{C_n\}$ as follows. Given $1^n$, the machine enumerates the first $n$ Turing machines (according to some efficient enumeration), simulates each of them for $n^k$ steps, and obtains $n$ arithmetic circuits over $\mathbb{F}_n$ of size $n^k$ with $n$ input gates and $n$ output gates, denoted $A_n^{(1)}, \ldots, A_n^{(n)}$. (If one of the $n$ Turing machines does not print such an arithmetic circuit in $n^k$ steps, we define the corresponding arithmetic circuit in some trivial way.)

For each $i \in [n]$, the machine modifies $A_n^{(i)}$ to a circuit $B_n^{(i)}$ that only computes the $i^{\text{th}}$ output gate of $A_n^{(i)}$. The machine prints a circuit $C_n$ that gets input $\vec{x}$ and outputs the $n$ elements

$$B_n^{(1)}(\vec{x}) + 1, B_n^{(2)}(\vec{x}) + 1, \ldots, B_n^{(n)}(\vec{x}) + 1.$$

Note that $C_n$ is indeed of polynomial size (i.e., size $O(n^{k+1})$). Also, for every $n^k$-time-uniform circuit family $\{C'_n\}$ printed by the $i^{\text{th}}$ machine, for all inputs $\vec{\alpha}$ of length at least $i$, we have $C'_n(\vec{\alpha}) \neq C_n(\vec{\alpha})$; this is the case since otherwise we would have $B_n^{(i)}(\vec{\alpha}) = A_n^{(i)}(\vec{\alpha})_i = C_n(\vec{\alpha})_i = B_n^{(i)}(\vec{\alpha}) + 1$, a contradiction. □

## 3.4 Randomized arithmetic networks

In this section we define two models of randomized arithmetic networks. As explained in Section 1, the first and simpler model will consist of networks with PIT gates (see Section 3.4.1) and the second and more general model consists of networks with random inputs (see Section 3.4.2).

### 3.4.1 Arithmetic networks with PIT gates

In the following definition of networks with PIT gates, the gates receive a description of an arithmetic circuit $C$, and compute PIT (as a Boolean-valued function) on $C$. That is:

**Definition 3.13** (arithmetic networks with PIT gates)**.** *An* arithmetic network with PIT gates $\mathcal{C}$ *is an arithmetic network that, in addition to arithmetic, Boolean, test, and selection gates described in Definition 3.6, has* PIT *gates that have unbounded fan-in and implement the following functionality: each* PIT *gate receives as input a universal encoding*[14] $\vec{\alpha}$ *of an arithmetic circuit* $C$ *and outputs a Boolean value* $v$ *such that* $v = 1$ *if and only if* $C$ *computes the zero polynomial.*

We remark that the choice of using universal encodings in the definition above over standard encodings (see Definition 3.22) is not arbitrary; the former guarantees that an encoding of length $n$ can only describe circuits computing polynomials of degree at most $n$, whereas this may not be the case for the latter.

[14] For a formal definition of a universal encoding, see Section 3.6.

The computation of a network with PIT gates on any given input is defined in the straightforward way (i.e., since the value of a PIT gate on an input encoding of a circuit $C$ is well-defined). And indeed, this definition will be, conceptually, our primary focus. However, in order to design algorithms that meet Definition 3.13, it will be convenient for us to work with auxiliary subroutines that compute relations, i.e., we consider these subroutines to be correct if they print any output in some predetermined set of valid outputs on this input (i.e., rather than insisting on a single canonical output).

Specifically, in the following definition, we consider a relation (representing a search problem) and define the network as correct on input $(\vec{\alpha}, \vec{\beta})$ if it outputs any element in the relation.

**Definition 3.14** (computation of arithmetic relations via networks with PIT gates)**.** *Let $C(\vec{x}, \vec{y})$ be an arithmetic network with* PIT *gates and $R \subseteq (\mathbb{F}^n \times \{0,1\}^m) \times (\mathbb{F}^\nu \times \{0,1\}^\mu)$ be a relation. For any $(\vec{\alpha}, \vec{\beta}) \in \mathbb{F}^n \times \{0,1\}^m$, define $R(\vec{\alpha}, \vec{\beta}) \coloneqq \left\{(\vec{\sigma}, \vec{\sigma}') \in \mathbb{F}^\nu \times \{0,1\}^\mu : ((\vec{\alpha}, \vec{\beta}), (\vec{\sigma}, \vec{\sigma}')) \in R\right\}$. We say that $C$* computes *$R$ at input $(\vec{\alpha}, \vec{\beta})$ if $C(\vec{\alpha}, \vec{\beta}) \in R(\vec{\alpha}, \vec{\beta})$, and we say that $C$* computes *$R$ if this property holds for every input $(\vec{\alpha}, \vec{\beta})$.*

We stress again that networks with PIT gates are a relatively restrictive model of randomized networks (which means that hardness assumptions for it are relatively weak hardness assumptions). In particular, if PIT can be solved by standard arithmetic networks, then arithmetic networks with PIT gates can be simulated by standard arithmetic networks (see, e.g., the proof of Theorem 7.2).

### 3.4.2 Arithmetic networks with random inputs

In this subsection, we define a randomized network model that is a more general notion than Definition 3.13 that equips arithmetic networks with an additional designated set of "random inputs" (that come from a prescribed finite set in the underlying field) to perform its computation, while still morally being algebraic in nature. Intuitively, the idea is that when compared to Definition 3.13, it is no longer restricted to using its randomness *only* to solve PIT for auxiliary circuits—instead, it may now use it to perform *other* auxiliary computation. However, it is still subject to the constraint that it eventually computes the correct value with high probability—and in particular, with error probability no more than the corresponding error bound guaranteed by Lemma 3.1 (for the randomized procedure that solves PIT by simply evaluating its input polynomial at a random point).

This slightly more general notion of randomized arithmetic networks is what is referenced in Theorem 1.4, as well as its corresponding formal statement in Theorem 7.4. Jumping ahead, the reason for assuming hardness against this subtly-generalized model there—instead of simply working with Definition 3.13—is because the proof of correctness of our corresponding (KI-based) targeted hitting-set generator relies on a reconstruction procedure that uses Kaltofen's [Kal89] classical factoring algorithm as a subroutine. Indeed, this subroutine (described in detail in Section A.4) uses its input randomness in a manner that does not seem to be easily simulated using PIT gates (see also Problem 8.2 in Section 8).

**Definition 3.15** (randomized arithmetic networks)**.** *A* randomized arithmetic network *$C$ is an arithmetic network that, in addition to arithmetic inputs $\vec{x}$ and Boolean inputs $\vec{y}$ (now referred to as "primary" inputs), receives a supplementary set $\vec{r}$ of arithmetic inputs (which we term the "random" set of arithmetic inputs).*

Definition 3.15 defines only the syntax of randomized arithmetic networks. We next define the semantics of randomized arithmetic networks.

**Definition 3.16** (computation by randomized arithmetic networks)**.** *Let $C((\vec{x}, \vec{y}), \vec{r})$ be an arithmetic network with primary inputs $\vec{x}, \vec{y}$ and supplementary ("random") inputs $\vec{r}$. We say that $C$* computes value *$\vec{\gamma}$* at primary input *$(\vec{\alpha}, \vec{\beta})$* with error degree *$e$ if for every finite set $S \subset \mathbb{F}$, when each random input $r_i$ is chosen independently from $S$, with probability at least $1 - e/|S|$ we have $C((\vec{\alpha}, \vec{\beta}), \vec{r}) = \vec{\gamma}$. When $\mathbb{F}$ is finite, we may drop the qualifier "error degree $e$", in which case we consider $S = \mathbb{F}$ and require probability at least $2/3$.*

Analogous to Definition 3.14, we next define the notion of computing a relation by a randomized arithmetic network.

**Definition 3.17** (computation of arithmetic relations via randomized arithmetic networks)**.** *Let $C((\vec{x}, \vec{y}), \vec{r})$ be an arithmetic network with primary inputs $\vec{x}, \vec{y}$ and supplementary ("random") inputs $\vec{r}$. Let $R \subseteq (\mathbb{F}^n \times$*

$\{0,1\}^m) \times (\mathbb{F}^\nu \times \{0,1\}^\mu)$ *be a relation, and define* $R(\vec{\alpha},\vec{\beta}) \coloneqq \left\{(\vec{\sigma},\vec{\sigma}') \in \mathbb{F}^\nu \times \{0,1\}^\mu : ((\vec{\alpha},\vec{\beta}),(\vec{\sigma},\vec{\sigma}')) \in R\right\}$ *for every* $\vec{\alpha},\vec{\beta}$.

*We say that* $C$ computes $R$ at primary input $(\vec{\alpha},\vec{\beta})$ with error degree $e$ *if for every finite set* $S \subseteq \mathbb{F}$, *when each random input* $r_i$ *is chosen independently from* $S$, *with probability at least* $1 - e/|S|$, *we have* $C((\vec{\alpha},\vec{\beta}),\vec{r}) \in R(\vec{\alpha},\vec{\beta})$. *We say that* $C$ *computes* $R$ *with* error degree $e$ *if this property holds for every primary input. When* $\mathbb{F}$ *is finite, we may drop the qualifier "error degree* $e$*", in which case we consider* $S = \mathbb{F}$ *and require probability at least* $2/3$.

### 3.4.3 Concatenation and composition of randomized arithmetic networks

In this section we define notions of concatenating randomized arithmetic networks and of composing randomized arithmetic networks, and state some basic properties of these operations. The two notions are relevant both for Definition 3.13 and for Definition 3.15. However, since the operations and their properties are obvious for the model presented in the former definition, we will only present the operations and their properties formally while referring to the latter definition.

**Definition 3.18** (Concatenation of randomized arithmetic networks)**.** *Let* $\mathcal{R}$ *and* $\mathcal{S}$ *be two randomized arithmetic networks that compute relations* $R_\mathcal{R} \subseteq (\mathbb{F}^{n_\mathcal{R}} \times \{0,1\}^{m_\mathcal{R}}) \times (\mathbb{F}^{\nu_\mathcal{R}} \times \{0,1\}^{\mu_\mathcal{R}})$ *and* $R_\mathcal{S} \subseteq (\mathbb{F}^{n_\mathcal{S}} \times \{0,1\}^{m_\mathcal{S}}) \times (\mathbb{F}^{\nu_\mathcal{S}} \times \{0,1\}^{\mu_\mathcal{S}})$, *and take* $r_\mathcal{R}, r_\mathcal{S}$ *many random arithmetic inputs, respectively. The* concatenated arithmetic network $(\mathcal{R},\mathcal{S})$ *computes a relation*

$$(R_\mathcal{R}, R_\mathcal{S}) \subseteq (\mathbb{F}^{n_\mathcal{R}} \times \{0,1\}^{m_\mathcal{R}}) \times (\mathbb{F}^{n_\mathcal{S}} \times \{0,1\}^{m_\mathcal{S}}) \times (\mathbb{F}^{\nu_\mathcal{R}} \times \{0,1\}^{\mu_\mathcal{R}}) \times (\mathbb{F}^{\nu_\mathcal{S}} \times \{0,1\}^{\mu_\mathcal{S}})$$

*as follows. The network* $(\mathcal{R},\mathcal{S})$ *takes as primary arithmetic input a vector* $(\vec{\alpha}_1,\vec{\alpha}_2) \in \mathbb{F}^{n_\mathcal{R}} \times \mathbb{F}^{n_\mathcal{S}}$, *a Boolean vector* $(\vec{b}_1,\vec{b}_2) \in \{0,1\}^{m_\mathcal{R}} \times \{0,1\}^{m_\mathcal{S}}$, *and random arithmetic inputs* $\vec{\gamma}_1 \in \mathbb{F}^{r_\mathcal{R}}$ *and* $\vec{\gamma}_2 \in \mathbb{F}^{r_\mathcal{S}}$, *and on input* $(\vec{\alpha}_1,\vec{b}_2,\vec{\alpha}_2,\vec{b}_2)$ *it outputs* $(\mathcal{R}(\vec{\alpha}_1,\vec{b}_2,\vec{\gamma}_1),\mathcal{S}(\vec{\alpha}_2,\vec{b}_2,\vec{\gamma}_2))$.

**Proposition 3.19** (Degree and error degree under concatenation)**.** *Let* $\mathcal{R}$ *and* $\mathcal{S}$ *be randomized arithmetic networks as in Definition 3.18. Let* $d_\mathcal{R}$ *and* $d_\mathcal{S}$ *denote the degrees of* $\mathcal{R}$ *and* $\mathcal{S}$, *respectively; let* $e_\mathcal{R}$ *and* $e_\mathcal{S}$ *denote the error degrees of* $\mathcal{R}$ *and* $\mathcal{S}$, *respectively. Then the degree of the concatenated arithmetic network* $(\mathcal{R},\mathcal{S})$ *is bounded by* $\max\{d_\mathcal{R}, d_\mathcal{S}\}$ *and the error degree of* $(\mathcal{R},\mathcal{S})$ *is bounded by* $e_\mathcal{R} + e_\mathcal{S}$.

**Proof Sketch.** The bound for the degree is straightforward. The bound for the error degree follows from a simple union bound. □

We also need a notion of composition for randomized arithmetic networks. We formalize this in the following definition, and then subsequently bound the degree and error degree of the composition.

**Definition 3.20** (Composition of randomized arithmetic networks)**.** *Let* $\mathcal{R}$ *and* $\mathcal{S}$ *be two randomized arithmetic networks that compute relations* $R_\mathcal{R} \subseteq (\mathbb{F}^{n_\mathcal{R}} \times \{0,1\}^{m_\mathcal{R}}) \times (\mathbb{F}^{\nu_\mathcal{R}} \times \{0,1\}^{\mu_\mathcal{R}})$ *and* $R_\mathcal{S} \subseteq (\mathbb{F}^{n_\mathcal{S}} \times \{0,1\}^{m_\mathcal{S}}) \times (\mathbb{F}^{\nu_\mathcal{S}} \times \{0,1\}^{\mu_\mathcal{S}})$, *and take* $r_\mathcal{R}, r_\mathcal{S}$ *many random arithmetic inputs, respectively. Then, we say that the* $\mathcal{R}$ *and* $\mathcal{S}$ *(as an ordered pair) are* composable *if* $\nu_\mathcal{S} = n_\mathcal{R}$ *and* $\mu_\mathcal{S} = m_\mathcal{R}$.

*If* $\mathcal{R}$ *and* $\mathcal{S}$ *are composable, we define the relation*

$$R_\mathcal{R} \circ R_\mathcal{S} \subseteq (\mathbb{F}^{n_\mathcal{S}} \times \{0,1\}^{m_\mathcal{S}}) \times (\mathbb{F}^{\nu_\mathcal{R}} \times \{0,1\}^{\mu_\mathcal{R}})$$

*to be the set of all tuples* $((\vec{\alpha}_1,\vec{b}_1),(\vec{\alpha}_3,\vec{b}_3))$ *such that there exists* $(\vec{\alpha}_2,\vec{b}_2)$ *for which both* $R_\mathcal{R}((\vec{\alpha}_1,\vec{b}_1),(\vec{\alpha}_2,\vec{b}_2))$ *and* $R_\mathcal{S}((\vec{\alpha}_2,\vec{b}_2),(\vec{\alpha}_3,\vec{b}_3))$ *are satisfied. The* composition $\mathcal{R} \circ \mathcal{S}$ *computes the relation* $R_\mathcal{R} \circ R_\mathcal{S}$ *as follows: it takes as primary arithmetic input a vector* $\vec{\alpha} \in \mathbb{F}^{n_\mathcal{S}}$, *a Boolean vector* $\vec{b} \in \{0,1\}^{m_\mathcal{S}}$, *and random arithmetic inputs* $\vec{\gamma}_1 \in \mathbb{F}^{r_\mathcal{R}}$ *and* $\vec{\gamma}_2 \in \mathbb{F}^{r_\mathcal{S}}$, *and outputs* $\mathcal{R}(\mathcal{S}(\vec{\alpha},\vec{b},\vec{\gamma}_2),\vec{\gamma}_1)$.

**Proposition 3.21** (Degree and error degree under composition)**.** *Let* $\mathcal{R}$ *and* $\mathcal{S}$ *be composable randomized arithmetic networks as in Definition 3.20. Furthermore, let* $d_\mathcal{R}$ *and* $d_\mathcal{S}$ *denote the degrees of* $\mathcal{R}$ *and* $\mathcal{S}$, *respectively; let* $e_\mathcal{R}$ *and* $e_\mathcal{S}$ *denote the error degrees of* $\mathcal{R}$ *and* $\mathcal{S}$, *respectively. Then the degree of the composed arithmetic network* $\mathcal{R} \circ \mathcal{S}$ *is bounded by the product* $d_\mathcal{R} \cdot d_\mathcal{S}$ *and the error degree of* $\mathcal{R} \circ \mathcal{S}$ *is bounded by* $e_\mathcal{R} + e_\mathcal{S}$.

**Proof.** The bound for the degree is straightforward. To show the bound on the error degree of the composition, from Definition 3.17, it suffices to show that for every input $(\vec{\alpha}, \vec{b}) \in \mathbb{F}^{n_S} \times \{0,1\}^{m_S}$ and any finite set $S \subset \mathbb{F}$,

$$\mathop{\mathbb{P}}_{\substack{\vec{\gamma}_1 \leftarrow S^{r_\mathcal{R}}, \\ \vec{\gamma}_2 \leftarrow S^{r_\mathcal{S}}}} \left[ \mathcal{R} \circ \mathcal{S}(\vec{\alpha}, \vec{b}, (\vec{\gamma}_1, \vec{\gamma}_2)) \notin R_\mathcal{R} \circ R_\mathcal{S}(\vec{\alpha}, \vec{b}) \right] \leq \frac{e_\mathcal{R} + e_\mathcal{S}}{|S|}. \tag{1}$$

Consider any fixed input $(\vec{\alpha}, \vec{b}) \in \mathbb{F}^{n_S} \times \{0,1\}^{m_S}$. Let $E_1$ denote the event that $\mathcal{R} \circ \mathcal{S}$ outputs in the relation $R_\mathcal{R} \circ R_\mathcal{S}$ on input $(\vec{\alpha}, \vec{b})$, and $E_2$ denote the event that $\mathcal{S}$ outputs in the relation $R_\mathcal{S}(\vec{\alpha}, \vec{b})$ (i.e., computes "correctly"). Note that again from Definition 3.17, we have that $\mathbb{P}[E_1|E_2] \geq 1 - e_\mathcal{R}/|S|$ and $\mathbb{P}[E_2] \geq 1 - e_\mathcal{S}/|S|$. Therefore, we have

$$\mathbb{P}[E_1] \geq \mathbb{P}[E_1 \cap E_2] = \mathbb{P}[E_1|E_2] \cdot \mathbb{P}[E_2] \geq \left(1 - \frac{e_\mathcal{R}}{|S|}\right)\left(1 - \frac{e_\mathcal{S}}{|S|}\right) \geq 1 - \frac{e_\mathcal{R}}{|S|} - \frac{e_\mathcal{S}}{|S|}.$$

Since this is true for an arbitrary choice of a primary input to $\mathcal{R} \circ \mathcal{S}$, we conclude that its error degree is at most $e_\mathcal{R} + e_\mathcal{S}$. □

## 3.5 The relation to BSS machines

The PIT algorithms in Theorems 1.2 and 1.4 work assuming lower bounds for P-uniform randomized networks, and in fact even for $\neq$-uniform randomized networks as mentioned in Remark 3.10. However, our proofs do not heavily capitalize on the specific features of this computational model, and we suspect that the PIT algorithms also work assuming lower bounds for other reasonable models of uniform randomized arithmetic computation. The key point is that the reconstruction procedures we present in Sections 5 and 6 can also be implemented in other models of uniform randomized arithmetic computation.[15]

In particular, we believe that our PIT algorithms should work assuming lower bounds for the more standard model of (randomized) constant-free Blum–Shub–Smale machines (BSS machines) [BSS89]. This is because constant-free BSS machines are seemingly at least as strong as $\neq$-uniform networks (and ditto if we allow randomness or PIT gates/oracle in both models), and thus hardness for the former seems to be a stronger assumption than hardness for the latter.

Uniform networks and BSS machines are very similar in spirit (see below). The reason we work with networks in this paper is the conceptually clearer comparison of the upper-bounds to the lower-bounds (i.e., both refer to circuits or networks), as well as the clean converse direction in Theorem 1.3. An additional (mostly cosmetic) benefit in networks is a clearer separation between auxiliary Boolean inputs and arithmetic inputs, especially in intermediate auxiliary computations in the reconstruction procedures.[16]

**More detail about BSS machines.** Recall that a BSS machine receives a vector of elements from a ring $R$ as input, performs ring operations on its input and storage (which consists of ring elements) and then outputs another vector of elements from $R$. In addition to ring operations, a BSS machine may also perform equality tests with zero (similarly to a network) to decide which ring operations to perform next; over ordered rings inequality tests with zero are often also allowed (i.e., tests of the form "$v \geq 0$?", which are not allowed in networks). The control flow of the machine is specified by a finite directed graph, just as a Turing machine is specified by a finite collection of states and transitions. The machine may also be supplied with a finite list of constants from $R$.

Randomized BSS machines are slightly more subtle, and we need to restrict the constants allowed in the machine's description (otherwise PIT can be solved in polynomial time, by using "unreasonable" real constants to represent non-uniform advice; see [BCSS98, Chapter 17, Theorem 7]). As mentioned above, we can simply consider constant-free BSS machines, which already seem strong enough to simulate $\neq$-uniform networks. To augment the BSS machine with randomness, we either allow oracle calls to PIT (analogously to networks with PIT gates) or allow random field elements (analogously to randomized networks).

[15] The main non-trivial operation that the reconstruction procedures use is branching according to zero-tests; that is, an operation of the form "if $v = 0$ continue with arithmetic procedure $A$, and if $v \neq 0$ continue with arithmetic procedure $B$", where $v$ is the result of an intermediary arithmetic computation.

[16] In contrast to networks, the inputs to a BSS machine are all from the underlying ring. Thus, if we want to allow for subroutines that operate only on Boolean data, it becomes necessary to speak of partial functions computed by BSS machines.

For more details on BSS machines, we refer the reader to Blum, Cucker, Shub, and Smale [BCSS98] and Meer and Michaux [MM97], and in particular to [BCSS98, Chapter 17] for the randomized variant.

## 3.6 Universal arithmetic circuits and networks

In this subsection, we define two notions of encodings for arithmetic circuits and observe that it is possible to uniformly construct arithmetic networks that can evaluate any circuit on a given point when given its encoding as input. These notions are repeatedly used by the reconstruction networks of our main results.

**Definition 3.22** (Standard encoding of an arithmetic circuit)**.** *Let $\mathbb{F}$ be a field and let $\Phi$ be an algebraic circuit over $\mathbb{F}$. We say that a tuple $(\vec{\alpha}, \vec{b}) \in \mathbb{F}^n \times \{0,1\}^m$ is a* standard encoding *or* standard description *of $\Phi$ if $\vec{b}$ is a bit-string that describes the wiring of $\Phi$ and the type of each gate and $\vec{\alpha}$ is a vector of field elements where $\alpha_i$ is the $i$-th field constant used by the circuit $\Phi$.*

**Proposition 3.23.** *If $C$ is an arithmetic circuit of size $s$, then $C$ admits a standard encoding of size $O(s \log s)$.*

**Proof.** For each gate $v$ in $C$, the type of $v$ can be specified with $O(1)$ bits, so the types of all gates can be specified with $O(s)$ bits. Each wire in $C$ can be described by $O(\log s)$ bits. As we assume our circuits to have fan-in two, a circuit of size $s$ necessarily has $O(s)$ wires, so we can describe the wires of $C$ using $O(s \log s)$ bits. Finally, a circuit of size $s$ uses at most $s$ field constants. Thus there is a standard description $(\vec{\alpha}, \vec{b}) \in \mathbb{F}^n \times \{0,1\}^m$ of $C$ of length $n + m \leq O(s \log s)$. □

**Proposition 3.24** (Arithmetic networks can evaluate arithmetic circuits)**.** *Let $\mathbb{F}$ be any field and let $d : \mathbb{N} \to \mathbb{N}$ be a time-constructible function. There is a* P*-uniform family of arithmetic networks of size* $\mathrm{poly}(n, s, d(n))$ *and degree* $\mathrm{poly}(n, d(n))$ *over $\mathbb{F}$ that takes as input (i) a standard encoding of an $n$-variate arithmetic circuit $C$ over $\mathbb{F}$ of size $s$ and formal degree $d(n)$ and (ii) a vector $\vec{\alpha} \in \mathbb{F}^n$ (as arithmetic data), and outputs (as arithmetic data in $\mathbb{F}$) the evaluation $C(\vec{\alpha})$.*

**Proof.** The network proceeds by evaluating the circuit $C$ gate-by-gate in topological order. For each $i \in [s]$, we compute the value of the $i$-th intermediate gate in the circuit $C$ on input $\vec{\alpha}$. The network does this by computing all pairwise sums of the values it has computed so far, along with all pairwise products subject to the constraint that the resulting product has formal degree at most $d(n)$. These computations can be carried out using $O(i^2) \leq O(s^2)$ many arithmetic gates. The network then reads the Boolean part of the description of the circuit $C$ to determine the type of the $i$-th gate in $C$, along with the names of the children of the $i$-th gate. Feeding this into select gates, the network can correctly compute the value of the $i$-th gate in $C$.

It follows from the description above this network has size $\mathrm{poly}(n, s, d(n))$ and degree $\mathrm{poly}(n, d(n))$. Moreover, the wiring of this network can be printed by a Turing machine running in time $\mathrm{poly}(n, s, d(n))$. □

**Proposition 3.25.** *There is a* P*-uniform arithmetic network of size* $\mathrm{poly}(n, m, s, s')$ *and degree* 1 *over any field $\mathbb{F}$ that takes as input (i) a standard description of an arithmetic circuit $C$ of size $s$ over $\mathbb{F}$ computing an $n$-variate polynomial and (ii) a standard description of a multi-output arithmetic circuit $C'$ of size $s'$ over $\mathbb{F}$ having $n$ output gates where each output gate computes an $m$-variate polynomial, and outputs a standard description of the circuit computing their composition i.e., $C \circ C'$.*

**Proof.** The arithmetic network computes the Boolean part of a standard encoding of $C \circ C'$ by replacing the input gates of $C$ labeled by variables with the output gates of $C'$. This rewiring can be done by a P-uniform family of Boolean circuits. The arithmetic part of a standard encoding of $C \circ C'$ corresponds to the concatenation of the arithmetic parts of standard encodings of $C$ and $C'$, which can clearly by computed by a P-uniform arithmetic network of size $s + s'$ and degree 1. □

**Definition 3.26** (Universal circuits)**.** *Let $\mathbb{F}$ be a field and let $n, d, s \in \mathbb{N}$. We say that a polynomial $C(\vec{x}, \vec{y})$ is* universal for $n$-variate, degree-$d$, size-$s$ computation over $\mathbb{F}$ *if for every polynomial $f(\vec{x})$ where $\deg(f) \leq d$ and $f$ is computable by an algebraic circuit of size $s$, there is a point $\vec{\beta} \in \mathbb{F}^m$ such that $C(\vec{x}, \vec{\beta}) = f(\vec{x})$. An algebraic circuit is* universal for $n$-variate, degree-$d$, size-$s$ computation *if it computes a polynomial that is universal for $n$-variate, degree-$d$, size-$s$ computation.*

**Definition 3.27** (Universal encoding)**.** *Let $\mathbb{F}$ be a field and fix a family $\mathcal{C} = \{C_n\}_{n\in\mathbb{N}}$ of* P*-uniform arithmetic circuits such that $C_n$ is universal for $n$-variate, degree-$d$, size-$s$ computation. Let $\Phi$ be an $n$-variate algebraic circuit of size $s$ over $\mathbb{F}$ that computes a polynomial of degree $d$. We say that a point $\vec{\alpha} \in \mathbb{F}^t$ is a* $\mathcal{C}$-universal encoding *of $\Phi$ if for $m = \max(n, s, d)$, the circuit $C_m$ satisfies $C_m(\vec{x}, \vec{\alpha}) = \Phi(\vec{x})$.*

Throughout this work, we make use of the universal encoding with respect to the family of universal circuits $\{\Psi_n\}_{n\in\mathbb{N}}$ constructed by Raz [Raz10]. As this is the only universal circuit we make use of, we abbreviate the term "$\Psi$-universal encoding" as "universal encoding."

**Theorem 3.28** ([Raz10, Proposition 2.8])**.** *Let $\mathbb{F}$ be a field. For every $n, d, s \in \mathbb{N}$, there is a constant-free circuit $\Psi$ that is universal for $n$-variate, degree-$d$, size-$s$ computation over $\mathbb{F}$. The size of $\Psi$ is bounded by $O(sd^4)$. Moreover, there is a polynomial-time algorithm that receives $(1^n, 1^s, 1^d)$ as input and outputs a description of the circuit $\Psi$.*

We conclude this subsection by extending some of the aforementioned notions to arithmetic networks with PIT gates. These extensions will be useful in establishing the randomness $\Rightarrow$ hardness direction of our results, i.e., Theorem 1.3 (or more precisely, its formalization, Theorem 7.2 in Section 7.1).

**Definition 3.29** (Standard encoding of an arithmetic network with PIT gates)**.** *Let $\mathbb{F}$ be a field and let $\mathcal{C}$ be an arithmetic network with* PIT *gates over $\mathbb{F}$. We say that a tuple $(\vec{\alpha}, \vec{b}) \in \mathbb{F}^n \times \{0,1\}^m$ is a* standard encoding *or* standard description *of $\mathcal{C}$ if $\vec{b}$ is a bit-string that describes the wiring of $\mathcal{C}$ and the type of each gate and $\vec{\alpha}$ is a vector of field elements where $\alpha_i$ is the $i$-th field constant used by the network $\mathcal{C}$.*

The following statement has a proof similar to that of Proposition 3.23.

**Proposition 3.30.** *If $C$ is an arithmetic network with* PIT *gates of size $s$, then $C$ admits a standard encoding of size $O(s \log s)$.*

**Proposition 3.31** (Arithmetic networks can evaluate arithmetic networks)**.** *Let $\mathbb{F}$ be any field and let $d : \mathbb{N} \to \mathbb{N}$ be a time-constructible function. There is a* P*-uniform family of arithmetic networks of size $\mathrm{poly}(n, s, d(n))$ and degree $\mathrm{poly}(n, d(n))$ over $\mathbb{F}$ that takes as input (i) a standard encoding of an arithmetic network $C$ over $\mathbb{F}$ of size $s$ and degree $d(n)$ and (ii) a vector $(\vec{\alpha}, \vec{\beta}) \in \mathbb{F}^n \times \{0,1\}^n$, and outputs the evaluation $C(\vec{\alpha}, \vec{\beta})$.*

**Proof Sketch.** The proof is similar to that Proposition 3.24. The network mimics the one in the proof of Proposition 3.24, with the only difference now being that it computes the value of the $i$-th intermediate gate in the network $C$ on input $(\vec{\alpha}, \vec{\beta})$ by not just accounting for all the pairwise sums and products (of degree at most $d(n)$) of the arithmetic values it has computed so far, but also for the corresponding Boolean operations performed on the Boolean values computed so far, along with the test gates values $\stackrel{?}{=}$, and the select gate values of their appropriate combinations. □

# 4 Polynomial decompositions for arithmetic circuits

In this section, we recall the notion of polynomial decompositions of a circuit, following Chen and Tell [CT21]. The definition provided in this section adapts [CT21, Definition 4.6] to algebraic circuits. This notion is an encoding of a circuit's computation (on a fixed input $\vec{\Lambda} \in \mathbb{F}^n$) as a sequence of downward self-reducible polynomials. After presenting the abstract notion of polynomial decompositions, we will show that any $\log^c$-uniform family of algebraic circuits has efficient polynomial decompositions.

## 4.1 Variable-extended formulations

Before introducing polynomial decompositions of circuits, we need the definition of a variable-extended formulation of a boolean function. This is a technical notion that, while not strictly necessary for the definition of polynomial decompositions, leads to an improvement in the parameters of a polynomial decomposition. In particular, the use of variable-extended formulations allows us to guarantee that the individual degrees of the hard polynomials are bounded by a constant.

**Definition 4.1** (Variable-extended formulation)**.** *Let $f\colon S^m \to \{0,1\}$ where $S$ is an arbitrary set. We say that $g(x_1,\ldots,x_m,y_1,\ldots,y_r)$ is a* variable-extended formulation *of $f$ if the following holds for every $\vec{x} \in S^m$.*

1. *If $f(\vec{x}) = 1$, then there exists a unique $\vec{y}(\vec{x}) \in S^r$ ("unique witness") such that $g(\vec{x},\vec{y}(\vec{x})) = f(\vec{x}) = 1$, and $g(\vec{x},\vec{y}) = 0$ for all $\vec{y} \neq \vec{y}(\vec{x})$.*
2. *If $f(\vec{x}) = 0$, then $g(\vec{x},\vec{y}) = 0$ for all $\vec{y} \in S^r$.*

**Observation 4.2.** Any variable-extended formulation $g$ of $f$ satisfies

$$f(\vec{x}) = \sum_{\vec{y}\in S^r} g(\vec{x},\vec{y})$$

for every $\vec{x} \in S^m$.

The following lemma asserts that any small Boolean formula has a variable-extended formulation that can be computed by a small arithmetic formula of constant individual degree. The proof follows [KLV23] and is based on a Cook–Levin-style trick.

**Lemma 4.3** (Modification of [KLV23, Lemma 5.3])**.** *Let $f : \{0,1\}^m \to \{0,1\}$ be a NAND-boolean formula of size $s$ and let $\mathbb{F}$ be any field. Then there exists a variable-extended formulation $g : \{0,1\}^{m+s} \to \{0,1\}$ of $f$ that can be computed by an arithmetic formula[17] of size $O(s\cdot m)$ over $\mathbb{F}$, which computes a polynomial of total degree $O(s\cdot m)$ and individual degree 2. Moreover, there is a $\mathsf{P}$-uniform family of arithmetic networks of size $\mathrm{poly}(s\cdot m)$ that, when given as input a standard description[18] of the NAND-formula for $f$, outputs a standard description of this arithmetic formula for $g$.*

**Proof.** We introduce $s$ new variables $y_1,\ldots,y_s$, one for each wire of the formula computing $f$. For convenience of notation, we assume that there is a wire coming out of the output gate that has the label $y_s$.

We identify each gate $\Phi = \Phi_i$ with the wire $i \in [s]$ going out of $\Phi$. Let

$$\Phi_i(\vec{y}) \coloneqq y_i + y_j y_k - 2y_i y_j y_k,$$

where $j,k \in [s]$ are the indices of the wires going into $\Phi_i$. For a zero-one-valued input $\vec{y} \in \{0,1\}^s$, we have

$$\Phi_i(\vec{y}) = \begin{cases} 1 & \text{if } y_i = \mathsf{NAND}(y_j,y_k), \\ 0 & \text{otherwise.} \end{cases}$$

For every index $(j,\sigma) \in [m]\times\{0,1\}$ of an input literal (i.e., $x_j$ or $\neg x_j$), we define

$$h_{j,\sigma}(\vec{x},\vec{y}) \coloneqq \ell_{j,\sigma} \prod_{i\in S_{j,\sigma}} y_i + (1-\ell_{j,\sigma}) \prod_{i\in S_{j,\sigma}} (1-y_i),$$

where

$$\ell_{j,\sigma} \coloneqq \begin{cases} x_j & \text{if } \sigma = 0, \\ 1-x_j & \text{if } \sigma = 1, \end{cases}$$

and $S_{j,\sigma}$ denotes the set of leaf wires corresponding to $\ell_{j,\sigma}$. Observe that for every $\vec{x},\vec{y} \in \{0,1\}^{m+s}$ we have that $h_{j,\sigma}(\vec{x},\vec{y}) = 1$ if and only if for all $i \in S_{j,\sigma}$, the variable $y_i$ is set to $\ell_{j,\sigma}$. Otherwise, we have $h_{j,\sigma}(\vec{x},\vec{y}) = 0$. Then we define

$$g(\vec{x},\vec{y}) \coloneqq y_s \prod_{i=1}^{s} \Phi_i(\vec{y}) \prod_{j=1}^{m} \left( \prod_{\sigma\in\{0,1\}} h_{j,\sigma}(\vec{x},\vec{y}) \right).$$

The claims about the formula size and total degree follow immediately from the definition of $g$. It follows from the definitions of the $\Phi_i$'s and $h_{j,\sigma}$'s that the restriction of $g(\vec{x},\vec{y})$ to $\{0,1\}^{m+s}$ corresponds to a Boolean function that is a variable-extended formulation of $f$.

[17] By this, we mean that the formula computes a polynomial whose evaluations on $\{0,1\}^{m+s}$ agree with a variable-extended formulation $g : \{0,1\}^{m+s} \to \{0,1\}$.

[18] Since $f$ is a Boolean formula, by a standard description, we simply mean a description that describes the wiring of $f$.

Finally, note that $g(\vec{x}, \vec{y})$ has individual degree 2, since (i) the variable $y_s$ appears only twice, (ii) each intermediate (non-output, non-leaf) variable only appears twice because the corresponding wire has exactly two endpoints, and (iii) the variables $\{x_j, y_i | j \in [m], i \in S_{j,\sigma}\}$ have degree at most 2 (they occur at most once in the first product, while the second product is multilinear). The efficient computation of a standard description of $g$ (see Definition 3.22) can be done via a $\mathsf{P}$-uniform family of arithmetic networks $\{\mathcal{N}_m\}_{m \in \mathbb{N}}$ as follows. First, from the given description of $f$, the arithmetic network $\mathcal{N}_m$ reads off the set of wires $S_{j,\sigma}$ adjacent to each input literal. Given $S_{j,\sigma}$, a standard description of $h_{j,\sigma}$ can be output using a constant overhead using its definition above. From the given wiring of $f$, the network is able to compute a description of each $\Phi_i$ of constant size. Putting everything together, the network $\mathcal{N}_m$ outputs a standard description of the arithmetic formula for $g$. □

## 4.2 Polynomial decompositions

We recall that the overall idea in the definition below is to arithmetize the circuit structure function layer-by-layer over an appropriate arithmetic setting, and "in between layers", we add additional polynomials that represent a suitable sumcheck protocol, which allows reducing claims about each layer to claims about the preceding layer. We allow some parameters to be arbitrary to allow for greater functionality in the reconstruction procedures later, in particular the size of an interpolating set $H \subseteq \mathbb{F}$ and an integer parameter $r$. Jumping ahead, we will instantiate the notion either with $H = \{0, 1\}$ and $r > 0$, or with a larger $H$ (e.g., $|H| = n^{1/\log\log(n)}$) and $r = 0$.

**Definition 4.4** (Polynomial decompositions of a circuit)**.** *Let $C$ be an algebraic circuit over a field $\mathbb{F}$ that has $n$ inputs, fan-in two, size $s$, and depth $\Delta$. For any $\vec{\Lambda} \in \mathbb{F}^n$, we call a collection of polynomials $\{f_{i,j}\}$ an $(h, m, r)$-*polynomial decomposition of $C(\vec{\Lambda})$ *if it satisfies the following requirements.*

1. *(Setting.) The polynomials $f_{i,j}$ are defined over the field $\mathbb{F}$. For some integer $h \leq |\mathbb{F}|$ such that $h$ is a power of 2, let $H \subseteq \mathbb{F}$ be of size $h$, let $m$ be the minimum positive integer such that $h^m \geq s$, and let $r \geq 0$ be an integer.*

2. *(Circuit structure polynomial.) For each $i \in [\Delta]$, let $\hat{\Phi}_i : H^{3m} \to \{0, 1\}$ be the function such that $\hat{\Phi}_i(\vec{\alpha}, \vec{\beta}, \vec{\gamma}) = 1$ if and only if the gate in layer $i$ of $C$ indexed by $\vec{\alpha}$ has as children the gates in layer $i-1$ indexed by $\vec{\beta}$ and $\vec{\gamma}$.[19] Let $\hat{\Phi}_i^e : H^{3m+r} \to \{0, 1\}$ be a variable-extended formulation of $\hat{\Phi}_i$ (so if $r = 0$, $\hat{\Phi}_i^e = \hat{\Phi}_i$). Let the polynomial $\Phi_i(w_1, \ldots, w_m, u_1, \ldots, u_m, v_1, \ldots, v_m, y_1, \ldots, y_r) \in \mathbb{F}[\vec{w}, \vec{u}, \vec{v}, \vec{y}]$ be any polynomial which agrees with the function $\hat{\Phi}_i^e$ on the set $H^{3m+r}$.*

3. *(Input-layer function.) Let $\hat{f}_0 : H^m \to \mathbb{F}$ represent[20] the input layer string $\vec{\Lambda}\vec{K} \in \mathbb{F}^s$ (interpreted as the concatenation of vectors $\vec{\Lambda}$ and $\vec{K}$), where $\vec{K}$ is the list of constants that $C$ uses. Let $f_0(w_1, \ldots, w_m)$ be any polynomial over $\mathbb{F}$ that agrees with $\hat{f}_0$ on all inputs in $H^m$.*

4. *(Layer polynomials.) For each $i \in [\Delta]$, let $\hat{f}_i : H^m \to \mathbb{F}$ represent[21] the values of the gates at the $i^{th}$ layer of $C$ in the computation of $C(\vec{\Lambda})$, with zeroes in locations that do not index valid gates. Define $f_i \in \mathbb{F}[w_1, \ldots, w_m]$ inductively by*

$$f_i(\vec{w}) := \begin{cases} \sum_{\vec{\beta}, \vec{\gamma} \in H^m, \vec{\delta} \in H^r} \Phi_i(\vec{w}, \vec{\beta}, \vec{\gamma}, \vec{\delta}) \left( f_{i-1}(\vec{\beta}) + f_{i-1}(\vec{\gamma}) \right), & \text{if } i \text{ is odd} \\ \sum_{\vec{\beta}, \vec{\gamma} \in H^m, \vec{\delta} \in H^r} \Phi_i(\vec{w}, \vec{\beta}, \vec{\gamma}, \vec{\delta}) \left( f_{i-1}(\vec{\beta}) \cdot f_{i-1}(\vec{\gamma}) \right), & \text{if } i \text{ is even.} \end{cases}$$

5. *(Sumcheck polynomials.) Denote $t = 2m + r$. For each $i \in [\Delta]$, let $f_{i,0}(\vec{w}, \sigma_1, \ldots, \sigma_t) \in \mathbb{F}[\vec{w}, \vec{\sigma}]$ be the polynomial*

$$f_{i,0}(\vec{w}, \vec{\sigma}) := \begin{cases} \Phi_i(\vec{w}, \sigma_1, \ldots, \sigma_t) \left( f_{i-1}(\sigma_1, \ldots, \sigma_m) + f_{i-1}(\sigma_{m+1}, \ldots, \sigma_{2m}) \right), & \text{if } i \text{ is odd} \\ \Phi_i(\vec{w}, \sigma_1, \ldots, \sigma_t) \left( f_{i-1}(\sigma_1, \ldots, \sigma_m) \cdot f_{i-1}(\sigma_{m+1}, \ldots, \sigma_{2m}) \right), & \text{if } i \text{ is even.} \end{cases}$$

[19] The function $\hat{\Phi}_i$ is defined to be zero if any of the three input strings is not a valid indexing of a gate.
[20] See Section 3 for a definition. Also, note that here, the elements of $H^m$ are assumed to be listed in the lexicographic order.
[21] See Footnote 20.

*For each $j \in [t-1]$, let $f_{i,j}(\vec{w}, \sigma_1, \ldots, \sigma_{t-j}) \in \mathbb{F}[\vec{w}, \vec{\sigma}]$ be the polynomial*

$$f_{i,j}(\vec{w}, \sigma_1, \ldots, \sigma_{t-j}) \coloneqq \begin{cases} \sum_{\sigma_{t-j+1}, \ldots, \sigma_t \in H^j} \Phi_i(\vec{w}, \sigma_1, \ldots, \sigma_t) \left(f_{i-1}(\sigma_1, \ldots, \sigma_m) + f_{i-1}(\sigma_{m+1}, \ldots, \sigma_{2m})\right), & \text{if } i \text{ is odd} \\ \sum_{\sigma_{t-j+1}, \ldots, \sigma_t \in H^j} \Phi_i(\vec{w}, \sigma_1, \ldots, \sigma_t) \left(f_{i-1}(\sigma_1, \ldots, \sigma_m) \cdot f_{i-1}(\sigma_{m+1}, \ldots, \sigma_{2m})\right), & \text{if } i \text{ is even} \end{cases}$$

*Finally, we define $f_{i,t}(\vec{w}) \coloneqq f_i(\vec{w})$.*

Towards constructing efficient polynomial decompositions, it will be useful—particularly for our targeted hitting set generator based on the Kabanets–Impagliazzo generator—to define an auxiliary notion of an enumeration of a set of given size within feasible field families (see Section 3 for a refresher on its definition).

**Definition 4.5** (enumeration of a set of prescribed size in feasible fields)**.** *Let $\mathbb{F} = \{\mathbb{F}_n\}_{n \in \mathbb{N}}$ be a feasible sequence of fields, and let $h(n) \leq |\mathbb{F}_n|$ be a given time-constructible function. We say that a family of sets $\{H_n\}$ is an* enumeration of $h(n)$ with respect to $\{\mathbb{F}_n\}$ *(or simply that $H$ is an* enumeration of $h$ in $\mathbb{F}$ *if the family is clear from context) if there is a Turing machine that takes as input $1^n$ and outputs the elements of a set $H_n \subset \mathbb{F}_n$ such that $|H_n| = h(n)$ and whose running time is bounded by $\mathrm{poly}(n, h(n))$. We call a function family $\{\phi_n : [h(n)] \to \mathbb{F}_n\}$ an* enumerator of $h(n)$ with respect to $\{\mathbb{F}_n\}$ *if $\phi_n$ is injective, its image is $H_n$, and it is computable in $\mathrm{poly}(n, h(n))$ time.*[22]

We now show that a polynomial decomposition exists for any $\log^c$-uniform family $\{C_n\}$ of arithmetic circuits (see Definition 3.11). Moreover, we show that there is a polynomial-sized P-uniform family of arithmetic networks that when given an input $\vec{\Lambda}$, outputs small arithmetic circuits for the layer polynomials as well as for the sumcheck polynomials. These small circuits ensure the efficiency of our targeted hitting set generators (defined later in Construction 6.7 and Construction 5.6). Furthermore, we record crucial downward self-reducibility properties of these polynomials, which will be essential in the reconstruction argument.

The proposition below actually constructs two polynomial decompositions, parameterized differently. The first one is parameterized with the set $H = \{0, 1\}$ and with an integer $r = \mathrm{polylog}(s)$ (where $s$ is the size of $\{C_n\}$), and the second one is parameterized with an arbitrary set $H$ and with $r = 0$. We crucially use both instantiations for our targeted hitting set generator based on the Kabanets–Impagliazzo generator (which is discussed in Section 6), whereas we only use the latter for our construction based on the hitting set generator of Guo–Kumar–Saptharishi–Solomon (which is discussed in Section 5).

**Proposition 4.6** (Polynomial decompositions for arithmetic circuits)**.** *Let $\{\mathbb{F}_n\}_{n \in \mathbb{N}}$ be a feasible sequence of fields where $\mathbb{F}_n$ has order $q(n) \leq \exp(n^{O(1)})$. Let $\{C_n\}_{n \in \mathbb{N}}$ be a $\log^c$-uniform family of arithmetic circuits correspondingly over $\{\mathbb{F}_n\}_{n \in \mathbb{N}}$ of size $s(n)$ and depth $\Delta(n)$. Let $h = h(n) \geq 2$ be a time-constructible function whose range is powers of two. Then, for every $n \in \mathbb{N}$ and every $\vec{\Lambda} \in \mathbb{F}_n^n$,*

1. **($r = \mathrm{polylog}(s)$ instantiation.)** *There is a $(h = 2, m = \lceil \log(s) \rceil, r = \log(s)^c)$-polynomial decomposition $\{f_{i,j} \mid i \in [\Delta(n)], j \in \{0, \ldots, t\}\}$ of $C_n(\vec{\Lambda})$, where $t \coloneqq 2m + r$.*
2. **($r = 0$ instantiation.)** *There is an $(h, m, 0)$-polynomial decomposition $\{f_{i,j} \mid i \in [\Delta(n)], j \in \{0, \ldots, t\}\}$ of $C_n(\vec{\Lambda})$, where $m$ is the minimal integer such that $h^m \geq s$ and $t \coloneqq 2m$.*

*where each of the foregoing decompositions satisfies the following properties.*

1. *(Faithful representation.) Let $H$ be any enumeration of $h$ in $\mathbb{F}_n$ and $\phi : [h] \to H$ be its corresponding enumerator. Then, for every $i \in [\Delta(n)]$ and $\vec{\alpha} \in H^m$ representing a gate in the $i^{th}$ layer, it holds that*[23] *$f_i(\vec{\alpha})$ is the value of the gate $\vec{\alpha}$ in $C_n(\vec{\Lambda})$, i.e., $f_i(\vec{\alpha}) = \hat{f}_i(\vec{\alpha})$.*
2. *(Layer Polynomial Computation.) It is possible to compute the layer polynomials efficiently by a uniform family of arithmetic networks as follows:*

[22] Note that the defintion of feasible field families implies that such enumerators exist.
[23] Recall from item 5 of Definition 4.4 that $f_i$ is defined to be the layer polynomial $f_{i,t}$.

(a) *(Input Layer.) There is a* P*-uniform family of arithmetic networks of size* $\mathrm{poly}(n, m, h, \log(s))$ *that takes as input* $\vec{\Lambda}$ *and outputs the description of an arithmetic circuit of size* $O(n \cdot m \cdot h \cdot \mathrm{polylog}(s(n)))$ *and degree* $n \cdot h \cdot \mathrm{polylog}(s)$ *that computes the polynomial* $f_0(\vec{w})$*.*

(b) *(Other Layers.) There is a* P*-uniform family of arithmetic networks of size* $\mathrm{poly}(s(n))$ *that takes as input* $\vec{\Lambda}$ *(as arithmetic data), and integers* $i \in [\Delta(n)]$ *and* $j \in \{0, ..., t\}$ *(as boolean data) and outputs the description of an arithmetic circuit of size* $\mathrm{poly}(s(n))$ *and degree* $h \cdot \mathrm{polylog}(s(n))$ *that computes* $f_{i,j}(\vec{w}, \vec{\sigma})$*.*

3. *(Layer downward self-reducibility.) There is a* P*-uniform arithmetic network family of size* $\mathrm{poly}(h, \log(s(n)))$ *that takes as input* $\vec{\Lambda}$ *(as arithmetic data) and an integer* $i \in [\Delta(n)]$ *(as boolean data), and outputs the description of an oracle arithmetic circuit of size and degree at most* $h \cdot \mathrm{polylog}(s(n))$ *that computes* $f_{i,0}(\vec{w}, \vec{\sigma})$*. The only oracle access that the output circuit requires is for querying* $f_{i-1}$ *twice on inputs in* $H^m$*.*

4. *(Sumcheck downward self-reducibility.) There is a* P*-uniform arithmetic network family of size* $\mathrm{poly}(h)$ *and degree* $O(1)$ *that takes as input* $\vec{\Lambda}$ *(as arithmetic data) and integers* $i \in [\Delta(n)]$ *and* $j \in \{0, ..., t\}$ *(as boolean data), and outputs the description of an oracle arithmetic circuit of size* $O(h)$ *and degree* 1 *that computes* $f_{i,j}(\vec{w}, \vec{\sigma})$*. The only oracle access that the output circuit requires is for querying* $f_{i,j-1}$ *on inputs in* $H^{m+t-j+1}$*.*

5. *(Low degree.) In the instantiation with* $r > 0$*, the maximum individual degree of* $f_{i,j}$ *for each* $(i, j) \in [\Delta] \times \{0, ..., t\}$ *is at most* $6(h-1)$*. Moreover, in either instantiation (*$r > 0$ *or* $r = 0$*), the total degree* $\deg(f_{i,j})$ *of each* $f_{i,j}$ *is bounded by* $h \cdot \mathrm{polylog}(s)$*.*

**Proof.** As $\{C_n\}_{n \in \mathbb{N}}$ is a $\log^c$-uniform family of circuits, we know that its circuit structure function $\hat{\Phi}$ (see Definition 3.11) is computable by a P-uniform family of formulas of size $(\log(s(n)))^c$. Note that the domain of $\hat{\Phi}$ is $[\Delta(n)] \times [s(n)]^3$. Let the $i$-th "slice" be denoted by $\hat{\Phi}_i(\cdot)$ (that is, $\hat{\Phi}_i(w, u, v) = \hat{\Phi}(i, w, u, v)$). In the rest of the proof, we use the shorthand $s = s(n)$, $\Delta = \Delta(n)$, and $\mathbb{F} = \mathbb{F}_n$.

We construct two instantiations of polynomial decompositions (as described in Definition 4.4): one corresponding to $r > 0$ and $H = \{0, 1\}$, and the other corresponding to $r = 0$. From the definition of the $f_i$'s in item 4 of Definition 4.4, together with Observation 4.2, in order to establish the claim about faithful representation, it suffices to find an appropriate arithmetization of the $\hat{\Phi}_i$'s. In the first case, we construct the arithmetization of an appropriate variable-extended formulation (see Definition 4.1) of $\hat{\Phi}_i$ for some $0 < r = \mathrm{polylog}(s(n))$ to be specified soon, whereas in the second case, we do so for $\hat{\Phi}_i$ itself. Thus, to complete the description of the polynomial decomposition and conclude the proof, we now specify the two arithmetizations (correspondingly in the two subclaims below), and subsequently prove our claims regarding downward self-reducibility and efficient computability (which is common to both cases).

**Subclaim 4.7** $(r > 0, H = \{0, 1\})$**.** *Let* $H = \{0, 1\}$ *and let* $r = \lfloor (\log s)^c \rfloor$*. For each* $i \in [\Delta]$*, there is a polynomial* $\Phi_i \in \mathbb{F}[z_1, \ldots, z_{3m+r}]$ *that satisfies the following:*

1. *For every* $(\vec{w}, \vec{u}, \vec{v}) \in \{0, 1\}^{3m}$ *(so that each of these vectors denotes the label of a gate in layer* $i$ *of* $C_n$*), we have that*
$$\hat{\Phi}_i(\vec{w}, \vec{u}, \vec{v}) = \sum_{\vec{y} \in \{0,1\}^r} \Phi_i(\vec{w}, \vec{u}, \vec{v}, \vec{y}).$$

2. *The maximum individual degree of* $\Phi_i$ *is* 2 *and the total degree is bounded by* $\mathrm{polylog}(s)$*.*

3. *There is an arithmetic circuit that computes* $\Phi_i$ *that has size at most* $\mathrm{polylog}(s)$*.*

*Moreover, there exists a* P*-uniform family of arithmetic networks of size* $\mathrm{polylog}(s)$ *that takes as input an integer* $i \in [\Delta]$ *and outputs the description of the foregoing arithmetic circuit.*

**Subproof.** From Definition 3.11, we have a P-uniform family of NAND-formulas of size $r' = \lfloor (\log s)^c \rfloor$ that compute $\hat{\Phi}_i$. For any given $n$, we assume that in the corresponding boolean NAND-formula, the domain of $\hat{\Phi}_i$ is $\{0, 1\}^{3m}$.

Next, we apply Lemma 4.3 to this boolean formula to obtain an arithmetic formula $\Phi'$ of size $5(\log s)^c$ over $\mathbb{F}$ that computes a variable-extended formulation of $\hat{\Phi}_i$. We define $\Phi_i$ to be the polynomial computed by this arithmetic formula. By Lemma 4.3, for every $(\vec{w},\vec{u},\vec{v}) \in \{0,1\}^{3m}$ we have $\sum_{\vec{y}\in\{0,1\}^r} \Phi_i(\vec{w},\vec{u},\vec{v},\vec{y}) = \hat{\Phi}_i(\vec{w},\vec{u},\vec{v})$, and the bounds on the individual degree, total degree, and size of $\Phi_i$ also follow immediately from the lemma's statement (note that $s_f = r' = \text{polylog}(s)$ and $m = \log(s)$).

As for the P-uniform family of networks that get $i$ and print a circuit for $\Phi_i$, the network has the descriptions of $\hat{\Phi}_1,\ldots,\hat{\Phi}_\Delta$ hard-wired. This can be done in a P-uniform manner, as each of $\hat{\Phi}_1,\ldots,\hat{\Phi}_\Delta$ can be printed in time $\text{polylog}(s)$, since the circuit family $\{C_n\}$ is $\log^c$-uniform. Given $i$, the network uses the network from Lemma 4.3 to transform the description of $\hat{\Phi}_i$ into a description of $\Phi_i$. □

**Subclaim 4.8** ($r = 0$, $H$ arbitrary)**.** *For each $i \in [\Delta]$, there is a polynomial $\Phi_i \in \mathbb{F}[z_1,\ldots,z_{3m}]$ that satisfies the following:*

1. *For every $(\vec{w},\vec{u},\vec{v}) \in H^{3m}$ (so that each of these vectors denotes the label of a gate in layer $i$ of $C_n$), we have that $\hat{\Phi}_i(\vec{w},\vec{u},\vec{v}) = \Phi_i(\vec{w},\vec{u},\vec{v})$.*
2. *The degree of $\Phi_i$ is bounded by $h \cdot \text{polylog}(s)$.*
3. *There is an arithmetic circuit that computes $\Phi_i$ that has size at most $h \cdot \text{polylog}(s)$.*

*Moreover, there exists a P-uniform family of arithmetic networks of size $h \cdot \text{polylog}(s)$ that takes as input an integer $i \in [\Delta]$ and outputs the description of the foregoing arithmetic circuit.*

**Subproof.** We think of $\hat{\Phi}_i$ as a function $\{0,1\}^{3\log s} \to \{0,1\}$, and note that it is computable by an arithmetic formula (over $\{0,1\}$), whose structure mimics the one of the original Boolean formula (i.e., each gate $\hat{w}$ in the original formula computes the NAND of two sub-formulas $\hat{u}$ and $\hat{v}$, and thus $\hat{w}$ can be replaced by the expression $w = 1 - u\cdot v$). The same arithmetic formula (now interpreted to be over $\mathbb{F}$) can be used to compute a polynomial $\Phi'_i : \mathbb{F}^{3\log s} \to \mathbb{F}$ of size at most $3 \cdot \text{size}(\hat{\Phi}) = O((\log s)^c)$ and degree bounded by $O((\log s)^c)$. The latter follows from the simple observation that the degree of a polynomial computed by an arithmetic formula is always bounded by its size[24]. We conclude that $\Phi'_i$ is a polynomial of degree $\text{polylog}(s(n))$ over $\mathbb{F}$ computable by an arithmetic formula of size $\text{polylog}(s(n))$ that agrees with $\hat{\Phi}_i$ on $\{0,1\}^{3\log s}$.

Next, let $\ell = \log h$, and for every $j \in [\ell]$ consider the function $\hat{\pi}_j : H \to \{0,1\}$ such that $\hat{\pi}_j(a)$ is the $j$-th bit in the binary representation of the integer $\phi(a)$, where $\phi^{-1}$ is an enumerator for $H$ (see Definition 4.5). Note that there is a polynomial $\pi_j : \mathbb{F} \to \mathbb{F}$ of degree at most $h$ that agrees with $\hat{\pi}_j$ on $H$. Finally, let $\Phi_i : \mathbb{F}^{3m} \to \mathbb{F}$ be the polynomial

$$\Phi_i(z_1,\ldots,z_{3m}) \coloneqq \Phi'_i(\pi_1(z_1),\ldots,\pi_\ell(z_1),\ldots,\pi_1(z_{3m}),\ldots,\pi_\ell(z_{3m})).$$

By definition, when $\Phi_i$ is given input $(\vec{w},\vec{u},\vec{v}) \in H^{3m}$, the $\pi_j$'s project each of the three inputs to a bit-string that is the index of a gate, and the arithmetic formula $\Phi'_i$ computes $\hat{\Phi}_i$ on the corresponding gates. Also, the size and degree of $\Phi_i$ are bounded by $h \cdot \text{size}(\Phi') = h \cdot (\log(s(n)))^c = h \cdot \text{polylog}(s(n))$. Finally, to see the "moreover" part of the claim, we note that the arithmetic formulas for $\Phi_i$ are defined by composing $\Phi'_i$ (whose formula descriptions are already output by the P-uniform family of arithmetic networks in Lemma 4.3) and $\pi_j$'s (which have small $O(h)$-sized P-uniform formulas as $H$ can be enumerated efficiently). Applying Proposition 3.25 then yields the result. □

To specify a polynomial decomposition, it suffices to specify $H, m, r, \Phi$, and $f_0$. Thus, given Subclaim 4.7 and Subclaim 4.8, all that is left in order to fully specify the two polynomial decompositions is $f_0$.

Let $\vec{\Lambda}\vec{K} \in \mathbb{F}^s$ be the input layer in $C_n(\vec{\Lambda})$. Since $\{C_n\}_{n\in\mathbb{N}}$ is $\log^c$-uniform, there is a P-uniform family $\{\Psi'_n : \mathbb{F}_n^n \times \mathbb{F}_n^{\log(s)} \to \mathbb{F}_n\}$ of arithmetic circuits such that $\Psi'_n(\vec{\Lambda}, w) = (\vec{\Lambda}\vec{K})_w$ for all $w \in \{0,1\}^{\log(s)}$. Consider an arithmetic circuit $C_{\vec{\Lambda}} : \mathbb{F}^m \to \mathbb{F}$ that has $\vec{\Lambda}$ hard-wired, gets input $\vec{w} = (w_1,\ldots,w_m) \in \mathbb{F}^m$, computes $z = (\pi_1(w_1),\ldots,\pi_\ell(w_1),\ldots,\pi_1(w_m),\ldots,\pi_\ell(w_m)) \in \{0,1\}^{\ell\cdot m = \log(s)}$ (where $\ell = \log(h)$ and the $\pi_i$'s are as in the

[24] Here is a simple proof by structural induction: the statement is clearly true for leaves, regardless of whether the leaf is labelled by a constant or a variable. Suppose $u$ is an addition gate with children $v$ and $w$; then by the inductive hypothesis for $v$ and $w$, $\deg(u) \le \max(\deg(v),\deg(w)) \le \deg(v)+\deg(w) \le \text{size}(v)+\text{size}(w) \le \text{size}(u)$. If $u$ is a multiplication gate, then $\deg(u) = \deg(v)+\deg(w) \le \text{size}(v)+\text{size}(w) \le \text{size}(u)$.

proof of Subclaim 4.8) and outputs $\Psi'_n(\vec{\Lambda}, z)$. The circuit $C_{\vec{\Lambda}}$ is of size $h \cdot m \cdot n \cdot \text{polylog}(s)$ and of degree $n \cdot h \cdot \text{polylog}(s)$, and for every $\vec{w} \in H^m$ it outputs the $w^{th}$ entry of $\vec{\Lambda}\vec{K}$. Also, there is a P-uniform arithmetic network that gets $\vec{\Lambda}$ and outputs the description of $C_{\vec{\Lambda}}$ in size $\text{poly}(n, m, h, \log(s))$. We thus define $f_0$ to be the polynomial that $C_{\vec{\Lambda}}$ computes.

**Layer Polynomial Computation:** We already proved the claim about $f_0$ above, so let us focus on other layers. Given $\vec{\Lambda}$ and $i \in [\Delta]$, consider the following arithmetic circuit for $f_i(\vec{w})$.

Consider all pairs of gates in the $(i-1)$-th layer of $C_n$ evaluated on the input $\vec{\Lambda}$ (represented by a pair of index vectors $(\vec{\beta}, \vec{\gamma})$), apply the suitable arithmetic operation (addition if $i$ is odd, and multiplication if $i$ is even), multiply the result with $\sum_{\vec{\delta} \in H^r} \Phi_i(\vec{w}, \vec{\beta}, \vec{\gamma}, \vec{\delta})$ if we are working with the $r > 0$ instantiation (whose circuit is obtained from Subclaim 4.7 above) or $\Phi_i(\vec{w}, \vec{\beta}, \vec{\gamma})$ if we are working with the $r = 0$ instantiation (whose circuit is obtained from Subclaim 4.8 above), and finally add the result for all such pairs of gates in the $(i-1)$-th layer of $C_n$ evaluated on the input $\vec{\Lambda}$ (using Proposition 3.24).

As the circuit size of $C_n$ is bounded by $s(n)$, the size of the sub-circuit that computes the evaluation of any gate in the $(i-1)$-th layer on input $\vec{\Lambda}$ is also bounded by $s(n)$. As the computation described above can also be performed by a sub-circuit of $\text{poly}(s(n))$ size, it follows that the size of the circuit for $f_i(\vec{w})$ constructed above is also bounded by $\text{poly}(s(n))$.

Moreover, given as input $j \in \{0, ..., t\}$, we can construct an arithmetic circuit that computes $f_{i,j}$ by simply using the formula defining $f_{i,j}(\vec{w}, \vec{\sigma})$ in item 5 of Definition 4.4: enumerate over the $h^j$ different values for $\sigma_{t-j+1}, \ldots, \sigma_t$ and use the foregoing circuits for $\Phi_i$ and $f_{i-1}$. As $h^j \leq h^m = O(s)$, the size of the overall circuit for $f_{i,j}$ is still $\text{poly}(s(n))$.

Finally, note that the implementation of the above using a P-uniform family of arithmetic networks follows from a suitably repeated application of Proposition 3.23, Proposition 3.24, and Proposition 3.25.

**Layer Downward Self-reducibility:** Given as input $\vec{\Lambda}$, an integer $i \in [\Delta(n)]$, and oracle access to $f_{i-1}$, in order to compute (the description of) an oracle arithmetic circuit for $f_{i,0}$, the arithmetic network does the following: It takes (the description of) the circuit for $\Phi_i$ as constructed above (accordingly in Subclaim 4.7 if $r > 0$ and Subclaim 4.8 if $r = 0$), and computes the description of a circuit that is obtained by performing the two oracle calls to $f_{i-1}$ (as described in item 4 of Definition 4.4) and the appropriate arithmetic operation on the two values (depending on the parity of $i$), and multiplying this result with the circuit for $\Phi_i$. The correctness of this circuit computing the desired polynomial $f_{i,0}$ again follows from item 5 of Definition 4.4. The size and degree bounds follow from Subclaim 4.7 and Subclaim 4.8. The claim about the the polynomial identity $f_{i,t}(\vec{w}) = f_i(\vec{w})$ is by item 5 of Definition 4.4.

**Sumcheck downward self-reducibility:** From item 5 of Definition 4.4, we have

$$f_{i,j}(\vec{\alpha}, \beta_1, \ldots, \beta_{t-j}) = \sum_{\beta_{t-j+1} \in H} f_{i,j-1}(\vec{\alpha}, \beta_1, \ldots, \beta_{t-j}, \beta_{t-j+1}),$$

and this expression can be computed by an oracle formula (with access to $f_{i,j-1}$) of size $O(h)$ and degree 1. Therefore, given an integer $j \in \{0, ..., t\}$ and oracle access to $f_{i,j-1}$, the arithmetic network computes its description simply using this equation (which uses the oracle to the circuit for $f_{i,j}$ $h$ times). The claims about the P-uniformity and the size and degree of the arithmetic network that computes this description are straightforward in this case.

**Low degree:** The maximum individual degree of $f_i$ for each $i \in [\Delta]$ is at most that of $\Phi_i$ by item 4 of Definition 4.4. Therefore, it satisfies

$$\text{i-deg}(f_i) \leq \begin{cases} 2(h-1) & \text{if } r > 0 \\ h \cdot \text{polylog}(s) & \text{if } r = 0. \end{cases}$$

In the $r > 0$ instantiation, for each $(i, j) \in [\Delta] \times \{0, ..., t\}$, we have that

$$\text{i-deg}(f_{i,j}) \leq \text{i-deg}(\Phi_i) + 2\text{i-deg}(f_{i-1}) \leq \text{i-deg}(\Phi_i) + 2\text{i-deg}(\Phi_{i-1}) \leq 6(h-1) \ ,$$

from Subclaim 4.7. Moreover, in either instantiation, the total degree $\deg(f_{i,j})$ of each $f_{i,j}$ is bounded by $h \cdot \text{polylog}(s)$. □

# 5 A targeted version of the Guo–Kumar–Saptharishi–Solomon generator

Our goal in this section is to construct an arithmetic reconstructive targeted hitting-set generator that works over fields of characteristic zero, or over sequences of finite fields of large characteristic. The generator is based on a family $\{C_n\}$ of circuits computing a hard problem, and is computable by a family of arithmetic networks. Whenever a distinguisher represented by a string $\vec{\Lambda}$ breaks the generator, another family $\{R_n\}$ of arithmetic networks computes the hard problem at input $\vec{\Lambda}$.

**Theorem 5.1** (algebraic GKSS-based targeted hitting-set generator)**.** *Let $d, \bar{n} : \mathbb{N} \to \mathbb{N}$ be time-constructible functions where $d$ is polynomially-bounded, and let $\varepsilon > 0$ be a constant. Let $\{C_n\}_{n\in\mathbb{N}}$ be a $\log^c$-uniform family of arithmetic circuits of size $s(n)$ and degree at most $s(n)$ having $n$ output gates, defined over a field family $\{\mathbb{F}_n\}_{n\in\mathbb{N}}$ where either (i) each $\mathbb{F}_n$ is a fixed field $\mathbb{F}$ of characteristic zero, or (ii) for each $n \in \mathbb{N}$, $\text{char}(\mathbb{F}_n) > s^{O(\varepsilon/c)} \cdot \text{polylog}(s) > \bar{n}$. Then, there are two $\mathsf{P}$-uniform families of arithmetic networks $\{G_n\}$ and $\{R_n\}$ satisfying the following.*

1. **(Generator.)** *Networks in $\{G_n\}$ are of size $\text{poly}(s)$. On input $\vec{\Lambda} \in \mathbb{F}_n^n$, the network $G_n$ outputs the description of an arithmetic circuit of size $\text{poly}(s, \bar{n})$ that computes a polynomial map*
$$\mathcal{H}_{\vec{\Lambda}} \colon \mathbb{F}_n^{O(1/\varepsilon)} \to \mathbb{F}_n^{\bar{n}}$$
*of degree $\deg(\mathcal{H}_{\vec{\Lambda}}) \leq n \cdot s^{\varepsilon}$.*
2. **(Reconstruction.)** *Networks in $\{R_n\}$ are networks with $\mathsf{PIT}$ gates and are of size $\text{poly}(n, \bar{n}^{1/\varepsilon}, s^{\varepsilon}, d(\bar{n}))$ and degree $\text{poly}(d(\bar{n}), s^{\varepsilon}, \bar{n})^{\text{polylog}(s)}$. When $\vec{\Lambda} \in \mathbb{F}_n^n$ is a universal encoding of a degree-$d(\bar{n})$ arithmetic circuit with $\bar{n}$ input gates such that $\vec{\Lambda} \circ \mathcal{H}_{\vec{\Lambda}} \equiv 0$, then $R_n$ computes $C_n(\vec{\Lambda})$ on input $\vec{\Lambda}$.*

We now prove Theorem 5.1 based on claims that will be established throughout the section. We include the proof here (i.e., in advance) to assist in verification. The definitions and analyses of the generator networks $\{G_n\}$ and reconstruction networks $\{R_n\}$ appear in Section 5.1 and Section 5.2, respectively.

**Proof of Theorem 5.1.** Let $c' > 1$ be a sufficiently large universal constant. We use Construction 5.6 with the circuit family $\{C_n\}$ and with parameters $h = 2^{\lceil (\varepsilon/c)\log(s)\rceil}$ and $m = c'/\varepsilon$ (so that indeed $h^m \geq s'$ is satisfied) and with output length $\bar{n}$. By Lemma 5.8, the degree of $\mathcal{H}_{\vec{\Lambda}}$ is indeed at most $n \cdot \tilde{O}(s^{\varepsilon/c}) < n \cdot s^{\varepsilon}$, for every $\vec{\Lambda} \in \mathbb{F}_n^n$. The reconstruction in Proposition 5.10, instantiated with degree bound $d(\bar{n})$, is a network of size at most $\text{poly}(n, \bar{n}^{1/\varepsilon}, s^{\varepsilon}, d(\bar{n}))$ and degree at most $\text{poly}(d(\bar{n}), s^{\varepsilon}, \bar{n})^{\text{polylog}(s)}$. □

## 5.1 The targeted hitting set generator

Loosely speaking, the targeted generator will get input $\vec{\Lambda}$ and apply the generator of Guo, Kumar, Saptharishi, and Solomon [GKSS22] to each of the polynomials in the polynomial decomposition of $C_n(\vec{\Lambda})$. We first recall their construction, and then describe our targeted generator.

**Construction 5.2** ([GKSS22])**.** *Let $n \in \mathbb{N}$ and let $\mathbb{F}$ be a field with $\text{char}(\mathbb{F}) = 0$ or $\text{char}(\mathbb{F}) \geq n$. Let $P(z_1, \ldots, z_\ell) \in \mathbb{F}[z_1, \ldots, z_\ell]$ be an $\ell$-variate polynomial. For $i \in [n]$, define $\Delta_{P,i}$ to be the $2\ell$-variate polynomial*
$$\Delta_{P,i}(z_1, \ldots, z_\ell, y_1, \ldots, y_\ell) \coloneqq \sum_{\substack{\vec{e}\in\mathbb{Z}^{\ell}_{\geq 0} \\ |\vec{e}|=i}} \frac{\vec{y}^{\vec{e}}}{\vec{e}!} \cdot \frac{\partial^{\vec{e}}}{\partial \vec{z}^{\vec{e}}}(P(z_1, \ldots, z_\ell)).$$
*Equivalently, the polynomial $\Delta_{P,i}$ is the component of $P(y_1 + z_1, \ldots, y_\ell + z_\ell)$ that is homogeneous of degree $i$ with respect to the $\vec{y}$ variables. The $n$-output $\mathsf{GKSS}$ generator with respect to $P$ is the polynomial map $\mathcal{H}_{GKSS,P,n} : \mathbb{F}^{2\ell} \to \mathbb{F}^n$ given by*
$$\mathcal{H}_{GKSS,P,n}(z_1, \ldots, z_\ell, y_1, \ldots, y_\ell) = (\Delta_{P,0}(\vec{z}, \vec{y}), \Delta_{P,1}(\vec{z}, \vec{y}), \ldots, \Delta_{P,n-1}(\vec{z}, \vec{y})).$$

**Preliminary observations.** We will later use some basic observations about the generator of Construction 5.2, and a well-known fact, which we state here for convenience.

**Observation 5.3** ([GKSS22, Observation 2.11])**.** Let $P \in \mathbb{F}[z_1, \ldots, z_\ell]$ and let $\vec{a} \in \mathbb{F}^\ell$. Letting $P'(\vec{z}) \coloneqq P(\vec{z} + \vec{a})$, we have

$$\Delta_{P',i}(\vec{z}, \vec{y}) = \Delta_{P,i}(\vec{z} + \vec{a}, \vec{y})$$

for all $i \in \mathbb{N}$. As a consequence, we have

$$\mathcal{H}_{\mathrm{GKSS},P',n}(\vec{z}, \vec{y}) = \mathcal{H}_{\mathrm{GKSS},P,n}(\vec{z} + \vec{a}, \vec{y})$$

for all $n \in \mathbb{N}$.

**Lemma 5.4.** *Let $P \in \mathbb{F}[z_1, \ldots, z_\ell]$. For $i \in \mathbb{N}$, let $P_i$ denote the degree-$i$ homogeneous component of $P$. Then with $\Delta_{P,i}(\vec{z}, \vec{y})$ as in Construction 5.2, we have $\Delta_{P,i}(\vec{0}, \vec{y}) = P_i(\vec{y})$. As a consequence, we have $\mathcal{H}_{GKSS,P,n}(\vec{0}, \vec{y}) = (P_0(\vec{y}), \ldots, P_{n-1}(\vec{y}))$.*

**Proof.** Recall that $\Delta_{P,i}(\vec{z}, \vec{y})$ is, by definition, the component of $P(\vec{z} + \vec{y})$ that is homogeneous of degree $i$ with respect to the $y$ variables. Applying the evaluation $\vec{z} \mapsto \vec{0}$, we see that $\Delta_{P,i}(\vec{0}, \vec{y})$ is the degree-$i$ homogeneous of component of $P(\vec{0} + \vec{y}) = P(\vec{y})$, which is precisely $P_i(\vec{y})$. □

**Fact 5.5** (Euler's formula for differentiation of homogeneous polynomials)**.** *If $A \in \mathbb{F}[x_1, \ldots, x_\ell]$ is a homogeneous polynomial of degree $t$, then*

$$\sum_{i=1}^{\ell} x_i \cdot \frac{\partial}{\partial x_i}(A(\vec{x})) = t \cdot A(\vec{x}).$$

**The targeted generator.** We now state the construction of our targeted hitting set generator that is based on Construction 5.2. The generator networks $\{G_n\}_{n\in\mathbb{N}}$ of Theorem 5.1 will output arithmetic circuits that compute this targeted hitting set generator.

**Construction 5.6** (GKSS-based targeted HSG)**.** *Let $\{C_n\}_{n\in\mathbb{N}}$ be a $\log^c$-uniform family of arithmetic circuits of size $s(n)$ and degree at most $s(n)$, defined over a field family $\{\mathbb{F}_n\}_{n\in\mathbb{N}}$ satisfying certain conditions specified in the next paragraph. Let $\{C'_n\}$ be the $\log^{c+O(1)}$-uniform family of arithmetic circuits of size $s'(n) = \mathrm{poly}(s(n))$ and depth $\Delta(n) = O(\log(s(n))^2)$ obtained by applying Proposition A.23 to $\{C_n\}$.*

**Parameters.** *Let $h, m, \bar{n} : \mathbb{N} \to \mathbb{N}$ be time-constructible functions such that $h(n)$ is a power of 2, $m(n)$ is the minimal integer for which $h(n)^{m(n)} \geq s'(n)$. The field family $\{\mathbb{F}_n\}_{n\in\mathbb{N}}$ is such that either (i) each $\mathbb{F}_n$ is a fixed field $\mathbb{F}$ of characteristic zero, or (ii) for each $n \in \mathbb{N}$, $\mathrm{char}(\mathbb{F}_n) > \max(\bar{n}, h(n) \cdot \mathrm{polylog}(s(n)))$.*

**Generator.** *Given $\vec{\Lambda} \in \mathbb{F}_n^n$, let $\{f_{i,j}\}$ be the $(h, m, 0)$-polynomial decomposition of $C'_n$ on input $\vec{\Lambda} \in \mathbb{F}_n^n$ obtained via Proposition 4.6 (i.e., the $r = 0$ instantiation). We view all polynomials as having $\ell \coloneqq 3m$ variables, padding them with extra variables as necessary. The $\bar{n}$-output GKSS-based targeted HSG for $\vec{\Lambda}$ is the polynomial map $\mathcal{H}_{\vec{\Lambda}}^{(h,m,\bar{n})} : \mathbb{F}_n^{2\ell+2} \to \mathbb{F}_n^{\bar{n}}$ given by*

$$\mathcal{H}_{\vec{\Lambda}}^{(h,m,\bar{n})}(w_1, w_2, \vec{z}, \vec{y}) \coloneqq \sum_{i=1}^{\Delta(n)} \sum_{j=0}^{2m} L_{i,j}(w_1, w_2) \cdot \mathcal{H}_{GKSS,f_{i,j},\bar{n}}(\vec{z}, \vec{y}), \tag{5.1}$$

*where $L_{i,j}(w_1, w_2)$ is the Lagrange interpolation polynomial that satisfies*[25]

$$L_{i,j}(w_1, w_2) = \begin{cases} 1 & \text{if } w_1 = i \text{ and } w_2 = j, \\ 0 & \text{if } (w_1, w_2) \in ([\Delta(n)] \times \{0, ..., 2m\}) \setminus \{(i, j)\}. \end{cases}$$

[25] Note that the parameter settings imply that both $m$ and $\Delta(n)$ are bounded by $\mathrm{polylog}(s(n))$, so the assumption on the field characteristic implies that $[\Delta(n)]$ and $[2m]$ can be regarded as subsets of $\mathbb{F}_n$.

**Remark 5.7.** *We remark here that the characteristic assumption* $\mathrm{char}(\mathbb{F}_n) > h(n) \cdot \mathrm{polylog}(s(n))$ *above is made primarily because it allows us to assume that* $\mathrm{char}(\mathbb{F}_n) > \deg(f_{i,j})$ *(see item 5 of Proposition 4.6) for each pair* $(i,j)$*. In particular, this allows us to apply Fact 5.5 to compute lower-order derivatives of* $f_{i,j}$ *from its higher-order derivatives, an operation that is used crucially later in the reconstruction procedure. For a more precise explanation of the reasons behind this assumption, see Remark 5.17.*

We now show that there is a $\mathsf{P}$-uniform family of arithmetic networks that take $\vec{\Lambda} \in \mathbb{F}_n^n$ as input and output a description of an arithmetic circuit that computes the corresponding GKSS-based targeted hitting set generator $\mathcal{H}_\Lambda$. This corresponds to item 1 in Theorem 5.1.

**Lemma 5.8** (The complexity of the targeted generator)**.** *Let* $\{C_n\}$*,* $h$*,* $m$*, and* $\bar{n}$ *be as in the statement of Construction 5.6. Then, the following statements hold.*

1. *There is a* $\mathsf{P}$*-uniform family of arithmetic networks* $\{G_n\}_{n\in\mathbb{N}}$ *where the network* $G_n$ *has size* $\mathrm{poly}(s(n))$ *and on input* $\vec{\Lambda} \in \mathbb{F}_n^n$ *outputs the description of an arithmetic circuit that computes* $\mathcal{H}_{\vec{\Lambda}}^{(h,m,\bar{n})}$*, the* $\bar{n}$*-output GKSS-based targeted hitting set generator for* $\vec{\Lambda}$*.*
2. *The degree of* $\mathcal{H}_{\vec{\Lambda}}^{(h,m,\bar{n})}$ *is at most* $n \cdot h(n) \cdot \mathrm{polylog}(s(n))$*.*

**Proof.** We stress that there are three algorithms involved in the claim. Our goal is to construct a Turing machine $M$ that on input $1^n$ prints an arithmetic network $N$. The arithmetic network $N$ receives as input $\vec{\Lambda} \in \mathbb{F}_n^n$ and computes the description of an arithmetic circuit $H_{\vec{\Lambda}}$ that computes the generator $\mathcal{H}_{\vec{\Lambda}}^{(h,m,\bar{n})}$. We note that we have the basic chain of inequalities $s \geq n \geq \bar{n}$; the former is because $s$ is the size of an $n$-input circuit, and the latter is because $n$ is the length of the encoding of a circuit computing an $\bar{n}$-variate polynomial.

We first describe the arithmetic circuit $H_{\vec{\Lambda}}$. We then explain how $N$ computes the description of $H_{\vec{\Lambda}}$ given $\vec{\Lambda}$ as input. Finally, we describe how $M$ prints a description of $N$ when given input $1^n$.

**The circuit** $H_{\vec{\Lambda}}$**.** We build the circuit $H_{\vec{\Lambda}}$ for $\mathcal{H}_{\vec{\Lambda}}^{(h,m,\bar{n})}$ by implementing Equation (5.1). To do this, we need small circuits for both the Lagrange interpolation polynomials $L_{i,j}$ and the generators $\mathcal{H}_{\mathrm{GKSS},f_{i,j},\bar{n}}$.

The Lagrange interpolation polynomial $L_{i,j}(w_1, w_2)$ can be written explicitly as

$$L_{i,j}(w_1, w_2) \coloneqq \left( \prod_{k\in[\Delta]} \frac{w_1 - k}{i - k} \right) \cdot \left( \prod_{\ell\in\{0,1,\ldots,2m\}} \frac{w_2 - \ell}{j - \ell} \right).$$

Note that from the given characteristic assumption $\mathrm{char}(\mathbb{F}_n) \geq h \cdot \mathrm{polylog}(s)$, it follows that $[\Delta], \{0,\ldots,2m\} \subseteq \mathbb{F}_n$ as we have $\mathrm{char}(\mathbb{F}_n) \geq \max\{2m, \Delta\}$ because $\Delta$ is assumed to be bounded by $\mathrm{polylog}(s)$. It is clear from this expression that $\deg(L_{i,j}) \leq O(\Delta m)$ and that there is an arithmetic formula of size $O(\Delta m) \leq \mathrm{polylog}(s)$ that computes $L_{i,j}(w_1, w_2)$.

To compute the generators $\mathcal{H}_{\mathrm{GKSS},f_{i,j},\bar{n}}$, we need arithmetic circuits that compute the layer polynomials $f_{i,j}$. By item 2 of Proposition 4.6, there are arithmetic circuits of size $\mathrm{poly}(s)$ and degree at most $n \cdot h \cdot \mathrm{polylog}(s)$ that compute the polynomials $f_0$ and $\{f_{i,j} : i \in [\Delta], j \in \{0,\ldots,2m\}\}$. Recall that the outputs of the generator $\mathcal{H}_{\mathrm{GKSS},f_{i,j},\bar{n}}$ correspond to the components of $f_{i,j}(\vec{z}+\vec{y})$ that are homogeneous with respect to the $\vec{y}$ variables and are of degree less than $\bar{n}$. Applying Lemma A.6 to extract the $\vec{y}$-homogeneous components of $f_{i,j}(\vec{z}+\vec{y})$ we obtain a multi-output arithmetic circuit of size $O(s\bar{n}^2)$ that computes $\mathcal{H}_{\mathrm{GKSS},f_{i,j},\bar{n}}$.

Combining the Lagrange interpolation polynomials $L_{i,j}$ with the generators $\mathcal{H}_{\mathrm{GKSS},f_{i,j},\bar{n}}$ can be done using additional $O(\Delta m) \leq \mathrm{polylog}(s)$ gates. Overall, we obtain a circuit $H_{\vec{\Lambda}}$ of size $\mathrm{poly}(s, \bar{n}) \leq \mathrm{poly}(s)$ and degree $n \cdot h \cdot \mathrm{polylog}(s)$ that computes $\mathcal{H}_{\vec{\Lambda}}^{(h,m,\bar{n})}$.

**The network** $N$**.** Recall that $N$ is an arithmetic network that receives $\vec{\Lambda} \in \mathbb{F}_n^n$ as input and outputs a description of the circuit $H_{\vec{\Lambda}}$. The outer summation in Equation (5.1) and the Lagrange interpolation polynomials $L_{i,j}$ do not depend on the value of $\vec{\Lambda}$, so the descriptions of these components can be hard-wired into the network $N$. It remains to describe an arithmetic network that computes descriptions of circuits for the generators $\mathcal{H}_{\mathrm{GKSS},f_{i,j},\bar{n}}$.

To do this, we invoke item 2 of Proposition 4.6, which provides us with a P-uniform family of arithmetic networks of size $\mathrm{poly}(s)$ that take as input $\vec{\Lambda}$ and print descriptions of arithmetic circuits of size $\mathrm{poly}(s)$ and degree at most $n \cdot h \cdot \mathrm{polylog}(s)$ for the polynomials $f_{i,j}$. From a description of a circuit that computes $f_{i,j}(\vec{y})$, we can easily obtain a description of a circuit that computes $f_{i,j}(\vec{z}+\vec{y})$. We then apply Lemma A.6 to the circuit for $f_{i,j}(\vec{z}+\vec{y})$, which yields the description of circuits that compute the components of $f_{i,j}(\vec{z}+\vec{y})$ that are homogeneous of degree less than $\bar{n}$ with respect to the $\vec{y}$-variables, which are precisely the outputs of the generator $\mathcal{H}_{\mathrm{GKSS},f_{i,j},\bar{n}}$.

Thus, for each $(i,j) \in [\Delta] \times \{0,\ldots,2m\}$, we have an arithmetic network of size $\mathrm{poly}(s)$ that takes as input $\vec{\Lambda}$ and outputs a description of an arithmetic circuit for the component generator $\mathcal{H}_{\mathrm{GKSS},f_{i,j},\bar{n}}$.

**The Turing machine $M$.** Finally, we describe a polynomial-time Turing machine $M$ that takes $1^n$ as input and outputs a description of the arithmetic network $N$. The network $N$ has both the outer summation in Equation (5.1) and the Lagrange interpolation polynomials hardwired as part of the network. These components can be computed in polynomial time by $M$.

The part of $N$ that takes as input $\vec{\Lambda}$ and produces circuits for the component generators $\mathcal{H}_{\vec{\Lambda}}^{(h,m,\bar{n})}$ is likewise computable in polynomial time: the reason is that Item 2 of Proposition 4.6 provides a polynomial-time algorithm that takes $1^n$ as input and computes the description of a network that maps $\vec{\Lambda}$ to descriptions of circuits for the layer polynomials $f_{i,j}$. The subsequent homogenization of these circuits can be done by a P-uniform arithmetic network using Lemma A.6. □

## 5.2 Reconstruction procedure

Before we describe the reconstruction procedure for our GKSS-based targeted HSG, we need the following definition.

**Definition 5.9** (Interpolating Sets, [GKSS22])**.** *Let $\mathbb{F}$ be a field and let $\mathcal{P}(\ell,d)$ denote the set of all $\ell$-variate degree-$d$ polynomials over $\mathbb{F}$. Let $M_{\ell,d} := \binom{\ell+d}{d}$ be the number of $\ell$-variate monomials of total degree at most $d$. We say that a set of points $\{\vec{a}_1,\ldots,\vec{a}_r\} \subseteq \mathbb{F}^\ell$ is an* interpolating set for $\mathcal{P}(\ell,d)$ *if the set of vectors*

$$\left\{(\vec{a}_i^{\vec{e}} : \vec{e} \in \mathbb{Z}^\ell_{\geq 0}, |\vec{e}| \leq d) : i \in [r]\right\} \subseteq \mathbb{F}^{M_{\ell,d}}$$

*spans $\mathbb{F}^{M_{\ell,d}}$. In other words, for every $\vec{e} \in \mathbb{Z}^\ell_{\geq 0}$ satisfying $|\vec{e}| \leq d$, there are field constants $\beta_1,\ldots,\beta_r \in \mathbb{F}$ such that for all $P(\vec{y}) \in \mathcal{P}(\ell,d)$, we have*

$$\mathrm{coeff}_{\vec{y}^{\vec{e}}}(P(\vec{y})) = \sum_{i=1}^{r} \beta_i \cdot P(\vec{a}_i).$$

We now describe the reconstruction procedure for our targeted hitting set generator based on the GKSS generator. This corresponds to item 2 in Theorem 5.1.

**Proposition 5.10.** *Let $\bar{n} : \mathbb{N} \to \mathbb{N}$ be a time-constructible function and $\{C_n\}_{n\in\mathbb{N}}$ be a $\log^c$-uniform family of $n$-output arithmetic circuits of size $s(n)$ and degree at most $s(n)$, defined over a field family $\{\mathbb{F}_n\}_{n\in\mathbb{N}}$ where either (i) each $\mathbb{F}_n$ is a fixed field $\mathbb{F}$ of characteristic zero, or (ii) for each $n \in \mathbb{N}$, $\mathrm{char}(\mathbb{F}_n) > h(n) \cdot \mathrm{polylog}(s)$ and for the same poly-logarithmically bounded function, we also have $h(n) \cdot \mathrm{polylog}(s) > \bar{n}(n)$. Let $d : \mathbb{N} \to \mathbb{N}$ be a polynomially-bounded time-constructible function. Then there is a P-uniform family of arithmetic networks with* PIT *gates $R = \{R_n\}_{n\in\mathbb{N}}$ that satisfies the following.*

1. *$R_n$ receives as input a vector $\vec{\Lambda} \in \mathbb{F}_n^n$, where the vector $\vec{\Lambda}$ is promised to be the universal encoding of an arithmetic circuit $D$ that computes a nonzero $\bar{n}$-variate polynomial of degree at most $d = d(\bar{n})$ that satisfies $D \circ \mathcal{H}_{\vec{\Lambda}}^{(h,m,\bar{n})} = 0$, where $\mathcal{H}_{\vec{\Lambda}}^{(h,m,\bar{n})}$ is the generator described in Construction 5.6.*
2. *The size[26] of the network $R_n$ is bounded by $\mathrm{poly}(n,d,h,\log s,\bar{n}^m)$.*

[26] In the statement of this proposition, the notation $\mathrm{poly}(T)$ indicates a bound of $T^{cK}$, where $K$ is a universal constant and $c$ is the given constant that parameterizes the uniformity of $\{C_n\}$.

3. *The degree of* $R_n$ *is bounded by* $\mathrm{poly}(d, h, \bar{n}, \log s)^{\mathrm{polylog}(s)}$.

4. $R_n$ *computes the vector* $C_n(\vec{\Lambda}) \in \mathbb{F}_n^n$ *at input* $\vec{\Lambda} \in \mathbb{F}_n^n$.

The rest of this subsection is devoted to the proof of Proposition 5.10, which follows a similar structure to that of Proposition 6.11. Recall that the generator $\mathcal{H}_{\vec{\Lambda}}^{(h,m,\bar{n})}$, defined in Construction 5.6, is obtained by instantiating copies of the GKSS generator (Construction 5.2) with polynomials coming from the polynomial decomposition of a circuit family $\{C'_n\}$ obtained by applying depth reduction to the circuit family $\{C_n\}$. On input $\vec{\Lambda}$, the network $R_n$ will output the value of $C'_n(\vec{\Lambda})$, which equals $C_n(\vec{\Lambda})$ by correctness of the depth reduction. The depth reduction ensures that the size of $C'_n$ is bounded by $\mathrm{poly}(s)$ and the depth of $C'_n$ is bounded by $\Delta \leq O(\log^2 s)$. For notational convenience, we will drop the prime symbol in $C'_n$ and instead refer to these circuits as $C_n$.

Before we describe the construction formally, we provide a brief sketch of its various components. On input $\vec{\Lambda}$, the network efficiently constructs a description of a circuit computing the polynomial $f_\Delta = f_{\Delta,2m}$. As this polynomial faithfully represents the output gate values of $C_n(\vec{\Lambda})$ (see item 1 of Proposition 4.6), the network eventually computes $C_n(\vec{\Lambda})$ by simply evaluating $f_\Delta$ over the first $n$ many vectors (in lexicographical order) in $H^m$. To construct the description of a circuit that computes the polynomial $f_\Delta$, the network $R_n$ iteratively constructs a small arithmetic circuit that computes $f_i$ for $i = 0, 1, \ldots, \Delta$. The circuit for $f_i$ is obtained by using query access to a circuit for $f_{i-1}$ to iteratively construct circuits for $f_{i,j-1}$ for $j = 1, \ldots, 2m$.

We will now describe the arithmetic network $R_n$, accounting for both the complexity of implementing each iteration (i.e., the complexity of the uniform arithmetic network) and the complexity of the arithmetic circuits whose descriptions are produced by each iteration. Since $\vec{\Lambda}$ is the universal encoding of $D$, it follows from Theorem 3.28 that $\mathrm{size}(D) \leq n$. To begin, $R_n$ employs the $\mathsf{P}$-uniform family of arithmetic networks from Lemma A.2 to convert $\vec{\Lambda}$ to a standard description of $D$ of $\mathrm{poly}(n, d)$ size. In the rest of this section, we abuse notation in the following way: whenever we refer to $\vec{\Lambda}$, we mean this standard description of $D$ of $\mathrm{poly}(n, d)$ size (unless specified otherwise).

We start the iterative procedure using Item 2a of Proposition 4.6, which allows the network $R_n$ to produce an arithmetic circuit of size $O(nmh \cdot \mathrm{polylog}(s))$ and degree $nh \cdot \mathrm{polylog}(s)$ that computes $f_0$. Then the arithmetic network $R_n$ successively uses the networks given by the following lemma to construct circuits for the subsequent layer polynomials $f_{i,j}$.

**Lemma 5.11** (One iteration: moving from a circuit for $f_{i,j-1}$ to a circuit for $f_{i,j}$)**.** *There is a* $\mathsf{P}$*-uniform family* $\mathcal{N} = \{\mathcal{N}_n\}$ *of arithmetic networks with* $\mathsf{PIT}$ *gates that when given as input* $\vec{\Lambda}$ *and a standard description of a size* $s_{i,j-1}$*, degree* $d_{i,j-1}$ *arithmetic circuit computing* $f_{i,j-1}$*, outputs a standard description of an arithmetic circuit computing* $f_{i,j}$ *of size* $O(n \cdot d \cdot h^3 \cdot \mathrm{polylog}(s) \cdot \bar{n}^{30m}) = \mathrm{poly}(n, d, h, \log s, \bar{n}^m)$ *and degree* $h \cdot \mathrm{polylog}(s)$*. The size of* $\mathcal{N}_n$ *is* $\mathrm{poly}(n, s_{i,j-1}, d_{i,j-1}, d, h, \log s, \bar{n}^m)$ *and its degree is at most* $\mathrm{poly}(d_{i,j-1}, d, h, \bar{n}, \log s)$*.*

*Furthermore, there is another family* $\mathcal{N}'$ *of arithmetic networks with* $\mathsf{PIT}$ *gates with the same complexity as* $\mathcal{N}$ *(i.e.,* $\mathsf{P}$*-uniformity, size, and degree) that when given as input* $\vec{\Lambda}$ *and a standard description of a size* $s_{i-1,2m}$*, degree* $d_{i-1,2m}$ *arithmetic circuit computing* $f_{i-1,2m}$*, outputs the description of an arithmetic circuit computing* $f_{i,0}$ *of size* $\mathrm{poly}(n, d, h, \log s, \bar{n}^m)$ *and degree* $h \cdot \mathrm{polylog}(s)$*.*

We defer the proof of Lemma 5.11 to Section 5.2.1. The remainder of this subsection completes the proof of Proposition 5.10 assuming Lemma 5.11.

The network $R_n$ makes repeated use of the networks $\mathcal{N}_n$ and $\mathcal{N}'_n$ from Lemma 5.11. If we have computed the description of a circuit for $f_{i-1}$, then applying the network $\mathcal{N}'_n$ of Lemma 5.11 to the description of the circuit for $f_{i-1}$ yields a description of a circuit for $f_{i,0}$. We then use $2m$ sequential applications of the network $\mathcal{N}_n$ of Lemma 5.11 to compute descriptions of circuits for $f_{i,1}, f_{i,2}, \ldots, f_{i,2m} = f_i$. After having computed the description of a circuit that computes $f_\Delta = f_{\Delta,2m}$, we use Proposition 3.24 to query $f_\Delta$ on the $n$ lexicographically first elements of $H^m$. Finally, the network outputs the values of $f_\Delta$ on these $n$ points, which by item 1 of Proposition 4.6, equals $C_n(\vec{\Lambda})$.

The network $R_n$ is obtained through one application of Item 2a of Proposition 4.6, $2m\Delta$ applications of Lemma 5.11, and $n$ applications of Proposition 3.24. For each application of Lemma 5.11, the size of the resulting circuit for $f_{i,j}$ is independent of the circuit for the preceding layer polynomial $f_{i,j-1}$. In other words, letting $s_{i,j}$ denote the size of the circuit obtained for $f_{i,j}$, we have

$$s_{i,j} \leq \mathrm{poly}(n, d, h, \log s, \bar{n}^m).$$

In addition, the degree of $f_{i,j}$ is bounded by $h \cdot \mathrm{polylog}(s)$. From the bound on the size of the circuit computing $f_{i,j}$, we see that the $2m\Delta \leq \mathrm{polylog}(s)$ applications of networks $\mathcal{N}_n$ and $\mathcal{N}'_n$ increase the size of $R_n$ by a total of $\mathrm{poly}(n, d, h, \log s, \bar{n}^m)$. The $n$ applications of Proposition 3.24 correspond to a cost of

$$\mathrm{poly}(n, s_{\Delta,2m}, \deg(f_{\Delta,2m})) \leq \mathrm{poly}(n, d, h, \log s, \bar{n}^m)$$

in the size of $R_n$. In total, the size of the network $R_n$ can be bounded by $\mathrm{poly}(n, d, h, \log s, \bar{n}^m)$ as claimed.

To see the degree bound, first note that from each application of Lemma 5.11, the degree of the resulting circuit for $f_{i,j}$ is independent of that of the circuit for the preceding layer polynomial $f_{i,j-1}$. In fact, we have the bound $d_{i,j} = h \cdot \mathrm{polylog}(s)$ for all $i$ and $j$. Next, the network produced by Item 2a of Proposition 4.6 that produces a description of $f_0$ has degree $n \cdot h \cdot \mathrm{polylog}(s)$. By Proposition 3.21, we can bound the degree of $R_n$ by the product of the degrees of its constituent networks. To this end, note that each of the $2m\Delta \leq \mathrm{polylog}(s)$ invocations of $\mathcal{N}_n$ or $\mathcal{N}'_n$ multiplies the degree of $R_n$ by $\mathrm{poly}(d, h, \bar{n}, \log s)$. Therefore, overall, we conclude that the degree of $R_n$ is bounded by $\mathrm{poly}(d, h, \bar{n}, \log s)^{\mathrm{polylog}(s)}$.

### 5.2.1 A single iteration: Proof of Lemma 5.11

We first prove the main part of the statement, corresponding to the family $\{\mathcal{N}_n\}_{n\in\mathbb{N}}$ of arithmetic networks with PIT gates. We then explain how to obtain the family $\{\mathcal{N}'_n\}_{n\in\mathbb{N}}$, relying on essentially the same argument as in the construction of $\{\mathcal{N}_n\}_{n\in\mathbb{N}}$.

Relabel the indices $i$ and $j$ in the lemma statement as $i'$ and $j'$, respectively, so that $i$ and $j$ are now free to use for other purposes in the remainder of the proof of this lemma. From now on, we assume the indices $i'$ and $j'$ are fixed, so we refer to $f_{i',j'}$ as $f$ for simplicity. We also use $s'$ and $d'$, respectively, to refer to the size and degree of the circuit for $f_{i',j'-1}$ that is given to $\mathcal{N}_n$ as input. Let $d_f$ be the degree of $f$ (which by item 5 of Proposition 4.6 is bounded by $h \cdot \mathrm{polylog}(s)$).

Recall that in Construction 5.6 we added dummy variables to all the $f_{i',j'}$'s so that they are all $3m$-variate polynomials. We will repeatedly use the fact that given the circuit for $f_{i',j'-1}$, the network can evaluate $f$ at any given point in $\mathbb{F}_n^{3m}$ by using the downwards self-reducibility of the decomposition $\{f_{i,j}\}$. Let us state this formally.

**Subclaim 5.12.** *There is a* P*-uniform network of size* $h \cdot \mathrm{poly}(s')$ *and degree* $O(1)$ *that gets as input* $(i', j') \in [\Delta] \times \{0, ..., 2m\}$ *and a standard description of an arithmetic circuit of size* $s'$ *and degree* $d'$ *computing* $f_{i',j'-1}$*, and outputs a standard description of an arithmetic circuit of size* $h \cdot \mathrm{poly}(s')$ *and degree* $d'$ *that computes* $f$*.*

**Subproof.** The network relies on Proposition 3.24 (to evaluate the circuit for $f_{i',j'-1}$) and on the downward self-reducibility of the polynomial decomposition $\{f_{i,j}\}$ (i.e., Item 4 of Proposition 4.6). □

**High-level overview.** For a better exposition, we first describe the ideas involved in the construction of the arithmetic network $\mathcal{N}_n$ in a list below and subsequently discuss the formal details of their implementation via a P-uniform family of arithmetic networks with PIT gates. The reconstruction closely follows the argument laid out in [GKSS22], with additional ideas needed to ensure the uniformity of the reconstruction.

- From the equation
$$D \circ \mathcal{H}_{\vec{\Lambda}}^{(h,m,\bar{n})}(w_1, w_2, \vec{z}, \vec{y}) = 0,$$
applying the partial evaluation $(w_1, w_2) \mapsto (i', j')$ yields
$$D \circ \mathcal{H}_{\mathrm{GKSS},f_{i',j'},\bar{n}}(\vec{z}, \vec{y}) = 0.$$
That is, the distinguisher $D$ for $\mathcal{H}_{\vec{\Lambda}}^{(h,m,\bar{n})}$ is necessarily a distinguisher for $\mathcal{H}_{\mathrm{GKSS},f_{i',j'},\bar{n}}$. Let the output of the generator $\mathcal{H}_{\mathrm{GKSS},f_{i',j'},\bar{n}}(\vec{z}, \vec{y})$ (or just $\mathcal{H}_{\mathrm{GKSS},f,\bar{n}}(\vec{z}, \vec{y})$ in our simplified notation) be denoted by $(g_0(\vec{z}, \vec{y}), \ldots, g_{\bar{n}}(\vec{z}, \vec{y}))$.
- **(Distinguisher to "predictor" transformation.)** For $a \in \{0, 1, \ldots, \bar{n}\}$, define the $a^{\mathrm{th}}$ hybrid polynomial
$$D_a(\vec{z}, \vec{y}, \vec{x}) = D(g_0(\vec{z}, \vec{y}), \ldots, g_a(\vec{z}, \vec{y}), x_{a+1}, \ldots, x_{\bar{n}}).$$

Because $D \circ \mathcal{H}_{\mathrm{GKSS},f,\bar{n}}(\vec{z},\vec{y}) = 0$ and $D \neq 0$, it follows that there exists some $a \in \{0,1,\ldots,\bar{n}-1\}$ for which $D_a(\vec{z},\vec{y},\vec{x}) \neq 0$ but $D_{a+1}(\vec{z},\vec{y},\vec{x}) = 0$. Since we do not know which index $a$ satisfies this condition, the arithmetic network $\mathcal{N}_n$ will simultaneously try all possible values of $a$ in $\bar{n}$ different branches. In what follows, we work within one such branch of the arithmetic network with a fixed value of $a$.

- **(Guessing multiplicity of the generator as a root.)** Suppose we have correctly guessed the value of $a$ for which $D_a(\vec{z},\vec{y},\vec{x}) \neq 0$ but $D_{a+1}(\vec{z},\vec{y},\vec{x}) = 0$. This implies that the distinguisher $D$ depends non-trivially on the variable $x_{a+1}$. Furthermore, since $D_a(\vec{z},\vec{y},\vec{x})$ vanishes on the substitution $x_{a+1} \mapsto g_{a+1}(\vec{z},\vec{y})$, it follows from Lemma 3.3 that $(x_{a+1} - g_{a+1}(\vec{z},\vec{y}))$ divides $D_a(\vec{z},\vec{y},\vec{x})$. Let $r$ be the highest power of $(x_{a+1} - g_{a+1})$ that divides $D_a$. Then we have

$$(\partial^{r-1}_{x_{a+1}} D_a) \circ \mathcal{H}_{\mathrm{GKSS},f,\bar{n}}(\vec{z},\vec{y}) = 0$$
$$(\partial^{r}_{x_{a+1}} D_a) \circ \mathcal{H}_{\mathrm{GKSS},f,\bar{n}}(\vec{z},\vec{y}) \neq 0.$$

  Note that we must have $r \leq \deg(D_a) \leq d$. We would like to work with this $r$, but since we do not know its value, the network $\mathcal{N}_n$ simultaneously tries using all values of $b \in [d]$ as the value of $r$. Just as in the previous bullet, we now work within one such branch of the arithmetic network with a fixed value of $b$.

- **(Within branch $(a,b)$.)** We have now fixed guesses for the values of $a$ and $b$. From Lemma A.5, we can compute a description of a circuit of size $O(\mathrm{size}(D) \cdot d) = O(nd)$ for the partial derivative $\partial^b_{x_{a+1}} D_a$ using a $\mathsf{P}$-uniform family of arithmetic networks. We make use of this circuit in the subsequent steps.

  - **(Testing for viability of the sub-branch.)** First, we verify, using $\mathsf{PIT}$ gates, whether our choice of $(a,b)$ will allow us to reconstruct a circuit for the polynomial $f$. For the reconstruction to succeed in this sub-branch, it is sufficient that the polynomial

$$\Phi^{(a,b)}(\vec{z},\vec{y}) \coloneqq (\partial^b_{x_{a+1}} D_a) \circ \mathcal{H}_{\mathrm{GKSS},f,\bar{n}}(\vec{z},\vec{y}) = (\partial^b_{x_{a+1}} D_a) \circ (\Delta_{f,0}(\vec{z},\vec{y}), \Delta_{f,1}(\vec{z},\vec{y}), \ldots, \Delta_{f,\bar{n}-1}(\vec{z},\vec{y}))$$

    is nonzero.[27] We first use the downward self-reducibility relation (Subclaim 5.12) to compute the description of an arithmetic circuit $F$ of size $h \cdot s'$ and degree $d'$ that computes $f$. Using this, we can generate the description of a circuit $G$ of size $h \cdot s' + 2\ell = O(h \cdot s')$ and the same degree $d'$ that computes the $2\ell$-variate polynomial $f(\vec{y}+\vec{z})$. We then use the arithmetic network in Lemma A.6 to extract the $\vec{y}$-homogeneous components of degree at most $\bar{n}$ of this circuit, i.e., the polynomials $\Delta_{f,0}(\vec{z},\vec{y}), \Delta_{f,1}(\vec{z},\vec{y}), \ldots, \Delta_{f,\bar{n}-1}(\vec{z},\vec{y})$. Let $G_i$ denote the circuit of size $O(hs'i^2)$ that computes $\Delta_{f,i}$. From the aforementioned circuit description for the partial derivative $\partial^b_{x_{a+1}} D_a$ along with these descriptions of $G_i$'s, and applying Proposition 3.25, it follows that we can compute a description of a circuit for $\Phi^{(a,b)}(\vec{y},\vec{z})$ of size $O(nd \cdot hs'\bar{n}^2)$.

    We feed this description[28] to a $\mathsf{PIT}$ gate and only proceed with subsequent steps in this sub-branch if the $\mathsf{PIT}$ gate outputs that $\Phi^{(a,b)}$ is nonzero. Otherwise, the current branch detects its failure to reconstruct $f$ and aborts, returning (a description of) the zero circuit.

  - **(Ensuring nonzeroness of an auxiliary polynomial.)** In the rest of the high-level overview, we work with the assumption that the polynomial $\Phi^{(a,b)}(\vec{z},\vec{y}) \coloneqq (\partial^b_{x_{a+1}} D_a) \circ \mathcal{H}_{\mathrm{GKSS},f,\bar{n}}(\vec{z},\vec{y})$ is nonzero. In fact, for the reconstruction to succeed in this sub-branch, it is sufficient that the polynomial

$$\Psi^{(a,b)}_f(\vec{y}) \coloneqq (\partial^b_{x_{a+1}} D_a) \circ \mathcal{H}_{\mathrm{GKSS},f,\bar{n}}(\vec{0},\vec{y})$$

    is nonzero. We would like to work with the assumption that the simpler polynomial $\Psi^{(a,b)}_f(\vec{y})$ is nonzero; however, it is possible that even in a sub-branch corresponding to "correct" guesses for $a$ and $b$ (i.e., when $\Phi^{(a,b)}$ is nonzero), the polynomial $\Psi^{(a,b)}_f(\vec{y})$ simplifies to the zero polynomial. In such a situation, since $\Phi^{(a,b)}(\vec{z},\vec{y})$ is a nonzero $\ell$-variate polynomial of degree at most $d \cdot d_f$ in the

[27] We do not justify this statement here; rather, this is an artifact of the reconstruction procedure of [GKSS22] and becomes evident after a complete reading of the present high-level overview.

[28] Strictly speaking, $\mathsf{PIT}$ gates expect universal encodings as inputs but we note that it is possible to convert a standard encoding to a universal one (and vice-versa) without significant overhead (see Lemma A.1 and Lemma A.2), and ignore this point about encoding format in the high-level overview.

$\vec{z}$ variables over $\mathbb{F}_n(\vec{y})$, Lemma 3.1 implies that $\Phi^{(a,b)}(\vec{c},\vec{y})$ is a nonzero polynomial for a random choice of $\vec{c} \in [2d \cdot d_f]^\ell$. By Observation 5.3, we have

$$\Phi_{(a,b)}(\vec{z}+\vec{c},\vec{y}) = (\partial^b_{x_{a+1}} D_a) \circ \mathcal{H}_{\mathrm{GKSS},f,\bar{n}}(\vec{z}+\vec{c},\vec{y}) = (\partial^b_{x_{a+1}} D_a) \circ \mathcal{H}_{\mathrm{GKSS},f',\bar{n}}(\vec{z},\vec{y}),$$

where $f'$ denotes the polynomial $f'(\vec{y}) \coloneqq f(\vec{y}+\vec{c})$ obtained by shifting the point $\vec{c}$ to the origin. In particular, $\Phi^{(a,b)}(\vec{c},\vec{y}) = (\partial^b_{x_{a+1}} D_a) \circ \mathcal{H}_{\mathrm{GKSS},f',\bar{n}}(\vec{0},\vec{y}) = \Psi^{(a,b)}_{f'}(\vec{y})$ is a nonzero polynomial for a random choice of $\vec{c} \in [2d \cdot d_f]^\ell$.

Therefore, the next goal for our reconstruction network is to repeatedly make use of PIT gates in order to find such a shift $\vec{c}$. The reconstruction network attempts to find such a shift by fixing one coordinate of $\vec{c}$ at a time, using the standard search-to-decision reduction for PIT. Fix a constant $c_1 \in [2d \cdot d_f]$. From the aforementioned description of a circuit for $\Phi^{(a,b)}_g(\vec{y},\vec{z})$ of size $O(nd \cdot hs'\bar{n}^2)$, we generate the description of a circuit for the $(2\ell-1)$-variate polynomial $\Phi^{(a,b)}_g(\vec{y},c_1,z_2,\ldots,z_\ell)$. We feed the resulting description to a PIT gate. If for this choice of $c_1$, the polynomial is zero, then we try the next value for $c_1$ in $[2d \cdot d_f]$, and continue in this manner. If we find a choice of $c_1$ for which the polynomial is nonzero, we fix it and move on to checking the nonzeroness of the choices for $z_2$ upon substituting constants $c_2 \in [2d \cdot d_f]$, and so on. Note that since $\Phi^{(a,b)}(\vec{z},\vec{y})$ is nonzero, from Lemma 3.1, we are guaranteed to find a shift $\vec{c}$ using this procedure such that $\Psi^{(a,b)}_{f'}(\vec{y})$ is nonzero.

In the rest of this high-level overview, the reconstruction is truly for $f'$, not $f$; however we remark that once we manage to reconstruct a small circuit for $f'$ at the end of this sub-branch, we can easily obtain one for $f$ by simply subtracting $\vec{c}$ from the inputs. For the sake of simpler notation, we ignore this point about translation by a random vector $\vec{c}$ in the rest of the high-level overview, and work with the assumption that we are in a sub-branch where $\Phi^{(a,b)}$ is nonzero, and furthermore, so is $\Psi(\vec{y}) \coloneqq (\partial^b_{x_{a+1}} D_a) \circ \mathcal{H}_{\mathrm{GKSS},f,\bar{n}}(\vec{0},\vec{y})$. From now on, we will drop the subscript $f$ in $\Psi$, as well as the superscript $(a,b)$, as we are working in a fixed sub-branch of the reconstruction.

– **(Constructing a constrained interpolating set.)** The next step is to construct an explicit interpolating set for $\mathcal{P}(\ell,\bar{n})$ with the crucial property that $\Psi$ is nonzero at every point in this set (the reason for this demand will be explained momentarily). Consider the following $M_{\ell,\bar{n}} \times M_{\ell,\bar{n}}$ variable matrix $A(\vec{y}_1,\ldots,\vec{y}_{M_{\ell,\bar{n}}})$: the rows of $A$ are indexed by $[M_{\ell,\bar{n}}]$ and the columns indexed by $\ell$-variate monomials $\vec{x}^{\vec{e}}$ of degree at most $\bar{n}$, and the entry at $(i,\vec{x}^{\vec{e}})$ is $\vec{y}_i^{\vec{e}}$ (the monomial $\vec{x}^{\vec{e}}$ evaluated at $\vec{y}_i$). The condition that a collection of vectors $\{\vec{a}_1,\ldots,\vec{a}_{M_{\ell,\bar{n}}}\}$ (where each $\vec{a}_i \in \mathbb{F}_n^\ell$) is an interpolating set is characterized by the nonzeroness of the determinant of $A(\vec{a}_1,\ldots,\vec{a}_{M_{\ell,\bar{n}}})$. Therefore, it suffices to find vectors $\vec{a}_1,\ldots,\vec{a}_{M_{\ell,\bar{n}}}$ such that the evaluation of the polynomial $Q(\vec{y}_1,\ldots,\vec{y}_{M_{\ell,\bar{n}}}) \coloneqq \det(A(\vec{y}_1,\ldots,\vec{y}_{M_{\ell,\bar{n}}})) \cdot \prod_{i=1}^{M_{\ell,\bar{n}}} \Psi(\vec{y}_i)$ at $(\vec{a}_1,\ldots,\vec{a}_{M_{\ell,\bar{n}}})$ does not vanish.

To do this, the network first uses a standard description of the P-uniform family of arithmetic circuits of $\mathrm{poly}(M_{\ell,\bar{n}})$ size and degree from [Ber84] to compute $\det(A)$, and the description of the circuit for $\Psi(\vec{y}_i) = \Phi^{(a,b)}(\vec{0},\vec{y}_i)$ computed in a previous bullet point to compute a description of $Q$. Note that our assumption about $\Psi$ implies that $Q$ is a nonzero polynomial, and furthermore, note that the determinant of $A$ as a polynomial over $\ell \cdot M_{\ell,\bar{n}}$ many variables has the property that each of its rows is indexed by a disjoint set of variables. As a consequence, the individual degree of $Q$ is bounded by $\text{i-deg}(A) \cdot \deg(\Psi) \leq \bar{n} + d\bar{n}$ and therefore, by Lemma 3.2, we have that when the vectors $\vec{a}_1,\ldots,\vec{a}_{M_{\ell,\bar{n}}}$ are chosen uniformly at random from $[2\bar{n}+2d\bar{n}]^\ell$, then with high probability, they form an interpolating set for $\mathcal{P}(\ell,\bar{n})$ and $\Psi(\vec{a}_i) \neq 0$ for each $i \in [M_{\ell,\bar{n}}]$.

The network finds such a non-vanishing point $(\vec{a}_1,\ldots,\vec{a}_{M_{\ell,\bar{n}}})$ for $Q$ by fixing one coordinate (out of $\ell \cdot M_{\ell,\bar{n}}$ in all) at a time, in a process similar to that described earlier for finding a good shift $\vec{c}$ described previously. We omit the details but note that this leads to at most $(2\bar{n}+2d\bar{n}) \cdot \ell \cdot M_{\ell,\bar{n}}$ calls to PIT gates, where each such PIT gate is fed a description of an arithmetic circuit of size bounded by $O(nd \cdot hs'\bar{n}^2) \cdot \mathrm{poly}(M_{\ell,\bar{n}}) = \mathrm{poly}(n,d,h,s',\bar{n}^m)$.

– **(Induction setup.)** We now describe the setup of the inductive reconstruction of a small circuit for $f$. The induction is on a parameter $j$ which takes values from 0 up to $d_f - \bar{n}$. Note that this is where we use the given bound on $\bar{n}$ in the statement of Proposition 5.10. At the end of the

$j$-th step, we will have a circuit that computes all partial derivatives of order at most $\bar{n}$ of all homogeneous components of $f$ of degree up to $j+\bar{n}$. We now describe the steps of the induction argument more formally.

– **(Base case ($j=0$).)** For each $\vec{e} \in \mathbb{Z}^{\ell}_{\geq 0}$ of weight $|\vec{e}| \leq \bar{n}$ and each $t \leq \bar{n}$, we can write the polynomial $\partial^{\vec{e}}_{\vec{z}} f_t$—where $f_t$ is the degree-$t$ homogeneous component of $f$—explicitly as a sum of at most $M_{\ell,\bar{n}} \coloneqq \binom{\bar{n}+\ell}{\ell}$ monomials,[29] where recall from Construction 5.6 that $\ell = 3m$ is a bound on the number of variables in $f$. Hence there is a multi-output circuit $B_0$ of size $s_0 = M^2_{\ell,\bar{n}}$ that computes the polynomials $\{\partial^{\vec{e}}_{\vec{z}} f_t : 0 \leq t \leq \bar{n}, |e| \leq \bar{n}\}$.

To construct the circuit $B_0$, we first use the downward self-reducibility relation (Subclaim 5.12) to compute the description of an arithmetic circuit $F$ of size $h \cdot s'$ and degree $d'$ that computes $f$. We use the arithmetic network in Lemma A.6 to extract the homogeneous components of degree at most $\bar{n}$ of this circuit. Let $F_i$ denote the circuit of size $O(hs'i^2)$ that computes $f_i$, the degree-$i$ homogeneous component of $f$. For $i \leq \bar{n}$, the polynomial $f_i$ is an $\ell$-variate polynomial of degree at most $\bar{n}$. We can interpolate $f_i$ using a circuit of size at most $O(\bar{n}^{\ell+1}\ell)$ via

$$f_i(\vec{z}) \coloneqq \sum_{\vec{\beta} \in T^{\ell}} f_i(\vec{\beta}) \prod_{k \in [\ell]} \prod_{\gamma \in T \setminus \{\beta_k\}} \frac{z_k - \gamma}{\beta_k - \gamma}, \tag{2}$$

where $T \subseteq \mathbb{F}_n$ is any finite set[30] of size at least $\bar{n}+1$. We can compute the value of $f_i(\vec{\beta})$ by evaluating the circuit $F_i$ at $\vec{\beta}$ using Proposition 3.24.

Likewise, for any $\vec{e} \in \mathbb{Z}^{\ell}_{\geq 0}$ and any $t \leq \bar{n}$, we can interpolate a circuit for $\partial^{\vec{e}}_{\vec{z}} f_t(\vec{z})$ of the same size. This requires a description of a circuit that computes $\partial^{\vec{e}}_{\vec{z}} f_t(\vec{z})$. We can obtain such a circuit of size $O(s \prod_{i=1}^{\ell} e_i)$ by repeatedly applying Lemma A.5 to the circuit $F$ to compute the appropriate partial derivative of $f$.

– **(Induction hypothesis.)** We have the description of a circuit $B_{j-1}(\vec{z})$ that has size at most $s_{j-1}$. The circuit $B_{j-1}$ computes the $M_{\ell,\bar{n}} \cdot (\bar{n}+j-1)$ polynomials $\partial^{\vec{e}}_{\vec{z}} f_t$ for $|\vec{e}| \leq \bar{n}$ and $t \leq \bar{n}+j-1$.

– **(Induction step.)** The goal by the end of this inductive step is to construct a circuit $B_j(\vec{z})$ of size at most $s_j$ (to be defined shortly) that computes $\partial^{\vec{e}}_{\vec{z}} f_t$ for $|\vec{e}| \leq \bar{n}$ and $t \leq \bar{n}+j$.

∗ For a point $\vec{a} \in \mathbb{F}^{\ell}_n$, define

$$\Gamma_{j-1,\vec{a}} \coloneqq (\Delta_{f_{\leq \bar{n}+j-1},0}(\vec{z},\vec{a}), \ldots, \Delta_{f_{\leq \bar{n}+j-1},\bar{n}}(\vec{z},\vec{a})),$$

where $f_{\leq \bar{n}+j-1}$ denotes the sum of the first $\bar{n}+j-1$ homogeneous components of $f$. The crucial observation in [GKSS22] is that $\Delta_{f_{n+j},n}$ can be computed using the evaluation of our distinguisher $D$ on the points $\Gamma_{j-1,\vec{a}}$ as $\vec{a}$ ranges over the interpolating set we have constructed. We state their main technical lemma below.[31]

**Lemma 5.13** ([GKSS22, Lemma 3.2])**.** *Let $\vec{a} \in \mathbb{F}^{\ell}_n$ be such that $\Psi(\vec{a}) \neq 0$. Then,*

$$\left(\frac{-1}{\Psi(\vec{a})}\right) (\partial^{b+1}_{x_{a+1}} D_a)(\Gamma_{j-1,\vec{a}}) = \Delta_{f_{\bar{n}+j},\bar{n}}(\vec{z},\vec{a}) \pmod{\langle \vec{z} \rangle^{j+1}}.$$

∗ We begin the reconstruction of $f_{\bar{n}+j}$ with the circuit $B_{j-1}(\vec{z})$ that computes every $\partial^{\vec{e}}_{\vec{z}} f_t$ for $|\vec{e}| \leq \bar{n}$ and $t \leq \bar{n}+j-1$. By taking suitable linear combinations of the output gates, $\mathcal{N}$ can create a new circuit $B$ of size at most $s_{j-1} + M^5_{\ell,\bar{n}}$ that computes the points $\{\Gamma_{j-1,\vec{a}_r} : r \in [M_{\ell,\bar{n}}]\}$. Using Lemma 5.13 for each $\vec{a}_i$, it then obtains a circuit of size $s_{j-1} + M^5_{\ell,\bar{n}} + n \cdot d \cdot M_{\ell,\bar{n}}$ that computes $\{\Delta_{f_{\bar{n}+j},\bar{n}}(\vec{z},\vec{a}_r) : r \in [M_{\ell,\bar{n}}]\}$ modulo the ideal $\langle \vec{z} \rangle^{j+1}$.

[29] This is overloaded notation, as $f_t$ has already been defined as a layer polynomial in Definition 4.4. However, since we are working with a fixed layer polynomial $f$ inside the proof of Lemma 5.11, there is no ambiguity, as we will never use $f_t$ to refer to a layer polynomial throughout the proof.

[30] For example, one can take $T = [\bar{n}]$, as the characteristic is large enough (see Construction 5.6).

[31] Strictly speaking, this lemma is only stated in [GKSS22] for $a = \bar{n}-1$ and $b = 0$ in our notation; but it is easy to verify that it also holds in this slightly more general form.

* We can regard $\Delta_{f_{\bar{n}+j},\bar{n}}$ as an $\ell$-variate degree-$\bar{n}$ polynomial in $\mathbb{F}_n(\vec{z})[\vec{y}]$. By definition, we can write $\Delta_{f_{\bar{n}+j},\bar{n}}(\vec{z},\vec{a})$ as a linear combination of the $\bar{n}$-th order partial derivatives of $f_{\bar{n}+j}(\vec{z})$. As $\{\vec{a}_1,\ldots,\vec{a}_{M_{\ell,\bar{n}}}\}$ is chosen to be an interpolating set for $\mathcal{P}(\ell,\bar{n})$, each $\partial^{\vec{e}}_{\vec{z}} f_{\bar{n}+j}$ with $|\vec{e}|=\bar{n}$ can be written as a suitable linear combination of $\{\Delta_{f_{\bar{n}+j},\bar{n}}(\vec{z},\vec{a}_r) : r \in [M_{\ell,\bar{n}}]\}$. More precisely, from Construction 5.2, including the definition of $\Delta_{P,i}$, and the "in other words" part of Definition 5.9, we have

$$\binom{1}{\vec{e}}\partial^{\vec{e}}_{\vec{z}} f_{\bar{n}+j}(\vec{z}) = \operatorname{coeff}_{\vec{y}^{\vec{e}}}(\Delta_{f_{\bar{n}+j},\bar{n}}(\vec{z},\vec{y})) = \sum_{i=1}^{M_{\ell,\bar{n}}} \beta_i \cdot \Delta_{f_{\bar{n}+j},\bar{n}}(\vec{z},\vec{a}_i) = \sum_{i=1}^{M_{\ell,\bar{n}}} \beta_i \cdot \sum_{\substack{\vec{\alpha}\in\mathbb{Z}^{\ell}_{\geq 0}\\ |\vec{\alpha}|=\bar{n}}} \frac{\vec{a}_i^{\vec{\alpha}}}{\vec{\alpha}!}\cdot\partial^{\vec{\alpha}}_{\vec{z}}(f_{\bar{n}+j}(\vec{z})) \tag{3}$$

for some $\{\beta_1,\ldots,\beta_N\} \subseteq \mathbb{F}_n(\vec{z})$. Note that this amounts to solving the linear system $L\vec{\beta} = \vec{\chi}_{\vec{e}}/\vec{e}!$. Here, $L$ is an $M_{\ell,\bar{n}} \times M_{\ell,\bar{n}}$ matrix whose rows are indexed by exponent vectors $\left\{\vec{\alpha}\in\mathbb{Z}^{\ell}_{\geq 0} : |\vec{\alpha}| \leq \bar{n}\right\}$, whose columns are indexed by $[M_{\ell,\bar{n}}]$, and whose $(\vec{\alpha},i)$ entry is $\vec{a}_i^{\vec{\alpha}}/\vec{\alpha}!$. The vector $\vec{\beta} = (\beta_1,\ldots,\beta_N)$ is the vector of indeterminates we want to solve for, and the right-hand side $\vec{\chi}_{\vec{e}}$ is the indicator vector for $\vec{e}$.
Because the set $\{\vec{a}_1,\ldots,\vec{a}_N\}$ is an interpolating set for $\mathcal{P}(\ell,\bar{n})$, the matrix $L$ is invertible. Hence we can solve this system of equations for $\vec{\beta}$ by inverting $L$ (which can be done with a $\mathsf{P}$-uniform arithmetic circuit of polynomial size [Ber84]) and computing the matrix-vector product $L^{-1}\vec{\chi}_{\vec{e}}$. For $\vec{e}\in\mathbb{Z}^{\ell}_{\geq 0}$ with $|\vec{e}|=\bar{n}$, this lets us write the partial derivative $\partial^{\vec{e}}_{\vec{z}} f_{\bar{n}+j}(\vec{z})$ as a linear combination of the evaluations $\left\{\Delta_{f_{\bar{n}+j},\bar{n}}(\vec{z},\vec{a}_r) : r\in[M_{\ell,\bar{n}}]\right\}$, which we have already computed in the previous step.

**Remark 5.14.** *We use the field characteristic assumption here to represent the derivative $\partial^{\vec{e}}_{\vec{z}} f_{\bar{n}+j}(\vec{z})$ as a linear combination of the evaluations $\left\{\Delta_{f_{\bar{n}+j},\bar{n}}(\vec{z},\vec{a}_r) : r\in[M_{\ell,\bar{n}}]\right\}$ (using the formula described in Construction 5.2 in eq. (3) and in the subsequent formulation of the linear system). The assumption that $\operatorname{char}(\mathbb{F}) = 0$ or $\operatorname{char}(\mathbb{F}_n) > d_f = h\cdot\operatorname{polylog}(s)$ ensures that the linear system $L\vec{\beta} = \vec{\chi}_{\vec{e}}/\vec{e}!$ is well-defined.*

* We have now computed all $\bar{n}$-th order partial derivatives of $f_{\bar{n}+j}$. Because $f_{\bar{n}+j}$ is a homogeneous polynomial, an arithmetic network can also compute all lower order derivatives via repeated applications of Euler's formula (Fact 5.5).

**Remark 5.15.** *It is crucial in these applications of Fact 5.5 that $\deg(f_{\bar{n}+j})$ is nonzero in $\mathbb{F}$. The value of $\deg(f_{\bar{n}+j})$ can be as large as $d_f = h\cdot\operatorname{polylog}(s)$. Because we assume that $\operatorname{char}(\mathbb{F}) = 0$ or $\operatorname{char}(\mathbb{F}_n) > h\cdot\operatorname{polylog}(s)$, we are guaranteed that $\deg(f_{\bar{n}+j})$ is nonzero in $\mathbb{F}$.*

Combined with the outputs of $B_{j-1}(\vec{z})$, we have a circuit $B'_j(\vec{z})$ of size $s_{j-1} + M^{10}_{\ell,\bar{n}} + n\cdot d\cdot M_{\ell,\bar{n}}$ that computes the polynomials

$$\{\partial^{\vec{e}}_{\vec{z}} f_t : |\vec{e}| \leq \bar{n}, t \leq \bar{n}+j-1\} \cup \{\partial^{\vec{e}}_{\vec{z}} \tilde{f}_{\bar{n}+j} : |\vec{e}| \leq \bar{n}\},$$

where $\partial^{\vec{e}}_{\vec{z}} \tilde{f}_{\bar{n}+j} = \partial^{\vec{e}}_{\vec{z}} f_{\bar{n}+j} \pmod{\langle z\rangle^{\bar{n}+j-|\vec{e}|+1}}$ for every $|\vec{e}| \leq \bar{n}$.

* The last remaining task is to fix the errors present in the polynomials $\partial^{\vec{e}}_{\vec{z}} \tilde{f}_{\bar{n}+j}$ and obtain circuits that correctly compute the desired partial derivatives $\partial^{\vec{e}}_{\vec{z}} f_{\bar{n}+j}$. The circuit $B'_j$ is a composition of a circuit of size $M^{10}_{\ell,\bar{n}} + n\cdot d\cdot M_{\ell,\bar{n}}$ with the homogeneous circuit $B_{j-1}$ of size $s_{j-1}$. Using Lemma A.7, we extract the lowest degree homogeneous parts of the outputs of $B'_j \circ B_{j-1}$ by constructing an equivalent homogeneous circuit (which we call $B_j$) of size at most $s_{j-1} + ((M^{10}_{\ell,\bar{n}} + n\cdot d\cdot M_{\ell,\bar{n}})\cdot d_f^2)$ that computes $\{\partial^{\vec{e}}_{\vec{z}} f_t : |\vec{e}| \leq \bar{n}, t \leq \bar{n}+j\}$. This completes the induction step.

– Unraveling the induction, for $d_f - \bar{n}$ steps, we eventually obtain a circuit of size at most $s_{d_f-\bar{n}} = O(n\cdot d\cdot d_f^3\cdot M^{10}_{\ell,\bar{n}}) = O(n\cdot d\cdot h^3\cdot\operatorname{polylog}(s)\cdot\bar{n}^{30m})$ that computes all the partial derivatives of order at most $\bar{n}$ of $f_0,\ldots,f_{d_f}$. The zeroth order partial derivatives are precisely $f_0,\ldots,f_{d_f}$, and these polynomials can be summed to produce a circuit for $f$ of size $O(n\cdot d\cdot h^3\cdot\operatorname{polylog}(s)\cdot\bar{n}^{30m})$.[32]

[32] It is at this point that the network undoes the initial translation by $\vec{c}$ to recover a circuit of the same asymptotic size for the original polynomial.

– The network outputs the description of a candidate circuit produced by any sub-branch of the computation which does not abort after testing viability, breaking ties arbitrarily.

**Details of implementation.** We now formalize the details of the implementation of this reconstruction procedure using a sequence of claims below.

**Subclaim 5.16** (evaluating a single sub-branch)**.** *There is a* $\mathsf{P}$*-uniform family* $\mathcal{N}^{\mathsf{in}} = \{\mathcal{N}^{\mathsf{in}}_n\}$ *of arithmetic networks with* $\mathsf{PIT}$ *gates with the following properties.*

- *It takes as input* $\vec{\Lambda} \in \mathbb{F}^n_n$*, integers* $a \in [\bar{n}-1]$ *and* $b \in [d]$*, and a standard description of a size* $s'$*, degree* $d'$ *arithmetic circuit computing* $f_{i',j'-1}$*.*
- *It outputs a Boolean value* $v_{a,b}$ *along with a standard description of a circuit* $\mathcal{C}_{a,b}$ *such that:*
  - *if* $(\partial^b_{x_{a+1}} D_a) \circ \mathcal{H}_{GKSS,f,\bar{n}}(\vec{z},\vec{y})$ *is nonzero, then* $v_{a,b} = 1$ *and* $\mathcal{C}_{a,b}$ *has size* $O(n \cdot d \cdot h^3 \cdot \mathrm{polylog}(s) \cdot \bar{n}^{30m})$*, degree* $d_f$*, and computes* $f$*, and*
  - *if* $(\partial^b_{x_{a+1}} D_a) \circ \mathcal{H}_{GKSS,f,\bar{n}}(\vec{z},\vec{y}) = 0$*, then* $v_{a,b} = 0$*, and* $\mathcal{C}_{a,b}$ *is a description of the zero circuit.*
- *Moreover,* $\mathcal{N}^{\mathsf{in}}_n$ *has size* $\mathrm{poly}(n, d, h, \log s, \bar{n}^m, s', d')$ *and degree at most* $\mathrm{poly}(d, d', h, \bar{n}, \log s)$*.*

**Subproof.** We verify that within any sub-branch $(a,b)$ where $(\partial^b_{x_{a+1}} D_a) \circ \mathcal{H}_{\mathrm{GKSS},f,\bar{n}}(\vec{z},\vec{y})$ is nonzero, the base case as well as the inductive step as laid out in the overview above can be performed using a $\mathsf{P}$-uniform family of arithmetic networks with $\mathsf{PIT}$ gates, and subsequently analyze the complexity of the subroutines involved.

**Testing for viability of** $(a,b)$**:** As described in the high-level overview, $\mathcal{N}^{\mathsf{in}}$ feeds a description of $\Phi^{(a,b)}$ to a $\mathsf{PIT}$ gate, and reports its output as $v_{a,b}$. If $v_{a,b} = 0$, then it outputs a description of the zero circuit (and if not, then it proceeds with the implementing the rest of the high-level overview and eventually outputs a standard description of $B_{d_f - \bar{n}}$ that we subsequently describe).

Let us now analyze the network size and degree complexity of the corresponding sub-procedure laid out in the high-level overview. First, the network that $\mathcal{N}^{\mathsf{in}}_n$ uses from Subclaim 5.12 as a subroutine to construct a description of $G$ has size $O(h \cdot s')$ and degree $O(1)$. Next, the network from Lemma A.6 used to extract circuits for $\Delta_{f,i}$ (for $0 \le i \le \bar{n}-1$) has size $\mathrm{poly}(\mathrm{size}(G), \deg(G)) = \mathrm{poly}(h, s', d')$ and degree $O(1)$. The network from Lemma A.5 that is used to compute a description of the partial derivative $\partial^b_{x_{a+1}} D_a$ has size $\mathrm{poly}(n,d)$ and degree $O(1)$. Finally, the arithmetic network from Proposition 3.25 used to generate the description of a circuit for $\Phi^{(a,b)}(\vec{y},\vec{z})$ (whose corresponding universal encoding after applying Lemma A.1 is fed into a $\mathsf{PIT}$ gate) has size $\mathrm{poly}(\bar{n}, \ell, \mathrm{size}(\partial^b_{x_{a+1}} D_a), \mathrm{size}(G)) = \mathrm{poly}(\bar{n}, \ell, n, d, h, s', d')$.

**Finding a good shift:** To find a shift $\vec{c} \in \mathbb{F}^\ell_n$ such that $\Psi^{(a,b)}_{f'}(\vec{y})$ is nonzero (where $f'(\vec{z}) \coloneqq f(\vec{z}+\vec{c})$), $\mathcal{N}^{\mathsf{in}}_n$ uses at most $2\ell \cdot d \cdot d_f$ $\mathsf{PIT}$ gates, each of which are fed a description of (an appropriate partial substitution of) the circuit described above for $\Phi^{(a,b)}(\vec{y},\vec{z})$.

**Computing a constrained interpolating set:** As described in the high-level overview, to construct an interpolating set, $\mathcal{N}^{\mathsf{in}}_n$ makes at most at most $(2\bar{n} + 2d\bar{n}) \cdot \ell \cdot M_{\ell,\bar{n}}$ calls to $\mathsf{PIT}$ gates, where each such $\mathsf{PIT}$ gate is fed a description of an arithmetic circuit for (an appropriate partial substitution of) $Q$ of size bounded by $O(nd \cdot hs'\bar{n}^2) \cdot \mathrm{poly}(M_{\ell,\bar{n}}) = \mathrm{poly}(n, d, h, s', \bar{n}^m)$.

**Base Case:** Given the description of a circuit for $f_{i',j'-1}$, the network $\mathcal{N}^{\mathsf{in}}_n$ uses it to construct the description of the circuit $F'$ for $f'(\vec{z}) = f(\vec{z}+\vec{c})$ in Subclaim 5.12 of size $O(hs')$ and degree $d'$. Next, it uses the network in Lemma A.6, which has size $\mathrm{poly}(h, s', d')$ and degree $O(1)$ to compute the descriptions of circuits for the homogeneous components $F'_0, \ldots, F'_{\bar{n}}$, where each of these circuits has size bounded by $\mathrm{poly}(h, s', \bar{n})$. Using eq. (2) and Proposition 3.24, the network $\mathcal{N}^{\mathsf{in}}_n$ constructs a circuit for each $f'_i$ (for $i \le \bar{n}$) of size at most $M^2_{\ell,\bar{n}}$.

Using the same procedure applied to circuits that compute partial derivatives of $F_i$, the network $\mathcal{N}^{\mathsf{in}}_n$ also constructs circuits for the derivatives $\left\{\partial^{\vec{e}}_{\vec{z}} f'_i : \vec{e} \in \mathbb{Z}^\ell_{\ge 0}, |\vec{e}| \le \bar{n}\right\}$ up to order $\bar{n}$ of $f'_i$. We can obtain a circuit

for $\partial_{\vec{z}}^{\vec{e}} F_i'$ of size $O(s' \prod_{i=1}^{\ell} e_i) \leq O(s' \bar{n}^{\ell}) = O(s' \bar{n}^m)$. by repeatedly applying Lemma A.5 to the circuit that computes $F_i'$. The interpolation procedure described in the previous paragraph then yields a circuit of size at most $M_{\ell,\bar{n}}^2$ for the corresponding partial derivative of $f_i'$. As in the high-level overview, we drop the primed notation (i.e., we continue to work with $f$ instead of $f'$) for the rest of this proof to simplify notation.

**Inductive Step:** Next, for each point $\vec{a}_i$, the network computes the vector $\Gamma_{j-1,\vec{a}} = (\Delta_{f_{\leq \bar{n}+j-1},0}(\vec{z},\vec{a}), \ldots, \Delta_{f_{\leq \bar{n}+j-1},\bar{n}}(\vec{z},\vec{a}))$ by using the formula for $\Delta_{P,i}$ in Construction 5.2. Note that each $\Delta_{f_{\leq \bar{n}+j-1},i}$ is a linear combination of the partial derivatives of $f_{\leq \bar{n}+j-1}(\vec{z})$ of order $i$, of which there are at most $M_{\ell,\bar{n}}$, and $\mathcal{N}_n^{\mathsf{in}}$ has access to the descriptions of all derivatives of $f_{\leq \bar{n}+j-1}(\vec{z})$ of order up to $\bar{n}$ by the inductive hypothesis. As mentioned in the overview, the network constructs a circuit $B$ of size at most $s_{j-1} + M_{\ell,\bar{n}}^5$, that computes $\{\Gamma_{j-1,\vec{a}_r} : r \in [M_{\ell,\bar{n}}]\}$. Furthermore, this computation can be performed using a $\mathsf{P}$-uniform family of networks of the same size and degree $O(1)$ (from Proposition 3.25).

Next, in order to apply Lemma 5.13, the network needs to evaluate $\Psi(\vec{a}_i)$. For this, it uses the circuits obtained in the base case for $f_i$ (for $i \leq \bar{n}$) together with Lemma 5.4 and Proposition 3.24. Also, from Lemma A.5, it can use a $\mathsf{P}$-uniform family of arithmetic networks of size $\mathrm{poly}(n,d)$ and degree $O(1)$ to compute a circuit of size $O(nd)$ for $(\partial_{x_{a+1}}^{b+1} D_a)(\vec{x})$. Overall, for each $i \in [M_{\ell,\bar{n}}]$, it computes the left-hand side of Lemma 5.13 using a $\mathsf{P}$-uniform family of arithmetic networks of size $\mathrm{poly}(n,d,M_{\ell,\bar{n}},s_{j-1})$ and degree $\mathrm{poly}(d,\bar{n})$ (this again follows from Proposition 3.25). Next, it uses a $\mathsf{P}$-uniform circuit of size and degree $\mathrm{poly}(M_{\ell,\bar{n}})$ to solve the appropriate linear system, writing the order-$\bar{n}$ partial derivatives of $f_{\bar{n}+j}$ in terms of the evaluations obtained through Lemma 5.13. It then uses the equation in Fact 5.5 to compute circuits for the derivatives of $f_{\bar{n}+j}$ of all lower orders. Finally, it uses the $\mathsf{P}$-uniform network family in Lemma A.7 of size $\mathrm{poly}(M_{\ell,\bar{n}},n,d,s_{j-1},\deg(B_j) = O(d_f))$ and degree $O(1)$ to construct the circuit $B_j$ to conclude the inductive step. Note that there are $d_f - \bar{n} = O(d_f) = h \cdot \mathrm{polylog}(s)$ iterations of this procedure.

As argued in the overview above, the size of the circuit $B_{d_f-\bar{n}}$ produced in the final iteration of the inductive step is bounded by $s_{d_f-\bar{n}} = O(n \cdot d \cdot d_f^3 \cdot M_{\ell,\bar{n}}^{10}) = \mathrm{poly}(n,d,h,\log s,\bar{n}^m)$. This implies that the bounds mentioned in this proof that rely on $s_{j-1}$ can also be bounded by this quantity. We conclude that $\mathcal{N}^{\mathsf{in}}$ has size $\mathrm{poly}(n,d,h,\log s,\bar{n}^m,s',d')$, and degree $\mathrm{poly}(\bar{n},d,d',h,\log s)$. □

**Description of $\mathcal{N}_n$.** Given this claim, we can now provide a complete description of $\mathcal{N}_n$ as follows. The output of $\mathcal{N}_n(\vec{\Lambda})$ is provided by the output of $\mathcal{S}$ (from Subclaim 6.16), which in turn, receives as input the output of the execution of $\mathcal{N}_n^{\mathsf{in}}$ (from Subclaim 5.16) on each of the indices $a \in [\bar{n}-1]$ and $b \in [d]$, i.e., the descriptions of the circuits $\{\mathcal{C}_{a,b}\}$ and the corresponding Boolean values $v_{a,b}$. More precisely, $\mathcal{S}$ receives as input the concatenation (see Definition 3.18) of $(\bar{n}-1) \cdot d$ networks with $\mathsf{PIT}$ gates ($\mathcal{N}_n^{\mathsf{in}}$ instantiated with each $a \in [\bar{n}-1]$ and $b \in [d]$).

**Complexity of $\mathcal{N}_n$.** Note that there are at most $\bar{n}d$ branches (i.e., instantiations of $\mathcal{N}_n^{\mathsf{in}}$) in all. Hence, by Subclaim 5.16, and Subclaim 6.16, the total size of $\mathcal{N}_n$ is at most $\mathrm{poly}(n,s',d',d,h,\log s,\bar{n}^m)$.

To see the degree bound, note that the degree of $\mathcal{N}_n^{\mathsf{in}}$ (the network from Subclaim 6.14) in each of its instantiations is bounded by $\mathrm{poly}(\bar{n},h,d,\log s,d')$, and as they are each run in parallel inside $\mathcal{N}_n$, the degree of the network that computes the overall set of candidate circuit descriptions $\{\langle \mathcal{C}_{a,b} \rangle | a \in [\bar{n}-1], b \in [d]\}$ is also bounded by $\mathrm{poly}(\bar{n},h,d,\log(s),d')$ (by Proposition 3.19). Finally, since $\mathcal{N}_n$ is the composition of $\mathcal{S}$ (which has degree $O(1)$) with these networks, the claimed overall degree bound of $\mathrm{poly}(\bar{n},h,d,\log(s),d')$ follows from Proposition 3.21.

**The "furthermore" part.** The proof is essentially identical to that of the main case, the only difference is that to prove Subclaim 5.12 we now rely on the downward self-reducibility relation between $f_{i-1,2m}$ and $f_{i,0}$ as specified in Item 3 of Proposition 4.6 (note the identity $f_{i,2m}(\vec{w}) = f_i(\vec{w})$). The size and degree of the circuit for downward self-reducibility both increase now to $h \cdot \mathrm{polylog}(s)$ (rather than $O(h)$ and 1 in the main case), and hence the circuit for evaluating $f$ that Subclaim 5.12 outputs now has size $h \cdot \mathrm{poly}(s',\log(s))$ and degree at most $d' \cdot h \cdot \mathrm{polylog}(s)$. This factors into the size and degree of the circuits from Subclaim 5.16, but does not change the final asymptotic bounds.

**Remark 5.17.** *The assumptions on the field characteristic made in Theorem 5.1 are used at the locations pointed to in Footnote 25, Remark 5.14, and Remark 5.15. The first instance can be remedied in arbitrary*

*characteristic by working over a sufficiently large field, and the second instance can likely be addressed by working with Hasse derivatives (a notion of derivative that is better suited to positive characteristic). However, it is not clear to the authors how one can remove the restriction on the characteristic arising from the use of Fact 5.5.*

# 6 A targeted version of the Kabanets–Impagliazzo generator

Our goal in this section is to construct an arithmetic reconstructive targeted hitting-set generator that works over sequences of finite fields, and is analogous to the generator from Theorem 5.1.

A technical subtlety is that the generator can be instantiated in two parametric "modes", represented by the parameter $\sigma \in \{0,1\}$ below. The mode $\sigma = 1$ useful when the output length is large, and the mode $\sigma = 0$ is useful for when the output length is small. (Jumping ahead, our PIT algorithm will compose this generator constantly many times, the first iteration with $\sigma = 1$ and subsequent iterations with $\sigma = 0$; see the proof of Theorem 7.4 for details.)

We say that a field family $\mathbb{F} = \{\mathbb{F}_n\}_{n\in\mathbb{N}}$ is efficiently constructible if there is a Turing machine that gets input $1^n$ and outputs a representation of $\mathbb{F}_n$ (i.e., a prime or a suitable irreducible polynomial).

**Theorem 6.1** (algebraic KI-based targeted hitting-set generator)**.** *Let $\{C_n\}_{n\in\mathbb{N}}$ be a $\log^c$-uniform family of algebraic circuits of size $s(n)$ and degree at most $s(n)$ having $n$ output gates, defined over a feasible sequence of finite fields $\{\mathbb{F}_n\}_{n\in\mathbb{N}}$, where $\mathbb{F}_n$ has order $q(n) \le \exp(n^{O(1)})$ and characteristic $p(n)$. Let $\varepsilon > 0$ be a constant, and let $m(n) < s(n)$ and $d(n)$ be time-constructible functions. Then, for $\sigma \in \{0,1\}$, there are two* P*-uniform families of arithmetic networks $\{G_n\}$ and $\{R_n\}$ satisfying the following.*

1. **(Generator.)** *Networks in $\{G_n\}$ are of size $\mathrm{poly}(s)$. When given input $\vec{\Lambda} \in \mathbb{F}_n^n$, the network $G_n(\vec{\Lambda})$ outputs the description of an arithmetic circuit*

$$\mathcal{G}_{\vec{\Lambda}} \colon \mathbb{F}_n^{\ell(n)} \times \mathbb{F}_n^2 \to \mathbb{F}_n^{m(n)}$$

*of degree $\deg(\mathcal{G}_{\vec{\Lambda}}) \le n \cdot s^{\Theta(1/\log(m))} \cdot \mathrm{polylog}(s(n))$, where $\ell(n) = \begin{cases} \mathrm{polylog}(s) & \text{if } \sigma = 1 \\ O(\log(m)) & \text{if } \sigma = 0 \end{cases}$.*

2. **(Reconstruction.)** *Networks in $\{R_n\}$ are randomized, and when $\vec{\Lambda} \in \mathbb{F}_n^n$ is a universal encoding of a degree-$d$ arithmetic circuit with $m$ input gates such that $\vec{\Lambda} \circ \mathcal{G}_{\vec{\Lambda}} \equiv 0$, then $R_n(\vec{\Lambda})$ computes $C_n(\vec{\Lambda})$ with error degree $\mathrm{poly}(n, d, \log(s))$.*

   *Each network $R_n$ is of size $\begin{cases} \mathrm{poly}(n, d, m, \log(s), \log(q)) & \text{if } \sigma = 1 \\ s^{\varepsilon} \cdot \mathrm{poly}(n, d, m^{\log\log(s)}, \log(q)) & \text{if } \sigma = 0 \end{cases}$ and of degree $(n \cdot d \cdot q)^{\mathrm{polylog}(s)}$.*

   *Moreover, when $\{\mathbb{F}_n\}$ is efficiently constructible, and if we allow each $R_n$ to use* $\mathsf{p}^{\mathsf{th}}$ root gates*, which are gates that map a field element $\alpha \in \mathbb{F}_q$ to $\alpha^{1/p(n)}$, the network $R_n$ is of degree $(n \cdot d \cdot p)^{\mathrm{polylog}(s)}$.*[33]

The generator networks $\{G_n\}$ and reconstruction networks $\{R_n\}$ are defined and analyzed in Sections 6.1 and 6.2, respectively. Our algorithms will make use of uniform versions of classical algorithms (i.e., versions implementable by uniform arithmetic circuits and networks), in particular the standard depth-reduction procedure of Valiant, Skyum, Berkowitz, and Rackoff [VSBR83] and Kaltofen's algorithm for factorization of arithmetic circuits [Kal89]. The uniform versions of these algorithms are described in Appendix A.

**Remark 6.2** (the reconstruction's degree, and the role of $p^{\text{th}}$ root gates)**.** *The reconstruction procedure underlying Theorem 6.1 computes $p^{th}$ roots of certain elements during its computation. Over a finite field of order $q$ and characteristic $p$, the $p^{th}$ root operation can be implemented via $x \mapsto x^{q/p}$, and this may cause a large (and generally, unavoidable) blowup in the network's degree. One alternative is to simply pay (in degree), as spelled out at the end of the theorem's statement. (We do so in a careful manner that bounds the number of $p^{th}$ root gates along any computational path; see Section 6.2.4.) The reason we also presented the reconstruction as networks using $p^{th}$ root gates is to clarify that this is the only part that uses high-degree computation; that is, other than $p^{th}$ root gates, the reconstruction is of quasipolynomial degree.*

[33] To formally define the degree of a network with $p^{\text{th}}$ root gates, we define it as the degree of the network obtained by replacing each $p^{\text{th}}$ root gate with a dummy gate that computes the identity function $\alpha \mapsto \alpha$. In other words, for this purpose, we consider the $p^{\text{th}}$ root gate as having degree 0.

## 6.1 The generator networks

Loosely speaking, the targeted generator will get input $\vec{\Lambda}$ and apply the construction of the Kabanets–Impagliazzo [KI04] generator to each of the polynomials in the polynomial decomposition of $C_n(\vec{\Lambda})$. We first recall the KI construction, and then describe our targeted generator.

**A reminder: The Kabanets–Impagliazzo generator.** The construction relies on combinatorial designs as in the work of Nisan and Wigderson [NW94]. Let us define these and state some standard constructions.

**Definition 6.3.** *Given $m, n \in \mathbb{N}$ with $n < 2^m$, a family of sets $S_1, \ldots, S_n \subseteq [\ell]$ is said to be a* Nisan–Wigderson $(m, a)$-design *if*

- *for every $1 \leq i \leq n$, $|S_i| = m$, and*
- *for every $1 \leq i < j \leq n$, $|S_i \cap S_j| \leq a$.*

**Lemma 6.4** (see, e.g., [Vad12, Problem 3.2])**.** *For every $m, n, a \in \mathbb{N}$ with $n < 2^m$ and $a < m$, there exists a Nisan–Wigderson $(m, a)$-design $S_1, ..., S_n \subseteq [\ell]$ that can be constructed deterministically in time $\mathrm{poly}(n, m)$, where $\ell = O(n^{1/a} \cdot m^2/a)$.*

**Corollary 6.5.** *For every $m, n \in \mathbb{N}$ with $n < 2^m$, there exists a Nisan–Wigderson $(m, \log n)$-design $S_1, ..., S_n \subseteq [\ell]$ that can be constructed deterministically in time $\mathrm{poly}(n, m)$, where $\ell = O(m^2/\log n)$.*

Given a hard polynomial $g(\vec{x}) \in \mathbb{F}[x_1, ..., x_n]$ and NW-designs as above, the KI construction is as follows.

**Construction 6.6** ([KI04])**.** *Let $n$ and $m$ be integers satisfying $n < 2^m$. Let $g(\vec{x}) \in \mathbb{F}[x_1, \ldots, x_m]$. Let $\bar{S} = S_1, \ldots, S_n \subseteq [\ell]$ be a Nisan–Wigderson $(m, a)$-design. The* Kabanets–Impagliazzo generator *$\mathcal{G}_{KI,g}(\vec{z}) : \mathbb{F}^\ell \to \mathbb{F}^n$ is the polynomial map given by*

$$\mathcal{G}^{\bar{S}}_{KI,g}(z_1, \ldots, z_\ell) \coloneqq (g(\vec{z}|_{S_1}), \ldots, g(\vec{z}|_{S_n})),$$

*where $\vec{z}|_{S_i}$ denotes the restriction of $\vec{z}$ to the variables indexed by $S_i$.*

**The targeted generator.** We now describe our construction of a targeted generator. Starting with a hard $\log^c$-uniform family of arithmetic circuits $\{C_n\}$ of size $s$, we apply uniformity-preserving depth reduction to obtain a uniform family $\{C'_n\}$ of size $\mathrm{poly}(s)$ and depth $\mathrm{polylog}(s)$, and given $\vec{\Lambda} \in \mathbb{F}^n$ we apply Construction 6.6 to each of the polynomials in the polynomial decomposition of $C'_n(\vec{\Lambda})$.

We first present a version of the generator with general parameters, and then present two specific instantiations that will be useful for us. To facilitate parsing the parameters, think of $n$ as the input length to the targeted generator, and of $\bar{n}$ as the number of pseudorandom field elements that the generator outputs. The parameters $h, m, r$ are auxiliary parameters, which will be used for instantiating the polynomial decomposition from Section 4.2. We will either set the parameters to get seed length $O(\log(\bar{n}))$ and reconstruction time proportional to $s^\varepsilon \cdot \bar{n}^{\log\log(s)}$, for a small constant $\varepsilon > 0$; or set the parameters to get seed length $\mathrm{polylog}(s)$ and reconstruction time proportional to $\mathrm{poly}(\bar{n})$.

**Construction 6.7** (KI-based targeted HSG)**.** *Let $\{C_n\}_{n\in\mathbb{N}}$ be a $\log^c$-uniform family of algebraic circuits of size $s(n)$ and degree at most $s(n)$, defined over a feasible sequence of fields $\{\mathbb{F}_n\}_{n\in\mathbb{N}}$, where $\mathbb{F}_n$ has order $q(n) \leq \exp(n^{O(1)})$. Let $\{C'_n\}$ be the $\log^{c'=c+O(1)}$-uniform family of arithmetic circuits of size $s'(n) = \mathrm{poly}(s(n))$ and depth $\Delta(n) = O(\log(s(n))^2)$ obtained from Proposition A.23.*

**Parameters.** *Let $h, m, r, \bar{n}$ be any time-constructible functions (interpreted as functions of $n$) such that $3m + r > \log(\bar{n})$, and one of the following holds:*

- *$h = 2$ and $m = \lceil \log(s') \rceil$ and $r = \log(s')^{c'}$, or*
- *$h$ is a power of two, and $m$ is the minimal integer such that $h^m \geq s'$, and $r = 0$.*

**Generator.** *Given $\vec{\Lambda} \in \mathbb{F}_n^n$, let $\{f_{i,j}\}$ be the $(h,m,r)$-polynomial decomposition of $C'_n$ on input $\vec{\Lambda}$ obtained via Proposition 4.6, and consider all polynomials as having $3m+r$ variables (i.e., add dummy variables if necessary). Let $\bar{S} = S_1, \ldots, S_{\bar{n}} \subseteq [\ell = O((3m+r)^2/\log(\bar{n}))]$ be the Nisan–Wigderson $(3m+r, \log \bar{n})$-design obtained via Corollary 6.5. Define the polynomial map $\mathcal{G}_{\vec{\Lambda}}^{(h,m,r,\bar{n})} : \mathbb{F}_n^{\ell+2} \to \mathbb{F}_n^{\bar{n}}$ (called the* KI-based targeted HSG *for $\vec{\Lambda}$) as follows:*

$$\mathcal{G}_{\vec{\Lambda}}^{(h,m,r,\bar{n})}(y_1, y_2, \vec{z}) \coloneqq \sum_{i=1}^{\Delta(n)} \sum_{j=0}^{2m+r} L_{i,j}(y_1, y_2) \cdot \mathcal{G}_{KI, f_{i,j}}^{\bar{S}}(\vec{z}), \tag{6.1}$$

*where $L_{i,j}(y_1, y_2)$ is the Lagrange interpolation polynomial, defined as*[34]

$$L_{i,j}(y_1, y_2) = \begin{cases} 1, & \text{if } y_1 = \phi_1(i) \text{ and } y_2 = \phi_2(j+1), \\ 0, & \text{if } (y_1, y_2) \in \phi_1([\Delta(n)]) \times \phi_2([2m+r+1]) \setminus \{(\phi_1(i), \phi_2(j+1))\}, \end{cases}$$

*where $\phi_1$ and $\phi_2$ are enumerators for $\Delta(n)$ and $t = 2m+r+1$, respectively.*

**Corollary 6.8** (KI-based targeted HSG, two useful instantiations)**.** *Let $\{C_n\}_{n\in\mathbb{N}}$ and $\{C'_n\}$ be as in Construction 6.7, and let $\bar{n} = \bar{n}(n) < n$ be time-constructible. Then,*

1. *Instantiating Construction 6.7 with $h = 2, m = \lceil \log(s) \rceil, r = \log(s')^{c'}$, we obtain a generator $\mathcal{G}_{\vec{\Lambda}}^{(h,m,r,\bar{n})}$ with seed length $\ell = \text{polylog}(s)$.*
2. *Instantiating Construction 6.7 with $h = 2^{\log(s')/O(\log(\bar{n}))}, m = O(\log(\bar{n})), r = 0$, we obtain a generator $\mathcal{G}_{\vec{\Lambda}}^{(h,m,r,\bar{n})}$ with seed length $\ell = O(\log(\bar{n}))$.*

The seed length in the second case above is strictly better (because $\bar{n} < 2^m < s$), but the choice of parameters $h, m, r$ affects the reconstruction time, so the second case is not always preferable to the first. (In the first case, the reconstruction time will be proportional to $\text{poly}(\bar{n})$, whereas in the second case the reconstruction time is proportional to $s^{\varepsilon} \cdot \bar{n}^{\log\log(s)}$; see Section 6.2.)

**Remark 6.9.** *The notation $\mathcal{G}_{\vec{\Lambda}}^{(h,m,r,\bar{n})}$ suppresses the fact that the generator also depends on the choice of a $\log^c$-uniform family of arithmetic circuits $\{C_n\}$ and the choice of a Nisan–Wigderson design. We will drop the superscript $(h,m,r,\bar{n})$ whenever the choice of these functions is clear from context.*

Let us now analyze the complexity of the generator. We stress that for a fixed $\vec{\Lambda}$, the generator $\mathcal{G}_{\vec{\Lambda}}(y_1, y_2, \vec{z})$ should be viewed as a polynomial map from $\mathbb{F}_n^{\ell+2} \to \mathbb{F}_n^{\bar{n}}$. This viewpoint is indeed necessary for repeated self-composition of this generator in order to reduce the final seed length to solve PIT.

**Lemma 6.10** (the complexity of the targeted generator)**.** *Let $\{C_n\}, h, m, r, \bar{n}$ be as in the statement of Construction 6.7. Then,*

1. *There is a* P*-uniform family of arithmetic networks of size $\text{poly}(s(n))$ that gets input $\vec{\Lambda}$ and outputs the description of the arithmetic circuit $\mathcal{G}_{\vec{\Lambda}}^{(h,m,r,\bar{n})}$.*
2. *The degree of $\mathcal{G}_{\vec{\Lambda}}^{(h,m,r,\bar{n})}$ is at most $n \cdot h(n) \cdot \text{polylog}(s(n))$.*

Note that the bounds on the complexity of the generator in Theorem 6.1, with $\sigma \in \{0,1\}$, follow from the two instantiations in Corollary 6.8 and from Lemma 6.10.

*Proof of Lemma 6.10.* We stress that there are three algorithms involved in the claim. We wish to construct a machine $M$ that prints an arithmetic network $N$, where $N$ gets input $\vec{\Lambda}$ and computes the description of an arithmetic circuit $G_{\vec{\Lambda}}$ that computes $\mathcal{G}_{\vec{\Lambda}}$.

We first describe $G_{\vec{\Lambda}}$, and then explain how $N(\vec{\Lambda})$ computes its description (and how $M(1^n)$ prints a description of $N$).

[34] The argument for $\phi_2$ is $j+1$ instead of the more natural $j$ because the domain of enumerators as given in Definition 4.5 starts from 1, not 0.

**The circuit $G_{\vec{\Lambda}}$.** The circuit $G_{\vec{\Lambda}}$ has hard-wired arithmetic circuits for $f_{i,j}$ for all $i \in [\Delta]$ and $j \in \{0,...,2m+r\}$. It also has hard-wired arithmetic circuits for $L_{i,j}$, for all $i,j$. It also has hard-wired the design $\bar{S}$. By Construction 6.6, using the hard-wired $\bar{S}$ and circuits for $f_{i,j}$, the circuit $G_{\vec{\Lambda}}$ can compute the mapping $\vec{z} \mapsto \mathcal{G}^{\bar{S}}_{KI,f_{i,j}}(\vec{z})$. Thus, the circuit $G_{\vec{\Lambda}}$ implements the functionality in Eq. (6.1) in the straightforward way. The size of $G_{\vec{\Lambda}}$ is at most

$$O(\log(s)^2 \cdot m \cdot r) \cdot \max_{i,j}\{\text{size}(L_{i,j})\} \cdot \max_{i,j}\{\text{size}(f_{i,j})\} + |\bar{S}|,$$

and its degree is at most

$$\max_{i,j}\{\deg(L_{i,j}) \cdot \deg(f_{i,j})\}.$$

**The polynomials $L_{i,j}$.** Let us present an explicit expression for the polynomials $L_{i,j}$. For $i \in [\Delta]$, define the univariate

$$Q_{i,1}(y) \coloneqq \prod_{k \in [\Delta] \setminus \{i\}} \frac{y - \phi_1(k)}{\phi_1(i) - \phi_1(k)},$$

and for $j \in \{0,...,2m+r\}$, define the univariate polynomials

$$Q_{j,2}(y) \coloneqq \prod_{k \in \{0,...,2m+r\} \setminus \{j\}} \frac{y - \phi_2(k+1)}{\phi_2(j+1) - \phi_2(k+1)}.$$

Then, for any $i \in [\Delta]$ and $j \in \{0,...,2m+r\}$, we define $L_{i,j}(y_1,y_2) \coloneqq Q_{i,1}(y_1) \cdot Q_{j,2}(y_2)$. The degree of $L_{i,j}$ is $O(\Delta \cdot m \cdot r)$, and it can be implemented in size $\text{poly}(m,r,\log(s)) = \text{polylog}(s)$. To see that there is a $\mathsf{P}$-uniform arithmetic network of size $\text{polylog}(s)$ that on any $n$-element input $\vec{\Lambda}$ computes a description of $L_{i,j}$, note that the machine that constructs the network can simply hard-wire the description of $L_{i,j}$ into the network, as this description does not depend on the input $\vec{\Lambda}$ given to the network. The Turing machine that prints this network can print the constants $\{\phi_1(k) : k \in [\Delta]\}$ and $\{\phi_2(k+1) : k \in \{0,\ldots,2m+r\}\}$ in $\text{poly}(n,\Delta,2m+r)$ time by invoking the Turing machines underlying the enumerators $\phi_1$ and $\phi_2$.

**The circuits for $f_{i,j}$.** By item 2 of Proposition 4.6, there is a $\mathsf{P}$-uniform family of arithmetic networks of size $\text{poly}(s)$ that takes input $\vec{\Lambda}$ and prints descriptions of arithmetic circuits of size $\text{poly}(s)$ and degree at most $n \cdot h \cdot \text{polylog}(s)$ for $\{f_0\} \cup \{f_{i,j}\}_{i,j}$.

**The design $\bar{S}$.** Recall that $\bar{S}$ is obtained via Corollary 6.5. In particular, the machine $M$ can compute this design and hard-wire it into $N$, as the designs do not depend on the input $\vec{\Lambda}$ to $N$. As a consequence, the network $N$ can hard-wire the design into the arithmetic circuit $G_{\vec{\Lambda}}$. The size of the design's description is at most $\bar{n} \cdot \ell = O(m \cdot r) < \text{polylog}(s)$.

**Putting things together.** Combining the constructions of the $L_{i,j}$'s, the $f_{i,j}$'s, and $\bar{S}$, we obtain a machine $M$ running in time $\text{poly}(s)$ and printing a network of size $\text{poly}(s)$, which maps $\vec{\Lambda}$ to $G_{\vec{\Lambda}}$ of size $\text{poly}(s)$ and degree $n \cdot h \cdot \text{polylog}(s)$. □

## 6.2 The reconstruction procedure

We now describe the reconstruction procedure for our generator. We first describe a version corresponding to the "mode" $\sigma = 1$, in which case the reconstruction time is roughly $\text{poly}(n,d,\log(s))$. Then we explain how to obtain the version for "mode" $\sigma = 0$, which has higher reconstruction time (see Proposition 6.17). In both cases, we assume that $\mathbb{F}$ is efficiently constructible, and we allow $p^{\text{th}}$ root gates; in Section 6.2.4 we explain how to obtain the claimed reconstruction networks if $\mathbb{F}$ is not efficiently constructible and/or when disallowing $p^{\text{th}}$ root gates.

### 6.2.1 The reconstruction in $\sigma = 1$ mode

In the statement below we keep $h, m, r$ as parameters, despite the fact that they are fixed to particular values in the $\sigma = 1$ "mode" (i.e., to $h = 2, m = \log(s), r = \text{polylog}(s)$). The reason is to present the construction in a way that will be easier to generalize later for the $\sigma = 0$ "mode".

**Proposition 6.11** (the reconstruction for Theorem 6.1 with $\sigma = 1$)**.** *Let $\{C_n\}_{n\in\mathbb{N}}$ be a $\log^c$-uniform family of algebraic circuits of size $s(n)$ having $n$ output gates, defined over a feasible sequence of fields $\{\mathbb{F}_n\}_{n\in\mathbb{N}}$, where each $\mathbb{F}_n$ has order $q(n) \leq \exp(n^{O(1)})$ and characteristic $p(n)$. Let $d(n)$ be time-constructible, let $\vec{\Lambda} \in \mathbb{F}_n^n$, and let $\mathcal{G}_{\vec{\Lambda}}^{(h,m,r,\bar{n})}$ be the generator described in Corollary 6.8, with the $r > 0$ instantiation. Then, there is a* P*-uniform family $R = \{R_n\}_{n\in\mathbb{N}}$ of randomized arithmetic networks that satisfies the following.*

1. *The input to $R_n$ is a vector $\vec{\Lambda} \in \mathbb{F}_n^n$ that is the universal encoding of an arithmetic circuit $D$ that computes an $\bar{n}$-variate, degree-$d$ nonzero polynomial and satisfies $D \circ \mathcal{G}_{\vec{\Lambda}}^{(h,m,r,\bar{n})}(y_1, y_2, \vec{z}) = 0$.*
2. *The size[35] of the network $R_n$ is bounded by $\text{poly}(n, d, \bar{n}^{\log h}, \log(s), \log(q))$.*
3. *The degree of $R_n$ is at most $(n \cdot d \cdot h \cdot p)^{\text{polylog}(s)}$ (allowing $p^{th}$ root gates, and assuming that $\mathbb{F}$ is efficiently constructible).*
4. *$R_n(\vec{\Lambda})$ computes the vector $C_n(\vec{\Lambda}) \in \mathbb{F}_n^n$ with error degree at most $\text{poly}(d, h, \bar{n}, \log(s))$.*

The rest of this section is devoted to the proof of Proposition 6.11. Recall that Construction 6.7 transforms $\{C_n\}$ into $\{C'_n\}$ of depth $\Delta = O((\log s)^2)$ and size $s' = \text{poly}(s)$, and $\mathcal{G}_{\vec{\Lambda}}$ is obtained by applying Construction 6.6 to the polynomials from the polynomial decomposition $\{f_{i,j}\}$ of $C'_n(\vec{\Lambda})$. The network $R_n(\vec{\Lambda})$ will compute $C'_n(\vec{\Lambda}) = C_n(\vec{\Lambda})$. For notational convenience, in what follows we will drop the prime symbol in $C'_n$ and $s'$, denoting them by $C_n$ and $s$, respectively.

Before we describe the construction formally, we provide a brief sketch of its various components. On input $\vec{\Lambda}$, the network efficiently constructs the description of a circuit computing the polynomial $f_\Delta = f_{\Delta,t}$. As this polynomial faithfully represents the output gate values of $C_n(\vec{\Lambda})$ (see item 1 of Proposition 4.6), the network eventually computes $C_n(\vec{\Lambda})$ by simply evaluating $f_\Delta$ over the first $n$ many vectors (in lexicographical order) in $H^m$. The main idea is that $R_n$ iteratively, for $i = 0, \ldots, \Delta$, constructs a small arithmetic circuit for $f_i$. At a finer level, $R_n$ constructs a circuit for $f_i$ using query access to a circuit for $f_{i-1}$ by sequentially constructing circuits for $f_{i,j}$ from $f_{i,j-1}$ for $j = 1, \ldots, t$.

We will now describe this arithmetic network, while accounting both for the complexity of implementing each iteration (i.e., the complexity of the uniform arithmetic network), and for the complexity of the arithmetic circuits whose descriptions are produced by each iteration. Note that the assumption $D \circ \mathcal{G}_{\vec{\Lambda}}^{(h,m,r,\bar{n})}(y_1, y_2, \vec{z}) = 0$ implies that $D \circ \mathcal{G}_{KI,f_{i,j}}(\vec{z}) = 0$ for every $\vec{z} \in \mathbb{F}^\ell$, and for all $i \in [\Delta]$ and $j \in \{0, ..., 2m + r\}$.

To begin, $R_n$ employs the P-uniform family of arithmetic networks from Lemma A.2 to convert $\vec{\Lambda}$ to a standard description of $D$ of $\text{poly}(n, d)$ size. In the rest of this section, we abuse notation in the following way: whenever we refer to $\vec{\Lambda}$, we mean this standard description of $D$ of $\text{poly}(n, d)$ size. To start the iterative procedure, by Item 2a in Proposition 4.6, the arithmetic network can produce an arithmetic circuit of size $O(n \cdot m \cdot h \cdot \text{polylog}(s))$ and degree $n \cdot h \cdot \text{polylog}(s)$ that computes $f_0$. Then the arithmetic network successively uses the network given by the following lemma.

**Lemma 6.12** (One iteration: moving from a circuit for $f_{i,j-1}$ to a circuit for $f_{i,j}$)**.** *There is a* P*-uniform family $\mathcal{N} = \{\mathcal{N}_n\}$ of randomized arithmetic networks with $p^{th}$ root gates that when given as input $\vec{\Lambda}$, a natural number $\beta \in \mathbb{N}$, and a standard description of a size $s_{i,j-1}$, degree $d_{i,j-1}$ arithmetic circuit computing $f_{i,j-1}^{p^\beta}$, outputs a natural number $\gamma \in \mathbb{N}$ and a standard description of an arithmetic circuit computing $f_{i,j}^{p^\gamma}$ of size $\text{poly}(n, d, \bar{n}^{\log h})$ and degree at most $\text{poly}(d, h, \log \bar{n})$. The size of $\mathcal{N}_n$ is at most $\text{poly}(n, d, \bar{n}^{\log h}, s_{i,j-1}, d_{i,j-1}, \log(s), \log(q))$, its degree is at most $\text{poly}(\log s, d, h, p, d_{i,j-1})$ (using $p^{th}$ root gates) and its error degree is at most $\text{poly}(\bar{n}, h, d, \log(s), d_{i,j-1})$.*

[35] In the statement of this proposition, the notation $\text{poly}(T)$ indicates a bound of $T^{cK}$, where $K$ is a universal constant and $c$ is the given constant that parameterizes the $\log^c$-uniformity of $\{C_n\}$.

*Furthermore, there is another* $\mathsf{P}$*-uniform family* $\mathcal{N}' = \{\mathcal{N}'_n\}$ *of randomized arithmetic networks with* $p^{th}$ *root gates that, when given as input* $\vec{\Lambda}$*, a natural number* $\beta \in \mathbb{N}$*, and the standard description of a size* $s_{i-1,2m+r}$*, degree* $d_{i-1,2m+r}$ *arithmetic circuit computing* $f_{i-1,2m+r}^{p^\beta}$*, outputs a natural number* $\gamma \in \mathbb{N}$ *and a standard description of an arithmetic circuit computing* $f_{i,0}^{p^\gamma}$ *of size* $\mathrm{poly}(n, d, \bar{n}^{\log h})$ *and degree at most* $\mathrm{poly}(d, h, \log \bar{n})$*. The size of* $\mathcal{N}'_n$ *is* $\mathrm{poly}(n, d, \bar{n}^{\log h}, s_{i-1,2m+r}, d_{i-1,2m+r}, \log(s), \log(q))$*, its degree is at most* $\mathrm{poly}(h, d, \log(s), p, d_{i-1,2m+r})$ *(using* $p^{th}$ *root gates) and its error degree is at most* $\mathrm{poly}(\bar{n}, h, d, \log(s), d_{i-1,2m+r})$*.*

We defer the proof of Lemma 6.12 to Section 6.2.2. The network $R_n$ makes repeated use of the networks $\mathcal{N}$ and $\mathcal{N}'$ from Lemma 6.12. After computing the description of a circuit for $f_0$ as explained above, $R_n$ uses the network $\mathcal{N}'$ (applied with $i = 1$) to output the description of a $p^{\text{th}}$ power of $f_{1,0}$. This description is then fed into the network $\mathcal{N}$ (applied with $i = 1$ and $j = 1$), which in turn then outputs the description of a $p^{\text{th}}$ power of $f_{1,1}$. This description is, in turn, fed into the network $\mathcal{N}$ (now applied with $i = 1$ and $j = 2$), and so on until $j = 2m + r$, at which point $R_n$ observes that $f_{1,2m+r} = f_1$ and subsequently invokes the network $\mathcal{N}'$ again (now applied with $i = 2$). This process repeats until $R_n$ eventually computes the description of a $p^{\text{th}}$ power of $f_\Delta = f_{\Delta,2m+r}$. Finally, $R_n$ uses Proposition 3.24 and $p^{\text{th}}$ root gates to query $f_\Delta$ on (the first $n$ elements of) the set $H^m$, and returns these elements as a vector $\vec{\alpha}$ (which we claim is equal to $C_n(\vec{\Lambda})$ with high probability).

This results in the application of Lemma 6.12 a total of $(2m + r) \cdot \Delta$ times, and the application of the universal arithmetic network in Proposition 3.24 to evaluate $f_\Delta$ a total of $n$ times. Furthermore, the size of the output circuit produced by each application of Lemma 6.12 is independent of the input circuit for the layer polynomial below, i.e., it follows that each $s_{i,j}$ is bounded by $\mathrm{poly}(n, d, \bar{n}^{\log h})$ and degree $d_{i,j}$ is bounded by $\mathrm{poly}(d, h, \log \bar{n})$. Hence, the overall size of $R_n$ is bounded by

$$(2m + r) \cdot \Delta \cdot \mathrm{poly}(n, d, \bar{n}^{\log h}, \log(q)) < \mathrm{poly}(n, d, \bar{n}^{\log h}, \log(s), \log(q)). \tag{6.2}$$

Next, we analyze the error degree of $R_n$. By Definition 3.16, this amounts to showing that for some integer $e$ bounded by $\mathrm{poly}(\bar{n}, d, h \log s)$, and for every finite set $S \subset \mathbb{F}$, when each random input $r_i$ (collectively represented by $\vec{r}$, say) is chosen independently and uniformly at random from $S$, with probability at least $1 - e/|S|$, we have $R_n(\vec{\Lambda}, \vec{w}, \vec{r}) = C_n(\vec{\Lambda})$, i.e., the vector $\vec{\alpha}$ as defined above equals $C_n(\vec{\Lambda})$.

From the construction of $R_n$, observe that it consists of a repeated composition of the networks $\mathcal{N}_n$ or $\mathcal{N}'_n$, successively instantiated appropriately for the indices $i \in [\Delta]$ and $j \in \{0, \ldots, t\}$, and the (deterministic) network from Item 2a of Proposition 4.6 that outputs a description of $f_0$. Note that since the network in Proposition 3.24 is also deterministic, conditioned on the event that a description of a $p^{\text{th}}$ power of $f_\Delta$ is computed correctly in the final instantiation of $\mathcal{N}$ (i.e., with $i = \Delta$ and $j = t$), the vector $\vec{\alpha}$ as defined above is guaranteed to equal $C_n(\vec{\Lambda})$ by definition. By Proposition 3.21, we can bound the error degree of $R_n$ by its constituent networks. The network produced by item 3 of Proposition 4.6 is deterministic, so it has error degree 0. Each of the $2m + r \leq \mathrm{polylog}(s)$ invocations of $\mathcal{N}_n$ or $\mathcal{N}'_n$ increase the error degree of $R_n$ by $\mathrm{poly}(\bar{n}, \log s, d, h)$. In total, we can bound the error degree of $R_n$ by $t\Delta \cdot \mathrm{poly}(\bar{n}, d, h, \log s) = \mathrm{poly}(\bar{n}, d, h \log s)$.

Finally, we analyze the degree of $R_n$. Using Proposition 3.21, we can again bound the degree of $R_n$ by its constituent networks. The network produced by item 2a of Proposition 4.6 has degree $n \cdot h \cdot \mathrm{polylog}(s)$. Each of the $2m + r \leq \mathrm{polylog}(s)$ invocations of $\mathcal{N}_n$ or $\mathcal{N}'_n$ multiply the degree of $R_n$ by $\mathrm{poly}(h, d, p, \log s)$. In total, we can bound the degree of $R_n$ by $(h \cdot d \cdot p \cdot \log s)^{O(t\Delta)} = (n \cdot d \cdot h \cdot p)^{\mathrm{polylog}(s)}$.

### 6.2.2 A single iteration: Proof of Lemma 6.12

We first prove the main part of the statement, and then explain how to deduce the "furthermore" part, relying on essentially the same argument.

Relabel the indices $i$ and $j$ in the lemma statement as $i'$ and $j'$, respectively, so that $i$ and $j$ are now free to use for other purposes in the remainder of the proof of this lemma. The indices $i'$ and $j'$ are henceforth assumed to be fixed and we call $f_{i',j'}$ as simply $f$. We also relabel the size and degree of the circuit for $f_{i',j'-1}$ that is given to the network by replacing $s_{i',j'-1}$ and $d_{i',j'-1}$ with $s'$ and $d'$, respectively.

Recall that in Construction 6.7 we added dummy variables to all the $f_{i',j'}$'s, so that they are all $(3m + r)$-variate polynomials. We will repeatedly use the fact that the network can evaluate $f$ at any given point in

$\mathbb{F}_n^{3m+r}$ using a circuit for $f_{i',j'-1}^{p^\beta}$ and the downward self-reducibility of the decomposition $\{f_{i,j}\}$. Let us state this formally.

**Subclaim 6.13.** *There is a* P*-uniform arithmetic network that gets as input* $(i', j') \in [\Delta] \times \{0, ..., 2m + r\}$*, a description of an arithmetic circuit of size* $s$ *and degree* $d$ *that computes* $f_{i',j'-1}^{p^\beta}$ *for some* $\beta \in \mathbb{N}$*, and a vector* $\vec{\alpha} \in \mathbb{F}_n^{3m+r}$*, and outputs the value of* $f(\vec{\alpha})^{p^\beta}$*. The size of this network is bounded by* $h \cdot \mathrm{poly}(m, r, s, d)$ *and its degree is bounded by* $\mathrm{poly}(m, r, d)$*.*

**Proof.** The network uses Proposition 3.24 to evaluate the given circuit for $f_{i',j'-1}^{p^\beta}$ and then applies the downward self-reducibility of the polynomial decomposition $\{f_{i,j}\}$ (i.e., Item 4 of Proposition 4.6) to compute $f_{i',j'}^{p^\beta}$. Since downward self-reducibility of the polynomial decomposition $\{f_{i,j}\}$ is only stated and proved for the polynomials $f_{i',j'-1}$ and $f$, we need to check that downward self-reducibility remains valid when we are working with $p^{\text{th}}$ powers of these polynomials. Over a field of characteristic $p > 0$, the $p^{\text{th}}$ power map is linear. That is, for any $a, b \in \mathbb{F}$, we have $(a + b)^p = a^p + b^p$. Using this identity, we can distribute the $p^{\text{th}}$ power over the sums appearing in the downward self-reducibility property.

In more detail, recall the sumcheck relation from Item 5 of Definition 4.4, which states

$$f(\alpha_1, \ldots, \alpha_{3m+r-j'}) = \sum_{\gamma \in H} f_{i',j'-1}(\alpha_1, \ldots, \alpha_{3m+r-j'}, \gamma).$$

Taking $p^\beta$-th powers of both sides and repeatedly using the fact that $(a + b)^p = a^p + b^p$, we have

$$f(\alpha_1, \ldots, \alpha_{3m+r-j'})^{p^\beta} = \sum_{\gamma \in H} f_{i',j'-1}(\alpha_1, \ldots, \alpha_{3m+r-j'}, \gamma)^{p^\beta}.$$

Thus, given a circuit that computes $f_{i',j'-1}^{p^\beta}$, we can correctly compute the value of $f^{p^\beta}$ at any point of our choosing.

Each application of the network of Proposition 3.24 increases the size of our network by an additive $\mathrm{poly}(m, r, s, d)$ and costs degree $\mathrm{poly}(m, r, d)$. We apply this network $h$ times in parallel and add the results together, so the size and degree of the whole network are bounded by $h \cdot \mathrm{poly}(m, r, s, d)$ and $\mathrm{poly}(m, r, d)$, respectively. □

**High-level overview.** For a better exposition, we first describe the ideas involved in the construction of this arithmetic network in a list below and subsequently discuss the formal details of their implementation via a P-uniform family of randomized arithmetic networks.

- The input $\vec{\Lambda}$ describes a nonzero arithmetic circuit $D$ ("distinguisher") satisfying $D \circ \mathcal{G}_{KI,f} = 0$.
- **(Distinguisher to "predictor" transformation.)** For each $0 \le i \le \bar{n}$, define the $i^{\text{th}}$ hybrid circuit

  $$C_i(\vec{x}, \vec{z}) \coloneqq D(f(\vec{z}|_{S_1}), \ldots, f(\vec{z}|_{S_i}), x_{i+1}, \ldots, x_{\bar{n}}).$$

  Here, both $\vec{z} = (z_1, \ldots, z_\ell)$ and $\vec{x} = (x_1, \ldots, x_{\bar{n}})$ are tuples of indeterminates, and the sets $S_1, \ldots, S_{\bar{n}} \subseteq [\ell]$ are the Nisan–Wigderson $(3m + r, \log \bar{n})$-design used in Construction 6.7.

  Since $C_0 = D \neq 0$ and $C_{\bar{n}} = D \circ \mathcal{G}_{KI,f} = 0$, there is $i \in \{0, ..., \bar{n} - 1\}$ for which $C_i(\vec{z}, \vec{x}) \neq 0$ but $C_{i+1}(\vec{z}, \vec{x}) = 0$. Since we do not know which index $i$ satisfies this condition, we will try out each possible value of $i$ in parallel in its own branch in the arithmetic network.
- **(Within branch $i$.)** In what follows, we work with one such fixed index $i$, i.e., within one such branch of the arithmetic network.
  - **(Fixing irrelevant variables.)** Call $\vec{z}|_{S_{i+1}}$ and $x_{i+1}$ the relevant variables ($3m + r + 1$ in all) and the remaining variables the irrelevant variables. Note that the polynomial computed by $C_i$ depends on at most $\ell - (3m + r) + \bar{n} - (i + 1) = O(\bar{n} + \ell)$ irrelevant variables. We use the given random set of arithmetic inputs to the network in order to fix the irrelevant variables to field constants in

$\mathbb{F}_n$.[36] Let us denote this assignment to irrelevant variables by the vector $\vec{\alpha}$. The notation $(\vec{z},\vec{x})\circ\vec{\alpha}$ refers to the variable vector in which the positions corresponding to the irrelevant $z$ variables (i.e., the indices not in $S_{i+1}$), along with the irrelevant $x$ variables (i.e., all $x$ variables except for $x_{i+1}$) are specified according to $\vec{\alpha}$, while the relevant variables are left unset. Define

$$C'_i = C'_i(\vec{z}|_{S_{i+1}}, x_{i+1}) \coloneqq C_i((\vec{z},\vec{x})\circ\vec{\alpha})$$

and

$$C'_{i+1} = C'_{i+1}(\vec{z}|_{S_{i+1}}, x_{i+1}) \coloneqq C_{i+1}((\vec{z},\vec{x})\circ\vec{\alpha}),$$

i.e., the circuits $C'_i$ and $C'_{i+1}$ are obtained from $C_i$ and $C_{i+1}$, respectively, after the partial substitution specified by $\vec{\alpha}$.

Note that $C'_{i+1}$ is obtained as a restriction of $C_{i+1}$ and is therefore the zero polynomial in the branch in which our guess for $i$ is correct. On the other hand, since $C_i$ is a nonzero polynomial, and since the assignment to the irrelevant variables comes from the random set of arithmetic inputs, with high probability $C'_i$ is a nonzero polynomial as well (see Subclaim 6.14).

– **(Efficiently constructing a circuit for $C'_i$.)** Since the polynomial obtained by replacing $x_{i+1}$ in $C'_i$ by the polynomial $f(\vec{z}|_{S_{i+1}})$ is the zero polynomial, it follows from Corollary 3.4 that $x_{i+1} - f(\vec{z}|_{S_{i+1}})$ is a factor of the polynomial computed by the circuit $C'_i$. Our goal is to construct a circuit for $C'_i$ and then use Corollary A.21 to factor this circuit, obtaining a circuit that computes $f(\vec{z}|_{S_{i+1}})$. Indeed, one caveat is that Corollary A.21 will yield a circuit that computes $f^{p^e}$ for some $e\in\mathbb{N}$, rather than $f$ itself.

To construct a circuit for $C'_i$, it suffices for the network to construct a circuit for the mapping $\vec{z}|_{S_j\cap S_{i+1}} \mapsto f(\vec{z}|_{S_j}\circ\vec{\alpha})$, for each $1\le j\le i$.[37]

To do so, first note that by Item (5) of Proposition 4.6, $f^{(j)}_{\vec{\alpha}}(\vec{z}|_{S_j\cap S_{i+1}}) \coloneqq f(\vec{z}|_{S_j}\circ\vec{\alpha})$ is a polynomial of individual degree at most $6h$ and total degree at most $h\cdot\mathrm{polylog}(s)$ over $|S_j\cap S_{i+1}|\le\log\bar{n}$ variables. As such, it can be interpolated using at most $(6h)^{\log\bar{n}}$ evaluations of $f$ on a grid specified by a fixed, explicit set $T\subseteq\mathbb{F}_n$ of size $6h$ (for which we can use an enumerator for the function $6h$) More precisely, we can explicitly write $f^{(j)}_{\vec{\alpha}}$ using multivariate interpolation as

$$f^{(j)}_{\vec{\alpha}}(\vec{z}|_{S_j\cap S_{i+1}}) \coloneqq \sum_{\vec{\beta}\in T^{S_j\cap S_{i+1}}} f(\vec{\beta}\circ\vec{\alpha}) \prod_{k\in S_j\cap S_{i+1}}\prod_{\beta\in T\setminus\{\beta_k\}} \frac{z_k-\beta}{\beta_k-\beta}, \tag{6.3}$$

where the evaluations $f(\vec{\beta}\circ\vec{\alpha})$ can be computed using Subclaim 6.13 and $p^{\text{th}}$ root gates.

Doing this for every $j\in[i]$ and plugging in the circuit $D$, we obtain a circuit for $C'_i$ whose size is bounded by

$$\mathrm{poly}(|T|^{\log\bar{n}+1}\cdot\log\bar{n}\cdot\bar{n}\cdot n) = \mathrm{poly}(n,\bar{n},h^{\log\bar{n}}) \le \mathrm{poly}(n,\bar{n}^{\log h}). \tag{6.4}$$

and degree is bounded by

$$O(d|T|\log\bar{n}) = O(dh\log\bar{n}). \tag{6.5}$$

– **(Kaltofen's factorization.)** We now invoke the network of Corollary A.21. This network takes as input a description of $C'_i$ and computes a list of (descriptions of) circuits $C_{i,1},\ldots,C_{i,t_i}$ and a list of natural numbers $\beta_{i,1},\ldots,\beta_{i,t_i}\in\mathbb{N}$, where $t_i\le\deg(C'_i)$. We are guaranteed that for every polynomial $g$ such that $C'_i(\vec{z}|_{S_{i+1}}, g(\vec{z}|_{S_{i+1}}))$ is the identically zero polynomial, at least one of the

[36] We stress that in each independent branch of the network, we will use different random arithmetic inputs. This ensures independence across branches.

[37] To see this, recall that $C'_i$ gets as input the variables $\vec{z}\restriction_{S_{i+1}}$ and $x_{i+1}$, and its goal is to output $D$ on $(f(\vec{z}\restriction_{S_1},\ldots,f(\vec{z}\restriction_{S_i}),x_{i+1},x_{i+2},\ldots,x_{\bar{n}})$, where $x_{i+2},\ldots,x_{\bar{n}}$ are fixed (according to $\vec{\alpha}$, which is hard-wired into $C'_i$) and the values in $\vec{z}$ with indices not in $S_{i+1}$ are also fixed (according to $\vec{\alpha}$). Thus, it suffices for $C'_i$ to be able to compute $\left\{f(\vec{z}\restriction_{S_j})\right\}_{j\le i}$ as a function of $\vec{z}\restriction_{S_j\cap S_{i+1}}$ (and of the hard-wired $\vec{\alpha}$).

circuits $C_{i,k}$ computes the polynomial $g^{p^{\beta_{i,k}}}$. In particular, one of the circuits in this list computes $f^{p^e}$ for some $e \in \mathbb{N}$. For each $k \in [t_i]$, the size of the circuit $C_{i,k}$ is bounded by

$$\mathrm{poly}(|C'_i|, \deg(C'_i), d, \bar{n}) < \mathrm{poly}(n, d, \bar{n}^{\log h}), \qquad \text{(by Eqs. (6.4), (6.5))}$$

and the degree of each $C_{i,k}$ is bounded by $\deg(C'_i) < dh \log(\bar{n})$ (by Eq. (6.5))

- (**Weeding candidate circuits.**)

  We know that for some choice of $i \in [\bar{n} - 1]$ and $k \in [t_i]$, the circuit $C_{i,k}$ computes $f^{p^{\beta_{i,k}}}$. To determine which circuit computes a $p^{\text{th}}$ power of $f$, we iteratively test each one against evaluations of $f^{p^{\beta_{i,k}}}$ using polynomial identity testing.

  Specifically, we evaluate each candidate circuit $C = C_{i,k}$ on a random point, which we choose using $3m + r$ arithmetic inputs taken from the random set of arithmetic values given to the network $R_{i,j}$. We can evaluate $C_{i,k}^{1/p^{\beta_{i,k}}}$ on this point by using Proposition 3.24 and $\beta_{i,k}$ many $p^{\text{th}}$ root gates, and we can evaluate $f$ on this point using Subclaim 6.13. We then use a $\stackrel{?}{=}$ gate (See Definition 3.6) to check if the two evaluations are equal.

  We choose the random evaluation point independently for each candidate circuit. If $C_{i,k}$ computes $f^{p^{\beta_{i,k}}}$, then this circuit always passes the test. Conversely, if $C_{i,k}$ does not compute $f^{p^{\beta_{i,k}}}$, then the Schwartz–Zippel lemma (Lemma 3.1) implies that if each coordinate of the evaluation point is chosen uniformly at random from a subset $S \subseteq \mathbb{F}_n$ of size at least $1000\bar{n}\Delta \deg(C'_i)^2$, then $C_{i,k}$ passes the test with probability at most $1/1000\bar{n}\Delta \deg(C'_i)$. Because there are at most $\bar{n} \deg(C'_i)$ candidate circuits to test, by the union bound, the probability that some circuit erroneously passes the test is at most $1/1000\Delta$. (We will use the iterative step in the current lemma $O(\Delta)$ times, so we require an error bound of $O(1/\Delta)$ here to ensure the overall algorithm has bounded error probability.)

- If one of the candidate circuits passes the test, we have a description of a circuit that computes $f^{p^e}$ for a known integer $e \in \mathbb{N}$. The network outputs the description of this circuit for $f^{p^e}$ together with a binary description of $e$. If more than one candidate circuit passes the test in the previous step, we break ties arbitrarily. (Even for a correct choice of $i$ for the hybrid, it is possible that after choosing the fixing $\vec{\alpha}$ the circuit $C'_i$ computes the zero polynomial, or that the arithmetic network from Corollary A.21 may err.)

**Details of implementation.** We formalize the implementation details using a sequence of claims below. The first claim helps describe the behavior of $\mathcal{N}$ inside a given branch, i.e., corresponding to a given setting of an index $i \in [\bar{n} - 1]$.

**Subclaim 6.14** (evaluating a single branch)**.** *There is a* $\mathsf{P}$*-uniform family* $\mathcal{N}^{\mathsf{in}} = \{\mathcal{N}^{\mathsf{in}}_n\}$ *of randomized arithmetic networks such that* $\mathcal{N}^{\mathsf{in}}_n$ *that has the following properties.*

- *It takes as input* $\vec{\Lambda} \in \mathbb{F}_n^n$*, an integer* $i \in [\bar{n} - 1]$*, an integer* $\beta \in \mathbb{N}$*, and a standard description of a size* $s'$*, degree* $d'$ *arithmetic circuit computing* $f^{p^\beta}_{i',j'-1}$.
- *It outputs standard descriptions of circuits* $C_{i,1}, \ldots, C_{i,t_i}$ *and a list of numbers* $\beta_{i,1}, \ldots, \beta_{i,t_i} \in \mathbb{N}$*, where* $t_i \le d'$ *and each circuit is of size* $\mathrm{poly}(n, \bar{n}^{\log h}, d)$ *and degree at most* $O(d \cdot h \cdot \log(n))$*. Moreover, if the input* $i$ *is so that* $C_i(\vec{z}, \vec{x}) \neq 0$ *but* $C_{i+1}(\vec{z}, \vec{x}) = 0$*, then there exists* $k \in [t_i]$ *for which* $C_{i,k}$ *computes a* $(3m + r)$*-variate polynomial* $g^{p^{\beta_{i,k}}}$ *such that* $C'_i(\vec{z}|_{S_{i+1}}, g(\vec{z}|_{S_{i+1}}))$ *is the zero polynomial, where* $C'_i$ *is a nonzero polynomial that is obtained from* $C_i$ *by a partial substitution to its irrelevant variables.*
- *Finally,* $\mathcal{N}^{\mathsf{in}}_n$ *has size* $\mathrm{poly}(n, d, s', d', \bar{n}^{\log h}, \log(q), \log(s))$*, degree bounded by* $\mathrm{poly}(h, d, p, \log s, d')$ *(using* $p^{th}$ *root gates), and error degree bounded by* $\mathrm{poly}(h, d, \log(s), d')$.

**Subproof.** Recall the three steps of the computation of $\mathcal{N}^{\mathsf{in}}_n$: fixing irrelevant variables, computing a description of $C'_i$, and evaluating the network from Corollary A.21.

Fixing irrelevant variables. To fix irrelevant variables to $\vec{\alpha}$, the Nisan–Wigderson design will be hard-wired into $\mathcal{N}_n^{\mathsf{in}}$. Recall that Construction 6.7 uses the designs from Corollary 6.5, which are computable in time $\mathrm{poly}(\bar{n}, m, r) < \mathrm{poly}(n, \log(s))$. The Turing machine that prints $\mathcal{N}_n^{\mathsf{in}}$ will compute these designs and hard-wire them. Given these designs and access to random elements, the step of fixing irrelevant variables can be implemented by a $\mathsf{P}$-uniform arithmetic network of size $\mathrm{poly}(s')$ and degree $O(1)$.

Computing a description of $C_i'$. Once $\vec{\alpha}$ is determined, the network $\mathcal{N}_n^{\mathsf{in}}$ computes a description of $C_i'$ that takes input $(\vec{z}\restriction_{S_{i+1}}, x_{i+1})$, computes $\left\{f(\vec{z}\restriction_{S_j})\right\}_{j\le i}$ and $x_{i+2}, \ldots, x_{\bar{n}}$ from its inputs and from $\vec{\alpha}$, and evaluates $D$ (i.e., the circuit represented by $\vec{\Lambda}$) on the resulting sequence. In order to compute the description of each $f_{\vec{\alpha}}^{(j)}$, which is defined as the circuit mapping $\vec{z}\restriction_{S_j \cap S_{i+1}} \mapsto f(\vec{z}|_{S_j} \circ \vec{\alpha})$, the network implements the interpolation described in Eq. (6.3), while computing the constants $f(\vec{\beta} \circ \vec{\alpha})$ using the network in Subclaim 6.13 together with $\beta$ applications of a $p^{\text{th}}$ root gate.

We argue that the computation of (a description of) $C_i'$ can be performed by a $\mathsf{P}$-uniform $\mathcal{N}_n^{\mathsf{in}}$ of size $\mathrm{poly}(n, \bar{n}^{\log h}, s', d', \log(s))$ and degree $\mathrm{poly}(d', log(s))$, using $p^{\text{th}}$ root gates. Recall that (as explained above) each $f_{\vec{\alpha}}^{(j)}$ is of size $\mathrm{poly}(n, \bar{n}^{\log h})$ and degree $O(h \cdot \log \bar{n})$; using Proposition 3.23, the network $\mathcal{N}_n^{\mathsf{in}}$ can produce a description for each $f_{\vec{\alpha}}^{(j)}$ of size $\mathrm{poly}(n, \bar{n}^{\log h})$, and the network for doing so is uniform and of size

$$\mathrm{poly}(n, \bar{n}^{\log h}) \cdot h \cdot \mathrm{poly}(m, r, s', d') \le \mathrm{poly}(n, \bar{n}^{\log h}, s', d', \log(s))$$

and degree $\mathrm{poly}(m, r, d') = \mathrm{poly}(d', \log(s))$, using $p^{\text{th}}$ root gates. (We used the fact that the field family $\mathbb{F}$ is feasible to argue that this network is uniform, i.e. a Turing machine can hard-code grid elements from $\mathbb{F}_n$ for the interpolation step.) The wiring of the functionality computing a description of $C_i'$ (from $\vec{\Lambda} = D$, $\vec{\alpha}$, and the circuits for $f_{\vec{\alpha}}^{(j)}$) does not depend on the inputs to $\mathcal{N}_n^{\mathsf{in}}$, and can thus be computed by the uniform machine that constructs $\mathcal{N}_n^{\mathsf{in}}$ and hard-wired into $\mathcal{N}_n^{\mathsf{in}}$ (without increasing the degree, and with size polynomial in $|C_i'|$). Thus, overall, computing the description of $C_i'$ can be done by a uniform network of size $\mathrm{poly}(n, \bar{n}^{\log h}, s', d', \log(s))$ and degree $\mathrm{polylog}(s)$ (relying on Proposition 3.25).

Factoring. The last step is applying Corollary A.21. The bounds on the number $t_i$ of output circuits and on the size and degree of each output circuit were established in the description above (i.e., they follow from Eqs. (6.4) and (6.5) and from Corollary A.21). The size of the network in Corollary A.21 is at most $\mathrm{poly}(|S_{i+1}| + 1, \deg(C_i'), \mathrm{size}(C_i'), \log(q)) = \mathrm{poly}(n, \bar{n}^{\log h}, d, \log(s), \log(q))$ and its degree is at most $\mathrm{poly}(|S_{i+1}| + 1, \deg(C_i'), p) \le \mathrm{poly}(d, h, \log s, p)$, where we again use the bounds on the size and degree of $C_i'$ from Eqs. (6.4) and (6.5).

Error degree. To see the error degree bound, we note that the sources of randomness in $\mathcal{N}_n^{\mathsf{in}}$ are (i) the choice of the vector $\vec{\alpha}$ to fix the irrelevant variables in $C_i$ (which has degree at most $d \cdot \deg(f) \le d \cdot h \cdot \mathrm{polylog}(s)$), and (ii) the execution of the factoring arithmetic network from Corollary A.21, which has error degree $\mathrm{poly}(d, h, \log \bar{n})$. From Lemma 3.1 applied to $C_i(\vec{z}, \vec{x})$ when viewed as a polynomial over the field extension $\mathbb{F}(\vec{z}|_{S_{i+1}}, x_{i+1})$, if $C_i$ is a nonzero polynomial, then fixing the irrelevant variables of $C_i$ to uniformly random elements of a set $S \subseteq \mathbb{F}_n$ results in the zero polynomial with probability $\deg(C_i)/|S|$. It follows that the sub-network that computes the description of $C_i'$ has error degree at most $\deg(C_i) \le d \cdot h \cdot \mathrm{polylog}(s)$. Therefore, the final error degree bound follows from Proposition 3.21. □

The next subclaim describes the behavior of the network $\mathcal{N}$ when testing each candidate circuit $C_{i,k}$ to see if it computes a $p^{\text{th}}$ power of $f$.

**Subclaim 6.15** (testing candidate circuits)**.** *There is a $\mathsf{P}$-uniform randomized arithmetic network $\mathcal{T}$ with $p^{th}$ root gates that gets as input $\vec{\Lambda}$, natural numbers $\beta, \gamma \in \mathbb{N}$, the description of a size $s'$, degree $d'$ arithmetic circuit that computes $f_{i',j'-1}^{p^\beta}$, and the description of a circuit $C$ of size $s''$ and degree at most $d''$ computing a $(3m+r)$-variate polynomial $g^{p^\gamma}$, and outputs a boolean value $b$ which indicates whether $f = g$ as polynomials. Moreover, $\mathcal{T}$ has size $\mathrm{poly}(s', s'', d'', h)$, degree at most $\mathrm{poly}(d', d'')$, and error degree at most $\max(d', d'')$.*

**Subproof.** The network first generates a random point $\vec{\alpha} \in \mathbb{F}^{3m+r}$ using $3m+r$ values from its set of random inputs. The network then evaluates $f^{p^\beta}$ and $g^{p^\gamma}$ at $\vec{\alpha}$. The value of $g(\vec{\alpha})^{p^\gamma}$ is obtained by applying the

network of Proposition 3.24 to the given description of the circuit for $g^{p^\gamma}$. The value of $f(\vec{\alpha})^{p^\beta}$ is computed by applying the network of Subclaim 6.13 to the given circuit that computes $f^{p^\beta}_{i',j'-1}$. To obtain the values of $f(\vec{\alpha})$ and $g(\vec{\alpha})$, the network applies $\beta$ copies of a $p^{\text{th}}$ root gate to the value of $f(\vec{\alpha})^{p^\beta}$, and likewise applies $\gamma$ copies of a $p^{\text{th}}$ root gate to the value of $g(\vec{\alpha})^{p^\gamma}$. Finally, the network feeds the difference of these two arithmetic values into a test gate and returns the value output by this test gate.

The degree and size bounds on the network, as well as the uniformity, follow from the corresponding bounds on the network of Proposition 3.24. The error degree bound follows from Lemma 3.1 and the observation that $\deg(f-g) \leq \max\{d', d''\}$. □

Finally, the subclaim below describes how the network $\mathcal{N}$ aggregates the results of the tests across different branches to choose one circuit that computes a $p^{\text{th}}$ power of $f$.

**Subclaim 6.16** (choosing a circuit according to indicator values)**.** *There is a* P*-uniform arithmetic network $\mathcal{S}$ that takes as input a sequence of descriptions of circuits $C_1, \ldots, C_t$ and boolean values $b_1, \ldots, b_t$, and outputs the description of the circuit $C_i$ with the smallest index $i$ such that $b_i = 1$ (we do not care about the output, if none of the $b_i$ are 1). Moreover, $\mathcal{S}$ has linear size and degree $O(1)$.*

**Subproof.** Note that each description consists of an arithmetic part and a boolean part. Let $(\vec{\alpha}_i, s_i)$ denote the description of $C_i$, where $\vec{\alpha}_i$ is the arithmetic part with each of its coordinates in $\mathbb{F}_n$, and $s_i$ is a boolean string denoting its boolean part. Moreover, using padding by $0 \in \mathbb{F}_n$ (respectively, the boolean value 0) if necessary, let us assume without loss of generality that the length of the arithmetic part (respectively, boolean part) of the description of $C_i$ is the same for every $i \in [t]$.

The task then is to construct an arithmetic network that outputs $(\vec{\alpha}_{i^*}, s_{i^*})$ for the smallest index $i^* \in [t]$ for which $b_i = 1$. Because $i^*$ is unique, the task can be performed independently for the boolean and the arithmetic part. We note that the boolean part of this task is easily performed by a linear sized P-uniform boolean circuit with $\wedge, \vee, \neg$ gates.

Next, we show that there is a simple P-uniform arithmetic network gadget that can output the *first* coordinate of $\vec{\alpha}_{i^*}$. This suffices for the proof of the proposition, as one can then simply repeat the gadget for all the coordinates. The idea is to repeatedly use $\mathsf{select}$ gates to achieve this. Notice that if $t = 2$, then simply $\mathsf{select}(b_1, \alpha_{2,1}, \alpha_{1,1})$ works, and if $t = 3$, then $\mathsf{select}(b_1, \mathsf{select}(b_2, \alpha_{3,1}, \alpha_{2,1}), \alpha_{1,1})$ works. Extrapolating this pattern, we conclude that there is a P-uniform linear sized arithmetic network that solves this task. □

Given these claims, we can now provide a complete description of $\mathcal{N}_n$ as follows. The output of $\mathcal{N}_n(\vec{\Lambda})$ is provided by the output of $\mathcal{S}$, which in turn, receives as input:

(i) The output of the execution of $\mathcal{N}^{\mathsf{in}}_n$ on each of the indices $i \in [\bar{n} - 1]$, i.e., the descriptions of the circuits $C_{i,1}, \ldots, C_{i,t_i}$, and

(ii) A boolean value $b_{i,k}$ corresponding to each circuit $C_{i,k}$, which is in turn obtained as the output of $\mathcal{T}$ when instantiated with the description of $C_{i,k}$.

We stress that every randomized arithmetic network that is used as a subroutine in the description above uses its own block (i.e., a fresh block) of random arithmetic inputs from the original set of random arithmetic inputs that is given to $\mathcal{N}_n$.

**Complexity of $\mathcal{N}_n$.** By Subclaim 6.14, each circuit $C_{i,k}$ fed into the network from Subclaim 6.15 is of size $s'' = \text{poly}(n, \bar{n}^{\log h}, d)$ and degree at most $d'' = O(d \cdot h \cdot \log(\bar{n}))$, and the number of output circuits in each of the $\bar{n}$ branches is at most $O(d \cdot h \cdot \log(\bar{n}))$. Hence, combining Claims 6.14, 6.15, and 6.16, the total size of $\mathcal{N}_n$ is at most $\text{poly}(n, d, \bar{n}^{\log h}, s', d', \log(q), \log(s))$.

To see the degree bound, first note that the degree of $\mathcal{T}$ (the network from Subclaim 6.15) in each of its instantiations is bounded by $\text{poly}(d', d'') = \text{poly}(h, d, \log \bar{n}, d')$. Also, note that since instantiation of $\mathcal{T}$ that is executed in order to compute the tuple of boolean values $(b_{i,k})_{i \in [\bar{n}-1], k \in [t_i]}$ is run in parallel inside $\mathcal{N}_n$, the degree of the (multi-output) sub-network that computes $(b_{i,k})_{i \in [\bar{n}-1], k \in [t_i]}$ is also bounded by the degree of $\mathcal{T}$ in any of its instantiations, and in particular, is also bounded by $\text{poly}(h, d, \log \bar{n}, d')$ (by Proposition 3.19). Next, the degree of $\mathcal{N}^{\mathsf{in}}_n$ (the network from Subclaim 6.14) in each of its instantiations is bounded by $\text{poly}(h, d, p, \log(s), d')$, and as they are each run in parallel inside $\mathcal{N}_n$, the degree of the network

that computes the overall set of candidate circuit descriptions $\{\langle C_{i,k}\rangle | i \in [\bar{n}-1], k \in [t_i]\}$ is also bounded by $\mathrm{poly}(h, d, p, \log(s), d')$ (by Proposition 3.19). Finally, since $\mathcal{N}_n$ is the composition of $\mathcal{S}$ (which has degree $O(1)$) with these networks, the claimed overall degree bound of $\mathrm{poly}(h, d, p, \log(s), d')$ follows from Proposition 3.25.

To see the error degree bound, using the same analysis as above, and once again using Proposition 3.19 and Proposition 3.21 and the three claims above, it follows that the error degree of $\mathcal{N}_n$ is bounded by $\mathrm{poly}(\bar{n}, h, d, \log(s), d')$, where we get the additional factor of $\bar{n}$ from an application of Proposition 3.19, since each of the $\bar{n}$ instantiations of $\mathcal{N}_n^{\mathsf{in}}$ are executed in parallel.

**The "furthermore" part.** The proof is essentially identical to that of the basic case. The only difference is that to prove Subclaim 6.13, we now rely on the downward self-reducibility relation between $f_{i-1,2m+r}$ and $f_{i,0}$ as specified in Item 5 of Definition 4.4. Recalling that $f_{i-1,2m+r}(\vec{w}) = f_{i-1}(\vec{w})$, we have the equality

$$f_{i,0}(\vec{w}, \vec{\sigma}) \coloneqq \begin{cases} \Phi_i(\vec{w}, \sigma_1, \ldots, \sigma_t)\left(f_{i-1}(\sigma_1, \ldots, \sigma_m) + f_{i-1}(\sigma_{m+1}, \ldots, \sigma_{2m})\right), & \text{if } i \text{ is odd} \\ \Phi_i(\vec{w}, \sigma_1, \ldots, \sigma_t)\left(f_{i-1}(\sigma_1, \ldots, \sigma_m) \cdot f_{i-1}(\sigma_{m+1}, \ldots, \sigma_{2m})\right), & \text{if } i \text{ is even.} \end{cases}$$

Taking $p^\beta$-th powers and using the fact that $(a+b)^p = a^p + b^p$ over a field of characteristic $p$, we have

$$f_{i,0}(\vec{w}, \vec{\sigma})^{p^\beta} = \begin{cases} \Phi_i(\vec{w}, \sigma_1, \ldots, \sigma_t)^{p^\beta}\left(f_{i-1}(\sigma_1, \ldots, \sigma_m)^{p^\beta} + f_{i-1}(\sigma_{m+1}, \ldots, \sigma_{2m})^{p^\beta}\right), & \text{if } i \text{ is odd} \\ \Phi_i(\vec{w}, \sigma_1, \ldots, \sigma_t)^{p^\beta}\left(f_{i-1}(\sigma_1, \ldots, \sigma_m)^{p^\beta} \cdot f_{i-1}(\sigma_{m+1}, \ldots, \sigma_{2m})^{p^\beta}\right), & \text{if } i \text{ is even.} \end{cases}$$

Thus, given circuits for $f_{i-1}^{p^\beta}$ and $\Phi_i^{p^\beta}$, we can evaluate $f_{i,0}^{p^\beta}$ at a point of our choosing using a network with similar complexity bounds to Subclaim 6.13. By assumption, we have such a circuit that computes $f_{i-1}^{p^\beta}$. A circuit of size $\mathrm{polylog}(s) \cdot \beta \log(p) \leq \mathrm{polylog}(s, d)$ for $\Phi_i^{p^\beta}$ can be obtained by using repeated squaring to take the $p^\beta$-th power of the circuit for $\Phi_i$ given by Subclaim 4.7.

### 6.2.3 The reconstruction in $\sigma = 0$ mode

We now describe the reconstruction procedure when it is instantiated in "mode" $\sigma = 0$. The proof is exactly identical to that of Proposition 6.17, the only difference being in the parameterization: Instead of using $h = 2, m = \lceil \log(s) \rceil, r = \mathrm{polylog}(s)$ we now use $r = 0$ and arbitrary $h, m$ such that $h^m \geq \mathrm{size}(C'_n)$. This yields the following result:

**Proposition 6.17** (the reconstruction for Theorem 6.1 with $\sigma = 0$)**.** *Let $\{C_n\}_{n\in\mathbb{N}}$ be a $\log^c$-uniform family of multi-output algebraic circuits of size $s(n)$ having $n$ output gates, defined over a feasible sequence of fields $\{\mathbb{F}_n\}_{n\in\mathbb{N}}$, where each $\mathbb{F}_n$ has order $q(n) \leq \exp(n^{O(1)})$, and let $\varepsilon > 0$ be an arbitrarily small constant. Let $\vec{\Lambda} \in \mathbb{F}_{q(n)}^n$, and let $\mathcal{G}_{\vec{\Lambda}}^{(h,m,0,\bar{n})}$ be the generator described in Corollary 6.8, with the $r = 0$ instantiation. Then, there is a $\mathsf{P}$-uniform family of randomized arithmetic networks $R = \{R_n\}_{n\in\mathbb{N}}$ with $p^{th}$ root gates that satisfies the following.*

1. *The input to $R_n$ is a vector $\vec{\Lambda} \in \mathbb{F}_n^n$ that is the universal encoding of an arithmetic circuit $D$ that computes an $\bar{n}$-variate, degree $d$ nonzero polynomial and satisfies $D \circ \mathcal{G}_{\vec{\Lambda}}^{(h,m,r,\bar{n})}(y_1, y_2, \vec{z}) = 0$, and a candidate output $w \in \mathbb{F}_n^n$.*
2. *The size of the network $R_n$ is at most $s^\varepsilon \cdot \mathrm{poly}(n, d, \bar{n}^{\log\log(s)}, \log(s))$.*
3. *The degree of $R_n$ is at most $(n \cdot d \cdot h \cdot p)^{\mathrm{polylog}(s)}$ (allowing $p^{th}$ root gates, and assuming that $\mathbb{F}$ is efficiently constructible).*
4. *$R_n(\vec{\Lambda})$ computes the vector $C_n(\vec{\Lambda}) \in \mathbb{F}_n^n$ with error degree at most $\mathrm{poly}(d, h, \bar{n}, \log(s))$.*

**Proof.** We instantiate Corollary 6.8 with $r = 0$ and with $h = 2^{\log(s')/(c\cdot\log(\bar{n}))}$ and $m = c' \cdot (\log(\bar{n}))$, where $c' > c > 1$ are sufficiently large constants that depend on $\varepsilon > 0$. We follow the proof of Proposition 6.11, using the new values for $m, h, r$. The changes are as follows:

1. The circuit for $f_0$ is of size $O(n \cdot m \cdot h \cdot \mathrm{polylog}(s)) = n \cdot s^{1/O(\log(\bar{n}))} \cdot \mathrm{polylog}(s) < s^\varepsilon \cdot n$.

2. The number of applications of Lemma 6.12 is $O(m \cdot \Delta) < \text{polylog}(s)$.

3. The main difference is in the complexity bounds of Lemma 6.12. Going through Section 6.2.2, when the network constructs a circuit for $C_i'$, the individual degree cannot be assumed to be $O(h)$ anymore, and thus we use the overall degree bound of $h \cdot \text{polylog}(s)$ for $f_{i,j}$ given in Item (5) of Proposition 4.6. The size of each $f_{\vec{\alpha}}^{(j)}$ thus becomes $(h \cdot \text{polylog}(s))^{\log(\bar{n})}$ and its degree becomes $h \cdot \text{polylog}(s) \cdot \log(\bar{n})$.

   Thus, in Subclaim 6.14, the size of each $C_{i,k}$ becomes $\text{poly}(n, (h \cdot \text{polylog}(s))^{\log(\bar{n})})$ and its degree is at most $\text{poly}(h, d, \log(s))$. (And these are the size and degree bounds on the circuit that $\mathcal{N}_n^{\mathsf{in}}$ outputs, since it just outputs one of the $C_{i,k}$'s.) The size and degree of $\mathcal{N}_n^{\mathsf{in}}$ increases appropriately, and thus the size of $\mathcal{N}_n$ is at most $\text{poly}(n, d, (h \cdot \text{polylog}(s))^{\log(\bar{n})}), s', d', \log(s), \log(q))$.

Replacing the calculation in Eq. (6.2), the final size of $R_n$ is now

$$(2m + r) \cdot \Delta \cdot \text{poly}(n, d, (h \cdot \text{polylog}(s))^{\log(\bar{n})}) < \text{poly}(n, d, (h \cdot \text{polylog}(s))^{\log(\bar{n})}, \log(q)),$$

and our claimed size bound follows since, by an appropriate choice of constant $c = c(\varepsilon) > 1$ for $h = 2^{\log(s')/(c \cdot \log(\bar{n}))}$, we have

$$(h \cdot \text{polylog}(s))^{\log(\bar{n})} = h^{\log(\bar{n})} \cdot (\text{polylog}(s))^{\log(\bar{n})} = s^{\varepsilon} \cdot \text{polylog}(s)^{\log(\bar{n})}.$$

Next, to analyze the error degree of $R_n$, we first note that the degree bound on $C_i'$ that is mentioned in (6.5) changes to $\text{poly}(h, d, \log s)$ as the individual degree cannot be assumed to be $O(h)$ anymore, and instead, we use a bound of $h \cdot \text{polylog}(s)$ for $\deg(f_{i,j})$. However, in Subclaim 6.14, the functional degree remains $\text{poly}(h, d, \log s)$ as argued at the end of its proof (note that randomly fixing the irrelevant coordinates still yields an upper bound of $\text{poly}(h, d, \log s)$). Next, we observe that while the size of the output circuits of Subclaim 6.14 increases to $\text{poly}(n, (h \cdot \text{polylog}(s))^{\log(\bar{n})})$, its degree remains at most $\text{poly}(h, d, \log(s))$. This is why subsequent applications of Subclaim 6.15 (the only remaining source of randomness now in $\mathcal{N}_n'$ after Subclaim 6.14) provide identical asymptotic upper bounds on the error degree, as we plug in an identical bound for $d''$.

Finally, we perform a similar analysis to conclude that the degree bound remains identical in each iteration of Lemma 5.11, and hence remains identical overall. □

### 6.2.4 Simulating $p^{\text{th}}$ root gates

The description of the reconstruction procedure so far assumed access to $p^{\text{th}}$ root gates, and assumed that the field family is constructible. We now explain how the construction differs when we do not assume this. In both cases, the main difference is the degree bound on the network.

First, when the field family is not necessarily constructible, the degree bound in Corollary A.21 increases from $\text{poly}(n, d, p)$ to $n^{O(1)} \cdot d^{O(\log q)}$. This increases the degree bounds in Subclaim 6.14 to $(h \cdot d \cdot \log s)^{O(\log q)} \cdot \text{poly}(d')$, which in turn increases the degree bound in Lemma 6.12 to $(h \cdot d \cdot \log s)^{O(\log q)} \cdot \text{poly}(d_{i,j-1})$, and the final degree bound for the reconstruction network becomes $O(n \cdot d \cdot h)^{\log(q) \cdot \text{polylog}(s)}$. (The calculation is identical for the $\sigma = 1$ mode and for the $\sigma = 0$ mode.)

To finish the proof, we now explain how to simulate $p^{\text{th}}$ root gates with standard arithmetic gates. Each gate can be simulate with a degree increase of at most $q$, but naively simulating all gates might increase the network's degree by a factor of $q$ raised to the network's size. We now explain how to avoid this, paying only $q^{\text{polylog}(s)}$.

**A single iteration.** The arithmetic networks $\mathcal{N}$ and $\mathcal{N}'$ constructed in Lemma 6.12 make use of $p^{\text{th}}$ root gates in two places. One is to obtain descriptions of circuits that compute the restricted polynomials $f_{\vec{\alpha}}^{(j)}$. The need to take $p^{\text{th}}$ roots here arises because we interpolate circuits for the $f_{\vec{\alpha}}^{(j)}$ using Equation (6.3), but we only have access to $p^{\text{th}}$ powers of the constants $f(\vec{\beta} \circ \vec{\alpha})$ appearing in that expression. The other use of $p^{\text{th}}$ root gates is in Subclaim 6.15, where we test if a candidate circuit indeed computes a $p^{\text{th}}$ power of $f$.

As remarked following the statement of Theorem 6.1, we can implement a $p^{\beta}$-th root by the operation $x \mapsto x^{q/p^{\beta}}$, where $q$ is the size of the underlying field. In one invocation of Lemma 6.12, these $p^{\text{th}}$ root operations can be batched into two parallel steps: one for the interpolation of circuits that compute the

polynomials $f_{\vec{\alpha}}^{(j)}$, and one to perform testing of candidate circuits. Implementing a single $p^\beta$-th root operation increases the degree of a network by a multiplicative factor of at most $q$. By Proposition 3.19, each batch of $p^{\text{th}}$ root operations performed in parallel can be simulated with the same multiplicative increase of $q$ in the degree. Using Proposition 3.21, simulating two parallel batches of $p^{\text{th}}$ root operations increases the degree of the network by a factor of $q^2$.

**The overall procedure.** The reconstruction procedure invokes Lemma 6.12 a total of $(2m+r)\cdot\Delta \leq$ polylog$(s)$ times in sequence, and uses $p^{\text{th}}$ root gates one last time (in parallel) after the final iteration. By Proposition 3.21, the degree of the reconstruction network is bounded by the product of the degrees of the networks used at each step of the reconstruction. We saw that simulating the $p^{\text{th}}$ root operations in a single application of Lemma 6.12 incurs a degree blowup of $q^2$ in each step, and there are polylog$(s)$ steps, so the degree of the full reconstruction procedure increases by a multiplicative factor of $q^{\text{polylog}(s)}$ when simulating $p^{\text{th}}$ root operations with multiplications in the field.

# 7 Proofs of the main results

Using the targeted generators from Theorem 5.1 and Theorem 6.1, we will now prove the main results. In Section 7.1 we show that hardness for arithmetic networks with PIT gates on almost all inputs is necessary for PIT, and in Section 7.2 we show that very similar hardness assumptions suffice for PIT.

**Uniformity over sequences of finite fields.** Consider an arithmetic circuit $C_n$ over a finite field $\mathbb{F}_n$. We would like to perform basic operations on $C_n$, which may increase its size or its degree, while still considering the modified circuit $C'_n$ as over $\mathbb{F}_n$. In particular, after the modification we may wish to perform PIT on $C'_n$, or to evaluate some hard polynomial on (the description of) $C'_n$. This should be a triviality, but it runs into an artificial complication when considering uniform families of circuits computing PIT or hard polynomials. Specifically, if we consider a uniform family of functions $f=\{f_n\}$ over a sequence of finite fields $\{\mathbb{F}_n\}$, where $f_n$ is defined over $\mathbb{F}_n$, then we cannot (for purely syntactic reasons) start with a description of $C_n$ over $\mathbb{F}_n$, modify it to $C'_n$ of size $n'$ over $\mathbb{F}_n$, and evaluate $f_{n'}$ (or $f_n$) on $C'_n$.

Indeed, this syntactic dependency is artificial and overly strict, and to avoid it, we will consider a more general notion that allows a polynomial gap between the index of the field and the size of the circuit. Specifically, we say that $\mathsf{sf}\colon \mathbb{N}\to\mathbb{N}$ is a shift function if $\mathsf{sf}$ is increasing, and $\mathsf{sf}(n)\leq n$ for all $n\in\mathbb{N}$, and $\mathsf{sf}$ is computable on input $n$ in time poly$(n)$. We say that a shift function is polynomially bounded if $\mathsf{sf}(n)=n^{\Omega(1)}$. For a sequence $\mathbb{F}=\{\mathbb{F}_n\}_{n\in\mathbb{N}}$ of finite fields and a shift function $\mathsf{sf}$, we define $\mathbb{F}^{\mathsf{sf}}=\left\{\mathbb{F}_n^{\mathsf{sf}}\right\}_{n\in\mathbb{N}}$ as the sequence of fields wherein $\mathbb{F}_n^{\mathsf{sf}}=\mathbb{F}_{\mathsf{sf}(n)}$. Then, when we think of applying operations on uniform circuit families over $\mathbb{F}$, we allow any polynomial slack, i.e. we allow circuits of size $n$ over $\mathbb{F}_{\mathsf{sf}(n)}$. For example, for PIT:

**Definition 7.1** (solving PIT over finite fields)**.** *Let $\mathbb{F}=\{\mathbb{F}_n\}$ be a sequence of finite fields. We say that a function*[38] *$f\colon \cup_n(\mathbb{F}_n)^*\to\{0,1\}$ solves PIT over $\mathbb{F}$ if for any polynomially bounded shift function $\mathsf{sf}$, and every sufficiently large $n\in\mathbb{N}$, and every $\vec{\Lambda}\in\mathbb{F}_{\mathsf{sf}(n)}^n$ describing an arithmetic circuit of size and degree at most $n$, the output is $f(\vec{\Lambda})=0$ if and only if the circuit described by $\vec{\Lambda}$ computes the zero polynomial.*

Similarly, when we will consider uniform families of arithmetic circuits (computing hard polynomials), we will also consider them with respect to all polynomially bounded shifts. We stress that both for PIT and for hard polynomials, we consider shifts both in the assumption and in the conclusion of all our results.

## 7.1 A necessary assumption for PIT

The following result asserts that hardness for networks with PIT gates on all but finitely many inputs is necessary for PIT. For convenience, we state the result simultaneously for fields of characteristic zero and for finite fields (i.e., we do not separate the cases).

[38] The notation $\mathbb{F}^*$ here means strings over the alphabet $\mathbb{F}$ of arbitrary length.

**Theorem 7.2** (PIT requires hardness for arithmetic networks with PIT gates on almost all inputs)**.** *Let $\mathbb{F} = \{\mathbb{F}_n\}_{n\in\mathbb{N}}$ be a sequence of fields, where either the fields are all finite, or there is a field $\mathbb{F}$ of characteristic zero such that $\mathbb{F}_n = \mathbb{F}$ for all $n \in \mathbb{N}$. Then, the following conditional implication holds.*

- **Assumption:** *There is a* $\mathsf{P}$*-uniform family of arithmetic networks of polynomial size solving* $\mathsf{PIT}$ *for arithmetic circuits over $\mathbb{F}$.*
- **Conclusion:** *For every $k \in \mathbb{N}$ and every polynomially bounded shift function* $\mathsf{sf}$ *there is a* $\mathsf{P}$*-uniform family of arithmetic networks $\{F_n\}_{n\in\mathbb{N}}$ of polynomial size over $\mathbb{F}^{\mathsf{sf}}$ satisfying the following. For every $n^k$-time-uniform family of arithmetic networks with* $\mathsf{PIT}$ *gates $\{C_n\}$ of size $n^k$ and degree $n^k$ over $\mathbb{F}^{\mathsf{sf}}$, and every sufficiently large $n \in \mathbb{N}$, and every $\vec{\Lambda} \in \mathbb{F}^n_{\mathsf{sf}(n)}$, we have $C_n(\vec{\Lambda}) \neq F_n(\vec{\Lambda})$.*

**Proof.** We prove the claim for the case of finite fields $\mathbb{F} = \{\mathbb{F}_n\}$. The proof for fields of characteristic zero is almost identical, except that the function $\mathsf{sf}$ does not matter in that case.

For $k \in \mathbb{N}$ and a shift function $\mathsf{sf}$, let $A = A_{k,\mathsf{sf}}$ be the following Turing machine. On input $1^n$, the machine $A$ simulates each of the first $n$ Turing machines (according to some predetermined efficient enumeration of TMs), each for $n^k$ steps. These machines print $n$ arithmetic networks with $\mathsf{PIT}$ gates, each with $n$ input gates and $n$ output gates, denoted by $\left\{C^{(i)} \in \mathbb{F}_{\mathsf{sf}(n)}[\vec{x}]\right\}_{i\in[n]}$, where $\vec{x} = (x_1, ..., x_n)$. (If a certain machine does not print an arithmetic network with $\mathsf{PIT}$ gates over $\mathbb{F}_{\mathsf{sf}(n)}$ that has the appropriate number of gates in time $n^k$, then $A$ discards this machine's output and defines $C^{(i)}$ as a dummy network.)

Let $\{P_n\}_{n\in\mathbb{N}}$ be the family of networks solving $\mathsf{PIT}$ over $\mathbb{F}$ with shift $\mathsf{sf}'$ such that $\mathsf{sf}'(n^k) = \mathsf{sf}(n)$ (note that $\mathsf{sf}'$ is polynomially bounded). The machine $A$ simulates the machine that prints $\{P_n\}$ on input $1^{n^k}$ to obtain a network $P_{n^k}$ solving $\mathsf{PIT}$ for arithmetic circuits of description size and degree $n^k$ over $\mathbb{F}_{\mathsf{sf}'(n^k)} = \mathbb{F}_{\mathsf{sf}(n)}$. Then $A$ prints a network $F_n$ that gets an $n$-element input $\vec{\Lambda}$ and executes as follows:

1. The network $F_n$ has the descriptions of $C^{(1)}, ..., C^{(n)}$ hard-wired, where each $\mathsf{PIT}$ gate in each $C^{(i)}$ is replaced with a copy of $P_{n^k}$. Denote the resulting networks by $D^{(1)}, ..., D^{(n)}$.
2. For each $i \in [n]$, the network $F_n$ computes $D^{(i)}(\vec{\Lambda})$ (using the network in Proposition 3.31), and letting $\sigma_i$ be the $i^{th}$ output element of this computation, it outputs

$$F_n(\vec{\Lambda}) = (\sigma_1 + 1, ..., \sigma_n + 1) \quad .$$

Observe that $F = \{F_n\}$ is a $\mathsf{P}$-uniform family of arithmetic networks of polynomial size. Assume towards a contradiction that there is an $n^k$-time-uniform family of arithmetic networks with $\mathsf{PIT}$ gates $\{B_n\}$ of size $n^k$ and degree $n^k$ and infinitely many $n \in \mathbb{N}$ and $\vec{\Lambda} \in \mathbb{F}^n_{\mathsf{sf}(n)}$ such that $B_n(\vec{\Lambda}) = F_n(\vec{\Lambda})$. Let $n \in \mathbb{N}$ be sufficiently large such that the Turing machine printing $\{B_n\}$ is one of the first $n$ machines, in which case there is $i \in [n]$ such that $C^{(i)} = B_n$. Fix $\vec{\Lambda} \in \mathbb{F}^n_{\mathsf{sf}(n)}$ such that $B_n(\vec{\Lambda}) = F_n(\vec{\Lambda})$. Since $P_{n^k}$ solves $\mathsf{PIT}$ correctly over $\mathbb{F}_n$, we have $C^{(i)}(\vec{\Lambda}) = D^{(i)}(\vec{\Lambda})$, and hence $B_n(\vec{\Lambda})_i = F_n(\vec{\Lambda})_i = D^{(i)}(\vec{\Lambda})_i + 1 = B_n(\vec{\Lambda})_i + 1$, a contradiction. □

## 7.2 A sufficient assumption for PIT

We now show that hardness for randomized arithmetic networks over all but finitely many inputs suffices to obtain polynomial-time (or nearly so) worst-case $\mathsf{PIT}$ algorithms. In Section 7.2.1 we show a result for fields with characteristic zero, and in Section 7.2.2 we show a result for finite fields.

### 7.2.1 Fields of characteristic zero

The following result uses the targeted generator from Theorem 5.1, which is based on the GKSS generator [GKSS22]. The description of the proof appeared in Section 2.1.

**Theorem 7.3** (hardness for networks with $\mathsf{PIT}$ gates on all inputs yields $\mathsf{PIT}$ algorithms; the case of characteristic zero)**.** *There is a constant $c_0 > 1$ such that for every sufficiently large constant $k > 1$ the following holds. Let $\mathbb{F}$ be a field of characteristic zero.*

- **Assumption:** *There is $c > 1$ and a family of $\log^c$-uniform arithmetic circuits $\{C_n\}_{n\in\mathbb{N}}$ of size $n^k$ and degree $n^k$ over $\mathbb{F}$ such that the following holds. For every* P*-time-uniform family of arithmetic networks with* PIT *gates $\{R_n\}_{n\in\mathbb{N}}$ of size $n^{c_0\cdot\sqrt{k}}$ and degree $2^{(\log(n))^{c_0}}$ over $\mathbb{F}$, and every sufficiently large $n \in \mathbb{N}$, and every $\vec{\Lambda} \in \mathbb{F}^n$, it holds that $R_n(\vec{\Lambda}) \neq C_n(\vec{\Lambda})$.*

- **Conclusion:** *There is a* P*-uniform family of arithmetic networks of polynomial size solving* PIT *over $\mathbb{F}$.*

**Proof.** The network for PIT is given a description $\vec{\Lambda} \in \mathbb{F}^n$ of an arithmetic circuit $\mathbb{F}^n \to \mathbb{F}$ computing a nonzero polynomial of degree at most $n$. Using Theorem 5.1 with the circuit family $\{C_n\}_{n\in\mathbb{N}}$, with $d(n) = n$ and $\bar{n} = n$, and with $\varepsilon = 1/\sqrt{k}$, the network computes a description of $\mathcal{H}_{\vec{\Lambda}}$. It then evaluates the composition $\vec{\Lambda} \circ \mathcal{H}_{\vec{\Lambda}}$ on a grid of side length $n^2 \cdot (n^k)^{\varepsilon} + 1 = \text{poly}(n)$ and dimension $O(1/\varepsilon) = O(1)$ to test if this polynomial is zero or nonzero. The precise field constants used to define this grid do not matter (e.g., we can use $1, 2, ..., \text{poly}(n)$), so the network can be printed by a polynomial-time Turing machine.

Assume towards a contradiction that for infinitely many arithmetic circuits, each represented by a vector $\vec{\Lambda}$, it holds that $\vec{\Lambda} \circ \mathcal{H}_{\vec{\Lambda}} = 0$. Then, the P-uniform network with PIT gates $R_n$ from Theorem 5.1 computes $R_n(\vec{\Lambda}) = C_n(\vec{\Lambda})$ in size

$$\text{poly}(n^{1/\varepsilon}, n^{\varepsilon\cdot k}) \leq n^{O(\sqrt{k})}$$

and degree

$$\text{poly}(n, n^{\varepsilon\cdot k})^{k\cdot\text{polylog}(n)} \leq 2^{\text{polylog}(n)} \quad ,$$

a contradiction. □

### 7.2.2 Finite fields

The following result uses the targeted generator from Theorem 6.1, which is based on the KI generator [KI04]. The description of the proof appeared in Section 2.2.

**Theorem 7.4** (hardness for randomized networks on all inputs yields PIT algorithms over finite fields)**.** *For every $c_0 > 1$ there is $k > 1$ such that for every feasible sequence $\mathbb{F} = \{\mathbb{F}_n\}_{n\in\mathbb{N}}$ of finite fields, where $\mathbb{F}_n$ has order $q(n)$ and satisfies $n^{\omega(1)} \leq q(n) \leq 2^{n^{c_0}}$, the following conditional implication holds.*

- **Assumption:** *For every polynomially bounded shift function* sf*, there is $c > 1$ and a $\log^c$-uniform family of arithmetic circuits $\{C_n\}_{n\in\mathbb{N}}$ over $\mathbb{F}^{\mathsf{sf}}$ of polynomial size and polynomial degree satisfying the following. For every* P*-uniform family of randomized arithmetic networks $\{R_n\}_{n\in\mathbb{N}}$ of size $n^k$ over $\mathbb{F}^{\mathsf{sf}}$, and every sufficiently large $n \in \mathbb{N}$, and every $\vec{\Lambda} \in \mathbb{F}^n_{\mathsf{sf}(n)}$, it holds that $R_n(\vec{\Lambda})$ does not compute $C_n(\vec{\Lambda})$ with error degree at most $n^k$.*

- **Conclusion:** *For every constant $C \geq 1$ there is a* P*-uniform family of arithmetic networks of size $n^{\log^{(C)}(n)}$ solving* PIT *over $\mathbb{F}$.*[39]

**Proof.** Let $k_0 > 1$ be a sufficiently large universal constant that bounds the degree of all the polynomially-bounded functions appearing in Theorem 6.1. Let $k = 4c_0 \cdot k_0$. Let $\{C_n\}$ be the $\log^c$-uniform family over $\mathbb{F}$ of size $s(n) = \text{poly}(n)$ and degree at most $s(n)$, from our hypothesis.

We first describe the PIT procedure, and later explain how to implement it as a P-uniform arithmetic network. Recall that to solve PIT, it suffices to construct an algorithm that gets as input an arithmetic circuit computing a nonzero polynomial, and finds an input on which this polynomial does not vanish.

Fixing any polynomially bounded shift function sf, we are given as input a description $\vec{\Lambda} \in \mathbb{F}^n_{\underline{n}}$ of an arithmetic circuit computing a nonzero polynomial of degree at most $n$ over $\mathbb{F}_{\underline{n}}$, where $\underline{n} = \mathsf{sf}(n)$. Let $n_0 = n$ and $\vec{\Lambda}_0 = \vec{\Lambda}$.

[39] Recall that $\log^{(i)}(n)$ is the $i^{\text{th}}$ iterated log function; that is, $\log^{(i)}(n) = \log(\log^{(i-1)}(n))$ for $i > 1$ and $\log^{(1)}(n) = \log(n)$.

**First step: Reducing the input length to be polylogarithmic.** Let $\{C_n\}$ be the hypothesized family of hard polynomials over $\mathbb{F}^{\mathsf{sf}}$. We use Theorem 6.1 with $\{C_n\}$ and $\mathbb{F}^{\mathsf{sf}}$, and with parameters $m = n_0$ and $d = n_0$ and $\sigma = 1$ and $\varepsilon = 1/2$. The network $G_n$ is defined over $\mathbb{F}_{\underline{n}}$, and given $\vec{\Lambda}_0 \in \mathbb{F}_{\underline{n}}^{n_0}$ it computes a description of the circuit $\mathcal{G}_{\vec{\Lambda}_0} : \mathbb{F}_{\underline{n}}^{\mathrm{polylog}(n_0)} \to \mathbb{F}_{\underline{n}}^{n_0}$. Denote

$$\vec{\Lambda}_1 := (\Lambda \circ \mathcal{G}_{\vec{\Lambda}}) \in \mathbb{F}_{\underline{n}}[x_1, ..., x_{\mathrm{polylog}(n)}] \quad .$$

We claim that $\vec{\Lambda}_1$ is nonzero as a polynomial. To see why this is true, assume otherwise. Then, the randomized network $R_{n_0}(\vec{\Lambda}_0)$ computes $C_{n_0}(\vec{\Lambda}_0)$ over $\mathbb{F}_{\underline{n}}$ with error degree

$$(n_0 \cdot d \cdot \log(s))^{k_0} < n_0^k \quad ,$$

where the size of $R_{n_0}$ is

$$(n_0 \cdot d \cdot m \cdot \log(s), \log(q))^{k_0} < n_0^{4c_0 \cdot k_0} = n_0^k \quad ,$$

contradicting the hardness of $\{C_n\}$.

By Theorem 6.1, the size of $\vec{\Lambda}_1$ is at most $n_0^{k'}$ for some constant $k' \in \mathbb{N}$ that depends on $\{C_n\}$, and the degree of $\vec{\Lambda}_1$ is also at most $n_0^{k'}$. We define $n_1 = n_0^{k'}$ and consider $\vec{\Lambda}_1 \in \mathbb{F}_{\underline{n}}^{n_1}$.

**Iterative step: Exponentially reducing the input length.** For $i \in [C-1]$, in iteration $i$ the network starts with a description $\vec{\Lambda}_i \in \mathbb{F}_{\underline{n}}^{n_i}$ of an arithmetic circuit $\mathbb{F}_{n_i}^{\ell_i} \to \mathbb{F}_{n_i}$ of degree at most $n_i \leq \mathrm{poly}(n)$, where $\ell_i = \begin{cases} \mathrm{polylog}(n_0) & i = 1 \\ O(\log(\ell_{i-1})) & i > 1 \end{cases}$.

Let $\mathsf{sf}_i$ be a polynomially bounded shift function such that $\mathsf{sf}_i(n_i) = \underline{n} = n_i^{\Omega(1)}$, and let $\{C_n\}$ be the corresponding hypothesized family of hard polynomials over $\mathbb{F}^{\mathsf{sf}}$. We use Theorem 6.1 with $\{C_n\}$, and with $m = \ell_i$ and $\sigma = 0$ and $d(n_i) = n_i$ and $\varepsilon = 1/r$ where $r$ is such that the size of $C_{n_i}$ is at most $n_i^r$. The network $G_{n_i}$ is defined over $\mathbb{F}_{\underline{n}}$, and given $\vec{\Lambda}_i \in \mathbb{F}_{\underline{n}}^{n_i}$ it computes a description of $\mathcal{G}_{\vec{\Lambda}_i} : \mathbb{F}_{\underline{n}}^{\ell_{i+1}} \to \mathbb{F}_{\underline{n}}^{\ell_i}$. Denote

$$\vec{\Lambda}_{i+1} := (\Lambda \circ \mathcal{G}_{\vec{\Lambda}}) \in \mathbb{F}_{\underline{n}}[x_1, ..., x_{\ell_{i+1}}] \quad .$$

Again, we claim that $\vec{\Lambda}_{i+1}$ is nonzero as a polynomial. Assuming otherwise, the network $R_{n_i}(\vec{\Lambda}_i)$ computes $C_{n_i}(\vec{\Lambda}_i)$ over $\mathbb{F}_{\underline{n}}$ with error degree $(n_i \cdot d \cdot \log(s)) < n_i^k$ and with size

$$s^{\varepsilon} \cdot (n_i, d, m^{\mathrm{loglog}(s)}, \log(q))^{k_0} = s^{\varepsilon} \cdot n_i^{2c_0 \cdot k_0} \cdot 2^{k_0 \cdot \log(\ell_i) \cdot \mathrm{loglog}(s)} < n_i^{4c_0 \cdot k_0} \quad ,$$

a contradiction. By Theorem 6.1, the size and degree of $\vec{\Lambda}_{i+1}$ are at most $n_i^{k'}$ for some constant $k' \in \mathbb{N}$ that depends on $\{C_n\}$,[40] and we define $n_{i+1} = n_i^{k'}$ and consider $\vec{\Lambda}_{i+1}$ as over $\mathbb{F}_{n_{i+1}}$.

**Final step.** After iteration $i = C - 1$ we obtain a nonzero $\vec{\Lambda}_C \in \mathbb{F}_{\underline{n}}[x_1, ..., x_{\ell_C}]$, where $n_C = \mathrm{poly}(n)$ and $\ell_C = \log^{(C)}(n)$ and the degree of $\vec{\Lambda}_C$ is at most $d' = \mathrm{poly}(n)$. We enumerate over the elements in a grid $S^{\ell_C} \subseteq \mathbb{F}_{\underline{n}}^{\ell_C}$ of width $|S| = d' + 1$, and since $\vec{\Lambda}_C$ is nonzero and $\mathbb{F}_{\underline{n}}$ is of size $\underline{n}^{\omega(1)} = n_C^{\omega(1)}$, we are guaranteed to find a nonzero element.

**A comment about uniformity.** Note that the algorithm considers constantly many shifts functions $\mathsf{sf}, \mathsf{sf}_1, ..., \mathsf{sf}_{C-1}$, and that in the proof we assume that the reconstruction procedure $\{R_n\}$ with each of these shift functions $\mathsf{sf}_i$ fails on the corresponding input $\vec{\Lambda}_i$. Since there are only $C = O(1)$ shift functions, this amounts to considering hardness for $C = O(1)$ uniform algorithms (i.e., each algorithm is a pairing of $\{R_n\}$ with a shift function), and we can safely assume that the initial input length $n$ is large enough so that all of these $C$ algorithms fail on all inputs of length $n$ or more.

[40] The size and degree also depend on $\varepsilon$, but we defined $\varepsilon = 1/r$ where $r$ is a function of the size $s$ of $\{C_n\}$. Thus, the size and degree indeed only depend on the family $\{C_n\}$.

**Implementation as a P-uniform arithmetic network.** Observe that the iterative procedure above repeats the following step: Given $\vec{\Lambda}_i$ (for $i \in \{0, ..., C-1\}$), pad its description, feed the description to the arithmetic network from Theorem 6.1, which outputs a description of $\vec{\Lambda}_{i+1}$, and continue to the next iteration. After iteration $C$, the procedure evaluates the final circuit on $S^{\ell_C}$.

To see that the procedure is indeed implementable as a P-uniform arithmetic network $\mathcal{N} = \{\mathcal{N}_n\}$, observe that there are only constantly many intermediary steps of padding and running arithmetic networks $\{G_n\}$, corresponding to the constantly many shifts $\mathsf{sf}_i$ and hard families $\{C_n\}$. The machines printing all of the $\{C_n\}$'s can be hard-wired into the machine that prints $\mathcal{N}_n$, and so can the constants bounding the various polynomials (i.e., the sizes of $\{C_n\}$'s and the polynomial size blow-ups from each $\vec{\Lambda}_i$ to $\vec{\Lambda}_{i+1}$). The machine printing $\mathcal{N}_n$ just adds padding gates and places the various $\{G_n\}$'s in the appropriate places in $\mathcal{N}_n$.

The last step of $\mathcal{N}_n$ is evaluating a circuit on $S^{\ell_C}$. Since $\mathbb{F}$ is feasible, the machine printing $\mathcal{N}_n$ can print $\mathrm{poly}(n)$ constants, and can thus also hard-wire all of the points in $S^{\ell_C}$ into $\mathcal{N}_n$. Thus, the only missing part is evaluating $\vec{\Lambda}_C$ on a point, which can be done by a P-uniform network using Proposition 3.24. □

**Remark 7.5** (degree bounds)**.** *The hardness assumption in Theorem 7.4 does not explicitly bound the degree of the randomized networks. (This is akin, for example, to the hardness assumption in [KI04] when working over finite fields.) Nevertheless, the degree of these networks is quasipolynomially bounded. Specifically, we can either assume hardness against randomized arithmetic networks with $p^{th}$ root gates[41] of degree quasipolynomial in $n$ and in the field's characteristic, similarly to Theorem 7.3; or assume hardness against randomized arithmetic circuits (without $p^{th}$ root gates) of degree that is quasipolynomial in both $n$ and $q$. Both statements follow directly from the degree bounds on the reconstruction in Theorem 6.1.*

# 8 Open problems

We believe that our result provide compelling motivation for studying PIT vs lower bounds in the uniform setting, and more generally for studying lower bounds for uniform arithmetic circuits. Let us therefore suggest several open problems that strike us as important and potentially tractable in this context:

**Problem 8.1.** *Relax the condition on the hard function in Theorem 1.2 to allow functions computable in arbitrary polynomial size (i.e., for any $k > 1$ allow an upper bound of $n^{c_k}$ where $c_k$ may be arbitrary).*

The reason for the upper bound of $n^{k^2}$ is our use of the generator of Guo, Kumar, Saptharishi, and Solomon [GKSS22], and in particular our implementation for the initial step in their reconstruction. Using a different implementation or a different generator (or reconstruction) might be a path forward for tackling Problem 8.1.

**Problem 8.2.** *Relax the hardness assumption in Theorem 1.4 to refer to networks with PIT gates, rather randomized networks with bounded error degree (see Theorem 7.4).*

The main source of trouble that currently prevents assuming hardness only for networks with PIT gates is the use of Kaltofen's [Kal89] classical factoring algorithm.

**Problem 8.3.** *Relax the hardness assumption in Theorems 1.2 and 1.4 to refer only to networks of polynomial degree.*

As mentioned in Sections 1.3 and 7, our results require hardness for networks of quasipolynomial degree. Technically, this degree stems from repeated applications of a universal arithmetic circuit (à la [Raz10]) in our construction.

**Problem 8.4.** *Strengthen the PIT conclusion in Theorem 1.4 to have a PIT algorithm running in strictly polynomial time.*

Indeed, the super-polynomial running time is reminiscent of running times obtained via bootstrapping techniques as in [AGS18; KST19], and the technical reasons for it are also very similar.

**Problem 8.5.** *Continue the study of lower bounds for uniform arithmetic circuits, by proving interesting lower bounds that go beyond what is known for non-uniform arithmetic circuits.*

[41] For a definition and discussion, see the comments after Theorem 6.1 as well as Section 6.2.4.

As mentioned in Section 1, lower bounds for uniform constant-free arithmetic circuits of polynomial size and bounded depth (against the permanent) are known (see [KP09], and also [JS13; CKK14]). However, in the Boolean setting we have lower bounds for general circuits, with no depth restriction (e.g., Santhanam and Williams [SW13] proved lower bounds in $\mathsf{P}$ for general uniform circuits of size $n^k$, for any fixed $k \in \mathbb{N}$), raising the possibility that stronger lower bounds for general uniform arithmetic circuits might be provable.


## Acknowledgements

Part of this work was done while Robert Andrews was at the Institute for Advanced Study and was supported by NSF grant CCF-1900460 and the Erik Ellentuck Endowed Fellowship Fund. Deepanshu Kush is thankful for financial support from an Ontario Graduate Scholarship (OGS-International) award. Roei Tell is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant RGPIN-2024-04490.


# Appendix A Uniformization of classical algorithms

In this section, we prove that various well-known algorithms in algebraic computation can be carried out in a uniform manner. We start in Section A.1, where we show that uniform arithmetic networks are capable of translating between the standard and universal encodings of arithmetic circuits. This affords us the flexibility to use either standard encodings or universal encodings of arithmetic circuits when designing algorithms. In Section A.2, we design uniform arithmetic networks for a variety of standard problems encountered when manipulating arithmetic circuits, such as polynomial interpolation and decomposition into homogeneous parts. Section A.3 studies the problem of factoring univariate polynomials over finite fields, showing that the Cantor–Zassenhaus algorithm, as modified by von zur Gathen [vzGat84], can be implemented as a uniform arithmetic network. Section A.4 proves that Kaltofen's algorithm to factor arithmetic circuits [Kal89] can be carried out by a uniform arithmetic network. Finally, we prove in Section A.5 that the depth reduction of Valiant, Skyum, Berkowitz, and Rackoff [VSBR83] for arithmetic circuits can be applied to a $\log^c$-uniform family of arithmetic circuits in a manner that preserves $\log^c$-uniformity.

The notation of this appendix deviates slightly from the rest of the paper. Throughout the appendix, we work with a fixed feasible sequence of fields $\mathbb{F} = \{\mathbb{F}_n\}_{n\in\mathbb{N}}$. Every algorithm in Section A receives $1^n$ as part of its input, so we adopt the convention that each algorithm operates over the corresponding field $\mathbb{F}_n$, suppressing the sequence of fields from the statement of the results. Whenever further assumptions on the field $\mathbb{F}_n$ are necessary, these will be made explicit in the statement of the corresponding result.

Throughout Sections A.1 and A.2, we use $\mathbb{F}_n$ to refer to the $n^{\text{th}}$ field in the sequence of fields. The field $\mathbb{F}_n$ may be a finite field of growing size (and not necessarily of order $n$), or $\mathbb{F}_n$ may be a field of characteristic zero (in which case $\mathbb{F}_n$ may not actually depend on the index $n$). In Sections A.3 and A.4, we restrict our attention to finite fields, so we change notation to $\mathbb{F}_{q(n)}$ to more transparently reflect that the $n^{\text{th}}$ field in the sequence $\mathbb{F}_{q(n)}$ is a finite field of order $q(n)$.

## A.1 Converting between universal and standard encodings

Many of our algorithms are uniform arithmetic networks that themselves operate on descriptions of arithmetic circuits. It is sometimes more convenient to describe an algorithm that operates on a standard description of an arithmetic circuit, even though the network receives a universal description as input, or vice-versa. The following lemmas show that uniform arithmetic networks can efficiently translate between the standard and universal encodings of arithmetic circuits. As a result, we may work with whichever encoding is more convenient in context at the expense of an additive polynomial increase in the complexity of our algorithms.

We start by converting standard encodings of arithmetic circuits (Definition 3.22) to universal encodings (Definition 3.27).

**Lemma A.1** (Standard to universal)**.** *There is a deterministic, polynomial-time Turing machine $M$ that receives as input $(1^n, 1^d, 1^s)$ and outputs an arithmetic network that satisfies the following properties.*

1. *The network receives as input the standard encoding of an $n$-variate size-$s$ arithmetic circuit $C$ that computes a polynomial of degree at most $d$.*

2. *The network outputs a universal encoding of length* $\mathrm{poly}(n, s, d)$ *of the circuit $C$.*

3. *The network has size* $\mathrm{poly}(n, s, d)$ *and degree* $O(1)$.

**Proof.** The network first uses Theorem 3.28 to print a circuit $\Psi(\vec{x}, \vec{y})$ that is universal for $n$-variate, degree-$d$, size-$s$ computation. The network then converts $C$ to a circuit $\Phi$ that computes the same polynomial, but whose underlying directed acyclic graph is the same as the universal circuit $\Psi$. This is done by applying the argument appearing in the proof of [Raz10, Proposition 2.8], noting that all steps in this transformation can be carried out by a uniform arithmetic network of size $\mathrm{poly}(n, s, d)$ and degree $O(1)$. As Raz's universal circuit only has constants appearing on edges that feed into sum gates, we propagate any constants in $\Phi$ that feed into a product gate into the sum gates that take this product gate as input. This rewiring can likewise be performed by a uniform arithmetic network of size $\mathrm{poly}(n, s, d)$ and constant degree. The resulting labeling of the constants in the circuit corresponds to the universal encoding of $C$, which the network then outputs. □

We now go in the other direction, converting universal encodings of arithmetic circuits into standard encodings.

**Lemma A.2** (Universal to standard)**.** *There is a deterministic, polynomial-time Turing machine $M$ that receives as input $(1^n, 1^d, 1^s)$ and outputs an arithmetic network that satisfies the following properties.*

1. *The network receives as input the universal encoding $\vec{\Lambda}$ of an $n$-variate size-$s$ arithmetic circuit $C$ that computes a polynomial of degree at most $d$.*

2. *The network outputs a standard encoding of length* $\mathrm{poly}(n, s, d)$ *of the circuit $C$.*

3. *The network has size* $\mathrm{poly}(n, s, d)$ *and degree* 1.

**Proof.** The network first invokes Theorem 3.28 to construct a circuit $\Psi(\vec{x}, \vec{y})$ that is universal for $n$-variate, degree-$d$, size-$s$ computation. Because $\vec{\Lambda}$ is a universal encoding of $C$, we have that $C(\vec{x}) = \Psi(\vec{x}, \vec{\Lambda})$ as polynomials. Thus, to output a standard description of $C$, the network replaces the $\vec{y}$ inputs to $\Psi$ with the given vector $\vec{\Lambda}$ and then outputs a standard description of the resulting circuit.

This network is the combination of Theorem 3.28 and a simple modification of the inputs to the universal circuit $\Psi$. It is clear that this network has size $\mathrm{poly}(n, s, d)$. Because this network does not do any arithmetic computation on the input $\vec{\Lambda}$, it has degree 1 as claimed.

To obtain a polynomial-time Turing machine that prints this network, the part of the network that builds the circuit $\Psi$ can be printed by running the Turing machine of Theorem 3.28. The remaining part of the network that replaces the $\vec{y}$ variables in the input to $\Psi$ with the given vector $\vec{\Lambda}$ can clearly be printed by a polynomial-time Turing machine. □

## A.2 Algorithms that manipulate arithmetic circuits

This subsection describes uniform families of arithmetic networks (sometimes with PIT gates) that perform a variety of tasks on arithmetic circuits, such as finding a variable a circuit depends on, homogenizing a circuit, and computing the greatest common divisor of two circuits. None of these algorithms are new. Rather, the goal of this subsection is to verify that each of these algorithms can be implemented by a uniform family of arithmetic networks, and to provide bounds on the complexity parameters of the resulting networks.

We start by describing a uniform arithmetic network (with PIT gates) that receives as input an arithmetic circuit $C(x_1, \ldots, x_n)$ and finds a variable $x_i$ that $C$ depends on. Of course, if the circuit $C$ computes a constant polynomial, then $C$ depends on no variables, in which case the arithmetic network correctly reports that $C$ is a constant polynomial.

**Lemma A.3** (Finding a variable a polynomial depends on)**.** *There is a deterministic, polynomial-time Turing machine $M$ that receives as input $(1^n, 1^d, 1^s)$ and outputs an arithmetic network with PIT gates that satisifes the following properties.*

1. *The network receives as input the standard description of an arithmetic circuit $C$ of size $s$ over $\mathbb{F}_n$ computing a polynomial $f \in \mathbb{F}_n[x_1, \ldots, x_n]$ of degree at most $d$.*

2. *The network outputs data $(b, \vec{a}) \in \{0,1\} \times \{0,1\}^{\log_2 n}$ satisfying the following.*

   (a) *The bit $b$ is set to 1 if and only if $f$ is a constant polynomial.*

   (b) *If $b = 0$ (and hence $f$ is nonconstant), then $\vec{a}$ is the binary encoding of an integer $i \in [n]$ such that $f$ depends on the variable $x_i$.*

3. *The network has size* $\mathrm{poly}(n, s, d)$ *and degree* $O(1)$.

**Proof.** For each $i \in [n]$, the network iteratively checks if $f$ depends on the variable $x_i$. To do this, the network computes a description of a circuit that computes the polynomial

$$f_i(\vec{x}, y) \coloneqq f(x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_n),$$

where $y$ is a fresh variable. A description of a circuit for $f_i$ can be obtained easily from a circuit that computes $f$ by replacing the variable $x_i$ with $y$ at the input gates. The network then uses its PIT gates to test if $f(\vec{x}) - f_i(\vec{x}, y) = 0$. If the PIT gate reports that $f(\vec{x}) - f_i(\vec{x}, y) \neq 0$, then $f$ depends on $x_i$, so the network outputs $(0, \mathrm{bin}(i))$, where $\mathrm{bin}(i) \in \{0,1\}^{\log_2 n}$ is the binary representation of $i$. Otherwise, the network proceeds to check if $f$ depends on the next variable $x_{i+1}$. If $f$ depends on none of the variables, then the network outputs $(1, 0^{\log_2 n})$ to indicate that $f$ is a constant polynomial.

To bound the size of the resulting network, we observe that the network makes simple modifications to the given circuit for $f$ and then performs PIT on the modified circuits. For each $i \in [n]$, replacing the variable $x_i$ with $y$ in the input to the circuit for $f$ can be done by a network of size $O(s)$. This produces a standard description of the circuit for $f - f_i$, again using a network of size $O(s)$. We then convert this to a universal description using Lemma A.1, incurring a $\mathrm{poly}(n, s, d)$ increase in the size of our network, and then feed this universal description into a PIT gate. The post-processing of the PIT gate's output can be done by a network of size $O(1)$. We perform this sequence of operations $n$ times, so the overall size of our network is bounded by $\mathrm{poly}(n, s, d)$. Each iteration can be implemented by a network of degree $O(1)$, so the degree of the overall network is likewise bounded by $O(1)$.

Finally, the network can be printed by a polynomial-time Turing machine as follows. The components of the network that modify the circuit for $f$ into a circuit for $f - f_i$, as well as the components that post-process the outcome of the PIT operation, can clearly be printed in polynomial time. The part of the network that translates from standard encodings to universal encodings of arithmetic circuits can be printed by repeatedly running the Turing machine that prints the network of Lemma A.1. □

Our next tool is polynomial interpolation. Given a multivariate polynomial $f(x_1, \ldots, x_n, y)$, we can always regard $f$ as a univariate polynomial in $y$ whose coefficients are themselves polynomials in $x_1, \ldots, x_n$. That is, there are polynomials $f_0, f_1, \ldots, f_d \in \mathbb{F}[\vec{x}]$ such that

$$f(\vec{x}, y) = \sum_{i=0}^{d} f_i(\vec{x}) y^i,$$

where $d$ is the degree of $f$. From a circuit that computes $f$, we can build a circuit of comparable size that computes these coefficients $f_0, \ldots, f_d$. The following lemma verifies that this transformation can be carried out by a uniform family of arithmetic networks. To avoid any constraints on the size of the underlying field $\mathbb{F}$, we carry out polynomial interpolation via a gate simulation argument, rather than by partially evaluating the circuit and then interpolating from these evaluations. If the circuit computing $f$ has size $s$, the gate simulation results in circuits of size $O(sd^2)$ for the coefficients $f_i$. This loses a factor of $d$ over the evaluation and interpolation approach, which produces circuits of size $O(sd)$ for the coefficients $f_i$. This extra factor of $d$ will not meaningfully impact any of our applications.

**Lemma A.4** (Polynomial interpolation)**.** *There is a deterministic, polynomial-time Turing machine $M$ that receives as input $(1^n, 1^d, 1^s)$ and outputs an arithmetic network satisfying the following properties.*

1. *The network receives as input the description of an arithmetic circuit $C$ of size $s$ over $\mathbb{F}_n$ computing a polynomial $f \in \mathbb{F}_n[x_1, \ldots, x_n]$ of degree at most $d$, and the binary encoding of an integer $i \in [n]$.*

2. *The network outputs the description a $(d+1)$-output arithmetic circuit $C'$ of size $O(sd^2)$ that does not depend on the variable $x_i$. Let $f_j(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$ be the polynomial computed by the $j^{th}$ output gate of $C'$. Then the polynomial identity*

$$f(x_1, \ldots, x_n) = \sum_{j=0}^{d} f_j(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) x_i^j$$

   *holds.*

3. *The network has size* $\mathrm{poly}(n, s, d)$ *and degree* $O(1)$.

**Proof.** We use gate simulation to convert the given circuit computing $f$ into a circuit that computes the coefficient polynomials $f_0, f_1, \ldots, f_d$. We first describe the structure of the circuit that computes the coefficient polynomials, and then explain how this transformation can be carried out by a uniform family of arithmetic networks.

Each gate $v$ of the input circuit $C$ will be split into $d+1$ gates $(v,0), (v,1), \ldots, (v,d)$. If $f_v(\vec{x})$ is the polynomial computed by the gate $v$ in $C$, then the gate $(v,j)$ is intended to compute a polynomial $f_{v,j}(\vec{x})$ such that

$$f_v(\vec{x}) = \sum_{j=0}^{d} f_{v,j}(\vec{x}) x_i^j$$

and none of the polynomials $f_{v,j}$ depend on the variable $x_i$. The gates $(v,j)$ are wired depending on the type of the gate $v$ in the circuit $C$.

- If $v$ is an input gate in $C$ labeled by a field element $\alpha$ or a variable $x_k$ where $k \neq i$, then we set $(v,0)$ to be an input gate labeled by $\alpha$ or $x_k$, respectively. For $j \geq 1$, we set $(v,j)$ to be an input gate labeled by zero.

- If $v$ is an input gate in $C$ labeled by the variable $x_i$, then we set $(v,1)$ to be an input gate labeled by 1. For $j \neq 1$, we set $(v,j)$ to be an input gate labeled by zero.

- If $v = u + w$ is an addition gate with children $u$ and $w$, then for each $j \in \{0, 1, \ldots, d\}$, we set $(v,j)$ to be an addition gate with children $(u,j)$ and $(w,j)$.

- If $v = u \times w$ is a multiplication gate with children $u$ and $w$, then for each $j \in \{0, 1, \ldots, d\}$, we set $(v,j)$ to be the output of the subcircuit given by

$$(v,j) = \sum_{k=0}^{j} (u,k) \times (w, j-k).$$

Denote this new circuit by $C'$. Using induction, one can prove that each gate $(v,j)$ of $C'$ correctly computes the polynomial $f_{v,j}(\vec{x})$. Each gate of $C$ is copied into $d+1$ gates in $C'$, and each of these gates is implemented by a subcircuit of size at most $O(d)$, so the total number of gates in $C'$ is bounded by $O(sd^2)$. Furthermore, it follows from the definition of $C'$ that $x_i$ does not appear on any of the input gates of $C'$, so no output of $C'$ depends on the variable $x_i$. This establishes the existence of the claimed circuits that compute the coefficients $f(\vec{x})$ as a polynomial in $x_i$.

It remains to prove that this transformation can be carried out by a uniform arithmetic network with the claimed bounds on size and degree. The circuit $C'$ is obtained by modifying the graph underlying the circuit $C$. Thanks to Lemma A.2, we may assume without loss of generality that we are given a standard description of $C$. In this case, the only modifications to $C$ are done on the boolean part of the circuit's description. We can obtain a polynomial-time Turing machine that modifies the gates and wiring of $C$ into that of $C'$ by directly implementing the transformation as described above. This implies there is a uniform family of polynomial-size boolean circuits that take as input the boolean part of $C$ and output the boolean part of $C'$.

These circuits are likewise arithmetic networks that operate on the boolean part of the circuit description, so we obtain a uniform family of arithmetic networks of size $\mathrm{poly}(n, s, d)$ and degree $O(1)$ that computes a description of $C'$ from the description of $C$ as desired. □

An easy application of polynomial interpolation allows us to take partial derivatives of arithmetic circuits with respect to one variable.

**Lemma A.5** (Partial derivatives)**.** *There is a deterministic, polynomial-time Turing machine that receives as input $(1^n, 1^d, 1^s)$ and outputs an arithmetic network satisfying the following.*

1. *The network receives as input the description of an arithmetic circuit $C$ of size $s$ over $\mathbb{F}_n$ computing a polynomial $f \in \mathbb{F}_n[x_1, \ldots, x_n]$ of degree at most $d$, and the binary encoding of two integers $i \in [n]$ and $k \in [d]$.*
2. *The network outputs a description of an arithmetic circuit $C'$ of size $O(sd^2)$. The arithmetic circuit $C'$ computes the polynomial*
$$\frac{\partial^k}{\partial x_i^k}(f(\vec{x})).$$
3. *The network has size $\mathrm{poly}(n, s, d)$ and degree $O(1)$.*

**Proof.** Write $f(\vec{x})$ as a polynomial in $x_i$ as

$$f(\vec{x}) = \sum_{j=0}^{d} f_j(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) x_i^j.$$

Then the derivative $\frac{\partial^k}{\partial x_i^k}(f(\vec{x}))$ is given by

$$\frac{\partial^k}{\partial x_i^k} = \sum_{j=k}^{d} j(j-1)\cdots(j-k+1) \cdot f_j(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) x_i^{j-k}.$$

Lemma A.4 shows that there is a multi-output arithmetic circuit of size $O(sd^2)$ that computes all the $f_j$ above, and that this circuit can be computed by a uniform family of arithmetic networks. By taking an appropriate linear combination of the $f_j$ as above, we obtain an arithmetic circuit of size $O(sd^2)$ that computes $\frac{\partial^k}{\partial x_i^k}(f(\vec{x}))$ as desired.

Given a description of the circuit computing the polynomials $f_j$, it is straightforward to add an extra addition gate on top and print the values of the falling factorials $j(j-1)\cdots(j-k+1)$. This shows that the description of the circuit for $\frac{\partial^k}{\partial x_i^k}(f(\vec{x}))$ can be computed by a uniform family of arithmetic networks as desired. □

Next on our list of circuit transformations is homogenization. Every multivariate polynomial $f(\vec{x})$ of degree at most $d$ can be written as a sum $f = f_0 + f_1 + \cdots + f_d$, where each $f_i$ is a polynomial consisting only of monomials of degree exactly $i$. Such polynomials are called homogeneous, and the polynomial $f_0, f_1, \ldots, f_d$ are called the homogeneous components of $f$. A gate simulation argument, very similar to the one used for interpolation, allows us to efficiently extract homogeneous components from arithmetic circuits. We sometimes apply this transformation in a manner where we only take homogeneous components with respect to a subset of the variables, pushing the others into the ground field. To accommodate this generality, we provide a list of variables as part of the input to the arithmetic network that performs homogenization, with the intended meaning that the network homogenizes the given circuit only with respect to these specified variables. By homogenizing with respect to a single variable, we essentially recover polynomial interpolation (Lemma A.4) as a special case of homogenization.

**Lemma A.6** (Decomposition into homogeneous parts)**.** *There is a deterministic, polynomial-time Turing machine $M$ that receives as input $(1^n, 1^d, 1^s)$ and outputs an arithmetic network satisfying the following properties.*

1. *The network receives as input the description of an arithmetic circuit $C$ of size $s$ over $\mathbb{F}_n$ computing a polynomial $f \in \mathbb{F}_n[x_1, \ldots, x_n]$ of degree at most $d$, as well as a set $S \subseteq [n]$ encoded as a bit vector in $\{0,1\}^n$.*

2. *The network outputs the description of a $(d+1)$-output arithmetic circuit $C'$ of size bounded by $O(sd^2)$. Let $f_j(\vec{x})$ be the polynomial computed by the $j^{th}$ output gate of $C'$. Then $f_j$ is a homogeneous polynomial of degree $j$ when viewed as a polynomial in the variables indexed by $S$, where the remaining variables are regarded as elements of the underlying field. Further, the polynomial identity*

$$f(x_1, \ldots, x_n) = \sum_{j=0}^{d} f_j(\vec{x})$$

*holds. That is, the polynomial $f_j$ is the degree-$j$ homogeneous component of $f$ with respect to the variables indexed by $S$.*

3. *The network has size* $\mathrm{poly}(n, s, d)$ *and degree* $O(1)$.

**Proof.** To obtain a circuit that computes the homogeneous components of $f$, we apply a gate simulation argument similar to the one used in the proof of Lemma A.4. We first describe the gate simulation argument used to obtain circuits for the homogeneous components of $f$, and then argue that there is a uniform family of arithmetic networks that implements this simulation. For the sake of completeness, we include the full details of the gate simulation.

Each gate $v$ of the given circuit $C$ will be split into $d+1$ gates $(v,0), (v,1), \ldots, (v,d+1)$. Letting $f_v(\vec{x})$ be the polynomial computed by gate $v$, the gate $(v,j)$ is intended to the degree-$j$ homogeneous component of $f_v$ with respect to the variables in $S$. The wiring of the gates $(v,j)$ depends on the label of the original gate $v$ in $C$.

1. If $v$ is an input gate labeled by a field constant $\alpha$ or a variable $x_k$ where $k \notin S$, we set $(v,0)$ to be an input gate labeled by $\alpha$ or $x_k$, respectively. For $j \geq 1$, we set $(v,j)$ to be an input gate labeled by zero.

2. If $v$ is an input gate labeled by a variable $x_k$ where $k \in S$, then we set $(v,1)$ to be an input gate labeled by $x_k$. For $j \neq 1$, we set $(v,j)$ to be an input gate labeled by zero.

3. If $v = u + w$ is an addition gate with children $u$ and $w$, then for each $j \in \{0,1,\ldots,d\}$, we set $(v,j)$ to be an addition gate with children $(u,j)$ and $(w,j)$.

4. If $v = u \times w$ is a product gate with children $u$ and $w$, then for each $j \in \{0,1,\ldots,d\}$, we set $(v,j)$ to be the output of the subcircuit given by

$$(v,j) = \sum_{k=0}^{j} (u,k) \times (w, j-k).$$

Let $C'$ be the new circuit obtained by applying this gate simulation to $C$. By induction, one can show that $(v,j)$ computes the degree-$j$ homogeneous component of $f_v$ with respect to the variables indexed by $S$. Every gate of $C$ is copied into $d+1$ gates in $C'$, each of which can be implemented by a subcircuit of size $O(d)$, so there are at most $O(sd^2)$ gates in $C'$. This proves the existence of the claimed circuits for the homogeneous components of $f$ with respect to the variables indexed by $S$.

To show that this transformation can be carried by a uniform family of arithmetic networks, we use the same argument as in the proof of Lemma A.4. By directly implementing the description of $C'$ above, we obtain a polynomial-time Turing machine that computes a description of the boolean part of $C'$ from a description of the boolean part of $C$. This implies the existence of a uniform family of boolean circuits—and hence a uniform family of arithmetic networks—that carries out the same computation. Using Lemma A.2, we may assume that we are given a standard description of $C$ as input, so we obtain the desired family of uniform arithmetic networks of size $\mathrm{poly}(n, s, d)$ and degree $O(1)$. □

Homogenization of arithmetic circuits is a useful subroutine in the reconstruction algorithm for our targeted hitting set generator based on the GKSS generator. There, we sometimes find ourselves in the situation where the circuit $C$ to be homogenized is already known to be partially homogeneous. That is, we know the circuit $C$ can be written as a composition $C = C_1 \circ C_2$ of two arithmetic circuits, where the circuit $C_2$ is already homogeneous. With this extra information, we can homogenize $C$ in a more efficient manner than what Lemma A.6 provides. Letting $s_1$ and $s_2$ denote the sizes of $C_1$ and $C_2$, respectively, Lemma A.6 constructs an equivalent homogeneous circuit of size $O((s_1 + s_2)d^2)$. The following lemma uses the fact that $C_2$ is already homogeneous to improve the size of the resulting homogeneous circuit to $s_2 + O(s_1 d^2)$, which is important when bounding the complexity of the reconstruction procedure in Section 5.2.

**Lemma A.7** (Partial homogenization [GKSS22, Lemma 2.10])**.** *There is a deterministic, polynomial-time Turing machine $M$ that receives as input $(1^n, 1^m, 1^d, 1^s, 1^t)$ and outputs an arithmetic network satisfying the following properties.*

1. *The network receives as input the description of a multi-output homogeneous arithmetic circuit $C$ of size $s$ over $\mathbb{F}_n$ with $m$ outputs computing polynomials $f_1, \ldots, f_m \in \mathbb{F}_n[x_1, \ldots, x_n]$, and the description of an $m$-input multi-output circuit $C'$ of size $t$.*
2. *If $\deg(C) \cdot \deg(C') \leq d$, then the network outputs the description of a homogeneous multi-output circuit $D$ of size $s + O(td^2)$ that computes the homogeneous components of degree at most $d$ of the outputs of $C' \circ C$. If $\deg(C) \cdot \deg(C') > d$, the output of the network is meaningless.*
3. *The network has size $\mathrm{poly}(n, m, s, t, d)$ and degree $O(1)$.*

**Proof.** As in the proof of [GKSS22, Lemma 2.10], we apply the standard gate simulation used to homogenize circuits (as done in Lemma A.6), but we only simulate the gates appearing in the not-necessarily-homogeneous circuit $C'$. Like our previous applications of gate simulation, we first describe the structure of the new circuit, and then explain how this new circuit can be computed by a uniform family of arithmetic networks.

We first create a new circuit $C''$ from $C'$ by applying gate simulation. For each gate $v$ of $C'$, we create $d + 1$ new gates $(v, 0), (v, 1), \ldots, (v, d)$ in $C''$. Similar to prior simulations, if $f_v$ denotes the polynomial computed by gate $v$ in the composed circuit $C' \circ C$, then the gate $(v, j)$ is intended to compute the degree-$j$ homogeneous component of $f_v$ when viewed as part of the composed circuit $C'' \circ C$. We wire the gates $(v, j)$ as follows.

1. If $v$ is an input gate labeled by a field constant $\alpha$, we set $(v, 0)$ to be an input gate labeled by $\alpha$. For all $j \geq 1$, we set $(v, j)$ to be an input gate labeled by zero.
2. If $v$ is an input gate labeled by the variable $y_i$, then we set $(v, \deg(f_i))$ to be an input gate labeled by $y_i$, where $f_i$ is the polynomial computed by the $i^{\text{th}}$ output gate of the circuit $C$. For $j \neq \deg(f_i)$, we set $(v, j)$ to be an input gate labeled by zero.
3. If $v = u + w$ is an addition gate with children $u$ and $w$, then we set $(v, j) = (u, j) + (w, j)$ for each $j \in \{0, 1, \ldots, d\}$.
4. If $v = u \times w$ is a product gate with children $u$ and $w$, then for each $j \in \{0, 1, \ldots, d\}$, we set $(v, j)$ to be the output of the subcircuit
$$(v, j) = \sum_{k=0}^{j} (u, k) \times (w, j - k).$$

Let $C''$ be the circuit obtained by applying this simulation to the gates of $C'$. It is clear that the size of $C''$ is bounded by $O(td^2)$. Using induction, one can prove that the circuit $C'' \circ C$ is homogeneous, and that the outputs of $C'' \circ C$ correspond to the homogeneous components of the outputs of $C' \circ C$. Thus, the circuit $D \coloneqq C'' \circ C$ is a homogeneous circuit of size $s + O(td^2)$ and computes the homogeneous components of the outputs of $C' \circ C$. This establishes the existence of a circuit with the desired behavior and complexity bounds.

It remains to describe a uniform family of arithmetic networks that receive descriptions of $C$ and $C'$ as input and output a description of $D$. To do this, it suffices to describe a uniform family of arithmetic networks that compute a description of the circuit $C''$ that simulates $C'$. If we are given the degrees of

the output polynomials of $f_1, \ldots, f_m$ of the circuit $C$, then the construction of $C''$ can be carried out by a uniform family of arithmetic networks in the same manner as in theproof of Lemma A.6. Thus, our task amounts to computing the degrees of the polynomials computed by the circuit $C$ in a uniform manner.

Recall that the circuit $C$ is homogeneous. This allows us to compute the degree of each gate of $C$ via dynamic programming, following a topological ordering of the gates of $C$. Starting at the input layer, each input gate of $C$ computes a polynomial either of degree zero or degree one, depending on whether the gate is labeled with a field element or a variable. In either case, we can easily determine the degree of the polynomial computed at the corresponding gate. If $v = u + w$ is an addition gate with children $u$ and $w$, then the fact that $C$ is homogeneous implies that $\deg(f_v) = \deg(f_u) = \deg(f_w)$, so from the degree of $u$ or $w$ we can immediately compute the degree of $v$. If $v = u \times w$ is a product gate, then we have $\deg(f_v) = \deg(f_u) + \deg(f_w)$, so we can efficiently compute the degree of $v$ from the degrees of $u$ and $w$.

The preceding algorithm yields a polynomial-time Turing machine that takes the description of $C$ as input and computes the degrees of each gate of $C$. This, in turn, produces a uniform family of boolean circuits—and hence a uniform family of arithmetic networks—that take the description of $C$ as input and output the degrees of each gate of $C$. □

We now turn our attention away from syntactic operations on arithmetic circuits and towards operations of a more algebraic nature, which will be particularly useful in our description of Kaltofen's algorithm to factor arithmetic circuits. We start by showing that uniform families of arithmetic networks with PIT gates can solve linear systems, even when the coefficients of the linear system are multivariate polynomials encoded as arithmetic circuits. This result is a natural combination of two algorithms. The first is a simple algorithm, due to Borodin, von zur Gathen, and Hopcroft [BvH82], that computes a basis for the nullspace of a matrix by finding a basis of the column space and then expressing the remaining columns of the matrix in terms of this basis. The second is a natural extension of this algorithm, appearing in the work of Kopparty, Saraf, and Shpilka [KSS15], that computes a single nonzero element of the nullspace of a matrix in the setting where the entries of the matrix are themselves multivariate polynomials encoded by arithmetic circuits.

**Lemma A.8** (Solving linear systems [BvH82, Theorem 5], [KSS15, Lemma 2.6])**.** *There is a deterministic, polynomial-time Turing machine $M$ that receives as input $(1^n, 1^d, 1^k, 1^\ell, 1^s)$ and outputs an arithmetic network with* PIT *gates satisfying the following properties.*

1. *The network receives as input the description of a multi-output arithmetic circuit $C$ of size $s$ that computes $k\ell$ polynomials $f_{1,1}, \ldots, f_{k,\ell} \in \mathbb{F}_n[x_1, \ldots, x_n]$, each of degree at most $d$.*
2. *Let $M \in \mathbb{F}_n[\vec{x}]^{k \times \ell}$ be the $k \times \ell$ matrix given by $M_{i,j} = f_{i,j}$. The network outputs an integer $m \in \{0, 1, \ldots, \ell\}$, encoded in binary, that corresponds to the dimension of the space of solutions to the linear system $M\vec{v} = \vec{0}$. In addition, the network outputs a description of a multi-output arithmetic circuit $D$ of size $\mathrm{poly}(n, k, \ell, s, d)$ that computes $m\ell$ polynomials $g_{1,1}, \ldots, g_{m,\ell} \in \mathbb{F}_n[\vec{x}]$ such that the vectors $\{(g_{i,1}, \ldots, g_{i,n}) : i \in [m]\} \subseteq \mathbb{F}_n[\vec{x}]^\ell$ are a basis for the solution space of $M\vec{v} = \vec{0}$.*
3. *The network has size $\mathrm{poly}(n, k, \ell, s, d)$ and degree $O(1)$.*

**Proof.** As described prior to the statement of the lemma, we combine the algorithm of Borodin, von zur Gathen, and Hopcroft [BvH82] with the observation of Kopparty, Saraf, and Shpilka [KSS15] that when the entries of the matrix are specified by arithmetic circuits, the necessary polynomial arithmetic can be carried out using PIT gates. We first describe the algorithm, and then verify that the algorithm can be implemented as a uniform family of arithmetic networks with PIT gates.

We first find a maximal nonsingular submatrix of $M$. This is done iteratively: for each $i \in [\ell]$, suppose we have found a set of $i - 1$ rows $R \subseteq [k]$ and a set of $i - 1$ columns $C \subseteq [\ell]$ such that the submatrix $M_{R,C}$ is nonsingular. For each $r \in [k] \setminus R$ and $c \in [\ell] \setminus C$, we check if the submatrix $M_{R \cup \{r\}, C \cup \{c\}}$ is nonsingular by computing $\det(M_{R \cup \{r\}, C \cup \{c\}})$. If $\det(M_{R \cup \{r\}, C \cup \{c\}}) \neq 0$, we have found an invertible $i \times i$ submatrix, so we update $R \leftarrow R \cup \{r\}$ and $C \leftarrow C \cup \{c\}$ and proceed to the next iteration. Otherwise, we have found a maximal nonsingular submatrix, which we will use to construct a basis for the nullspace of $M$ in the next step.

Fix subsets $R \subseteq [k]$ and $C \subseteq [\ell]$ with $|R| = |C|$ such that $M_{R,C}$ is a maximal nonsingular submatrix. If $C = [\ell]$, then the matrix $M$ has no nonzero vectors in its kernel, so our algorithm reports that the system

$M\vec{v} = \vec{0}$ has a solution space of dimension zero. Otherwise, for each $j \in [\ell] \setminus C$, the $j^{\text{th}}$ column of $M$ is in the span of the columns indexed by $C$, and the resulting linear relation between the columns of $M$ corresponds to an element of the nullspace of $M$. In more detail, let $\vec{v}^{(j)} \in \mathbb{F}_n[\vec{x}]^\ell$ be the vector

$$\vec{v}_i^{(j)} := \begin{cases} \det(M_{R,C\setminus\{i\}\cup\{j\}}) & \text{if } i \in C, \\ -\det(M_{R,C}) & \text{if } i = j, \\ 0 & \text{otherwise,} \end{cases}$$

where $M_{R,C\setminus\{i\}\cup\{j\}}$ is the matrix obtained by replacing the $i^{\text{th}}$ column of $M_{R,C}$ with $M_{R,j}$. Cramer's rule implies that $M\vec{v}^{(j)} = \vec{0}$. Because only the vector $\vec{v}^{(j)}$ has a nonzero entry in the $j^{\text{th}}$ coordinate, the vectors $\left\{\vec{v}^{(j)} : j \in [\ell] \setminus C\right\}$ are linearly independent. Since the nullity of $M$ is $\ell - |C|$ and we have found $\ell - |C|$ linearly independent vectors in the nullspace of $M$, these vectors form a basis for the nullspace of $M$.

It remains to describe an implementation of this algorithm as a uniform family of arithmetic networks with PIT gates. To find a maximal nonsingular submatrix, the network tries all $k\ell$ possible choices of which row and column to add to the current submatrix, selecting the lexicographically-first choice of row $r$ and column $c$ where $M_{R\cup\{r\},C\cup\{c\}}$ is nonsingular, if such a choice exists. To test if $M_{R\cup\{r\},C\cup\{c\}}$ is nonsingular, the network computes $\det(M_{R\cup\{r\},C\cup\{c\}})$ and checks if this determinant is nonzero. The entries of $M$ are multivariate polynomials specified by arithmetic circuits, so to check if this determinant is nonzero, the networks passes a description of a circuit computing $\det(M_{R\cup\{r\},C\cup\{c\}})$ to a PIT gate. We compute a description of a circuit computing $\det(M_{R\cup\{r\},C\cup\{c\}})$ by composing the circuits computing the entries of $M_{R\cup\{r\},C\cup\{c\}}$ with the uniform arithmetic circuit for the determinant constructed by Berkowitz [Ber84].

Once we have found a maximal nonsingular submatrix $M_{R,C}$, we report that $M\vec{v} = \vec{0}$ has a solution space of dimension $m = \ell - |C|$. In case that $m > 0$, a description of a multi-output circuit computing a basis for the nullspace of $M$ can be obtained from the explicit description of the vectors $\vec{v}^{(j)}$ above. This is again done by composing the uniform arithmetic circuits of Berkowitz [Ber84] for the determinant with the circuits computing the relevant entries of $M$.

It is clear from the above description that the resulting arithmetic network has size bounded by $\text{poly}(n, k, \ell, s, d)$. Since the network only manipulates descriptions of arithmetic circuits and does no arithmetic computation itself, its degree is bounded by $O(1)$. Finally, it is evident from the uniform description of this arithmetic network that it can be printed by a polynomial-time Turing machine. □

Next, we consider the problem of divisibility testing: we are given two polynomials $f(\vec{x})$ and $g(\vec{x})$, represented as arithmetic circuits, and we want to test if $f$ divides $g$. There is a simple reduction from divisibility testing to PIT due to Forbes [For15], based on Strassen's method of eliminating divisions in arithmetic circuits [Str73]. As the following lemma shows, this reduction (and hence divisibility testing itself) can be implemented as a uniform family of arithmetic networks with PIT gates.

**Lemma A.9** ([For15, Section 7])**.** *There is a deterministic, polynomial-time Turing machine that receives as input $(1^n, 1^d, 1^s)$ and outputs an arithmetic network with PIT gates that satisfies the following properties.*

1. *The network receives as input the descriptions of two arithmetic circuits $C_1$ and $C_2$ of size $s$ over $\mathbb{F}_n$, computing polynomials $f, g \in \mathbb{F}_n[x_1, \ldots, x_n]$ respectvely, both of degree at most $d$.*
2. *If $|\mathbb{F}_n| > d$, the network outputs a bit indicating whether $f$ divides $g$. Otherwise, the output of the network is meaningless.*
3. *The network has size $\text{poly}(n, s, d)$ and degree $O(1)$.*

**Proof.** We follow Forbes's [For15] reduction of divisibility testing to PIT. We first present this reduction, and then explain how to implement it as a uniform family of arithmetic networks with PIT gates.

We begin by handling the special case where either $f$ or $g$ is zero. If $g$ is identically zero, then every polynomial divides $g$, so we report that $f$ divides $g$. If $g$ is nonzero and $f$ is zero, then $f$ does not divide $g$ and we output this accordingly. Otherwise, we proceed with the case where both $f$ and $g$ are nonzero.

Following Forbes [For15], we execute Strassen's division elimination procedure [Str73] on an arithmetic circuit that computes $g(\vec{x})/f(\vec{x})$. This produces a circuit that computes a polynomial $h(\vec{x})$ which satisifes the identity $g(\vec{x}) - f(\vec{x})h(\vec{x}) = 0$ if and only if $f$ divides $g$. Thus, to check if $f$ divides $g$, it suffices to construct a

circuit that computes the polynomial $h(\vec{x})$ and then test the identity $g(\vec{x}) - f(\vec{x})h(\vec{x}) = 0$. We proceed to construct a circuit for the polynomial $h(\vec{x})$.

Because $f$ is nonzero and $|\mathbb{F}_n| > d \geq \deg(f)$, there is a point $\vec{\alpha} \in \mathbb{F}_n^n$ such that $f(\vec{\alpha}) \neq 0$. Translating this point to the origin, we may assume that $f(\vec{0}) \neq 0$. Since $f$ has a nonzero constant term, we can find an inverse of $f$ as a formal power series. Let $\hat{f}(\vec{x}) \coloneqq f(\vec{x}) - f(\vec{0})$ be the nonconstant part of $f(\vec{x})$. Then we have the equality of formal power series

$$\frac{1}{f(\vec{x})} = \frac{1}{f(\vec{0})} \cdot \frac{1}{1 - \left(\frac{-\hat{f}(\vec{x})}{f(\vec{0})}\right)} = \frac{1}{f(\vec{0})} \sum_{i=0}^{\infty} \left(\frac{-\hat{f}(\vec{x})}{f(\vec{0})}\right)^i.$$

Multiplying by $g(\vec{x})$, we have

$$\frac{g(\vec{x})}{f(\vec{x})} = \frac{g(\vec{x})}{f(\vec{0})} \sum_{i=0}^{\infty} \left(\frac{-\hat{f}(\vec{x})}{f(\vec{0})}\right)^i.$$

In the case where $f$ divides $g$, we know that the quotient $g/f$ is a polynomial of degree at most $d$, so the terms of degree at most $d$ on the right-hand side above will correspond to the quotient. Let $h(\vec{x})$ be the polynomial defined by

$$h(\vec{x}) \coloneqq \mathrm{Hom}_{\leq d}\left[\frac{g(\vec{x})}{f(\vec{0})} \sum_{i=0}^{\infty} \left(\frac{-\hat{f}(\vec{x})}{f(\vec{0})}\right)^i\right].$$

where $\mathrm{Hom}_{\leq d}[\bullet]$ is the projection onto the homogeneous components of degree at most $d$. Because $\hat{f}(\vec{x})$ has no constant term, the polynomial $\hat{f}(\vec{x})^i$ is only supported on monomials of degree $i$ and higher. This implies that when $i > d$, the term $\hat{f}(\vec{x})^i$ does not contribute to the homogeneous components of degree at most $d$ of the power series above. This lets us write

$$h(\vec{x}) = \mathrm{Hom}_{\leq d}\left[\frac{g(\vec{x})}{f(\vec{0})} \sum_{i=0}^{d} \left(\frac{-\hat{f}(\vec{x})}{f(\vec{0})}\right)^i\right],$$

so $h$ can be obtained from the homogeneous components of the finite sum above. When $f$ divides $g$, the preceding discussion implies that $h(\vec{x}) = g(\vec{x})/f(\vec{x})$, so we have the identity $g(\vec{x}) - f(\vec{x})h(\vec{x}) = 0$. On the other hand, if $f$ does not divide $g$, then we must have $g(\vec{x}) \neq f(\vec{x})h(\vec{x})$, as otherwise $f$ would be a divisor of $g$, so we have the nonidentity $g(\vec{x}) - f(\vec{x})h(\vec{x}) \neq 0$. Thus, to test if $f$ divides $g$, it suffices to test the identity $g(\vec{x}) - f(\vec{x})h(\vec{x}) = 0$.

We now explain how to carry out this reduction using an arithmetic network. The initial steps to check if either $f$ or $g$ is identically zero can be carried out by $\mathsf{PIT}$ gates. In the case where $f$ and $g$ are both nonzero, we need to find a point $\vec{\alpha} \in \mathbb{F}_n^n$ where $f(\vec{\alpha}) \neq 0$. We do this by carrying out the search-to-decision reduction for $\mathsf{PIT}$.

Fix a subset $S \subseteq \mathbb{F}_n$ of size $d + 1$. We will find such an $\vec{\alpha}$ in $S^n$ one coordinate at a time. For each $i \in [n]$, suppose we have determine the first $i - 1$ coordinates of $\vec{\alpha}$. Because $|S| > \deg(f)$ and $f(\alpha_1, \ldots, \alpha_{i-1}, x_i, \ldots, x_n) \neq 0$, one of the polynomials $\{f(\alpha_1, \ldots, \alpha_{i-1}, \beta, x_{i+1}, \ldots, x_n) : \beta \in S\}$ must be nonzero. We use $\mathsf{PIT}$ gates to determine which of these are nonzero, and we set $\alpha_i$ to be any one of the $\beta$ that results in a nonzero polynomial. Because this step only manipulates the circuit computing $f$ and evaluates $\mathsf{PIT}$ gates on the result, this step can be carried out by an arithmetic network of size $\mathrm{poly}(n, s, d)$ and degree $O(1)$. Further, it is clear that a polynomial-time Turing machine can print this part of the network.

With $\vec{\alpha}$ in hand, we replace $f(\vec{x})$ and $g(\vec{x})$ by $f'(\vec{x}) \coloneqq f(\vec{x} - \vec{\alpha})$ and $g'(\vec{x}) \coloneqq g(\vec{x} - \vec{\alpha})$, respectively. Because $f(\vec{x})$ divides $g(\vec{x})$ if and only if $f'(\vec{x})$ divides $g'(\vec{x})$, it suffices to test divisibility of $f'(\vec{x})$ and $g'(\vec{x})$. We know that $f'(\vec{0}) = f(\vec{\alpha}) \neq 0$. Let

$$h'(\vec{x}) \coloneqq \mathrm{Hom}_{\leq d}\left[\frac{g'(\vec{x})}{f'(\vec{0})} \sum_{i=0}^{d} \left(\frac{f'(\vec{0}) - f'(\vec{x})}{f'(\vec{0})}\right)^i\right]$$

as above. We can compute a description of a circuit computing $h'(\vec{x})$ by directly writing a circuit that computes the polynomial

$$\frac{g'(\vec{x})}{f'(\vec{0})} \sum_{i=0}^{d} \left( \frac{f'(\vec{0}) - f'(\vec{x})}{f'(\vec{0})} \right)^i$$

and then applying Lemma A.6 to extract the homogeneous components of degree up to $d$. This results in a description of a circuit of size $O(sd^3)$ that computes $h$, and this description can be computed by a network of size $\mathrm{poly}(n, s, d)$ and degree $O(1)$.

Once we have a description of a circuit that computes $h'(\vec{x})$, we compute a description of a circuit that computes $g'(\vec{x}) - f'(\vec{x})h'(\vec{x})$. We then use a PIT gate to test if this circuit computes the zero polynomial, and we report that $f$ divides $g$ exactly when $g' - f'h'$ is the zero polynomial. The correctness of this algorithm follows from the preceding discussion. This last step can likewise be carried out by a uniform network of size $\mathrm{poly}(n, s, d)$ and degree $O(1)$ as desired. □

Our next problem of interest is computing the greatest common divisor of polynomials. The following lemma shows that the greatest common divisor of two arithmetic circuits $C_1$ and $C_2$ can be computed by an arithmetic circuit $D$ of comparable size, and that a description of $D$ can be computed efficiently by a uniform family of arithmetic networks.

**Lemma A.10** (Greatest common divisor of arithmetic circuits)**.** *There is a deterministic, polynomial-time Turing machine that receives as input $(1^n, 1^d, 1^s)$ and outputs an arithmetic network with* PIT *gates that satisfies the following properties.*

1. *The network receives as input the descriptions of two arithmetic circuits $C_1$ and $C_2$ of size $s$ over $\mathbb{F}_n$, computing polynomials $f, g \in \mathbb{F}_n(x_1, \ldots, x_n)[y]$, respectively, both of degree at most $d$.*
2. *The network outputs a description of an arithmetic circuit $D$ of size $\mathrm{poly}(s, d)$. The arithmetic circuit $D$ computes the polynomial $\gcd(f, g) \in \mathbb{F}_n(\vec{x})[y]$.*
3. *The network has size $\mathrm{poly}(n, s, d)$ and degree $\mathrm{poly}(d)$.*

**Proof.** Use the network of Lemma A.4 to obtain descriptions of arithmetic circuits of size $O(sd^2)$ that compute the coefficients of $f$ and $g$ with respect to $y$. To compute the GCD of $f$ and $g$, we execute the Euclidean algorithm on $f$ and $g$, using PIT gates to both implement division with remainder[42] and to detect when to stop. The Euclidean algorithm runs for at most $d$ iterations. In each iteration, we have descriptions of arithmetic circuits that compute the coefficients of two polynomials $r_{i-1}, r_i \in \mathbb{F}_n(\vec{x})[y]$, and we compute $r_{i+1} \coloneqq r_i \bmod r_{i-1}$, the remainder when $r_i$ is divided by $r_{i-1}$.

Because $r_{i-1}$ and $r_i$ have degree at most $d$, the coefficients of $r_{i+1}$ can be computed from the coefficients of $r_{i-1}$ and $r_i$ using $\mathrm{poly}(d)$ arithmetic operations. This implies that the circuit complexity of the coefficients of $r_{i+1}$ increases by an additive $\mathrm{poly}(d)$ factor in each step of the Euclidean algorithm, so throughout the execution of the algorithm we only need to operate on arithmetic circuits of size $\mathrm{poly}(s, d)$. This yields the claimed bound of $\mathrm{poly}(n, s, d)$ on the size of the arithmetic network. The bound of $\mathrm{poly}(d)$ on the degree of the network follows from the fact that we only perform division with remainder on degree-$d$ polynomials, which can be implemented by a network of degree $\mathrm{poly}(d)$. The uniformity of the network follows from the uniformity of the Euclidean algorithm. □

For polynomials $f, g, h \in \mathbb{F}[x]$, we have the identity

$$\gcd(f, g, h) = \gcd(\gcd(f, g), h),$$

so we can repeatedly apply Lemma A.10 to compute the GCD of more than two polynomials. To compute the GCD of $k$ polynomials, each of degree $d$, we can apply Lemma A.10 a total of $O(\log k)$ times in parallel, resulting in a network of degree bounded by $d^{O(\log k)}$. Later in Section A.3, we will want to compute the GCD of $O(d)$ polynomials of degree $d$. Taking GCD's one-by-one results in a network of degree $d^{O(\log d)}$. We can

[42] Strictly speaking, the use of PIT gates is not necessary to implement division with remainder. Given two polynomials $f$ and $g$, let $f = qg + r$ with $\deg(r) < \deg(g)$. The coefficients of $q$ and $r$ are rational functions of the coefficients of $f$ and $g$, and so can be computed by arithmetic circuits without the use of branching or PIT gates.

improve this degree bound to $\mathrm{poly}(d)$ by using an algorithm that is designed to compute the GCD of many polynomials at once. Our application in Section A.3 will only require us to compute the GCD of polynomials whose coefficients are field elements, not rational functions computed by small arithmetic circuits, so we specialize the statement of Lemma A.11 below to this setting. We use an algorithm due to Steve Cook as reported by von zur Gathen [vzGat84, Theorem 3.1]. The recent GCD algorithm of Andrews and Wigderson [AW24] would work equally well for our purposes here.

**Lemma A.11** (Greatest common divisor of many polynomials [vzGat84, Theorem 3.1])**.** *There is a deterministic, polynomial-time Turing machine that receives as input $(1^n, 1^d)$ and outputs an arithmetic network that satisfies the following properties.*

1. *The network receives as input the coefficients of $n$ univariate polynomials $f_1, \ldots, f_n \in \mathbb{F}_n[x]$, each of degree at most $d$.*
2. *The network outputs the coefficients of $\gcd(f_1, \ldots, f_n) \in \mathbb{F}_n[x]$.*
3. *The network has size and degree $\mathrm{poly}(n, d)$.*

**Proof.** As shown by von zur Gathen [vzGat84, Theorem 3.1], we can compute $\gcd(f_1, \ldots, f_n)$ by solving a collection of linear systems. For each $k \in \{0, 1, \ldots, d\}$, we form a system of linear equations that expresses the existence of polynomials $g_1, \ldots, g_n$ of degree less than $d$ such that $\sum_{i=1}^{n} f_i(x) g_i(x)$ is a monic polynomial of degree $k$. Write $f_i(x) = \sum_{j=0}^{d} f_{i,j} x^j$ and let $g_i(x) = \sum_{j=0}^{d-1} g_{i,j} x^j$. Consider the system of linear equations

$$\sum_{i=1}^{n} \sum_{j=0}^{\ell} g_{i,j} f_{i,\ell-j} = \begin{cases} 1 & \text{if } \ell = k, \\ 0 & \text{if } k < \ell \leq 2d, \end{cases}$$

where the $f_{i,j}$ are known and the $g_{i,j}$ are unknown. This system has a solution exactly when there are $g_1, \ldots, g_n \in \mathbb{F}_n[x]$ such that $\sum_{i=1}^{n} f_i g_i$ is monic of egree $k$.

To compute the GCD, we solve all of these systems in parallel for $k \in \{0, 1, \ldots, d\}$ using the network of Lemma A.8. Letting $g_1(x), \ldots, g_n(x)$ be the solution of this system, we output $\sum_{i=1}^{n} f_i(x) g_i(x)$ as $\gcd(f_1, \ldots, f_n)$. The correctness of this algorithm follows from [vzGat84, Theorem 3.1]. The network of Lemma A.8 has size $\mathrm{poly}(n, d)$ and degree $O(1)$, so by Proposition 3.19, the parallel invocation of $d$ copies of this network likewise has size $\mathrm{poly}(n, d)$ and degree $O(1)$. Lemma A.8 provides us a network with $\mathsf{PIT}$ gates, but since the coefficients of the polynomials $f_1, \ldots, f_n$ are provided directly as input (and are not represented as arithmetic circuits), we can emulate the network of Lemma A.8 without using $\mathsf{PIT}$ gates. □

In addition to the GCD, we can also compute the Bézout coefficients of two polynomials efficiently by means of uniform arithmetic networks. Recall that for two univariate polynomials $f, g \in \mathbb{F}_n[x]$, their Bézout coefficients are two polynomials $a, b \in \mathbb{F}_n[x]$ satisfying $\deg(a) < \deg(g)$, $\deg(b) < \deg(f)$, and

$$af + bg = \gcd(f, g).$$

The Bézout coefficients are a useful tool in computer algebra. For our purposes, we will make use of them in Lemma A.13 to implement Hensel lifting as a uniform family of arithmetic networks. The interested reader can find more information about the Bézout coefficients in the book of von zur Gathen and Gerhard [vzGG13].

**Lemma A.12** (Bézout coefficients)**.** *There is a deterministic, polynomial-time Turing machine that receives as input $(1^n, 1^d)$ and outputs an arithmetic network satisfying the following properties.*

1. *The network receives as input the coefficients of two univariate polynomials $f, g \in \mathbb{F}_n[x]$, both of degree at most $d$.*
2. *The network outputs the coefficients of two polynomials $a, b \in \mathbb{F}_n[x]$ such that*
   *(a)* $\deg(a) < \deg(g)$,
   *(b)* $\deg(b) < \deg(f)$, *and*

(c) $af + bg = \gcd(f, g)$.

3. *The network has size and degree* $\mathrm{poly}(d)$.

**Proof.** Execute the extended Euclidean algorithm on the polynomials $f$ and $g$. The desired polynomials $a$ and $b$ are computed as part of the output of the extended Euclidean algorithm. The extended Euclidean algorithm runs for at most $d$ steps, and in each step computes polynomial division with remainder, which can be implemented by an arithmetic network of size and degree $\mathrm{poly}(d)$, so the extended Euclidean algorithm can likewise be implemented by an arithmetic network of size and degree $\mathrm{poly}(d)$. The uniformity of the network follows from the uniformity of the extended Euclidean algorithm. □

We conclude this section by showing that Hensel lifting can be implemented by a uniform family of arithmetic networks. Hensel lifting is an iterative procedure that, for our purposes, starts with an approximate factorization of a polynomial $f$ and improves the accuracy of this factorization. More specifically, for an ideal $I \subseteq \mathbb{F}[\vec{x}]$, if we have a factorization

$$f(\vec{x}) = g_0(\vec{x})h_0(\vec{x}) \bmod I,$$

then Hensel lifting computes, for each $k \in \mathbb{N}$, a pair of polynomials $g_k$ and $h_k$ such that

$$f(\vec{x}) = g_k(\vec{x})h_k(\vec{x}) \bmod I^{2^k}.$$

For us, Hensel lifting will be used as a subroutine in Kaltofen's algorithm [Kal89] to factor arithmetic circuits. We will not need the full details of the Hensel lifting procedure, so we do not cover them all here. The interested reader can find more details on Hensel lifting in the work of Kopparty, Saraf, and Shpilka [KSS15] or the book of von zur Gathen and Gerhard [vzGG13].

**Lemma A.13** (Hensel lifting [KSS15, Lemma 3.6])**.** *There is a deterministic, polynomial-time Turing machine that receives as input* $(1^n, 1^d, 1^s, 1^{2^k})$ *and outputs an arithmetic network satisfying the following properties.*

1. *The network receives as input the description of an arithmetic circuit* $C$ *of size* $s$ *that computes a polynomial* $f(\vec{x}, y, z) \in \mathbb{F}_n[x_1, \ldots, x_n][y, z]$ *of degree at most* $d$*. It also receives the coefficients of polynomials* $g_0, h_0, a_0, b_0 \in \mathbb{F}_n[y]$ *that satisfy the following conditions.*
   (a) *The polynomial* $g_0(y)$ *is an irreducible monic factor of* $f(\vec{0}, y, 0)$.
   (b) *The polynomials* $g_0(y)$ *and* $h_0(y)$ *satisfy the identity* $f(\vec{0}, y, 0) = g_0(y) \cdot h_0(y)$.
   (c) *The polynomials* $a_0(y)$ *and* $b_0(y)$ *satisfy the identity* $a_0(y)g_0(y) + b_0(y)h_0(y) = 1$*. Furthermore, we have* $\deg(a_0) < \deg(h_0)$ *and* $\deg(b_0) < \deg(g_0)$.
2. *The network outputs the description of a multi-output arithmetic circuit* $D$ *over* $\mathbb{F}_n$*. The circuit* $D$ *has size* $\mathrm{poly}(n, s, d, 2^k)$*, degree* $\mathrm{poly}(d, 2^k)$*, and outputs* $\mathrm{poly}(d, 2^k)$ *many polynomials. The outputs of* $D$ *are the the coefficients of two polynomials* $g_k(y, z), h_k(y, z) \in \mathbb{F}_n[\vec{x}][y, z]$ *of individual degree at most* $\max(d, 2^k)$ *that satisfy the following conditions.*
   (a) *Letting* $\langle z^{2^k} \rangle \subseteq \mathbb{F}_n(\vec{x})[y, z]$ *denote the ideal generated by* $z^{2^k}$*, the polynomials* $g_k$ *and* $h_k$ *satisfy the equality*
   $$f(\vec{x}, y, z) = g_k(y, z) \cdot h_k(y, z) \bmod \langle z^{2^k} \rangle.$$
   (b) *The polynomial* $g_k(y, z)$ *satisfies* $g_k(y, 0) = g_0(y)$.
   (c) *The polynomial* $g_k(y, z)$ *is monic with respect to* $y$.
3. *The network has size* $\mathrm{poly}(n, s, d, 2^k)$ *and degree* $O(1)$.

**Proof.** We construct the circuit $C$ iteratively. For $i \in \{0, 1, \ldots, k-1\}$, suppose we have constructed a circuit $C_i$ of size $\mathrm{poly}(n, s, d, 2^i)$ and degree $\mathrm{poly}(d, 2^i)$ that computes the coefficients of polynomials $g_i, h_i, a_i, b_i \in \mathbb{F}_n[\vec{x}][y, z]$ such that

$$f(\vec{x}, y, z) = g_i(y, z) \cdot h_i(y, z) \bmod \langle z^{2^i} \rangle,$$
$$a_i(y, z)g_i(y, z) + b_i(y, z)h_i(y, z) = 1 \bmod \langle z^{2^i} \rangle,$$

and $g_i(y,z)$ is monic with respect to $y$. From the coefficients of $g_i$, $h_i$, $a_i$, and $b_i$, we will compute the coefficients of polynomials $g_{i+1}, h_{i+1}, a_{i+1}, b_{i+1} \in \mathbb{F}_n[\vec{x}][y,z]$ such that the same equalities hold modulo the ideal $\langle z^{2^{i+1}} \rangle$. That is, we will have

$$f(\vec{x},y,z) = g_{i+1}(y,z) \cdot h_{i+1}(y,z) \bmod \langle z^{2^{i+1}} \rangle,$$
$$a_{i+1}(y,z)g_{i+1}(y,z) + b_{i+1}(y,z)h_{i+1}(y,z) = 1 \bmod \langle z^{2^{i+1}} \rangle,$$

and $g_{i+1}$ is likewise monic with respect to $y$.

To compute the coefficients of $g_{i+1}, h_{i+1}, a_{i+1}$, and $b_{i+1}$, we implement a single step of Hensel lifting as described in, e.g., [KSS15, Lemma 3.4]. For our purposes, the precise details of a Hensel lifting step are not important; it is enough to know that a single step consists of a constant number of arithmetic operations and one division with remainder, and that there is a uniform description of a lifting step. This implies that we can construct the circuit $C_{i+1}$ by adding an additional $\mathrm{poly}(d, 2^k)$ gates to the circuit $C_i$. To bound the degree of the final circuit $C_k$, we use the fact that each output of $C_k$ has degree bounded by $d2^k$, so we can homogenize $C_k$ using lemma A.6 to obtain a circuit whose degree is likewise bounded by $d2^k$.

The bound on the size of the resulting arithmetic network is clear. The network itself does not do any nontrivial arithmetic computation; it only adds gates to the given circuit $C$. This implies that the degree of the network is bounded by $O(1)$. The uniformity of the arithmetic network follows from the uniformity of the Hensel lifting algorithm. □

## A.3 Univariate factorization over finite fields

An important step in factoring polynomials given by arithmetic circuits is factorization of univariate polynomials. For our purposes (in particular, our targeted hitting set generator based on the Kabanets–Impagliazzo generator in Section 6), it will be enough to have an algorithm to factor univariate polynomials over finite fields. A randomized polynomial-time algorithm to factor polynomials over finite fields was first described by Berlekamp [Ber70]. Later work by von zur Gathen [vzGat84] showed that the randomized algorithm of Cantor and Zassenhaus [CZ81] admits an efficient parallel implementation.

The algorithm of [vzGat84] is essentially formalized as a uniform family of low-depth arithmetic networks, so we can easily conclude that univariate factorization over finite fields can be solved by uniform arithmetic networks. However, our notions of the degree and error degree of a randomized arithmetic network do not appear in [vzGat84], so we must be careful to bound these quantities. We state and prove this result only for factoring squarefree polynomials over finite fields. To factor general polynomials, we need the additional ability to compute $p^{\text{th}}$ roots, where $p$ is the characteristic of the underlying field, and this necessitates either the use of $p^{\text{th}}$ root gates in our network or operations of large degree. For more on factorization over finite fields, we refer the reader to [vzGG13, Chapter 14].

**Theorem A.14** (Univariate factorization over finite fields [vzGat84, Theorem 4.1])**.** *Let $\mathbb{F} = \{\mathbb{F}_{q(n)}\}_{n\in\mathbb{N}}$ be a feasible family of finite fields, where $\mathbb{F}_{q(n)}$ is a finite field of order $q(n)$ and characteristic $p(n)$. There is a deterministic, polynomial-time Turing machine $M_1$ that receives as input $(1^n, 1^d)$ and outputs a randomized arithmetic network satisfying the following properties.*

1. *The network receives as input the coefficients of a degree-$d$ polynomial $f \in \mathbb{F}_{q(n)}[x]$.*
2. *If the input polynomial $f$ is squarefree, then the network outputs the complete factorization of $f$ into irreducible polynomials. Otherwise, the output of the network is meaningless.*
3. *The network has size $\mathrm{poly}(d, \log q(n))$, degree $d^{O(\log(q(n)))}$, and error degree 1.*

*In addition, there is another deterministic, polynomial-time Turing machine $M_2$ that receives as input $(1^n, 1^d)$ and outputs a randomized arithmetic network satisfying the following properties.*

1. *The network receives as input the coefficients of an irreducible polynomial $t \in \mathbb{F}_{p(n)}[x]$ such that $\mathbb{F}_{q(n)}[x] = \mathbb{F}_{p(n)}[x]/\langle t(x) \rangle$. The network also receives as input the coefficients of a degree-$d$ polynomial $f \in \mathbb{F}_{q(n)}[x]$, each represented as a tuple of elements from $\mathbb{F}_{p(n)}$.*

2. *If the input polynomial $f$ is squarefree, then the network outputs the complete factorization of $f$ into irreducible polynomials. Otherwise, the output of the network is meaningless.*

3. *The network has size* $\mathrm{poly}(d, p)$*, degree* $\mathrm{poly}(d, p)$*, and error degree* 1*.*

**Proof.** We first describe von zur Gathen's modified version of the Cantor–Zassenhaus algorithm [vzGat84, Theorem 4.1], bounding the size and degree of the resulting arithmetic network along the way, and then bound the error degree of the algorithm. As our concern is primarily on the complexity of the algorithm and not its correctness, we address issues of correctness only at a level of detail that suffices to establish the claimed bounds on the complexity of the algorithm. For simplicity, we write $q := q(n)$ and $p := p(n)$ for the order and characteristic of $\mathbb{F}_{q(n)}$, respectively. Let $k \in \mathbb{N}$ be a natural number such that $q = p^k$. Throughout, let $R := \mathbb{F}_q[x]/\langle f \rangle$ be the quotient of $\mathbb{F}_q[x]$ by the ideal generated by $f$. We assume, without loss of generality, that our input polynomial $f$ is monic.

We begin by recalling the algorithm of von zur Gathen [vzGat84].

1. In the case where we do not have a representation of $\mathbb{F}_q$ as an extension of $\mathbb{F}_p$, we start by computing the matrix $Q \in \mathbb{F}_q^{d \times d}$ corresponding to the linear map $R \to R$ given by $a \mapsto a^q$. Since $\{1, x, x^2, \ldots, x^{d-1}\}$ is a basis of $R$ over $\mathbb{F}_q$, it suffices to compute representatives of $x^{iq}$ in $R$ for $i \in \{0, 1, \ldots, d-1\}$. The representative of $x^{iq}$, written in the basis $\{1, x, \ldots, x^{d-1}\}$, is precisely the remainder when $x^{iq}$ is divided by $f(x)$.

   To compute the remainder of $x^{iq}$ on division with $f(x)$, we compute $x^{iq}$ via repeated squaring in the quotient ring $R = \mathbb{F}_q[x]/\langle f \rangle$. Given a polynomial $g(x)$ of degree at most $d$ such that $g(x) = x^k \bmod f(x)$, we can compute a representative of $x^{2k}$ by computing $g(x)^2 \bmod f(x)$. The polynomial $g(x)^2$ has degree at most $2d$, so this polynomial division with remainder can be carried out with an arithmetic network of size and degree $\mathrm{poly}(d)$. There are $O(\log(qd))$ squaring operations, so computing representatives of the $x^{iq}$ can be done with size $\mathrm{poly}(d, \log(q))$ and degree $d^{O(\log(d)+\log(q))}$.

   In the case where we are given a representation of $\mathbb{F}_q$ as an extension of $\mathbb{F}_p$, we instead let $Q \in \mathbb{F}_q^{d \times d}$ be the matrix corresponding to the linear map $R \to R$ given by $a \mapsto a^p$. As above, to compute this matrix, we find representations of the polynomials $\{x^{ip} : i \in \{0, 1, \ldots, d-1\}\}$ in the basis $\{1, x, \ldots, x^{d-1}\}$ of $R$. We do this using polynomial division with remainder, but instead of using repeated squaring in $R$, we simply write down the polynomial $x^{ip}$ and then find its remainder on division with $f(x)$. Each division with remainder can be carried out by an arithmetic network of size $\mathrm{poly}(d, p)$ and degree $\mathrm{poly}(d, p)$, and there are $d$ such networks run in parallel, so this step can likewise be implemented in size $\mathrm{poly}(d, p)$ and degree $\mathrm{poly}(d, p)$.

2. We next compute the dimension of and a basis for the nullspace of $Q - I_d$, where $I_d \in \mathbb{F}_q^{d \times d}$ is the identity matrix. Let $r$ be the dimension of this nullspace, and let $g_1, \ldots, g_r \in \mathbb{F}_q[x]$ be polynomials such that $\{g_i \bmod f : i \in [r]\}$ is a basis for the nullspace of $Q - I_d$. The value of $r$ corresponds to the number of irreducible factors of $f$. If $r = 1$, then we report that $f$ is irreducible and the algorithm terminates. Otherwise, we proceed with factorization of $f$. (Here, we used the assumption that $f$ is squarefree: if $f$ is squarefree and has only one irreducible factor, then $f$ itself must be this irreducible factor, hence $f$ is irreducible.)

   We can compute $r$ and a basis for the nullspace of $Q - I_d$ using the network of Lemma A.8. In this case, the entries of the matrix $Q - I_d$ are field elements, so the basis produced by Lemma A.8 will likewise consist of vectors over the field $\mathbb{F}_q$. In particular, this invocation of Lemma A.8 does not require the use of PIT gates.

3. In the case where $r \geq 2$, let $m := 5 \log_{9/5}(r)$. Let $v_{1,1}, \ldots, v_{m,r}$ be chosen independently at random from $\mathbb{F}_q$ or $\mathbb{F}_p$, depending on whether we have a representation of $\mathbb{F}_q$ as an extension of $\mathbb{F}_p$. For $i \in [m]$, define $h_i(x) := \sum_{j=1}^{r} v_{i,j} g_j(x)$. The coefficients of each $h_i$ can be computed by a network of size $O(rd) \leq O(d^2)$ and degree 2.

4. For each $i \in [m]$, we compute a polynomial $c_i(x)$ that is likely to be a proper factor of $f(x)$. If we are

not given a representation of $\mathbb{F}_q$ as an extension of $\mathbb{F}_p$, we compute

$$c_i(x) \coloneqq \begin{cases} \gcd(f(x), h_i(x)^{\frac{q-1}{2}} - 1) & \text{if } p \text{ is odd,} \\ \gcd(f(x), \sum_{j=0}^{\log_2(q)-1} h_i(x)^{2^j}) & \text{otherwise.} \end{cases}$$

If instead we have a representation of $\mathbb{F}_q$ as an extension of $\mathbb{F}_p$, we compute

$$c_i(x) \coloneqq \begin{cases} \gcd(f(x), h_i(x)^{\frac{p-1}{2}} - 1) & \text{if } p \text{ is odd,} \\ \gcd(f(x), h_i(x)) & \text{otherwise.} \end{cases}$$

Some of the polynomials $c_i$ will be proper factors of $f$, which can be thought of as a partial factorization of $f$. We improve this to a finer factorization of $f$ as follows. For every $I \subseteq \{0,1\} \times [m]$, we compute

$$s_I(x) \coloneqq \gcd\left(\{c_i(x) : (0,i) \in I\} \cup \left\{\frac{f(x)}{c_i(x)} : (1,i) \in I\right\}\right).$$

We then compute the set

$$T \coloneqq \{I \subseteq \{0,1\} \times [m] : s_I \neq 1 \text{ and for all } J \supseteq I, \text{ either } s_J = 1 \text{ or } s_J = s_I\},$$

which corresponds to the minimal choices of $I$ that produce the finest nontrivial factorization of $f$.

If $|T| = r$, then the polynomials $\{s_I : I \in T\}$ are the irreducible factors of $f$. In this case, the network outputs the coefficients of these polynomials and terminates. Otherwise, we have $|T| \neq r$, in which case the network fails to factor $f$ into irreducibles and reports this failure accordingly.

To compute the polynomial $c_i(x)$ when we don't have a representation of $\mathbb{F}_q$ as an extension of $\mathbb{F}_p$, we first compute

$$g_i(x) \coloneqq \begin{cases} h_i(x)^{\frac{q-1}{2}} - 1 \bmod f(x) & \text{if } q \text{ is odd,} \\ \sum_{j=0}^{\log_2(q)-1} h_i(x)^{2^j} \bmod f(x) & \text{otherwise} \end{cases}$$

in a manner similar to how we computed $x^{iq} \bmod f(x)$ in step (1). In both situations, we compute the necessary powers of $h_i$ via repeated squaring, performing division with remainder after each squaring operation to ensure we always square a polynomial of degree bounded by $d$. This requires $O(\log(q))$ squaring operations, each of which can be implemented by a network of size and degree bounded by $\mathrm{poly}(d)$, so each $g_i$ can be computed by a network of size $\mathrm{poly}(d, \log(q))$ and degree $d^{O(\log q)}$. Once we have computed $g_i$, we can compute $c_i(x) = \gcd(f(x), g_i(x))$ using Lemma A.10, which is implemented as a network of size and degree bounded by $\mathrm{poly}(d)$.

If we instead have a representation of $\mathbb{F}_q$ as an extension of $\mathbb{F}_p$, we directly compute the coefficients of $h_i(x)^{\frac{p-1}{2}} - 1$ (if necessary) and then apply the network of Lemma A.10 to compute the polynomial $c_i(x)$. This can be implemented by a network of size and degree $\mathrm{poly}(d, p)$.

Once the $c_i(x)$'s have been computed, we can compute each $s_I(x)$ using Lemma A.11. There are at most $2^{2m} \leq \mathrm{poly}(r)$ choices of $I \subseteq \{0,1\} \times [m]$, and for each such $I$, the polynomial $s_I(x)$ can be computed by direct application of the network of Lemma A.11. This computes each $s_I(x)$ using a network of size $\mathrm{poly}(r, d) \leq \mathrm{poly}(d)$ and degree $\mathrm{poly}(r, d) \leq \mathrm{poly}(d)$. In all, we have computed each $s_I(x)$ using a network of size $\mathrm{poly}(d, \log q)$ and degree $d^{O(\log q)}$ when $\mathbb{F}_q$ is not presented as an extension of $\mathbb{F}_p$, and size $\mathrm{poly}(d, p)$ and degree $\mathrm{poly}(d, p)$ when $\mathbb{F}_q$ is given as an extension of $\mathbb{F}_p$. By Proposition 3.19 we can compute all the $s_I(x)$ in parallel using a network of size $\mathrm{poly}(d)$ and degree $d^{O(\log q)}$ in the former case, and size $\mathrm{poly}(d, p)$ and degree $\mathrm{poly}(d, p)$ in the latter case.

The set $T$ can be computed directly by a network of $\mathrm{poly}(d)$ size and degree $O(1)$: for $I, J \subseteq \{0,1\} \times [m]$ with $I \subseteq J$, if either $s_I(x) = 1$ or $s_J(x) \neq 1$ and $s_J(x) \neq s_I(x)$, then we know that $I \notin T$. Thus, we can check if $I \in T$ by checking all choices of $J \supseteq I$, of which there are at most $2^{2m} \leq \mathrm{poly}(r) \leq \mathrm{poly}(d)$. This clearly costs $\mathrm{poly}(d)$ size. Since the computation of $T$ is done by the boolean part of the network, this computation only incurs degree $O(1)$.

The preceding description bounds the size and degree of the resulting arithmetic network. The uniformity of the arithmetic network follows from the uniform description of the algorithm above. To bound the error degree of the network, we need to understand when the polynomials $s_I(x)$ appearing in step (4) fail to provide a complete factorization of $f$ into irreducible factors.

To do this, we follow von zur Gathen's analysis of his algorithm. Let $f = f_1 \cdots f_t$ be the factorization of $f$ into irreducible factors. (We assume that the given polynomial $f$ is squarefree, so each irreducible factor occurs with multiplicity 1.) Recall that $R = \mathbb{F}_q[x]/\langle f \rangle$ and define $R' \coloneqq \mathbb{F}_q[x]/\langle f_1 \rangle \times \cdots \times \mathbb{F}_q[x]/\langle f_t \rangle$. By the Chinese remainder theorem, there is an isomorphism of rings $\varphi : R \to R'$.

We first consider the case when $p$ is odd and we are not given a representation of $\mathbb{F}_q$ as an extension of $\mathbb{F}_p$. For a polynomial $h \in \mathbb{F}_q[x]$, we say that $h$ separates the factors $f_i$ and $f_j$ if exactly one of $f_i$ and $f_j$ divides $h^{\frac{q-1}{2}} - 1$. Let $\overline{h}$ be the image of $h$ in $R$ and let $\alpha(h) = (u_1, \ldots, u_r) \in \mathbb{F}_q^r$. We know from [vzGat84] that $h$ separates $f_i$ and $f_j$ exactly when one of $u_i$ and $u_j$ satisfies $u^{\frac{q-1}{2}} - 1 = 0$. There are exactly $\frac{q^2-1}{2}$ such pairs $(u_i, u_j)$, so the probability that a single randomly-chosen $h$ separates $f_i$ and $f_j$ is

$$\frac{q^2 - 1}{2q^2} \geq \frac{4}{9},$$

since $q \geq 3$. This implies that the probability that none of the $h_k$ separate the factors $f_i$ and $f_j$ is bounded by $(5/9)^m$. By a union bound, the probability that some pair of factors $f_i$ and $f_j$ is not separated is bounded by

$$\binom{r}{2}\left(\frac{5}{9}\right)^m \leq r^2 \left(\frac{5}{9}\right)^{\log_{9/5}(r)} = \frac{1}{r^3} \leq \frac{1}{8}.$$

The last inequality above uses the fact that the algorithm only errs when $f$ is reducible, in which case $r \geq 2$. If every pair of factors is separated, then the algorithm correctly returns a factorization of $f$ into irreducible factors, so the error probability of the algorithm is bounded by $1/8$. This bounds the error degree of the algorithm by $q/8$.

If $p$ is odd and we are given a representation of $\mathbb{F}_q$ as an extension of $\mathbb{F}_p$, then the preceding analysis still applies, replacing $q$ with $p$ everywhere. Since our random elements are drawn from $\mathbb{F}_p$ and not $\mathbb{F}_q$, we instead obtain a bound of $p/8$ on the error degree of the algorithm.

If $p$ is even and we are not given a representation of $\mathbb{F}_q$ as an extension of $\mathbb{F}_p$, we instead say that $h$ separates the factors $f_i$ and $f_j$ if exactly one of $f_i$ and $f_j$ divides $\sum_{j=0}^{\log_2(q)-1} h_i(x)^{2^j}$. von zur Gathen [vzGat84] shows that a randomly-chosen $h$ separates $f_i$ and $f_j$ with probability exactly $1/2$, so the same analysis as in the case of odd $p$ applies. If we are also given a representation of $\mathbb{F}_q$ as an extension of $\mathbb{F}_p$, then von zur Gathen [vzGat84] likewise shows that for a randomly-chosen $h$, with probability $1/2$ exactly one of $f_i$ and $f_j$ will divide $h(x)$. Again, the same analysis as in the case of odd $p$ applies.

For the improved bound of 1 on the error degree, we run $\log_8(q)$ or $\log_8(p)$ copies of the preceding algorithm in parallel, depending on whether or not $\mathbb{F}_q$ is presented as an extension of $\mathbb{F}_p$. Each independent run fails with probability $1/8$, so the probability that all copies fail is bounded by either $(1/8)^{\log_8(q)} = 1/q$ or $(1/8)^{\log_8(p)} = 1/p$. All successful runs of the algorithm will output the same set of irreducible factors of $f$, possibly up to permutation, so we may output the results of the first successful run of the algorithm. A run of the algorithm succeeds if the set $T$ computed in step (4) has size $r$, so we can determine which runs of the algorithm are successful and output the first such run. Hence the algorithm fails either with probability $1/q$ or $1/p$ as appropriate, so the error degree is bounded by 1. □

## A.4 Factorization of arithmetic circuits over finite fields

Our goal in this subsection is to argue that Kaltofen's [Kal89] algorithm to factor arithmetic circuits over finite fields can itself be implemented as a uniform family of randomized arithmetic networks.

### A.4.1 The resultant and discriminant

Before presenting Kaltofen's algorithm, we need to recall the notions of the resultant and the discriminant. These are well-known tools that are useful in the study of polynomial factorization, and we will make use of them to argue correctness of our presentation of Kaltofen's algorithm. For more on the resultant, we refer the reader to [vzGG13, Chapter 6].

**Definition A.15** (Resultant)**.** *Let $\mathbb{F}$ be a field and let $f, g \in \mathbb{F}[x]$ be univariate polynomials. Write $f(x) = \sum_{i=0}^{n} f_i x^i$ and $g(x) = \sum_{i=0}^{m} g_i x^i$, with $f_n \neq 0$ and $g_m \neq 0$. The* resultant *of $f$ and $g$, denoted $\operatorname{res}(f, g)$, is given by the determinant*

$$\operatorname{res}(f,g) \coloneqq \det \begin{pmatrix} f_n & & & & g_m & & & & \\ f_{n-1} & f_n & & & g_{m-1} & g_m & & & \\ \vdots & \vdots & \ddots & & \vdots & \vdots & \ddots & & \\ \vdots & \vdots & & f_n & g_1 & \vdots & & \ddots & \\ \vdots & \vdots & & f_{n-1} & g_0 & \vdots & & & \ddots \\ \vdots & \vdots & & \vdots & & g_0 & & & g_m \\ f_0 & \vdots & & \vdots & & & \ddots & & \vdots \\ & f_0 & & \vdots & & & & \ddots & \vdots \\ & & \ddots & \vdots & & & & & \ddots \\ & & & f_0 & & & & & g_0 \end{pmatrix},$$

*where the above matrix is of size $(n+m) \times (n+m)$, the first parallelogram consists of $m$ columns, and the second parallelogram consists of $n$ columns.*

The utility of the resultant arises from the fact that it detects when two polynomials share a common factor.

**Lemma A.16** (see, e.g., [vzGG13, Corollary 6.17])**.** *Let $\mathbb{F}$ be a field and let $f, g \in \mathbb{F}[x]$ be univariate polynomials. Then $\operatorname{res}(f, g) = 0$ if and only if $f$ and $g$ have a common factor.*

The resultant of two polynomials $f$ and $g$ can always be written as $s(x)f(x) + t(x)g(x)$, where $s, t \in \mathbb{F}[x]$ are univariate polynomials satisfying $\deg(s) < \deg(g)$ and $\deg(t) < \deg(f)$. We will not need the fact that this linear combination can be found in low degree, so we do not quote that here.

**Lemma A.17** (see, e.g., [vzGG13, Corollary 6.21])**.** *Let $\mathbb{F}$ be a field and let $f, g \in \mathbb{F}[x]$ be univariate polynomials. There are polynomials $s, t \in \mathbb{F}[x]$ such that $s(x)f(x) + t(x)g(x) = \operatorname{res}(f, g)$.*

By specializing $g$ to $\frac{\partial f}{\partial x}$ in the resultant, we obtain the discriminant of a single polynomial.

**Definition A.18** (Discriminant)**.** *Let $\mathbb{F}$ be a field and let $f \in \mathbb{F}[x]$ be a univariate polynomial. The* discriminant *of $f$, denoted by $\operatorname{disc}(f)$, is the resultant $\operatorname{res}(f, \frac{\partial f}{\partial x})$.*

Just as the resultant checks if two polynomials share a common factor, the discriminant checks if a single polynomial has repeated factors.

**Lemma A.19** (see, e.g., [vzGG13, Corollaries 6.17 and 14.25])**.** *Let $\mathbb{F}$ be a field and let $f \in \mathbb{F}[x]$ be a univariate polynomial. Suppose $f$ is nonconstant. Then $\operatorname{disc}(f) \neq 0$ if and only if $f$ is squarefree.*

### A.4.2 Kaltofen's algorithm

We now present Kaltofen's algorithm to factor arithmetic circuits over finite fields, closely following the implementation appearing in the work of Kopparty, Saraf, and Shpilka [KSS15].

**Proposition A.20.** *Let $\mathbb{F} = \{\mathbb{F}_{q(n)}\}_{n \in \mathbb{N}}$ be a feasible family of finite fields, where the $n^{th}$ field has order $q(n)$. There is a deterministic, polynomial-time Turing machine that receives as input $(1^n, 1^d, 1^s)$ and outputs a randomized arithmetic network satisfying the following properties.*

1. *The network receives as input the description of an arithmetic circuit $C$ of size $s$ over $\mathbb{F}_{q(n)}$ that computes a polynomial $f \in \mathbb{F}_{q(n)}[x_1, \ldots, x_n]$ of degree at most $d$.*

2. *The network outputs the descriptions of arithmetic circuits $C_1, \dots, C_t$ together with natural numbers $e_1, \dots, e_t \in \mathbb{N}$ and $\beta_1, \dots, \beta_t \in \mathbb{N}$ encoded in binary. For each $i \in [t]$, let $g_i \in \mathbb{F}_{q(n)}[x_1, \dots, x_n]$ be the polynomial computed by the circuit $C_i$. The output of the network satisfies the following properties.*

   (a) *Each circuit $C_i$ is of size $\mathrm{poly}(n, s, d)$.*

   (b) *For each $i \in [t]$, the polynomial $g_i$ is either irreducible or a $p^{\beta_i}$-th power of an irreducible polynomial, where $p = \mathrm{char}(\mathbb{F}_{q(n)})$.*

   (c) *For each $i \in [t]$, the natural number $e_i$ is not divisible by $p$.*

   (d) *For $i \neq j$, the polynomials $g_i$ and $g_j$ are coprime.*

   (e) *The polynomial identity $f(\vec{x}) = \prod_{i=1}^{t} g_i(\vec{x})^{e_i}$ holds.*

3. *The network has size $\mathrm{poly}(n, s, d, \log(q))$, degree $n^{O(1)} d^{O(\log(q))}$, and error degree $\mathrm{poly}(d)$. If we are additionally provided with a representation of $\mathbb{F}_{q(n)}$ as an extension of $\mathbb{F}_{p(n)}$, where $p(n)$ is the characteristic of $\mathbb{F}_{q(n)}$, then the size of the network becomes $\mathrm{poly}(n, s, d, p)$ and the degree becomes $\mathrm{poly}(n, d, p)$.*

The remainder of this subsubsection consists of the proof of Proposition A.20. We make use of several subroutines for manipulating polynomials and arithmetic circuits, all of which can be implemented as uniform randomized arithmetic networks in a straightforward manner. The curious reader may find the details of these implementations in Section A.2. For simplicity, we abbreviate $q = q(n)$ throughout, so we work over the finite field $\mathbb{F}_q$ of order $q$.

We first check if $f(\vec{x})$ is a constant polynomial. We do this by evaluating $f$ on two random points using Proposition 3.24, declaring $f$ to be a constant polynomial exactly when these evaluations match. If we determine that $f$ is a constant polynomial, we output a single circuit $C_1$ that computes the corresponding constant single gate labeled by either of the evaluations of $f$ we have computed. We also output a multiplicity of $e_1 = 1$ for this factor of $f(\vec{x})$. Otherwise, we know that $f(\vec{x})$ is nonconstant and we proceed to factor $f$. This step can be implemented by a network of size $\mathrm{poly}(n, s, d)$, degree $\mathrm{poly}(n, d)$, and error degree $d$, and this network can be printed by invoking the corresponding Turing machine from Proposition 3.24.

**Preparations for Factorization** Before factoring $f(\vec{x})$, we need some preparatory steps to ensure $f$ satisfies several convenient properties. In particular, we will ensure that $f$ is not a $p^{\text{th}}$ power, that $f$ is monic in a known variable, and that $f$ is squarefree.

1. First, we ensure that $f$ is not a $p^{\text{th}}$ power. Suppose that $f$ is a $p^e$-th power, but not a $p^{e+1}$-th power. Then there is a variable $x_i$ such that $f$ depends on $x_i$ and some monomial of $f$ is divisible by $x_i$, but is not divisible by $x_i^{p^{e+1}}$. Write $f$ as a polynomial in $x_i$ as

$$f(\vec{x}) = \sum_{j=0}^{d} f_j(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) x_i^j.$$

   Because $f$ depends on $x_i$, at least one of the $f_j$ is nonzero. By the assumption that $f$ is a $p^e$-th power, we know that $f_j(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) \neq 0$ if and only if $j = p^e k$ for some $k \in \mathbb{N}$. Consider the polynomial

$$f^*(\vec{x}) = \sum_{j=0}^{d/p^e} f_{p^e j}(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) x_i^j.$$

   That is, whenever $x_i^{p^e j}$ appears in a monomial of $f$, we replace it with $x_i^j$. By our choice of the variable $x_i$, there is some $j$ with $p \nmid j$ such that $f_{p^e j}(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) \neq 0$, and because $f$ depends on the variable $x_i$, there exists such a $j$ that is nonzero. This implies that $x_i$ appears in $f^*$ with an exponent that is not divisible by $p$, so $f^*$ is not a $p^{\text{th}}$ power.

   We claim that the irreducible factors of $f$ are in one-to-one correspondence with the irreducible factors of $f^*$.

- In one direction, if $h(x_1, \ldots, x_n)$ is an irreducible factor of $f$, then because $f$ is a $p^e$-th power, it follows that $h(x_1, \ldots, x_n)^{p^e}$ divides $f(x_1, \ldots, x_n)$. We can write $h^{p^e}$ as

$$h(x_1, \ldots, x_n)^{p^e} = h^*(x_1, \ldots, x_{i-1}, x_i^{p^e}, x_{i+1}, \ldots, x_n)$$

  for some polynomial $h^* \in \mathbb{F}_q[\vec{x}]$. Because $h^*(x_1, \ldots, x_{i-1}, x_i^{p^e}, x_{i+1}, \ldots, x_n)$ divides the polynomial $f^*(x_1, \ldots, x_{i-1}, x_i^{p^e}, x_{i+1}, \ldots, x_n)$, it follows that $h^*(\vec{x})$ divides $f^*(\vec{x})$.
- In the other direction, if $h^*(\vec{x})$ is an irreducible factor of $f^*(\vec{x})$, then it is clear that the polynomial $h(x_1, \ldots, x_{i-1}, x_i^{p^e}, x_{i+1}, \ldots, x_n)$ divides

$$f^*(x_1, \ldots, x_{i-1}, x_i^{p^e}, x_{i+1}, \ldots, x_n) = f(\vec{x}).$$

In particular, given an irreducible factor of $f^*$, we can recover a $p^e$-th power of the corresponding irreducible factor of $f$. Thus, to factor $f$ (up to $p^e$-th powers), it suffices to instead factorize $f^*$. (For details, see Kopparty, Saraf, and Shpilka [KSS15, Section 3.1].)

We now argue that the description of an arithmetic circuit computing $f^*$ can be efficiently computed from the description of a circuit computing $f$. We first find a natural number $e \in \mathbb{N}$ such that $f$ is a $p^e$-th power but not a $p^{e+1}$-th power. For each variable $x_i$, we can write $f$ as a polynomial in $x_i$ whose coefficients are polynomials in the remaining variables, i.e.,

$$f(\vec{x}) = \sum_{j=0}^{d} f_{i,j}(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) x_i^j.$$

Using Lemma A.4, we can compute descriptions of circuits that compute the polynomials $f_{i,j}$ above. For each pair $(i, j) \in [n] \times \{0, 1, \ldots, d\}$, we evaluate $f_{i,j}$ on a random point using Proposition 3.24 to determine if the polynomial $f_{i,j}$ is zero or nonzero. We then compute, for each $i \in [n]$, the largest natural number $e_i \in \mathbb{N}$ such that for all $j$, we have $f_{i,j} \neq 0$ if and only if $p^{e_i} \mid j$. The desired exponent $e$ is given by $e \coloneqq \min_{i \in [n]} e_i$.

Let $i^\star \coloneqq \operatorname{argmin}_{i \in [n]} e_i$. By our choice of $i^\star$, there is a nonzero $j$ such that $p^{e+1} \nmid j$ and $f_{i^\star, j} \neq 0$. We can take $f^*(\vec{x})$ to be the polynomial

$$f^*(\vec{x}) = \sum_{j=0}^{d/p^e} f_{i^\star, p^e j}(x_1, \ldots, x_{i^\star - 1}, x_{i^\star + 1}, \ldots, x_n) x_{i^\star}^j.$$

Our choice of $i^\star$ ensures that $f^*$ is not a $p^{\text{th}}$ power. We already have computed descriptions of circuits for the polynomials $\{f_{i^\star, p^e j} : j \in \{0, 1, \ldots, d/p^e\}\}$, so we can efficiently compute a description of a circuit that computes $f^*(\vec{x})$. As described above, we can recover the factorization of $f$ from the factorization of $f^*$, so we will proceed to factor $f^*$. We also record the integer $e$, which will correspond to the output $\beta_j$ for any factor $g_j$ of $f$ we compute by factoring $f^*$.

We now analyze the cost of this step. We invoke Lemma A.4 in parallel $n$ times, and each invocation is implemented by a network of size $\operatorname{poly}(n, s, d)$ and degree $O(1)$, so computing the coefficients $f_{i,j}$ can be done by a network of size $\operatorname{poly}(n, s, d)$ and degree $O(1)$. Evaluating the coefficients $f_{i,j}$ on random points is done in parallel using Proposition 3.24, which is implemented by a network of size $\operatorname{poly}(n, s, d)$ and degree $\operatorname{poly}(n, d)$. Computing $e$ and $i^\star$ can be done by a boolean subcircuit of size $\operatorname{poly}(n, d)$. Once $i^\star$ has been computed, a description of a circuit for $f^*$ can be computed in a straightforward manner using a network of size $\operatorname{poly}(n, s, d)$ and degree $O(1)$. In total, this step can be implemented by a network of size $\operatorname{poly}(n, s, d)$ and degree $\operatorname{poly}(n, d)$.

The only source of error in this step is when a polynomial $f_{i,j}$ is nonzero, but evaluates to zero on a random point. Since each $f_{i,j}$ has degree at most $d$, when we sample the coordinates of the evaluation point at random from a set $S \subseteq \mathbb{F}_q$, we err with probability at most $d/|S|$. Although there are $n(d+1)$ such polynomials $f_{i,j}$, we can avoid paying a union bound in the error degree. Suppose $i^\star \in [n]$ is the index of a variable $x_{i^\star}$ such that $f$ depends on $x_{i^\star}$ and some monomial of $f$ is divisible by $x_{i^\star}$, but is

not divisible by $x_{i^\star}^{p^{e+1}}$. Then there is some $j^\star \in [d]$ such that $p \nmid j$ and $f_{i^\star,p^e j^\star} \neq 0$. As long as the evaluation of $f_{i^\star,j^\star}$ is nonzero, the algorithm will correctly modify $f$ into a non-$p^{\text{th}}$-power $f^*$, so the error degree of this step is bounded by $d$.

From here on, we use $f(\vec{x})$ to denote the polynomial $f^*(\vec{x})$ obtained above for the sake of notational ease.

2. To make $f$ monic, we introduce a new variable $y$ and perform a random linear change of variables. Let $\vec{\alpha} \in \mathbb{F}_q^n$ be chosen at random and define

$$\hat{f}(\vec{x}, y) \coloneqq f(\vec{x} + y \cdot \vec{\alpha}) = f(x_1 + y\alpha_1, \ldots, x_n + y\alpha_n).$$

Let $f_d(\vec{x})$ be the top-degree homogeneous component of $f(\vec{x})$. A standard calculation shows that if $f_d(\vec{\alpha}) \neq 0$, then the polynomial $\hat{f}(\vec{x}, y)$ is monic in the fresh variable $y$.

Given $\vec{\alpha}$, we can compute a description of a circuit computing $\hat{f}$ from the description of the circuit computing $f$. This is done by adding new gates computing the new input $\vec{x} + y\vec{\alpha}$, which can be implemented by an arithmetic network of size $\text{poly}(n, s)$ and degree $O(1)$.

Because the irreducible factors of $f$ and $\hat{f}$ are in one-to-one correspondence, in order to factor $f$, it suffices to factor $\hat{f}$. After factoring $\hat{f}$, we can recover the factors of $f$ by applying the change of variables $(\vec{x}, y) \mapsto (\vec{x} - y\vec{\alpha}, y)$, which can be performed by an arithmetic network of size $\text{poly}(n, s)$ and degree $O(1)$.

This step of the algorithm errs if the point $\vec{\alpha}$ is chosen so that $f_d(\vec{\alpha}) = 0$. Because $f_d(\vec{x})$ is a polynomial of degree $d$, if we choose the coordinates of $\vec{\alpha}$ at random from a subset $S \subseteq \mathbb{F}_q$, then Lemma 3.1 implies that $f_d(\vec{\alpha}) = 0$ with probability at most $d/|S|$. This implies that the error degree of this step is bounded by $d$.

In the following steps, we write $f(\vec{x}, y)$ for $\hat{f}(\vec{x}, y)$ to avoid cluttered notation.

3. Finally, we need to ensure that $f(\vec{x}, y)$ is squarefree. It is a standard fact that the squarefree part of $f$ can be written as

$$\frac{f(\vec{x}, y)}{\gcd\left(f, \frac{\partial f}{\partial y}\right)},$$

assuming that $\frac{\partial f}{\partial y} \neq 0$. Because $f$ is nonzero, not a $p^{\text{th}}$ power, and depends on the variable $y$, it follows that $\frac{\partial f}{\partial y} \neq 0$. The description of a circuit for $\frac{\partial f}{\partial y}$ can be computed from the description of a circuit for $f$ by using Lemma A.5. A description of a circuit computing $\gcd(f, \frac{\partial f}{\partial y})$ can then be computed using Lemma A.10.

The applications of Lemma A.5 and Lemma A.10 can be implemented by arithmetic networks of size $\text{poly}(n, s, d)$ and degree $\text{poly}(d)$. The corresponding arithmetic networks use no randomness, so they have error degree 0.

In total, the preparatory steps above can be implemented by an arithmetic network of size $\text{poly}(n, s, d)$. Using Proposition 3.21, we can bound the degree and error degree of the resulting network by $\text{poly}(d)$ and $O(d)$, respectively. The description of the preparatory steps is a uniform algorithm, so the resulting arithmetic network can indeed be printed by a polynomial-time Turing machine.

**Reduction to Bivariate Factoring** We are now in the situation where the polynomial $f(\vec{x}, y)$ to be factored is not a $p^{\text{th}}$ power, is monic in $y$, and is squarefree. We now reduce the task of factoring the multivariate polynomial $f(\vec{x}, y)$ in $\mathbb{F}_q[\vec{x}, y]$ to factoring a bivariate polynomial, albeit over the larger field $\mathbb{F}_q(x_1, \ldots, x_n)$.

The discriminant will help us find a good reduction to bivariate factorization. We know that the polynomial $f(\vec{x}, y)$ to be factored is squarefree, and we would like to find a substitution $\vec{x} \mapsto \vec{\alpha}$ so that the resulting polynomial $f(\vec{\alpha}, y)$ remains squarefree. Because $f(\vec{x}, y)$ is squarefree and is nonconstant as a polynomial in $y$, viewing $f$ as an element of $\mathbb{F}_q(\vec{x})[y]$, Lemma A.19 implies that the discriminant of $f$ is a nonzero polynomial

in $\mathbb{F}_q[\vec{x}]$. To guarantee that $f(\vec{\alpha}, y)$ is squarefree, it suffices to find a point $\vec{\alpha}$ where the discriminant of $f$ evaluates to a nonzero value, as this implies (again, by Lemma A.19) that the polynomial $f(\vec{\alpha}, y)$ is squarefree.

Let $\Delta(\vec{x}) \coloneqq \mathrm{disc}_y(f(\vec{x}, y))$ be the discriminant of $f$ with respect to $y$. We know that $\Delta(\vec{x})$ is a nonzero polynomial and that $\Delta(\vec{x})$ has degree bounded by $2d^2$. If we sample a point $\vec{\alpha} \in \mathbb{F}_q^n$ whose coordinates are chosen uniformly at random from a set $S \subseteq \mathbb{F}_q$, then Lemma 3.1 the probability that $\Delta(\vec{\alpha}) = 0$ is at most $2d^2/|S|$, so we can find such a point $\vec{\alpha}$ with error degree $2d^2$. Applying the change of variables $\vec{x} \mapsto \vec{x} - \vec{\alpha}$, we may assume that $f$ has nonzero discriminant at the origin, which will be convenient for us later.

Consider the polynomial

$$\overline{f}(\vec{x}, y, z) \coloneqq f(x_1 z, \ldots, x_n z, y) \in \mathbb{F}(x_1, \ldots, x_n)[y, z].$$

One can show that the irreducible factors of $\overline{f}$ are in one-to-one correspondence with those of $f$ (see [KSS15, Section 3.4] for details), so to factor $f(\vec{x}, y)$, it suffices to factor $\overline{f}(\vec{x}, y, z)$ as a polynomial in the ring $\mathbb{F}(\vec{x})[y, z]$. Given a factor $\overline{g}(\vec{x}, y, z)$ of $\overline{f}$, the corresponding factor of $f(\vec{x}, y)$ is simply $\overline{g}(\vec{x}, y, 1)$. We can compute a description of $\overline{f}$ from a description of $f$ using a network of size $\mathrm{poly}(n, s, d)$ and degree $O(1)$.

In total, the reduction to bivariate factoring can be implemented by an arithmetic network of size $\mathrm{poly}(n, s, d)$, degree $O(1)$, and error degree $2d^2$. It is clear that this network can be printed by a polynomial-time Turing machine.

**Univariate factorization** The first step in Kaltofen's algorithm is to factor the univariate polynomial $\overline{f}(\vec{0}, y, 0)$ in $\mathbb{F}_q[y]$. Because we know that $f(\vec{x}, y)$ has nonzero discriminant at the origin and $f(\vec{x}, y)$ is monic in $y$, it follows that the discriminant of $f(\vec{0}, y) = \overline{f}(\vec{0}, y, 0)$ is nonzero, which implies that $\overline{f}(\vec{0}, y, 0)$ is squarefree. This allows us to factor $\overline{f}(\vec{0}, y, 0)$ in $\mathbb{F}_q[y]$ by applying the network of Theorem A.14, which has size $\mathrm{poly}(d, \log q)$, degree $d^{O(\log(q))}$, and error degree 1. If we are also given a representation of $\mathbb{F}_{q(n)}$ as an extension of $\mathbb{F}_{p(n)}$, then Theorem A.14 instead provides us with a network of size $\mathrm{poly}(d, p)$ and degree $\mathrm{poly}(d, p)$.

This is the only part of the arithmetic network whose size depends on $\log(q)$ or $p$. Because a univariate polynomial of degree $d$ can be computed by an arithmetic circuit of size $O(d)$, the factors produced in this step can be represented by circuits of size $O(d)$. The remaining part of the network that implements Proposition A.20 has size $\mathrm{poly}(n, s, d)$, so it follows that the circuits $C_1, \ldots, C_t$ produced for the factors of $f$ likewise have size bounded by $\mathrm{poly}(n, s, d)$ as claimed.

**Hensel lifting** We now perform Hensel lifting to lift the factorization of $\overline{f}(\vec{0}, y, 0)$ to an approximate factorization of $\overline{f}(\vec{x}, y, z)$ in $\mathbb{F}_q(\vec{x})[y, z]$. Let $g_0(y) \in \mathbb{F}_q[y]$ be an irreducible factor of $\overline{f}(\vec{0}, y, 0)$, and let $h_0(y) \in \mathbb{F}_q[y]$ be a polynomial such that

$$\overline{f}(\vec{0}, y, 0) = g_0(y) \cdot h_0(y).$$

We know that $\overline{f}(\vec{0}, y, 0)$ is squarefree, so $\gcd(g_0(y), h_0(y)) = 1$. Because $g_0$ and $h_0$ are coprime, there are polynomials $a_0, b_0 \in \mathbb{F}_q[y]$ of small degree such that $a_0 g_0 + b_0 h_0 = 1$. Moreover, from the coefficients of $g_0$ and $h_0$, we can compute the coefficients of $a_0$ and $b_0$ using the network of Lemma A.12, which has size and degree $\mathrm{poly}(d)$.

We now use Hensel lifting to lift the factorization $\overline{f}(\vec{0}, y, 0) = g_0(y) \cdot h_0(y)$ to an approximate factorization of $\overline{f}(\vec{x}, y, z)$. We apply the network of Lemma A.13 with $k = 2\log_2(d) + 2$, providing as input the description of the circuit that computes $\overline{f}(\vec{x}, y, z)$ as well as the coefficients of $g_0(y)$, $h_0(y)$, $a_0(y)$, and $b_0(y)$. This network outputs the description of an arithmetic circuit $D$ of size $\mathrm{poly}(n, s, d)$ and degree $\mathrm{poly}(d)$. The outputs of $D$ are the coefficients of two polynomials $g_k(y, z), h_k(y, z) \in \mathbb{F}[\vec{x}][y, z]$ such that

$$\begin{aligned} \overline{f}(\vec{x}, y, z) &= g_k(y, z) \cdot h_k(y, z) \bmod \langle z^{4d^2} \rangle \\ g_k(y, 0) &= g_0(y), \end{aligned}$$

and $g_k(y, z)$ is monic with respect to $y$. The network that computes $D$ has size $\mathrm{poly}(n, s, d)$ and degree $O(1)$.

**Factor reconstruction** From Hensel lifting, we have computed an approximate factorization

$$\overline{f}(\vec{x}, y, z) = g_k(y, z) \cdot h_k(y, z) \bmod \langle z^{4d^2} \rangle.$$

Our goal is to use this to recover a genuine factor of $\overline{f}(\vec{x}, y, z)$. At this point, our presentation of Kaltofen's algorithm deviates slightly from prior work. Normally, we would use this approximate factorization to set up a system of linear equations where any nontrivial solution of the linear system will provide us with a nontrivial factor $\overline{g}(\vec{x}, y, z)$ of $\overline{f}(\vec{x}, y, z)$. We would then recursively continue factoring $\overline{g}$ and $\overline{f}/\overline{g}$. In the worst case, this involves repeating the entire factoring algorithm $d-1$ times in sequence, which (using Proposition 3.21) would result in an arithmetic network of degree bounded by $d^{O(d \log(d) + d \log(q))}$. By slightly modifying this reconstruction step, we can avoid re-running the algorithm $d-1$ times in sequence, saving a factor of $d$ in the exponent of the resulting degree bound.

Because $g_k$ has individual degree at most $\max(d, 2^k) = 4d^2$, we can write $g_k(y, z)$ as

$$g_k(y, z) = \sum_{i,j=1}^{4d^2} g_{i,j}(\vec{x}) y^i z^j.$$

Consider the system of linear equations over $\mathbb{F}_q(\vec{x})$ in the variables $a_{i,j}$ and $b_{i,j}$ given by

$$\sum_{i<d, j \le d} a_{i,j} y^i z^j = g_k(y, z) \sum_{i \le 4d^2, j \le 4d^2} b_{i,j} y^i z^j \bmod \langle z^{4d^2} \rangle.$$

It is a standard fact (see, e.g., [KSS15, Claims 3.8 and 3.10]) that this is system has a nontrivial solution if and only if $\overline{f}(\vec{x}, y, z)$ is reducible, and that if $(a(y, z), b(y, z))$ is a nontrivial solution, then $a(y, z)$ and $\overline{f}(\vec{x}, y, z)$ have nontrivial GCD in $\mathbb{F}_q(\vec{x})[y, z]$. We will need more control over the factor that we obtain from this GCD—in particular, we want to guarantee that we obtain an irreducible factor of $\overline{f}(\vec{x}, y, z)$.

Suppose $\overline{f}(\vec{x}, y, z)$ is reducible $\mathbb{F}_q(\vec{x})[y, z]$. Let $r(\vec{x}, y, z)$ be an irreducible factor of $\overline{f}(\vec{x}, y, z)$ such that $g_0(y)$ divides $r(\vec{0}, y, 0)$. Because $\overline{f}(\vec{0}, y, 0)$ is squarefree, the polynomial $g_0(y)$ is a factor of $\overline{f}(\vec{0}, y, 0)$ of multiplicity 1, so there is in fact a unique irreducible factor $r(\vec{x}, y, z)$ of $\overline{f}(\vec{x}, y, z)$ such that $g_0(y)$ divides $r(\vec{0}, y, 0)$. We will argue that for every nontrivial solution $(a(y, z), b(y, z))$ of the above system of linear equations, we have that $r(\vec{x}, y, z)$ divides $a(y, z)$.

We first show that there is a solution to the linear system above whose left-hand side is precisely the irreducible factor $r(\vec{x}, y, z)$. Let $s_0(y)$ be a polynomial such that

$$r(\vec{0}, y, 0) = g_0(y) \cdot s_0(y).$$

Although we do not have $r(\vec{0}, y, 0)$ nor the factorization above, for the sake of analysis, we will consider what happens when we apply Hensel lifting to the above factorization. If we apply Hensel lifting for $k = 2 \log_2(d) + 2$ steps, we obtain an approximate factorization

$$r(\vec{x}, y, z) = \tilde{g}_k(y, z) \cdot s_k(y, z) \bmod \langle z^{4d^2} \rangle.$$

This, together with degree bounds on $\tilde{g}_k$ and $s_k$ implied by Lemma A.13, implies that $(r, s_k)$ solves the linear system above if we replace $g_k$ with $\tilde{g}_k$, i.e., that $r = \tilde{g}_k s_k \bmod \langle z^{4d^2} \rangle$. As we will see, the pair $(r, s_k)$ also solves the original linear system of interest. Writing $\overline{f}(\vec{x}, y, z) = r(\vec{x}, y, z) \cdot t(\vec{x}, y, z)$, this yields an approximate factorization of $\overline{f}(\vec{x}, y, z)$ as

$$\overline{f}(\vec{x}, y, z) = \tilde{g}_k(y, z) \cdot s_k(y, z) \cdot t(\vec{x}, y, z) \bmod \langle z^{4d^2} \rangle.$$

Because $\tilde{g}_k(y, z)$ is monic with respect to $y$ and satisfies $\tilde{g}_k(y, 0) = g_0(y)$, uniqueness of Hensel lifting ([KSS15, Lemma 3.4]) implies that

$$\tilde{g}_k(y, z) = g_k(y, z) \bmod \langle z^{4d^2} \rangle.$$

This lets us write

$$r(\vec{x}, y, z) = g_k(y, z) \cdot s_k(y, z) \bmod \langle z^{4d^2} \rangle.$$

Because $\overline{f}$ is monic with respect to $y$ and $r$ is a proper factor of $\overline{f}$, it must be the case that $r$ has individual degree less than $d$ with respect to $y$. Lemma A.13 implies that $s_k(y,z)$ is a polynomial of degree at most $4d^2$ with respect to $y$ and $z$. Thus $(r, s_k)$ is a solution to the linear system $a = g_k b \bmod \langle z^{4d^2} \rangle$ as claimed.

Of course, there may be other solutions to this linear system whose left-hand side is not the polynomial $r(\vec{x}, y, z)$. Let $(a(y,z), b(y,z))$ be one such solution. We will show that $r(\vec{x}, y, z)$ divides $a(y,z)$. Let

$$\rho(\vec{x}, z) \coloneqq \operatorname{res}_y(r(\vec{x}, y, z), a(y,z))$$

be the resultant of $r(\vec{x}, y, z)$ and $a(y,z)$ with respect to $y$. Importantly, the resultant $\rho(\vec{x}, z)$ does not depend on the variable $y$. From Lemma A.17, there are polynomials $u, v \in \mathbb{F}_q(\vec{x})[y,z]$ such that

$$u(\vec{x}, y, z) r(\vec{x}, y, z) + v(\vec{x}, y, z) a(y,z) = \rho(\vec{x}, z).$$

This lets us write

$$\begin{aligned}
\rho(\vec{x}, z) &= u(\vec{x}, y, z) r(\vec{x}, y, z) + v(\vec{x}, y, z) a(y,z) \bmod \langle z^{4d^2} \rangle \\
&= u(\vec{x}, y, z) g_k(y,z) s_k(y,z) + v(\vec{x}, y, z) g_k(y,z) b(y,z) \bmod \langle z^{4d^2} \rangle \\
&= g_k(y,z) \left( u(\vec{x}, y, z) s_k(y,z) + v(\vec{x}, y, z) b(y,z) \right) \bmod \langle z^{4d^2} \rangle.
\end{aligned}$$

Recall that the polynomial $g_k(y,z)$ is monic with respect to $y$. This implies that any multiple of $g_k(y,z)$ that is nonzero modulo $\langle z^{4d^2} \rangle$ must depend on the variable $y$. The left-hand side $\rho(\vec{x}, z)$ above does not depend on $y$, so it must be the case that

$$\rho(\vec{x}, z) = 0 \bmod \langle z^{4d^2} \rangle.$$

The resultant $\rho(\vec{x}, z)$ is the determinant of a matrix of size $2d \times 2d$. Each entry of this matrix is a polynomial of degree at most $d$, so we have $\deg(\rho(\vec{x}, z)) \leq 2d^2 < 4d^2$. Because $\rho(\vec{x}, z)$ has degree less than $4d^2$, the fact that $\rho(\vec{x}, z) = 0 \bmod \langle z^{4d^2} \rangle$ implies that $\rho(\vec{x}, z) = 0$ identically. This means that $r(\vec{x}, y, z)$ and $a(y,z)$ have a nontrivial common factor over $\mathbb{F}_q(\vec{x}, z)[y]$. Because $r(\vec{x}, y, z)$ is an irreducible factor of $\overline{f}(\vec{x}, y, z)$ and $\overline{f}$ is monic in $y$, it must be the case that $r$ is also monic in $y$. Gauss's Lemma (Lemma 3.3) then implies that $r$ is also irreducible in $\mathbb{F}_q(\vec{x}, z)[y]$. Since $r$ is irreducible in $\mathbb{F}_q(\vec{x}, z)[y]$ and $r$ and $a$ have a nontrivial common factor in $\mathbb{F}_q(\vec{x}, z)[y]$, it must be the case that $r$ divides $a$, as claimed.

We now describe how to find a nontrivial factor of $\overline{f}(\vec{x}, y, z)$. For every choice of $d_y, d_z \in [d]$, we solve the linear system

$$\sum_{i < d_y, j \leq d_z} a_{i,j} y^i z^j = g_k(y,z) \sum_{i \leq 4d^2, j \leq 4d^2} b_{i,j} y^i z^j \bmod \langle z^{4d^2} \rangle$$

using the arithmetic network of Lemma A.8, simulating the PIT gates by instead evaluating an input to a PIT gate on a randomly-chosen point. The network of Lemma A.8 has size $\operatorname{poly}(n, d, s)$ and degree $O(1)$, and we invoke this network $O(d^2)$ times in parallel, so we can solve all of these systems with a network of size $\operatorname{poly}(n, d, s)$ and degree $O(1)$. In our setting, each PIT gate in the network of Lemma A.8 tests a polynomial of degree at most $\operatorname{poly}(d)$, and we perform at most $\operatorname{poly}(d)$ such tests, so the error degree of the resulting network is bounded by $\operatorname{poly}(d)$.

If there is no nontrivial solution to any of these linear systems, then the polynomial $\overline{f}(\vec{x}, y, z)$ is irreducible ([KSS15, Claim 3.8]). In this case, we report that $f(\vec{x})$ is irreducible and the algorithm terminates. On the other hand, if at least one of these systems has a nontrivial solution, let $d_y, d_z \in [d]$ be chosen such that the corresponding linear system has a solution and the sum $d_y + d_z$ is minimized, and let $(a(y,z), b(y,z))$ be a nontrivial solution to the corresponding linear system. (In other words, we find a solution $a(y,z)$ of minimal degree.) Since $r(\vec{x}, y, z)$ is one such solution of degree at most $D \coloneqq \deg(r)$, we know that we have a solution $a(y,z)$ of degree at most $D$. By the analysis above, we know that $r$ divides $a$, so it must be the case that $r(\vec{x}, y, z) = a(y,z)$, i.e., that $a(y,z)$ is an irreducible factor of $\overline{f}(\vec{x}, y, z)$. We report that $a(y,z)$ is an irreducible factor of $\overline{f}(\vec{x}, y, z)$, which in turn yields a nontrivial factor of the original polynomial $f(\vec{x})$. This step can be implemented by an arithmetic network of size $\operatorname{poly}(n, s, d)$, degree $\operatorname{poly}(n, d)$, and error degree $\operatorname{poly}(d)$.

**Complete factorization** To obtain a complete factorization of $\overline{f}(\vec{x}, y, z)$ into irreducibles, we run the Hensel lifting and factor reconstruction steps in parallel on all the irreducible factors of $\overline{f}(\vec{0}, y, 0)$. This produces a collection of arithmetic circuits that compute the irreducible factors of $\overline{f}(\vec{x}, y, z)$. There may be multiple circuits that compute the same irreducible factor of $\overline{f}(\vec{x}, y, z)$ in this list. To prune these duplicates, we evaluate all circuits on a randomly-chosen point, declaring two circuits to be duplicates if they have the same evaluation at this point. For each unique irreducible factor, we output the lexicographically-first circuit that computes it. These evaluations can be carried out by the network of Proposition 3.24, which has size $\text{poly}(n, s, d)$ and degree $\text{poly}(n, d)$. The polynomials we are testing have degree at most $d$, so by Lemma 3.1, the error degree of this procedure is bounded by $d$. In total, this step results in a network of size $\text{poly}(n, s, d)$ and, by Proposition 3.19, degree $\text{poly}(n, d)$ and error degree $\text{poly}(d)$.

**Computing factor multiplicities** We have now computed descriptions of a collection of arithmetic circuits $C_1, \ldots, C_t$, where each $C_i$ computes a polynomial $g_i$ that is either irreducible or a $p^{\beta_i}$-th power of an irreducible polynomial. The last step in the proof of Proposition A.20 is to determine the multiplicities of each $g_i(\vec{x})$ as a factor of $f(\vec{x})$. For each $i \in [t]$ and $j \in [d]$, we test if $g_i(\vec{x})^j$ divides $f(\vec{x})$ using the network of Lemma A.9. We set the multiplicity of $g_i$ as a factor of $f$ to be the largest $j$ for which $g_i(\vec{x})^j$ divides $f(\vec{x})$. The network of Lemma A.9 is invoked $O(d^2)$ times, each of which costs size $\text{poly}(n, s, d)$ and degree $O(1)$, so the total cost of this step is likewise $\text{poly}(n, s, d)$ size. Because we do not have access to PIT gates, we have to simulate the PIT gates in the network of Lemma A.9 by evaluating circuits on random points using Proposition 3.24. Every polynomial that is tested in the network of Lemma A.9 has degree bounded by $2d$, so this simulation can be implemented in size $\text{poly}(n, s, d)$, degree $\text{poly}(n, d)$, and error degree $O(d)$.

This completes the proof of Proposition A.20.

### A.4.3 An application to root-finding

As a direct application of Proposition A.20, we can solve the root-finding problem for arithmetic circuits over $\mathbb{F}_q$ efficiently using a randomized arithmetic network. The root-finding problem is the following: given a polynomial $f(\vec{x}, y)$, find all polynomials $g(\vec{x})$ such that $f(\vec{x}, g(\vec{x})) = 0$. This is directly related to factorization, as Lemma 3.3 implies that such roots are precisely the factors of $f(\vec{x}, y)$ of the form $y - g(\vec{x})$. Since Proposition A.20 allows us to factor an arithmetic circuit, we have a solution for the root-finding problem as an immediate corollary. We state this as a separate corollary, as this is result we ultimately use in our reconstruction algorithm in Section 6.2.

**Corollary A.21.** *Let $\mathbb{F} = \{\mathbb{F}_{q(n)}\}_{n \in \mathbb{N}}$ be a feasible family of finite fields, where the $n^{th}$ field has order $q(n)$. There is a deterministic, polynomial-time Turing machine that receives as input $(1^n, 1^d, 1^s)$ and outputs a randomized arithmetic network satisfying the following properties.*

1. *The network receives as input the description of an arithmetic circuit $C$ of size $s$ over $\mathbb{F}_{q(n)}$ that computes a polynomial $f(\vec{x}, y) \in \mathbb{F}_q[x_1, \ldots, x_n]$ of degree at most $d$.*
2. *The network outputs the descriptions of arithmetic circuits $C_1, \ldots, C_t$ for some $t \leq d$, together with natural numbers $\beta_1, \ldots, \beta_t \in \mathbb{N}$ encoded in binary. For each $i \in [t]$, let $g_i \in \mathbb{F}_{q(n)}[x_1, \ldots, x_n]$ be the polynomial computed by the circuit $C_i$. The output of the network satisfies the following properties.*
   (a) *Each circuit $C_i$ is of size $\text{poly}(n, s, d)$.*
   (b) *For each $i \in [t]$, the polynomial $g_i$ has degree at most $d$.*
   (c) *For every polynomial $g(\vec{x})$ such that $f(\vec{x}, g(\vec{x})) = 0$, there is an index $i^\star \in [t]$ such that $g_{i^\star}(\vec{x}) = g(\vec{x})^{p^{\beta_{i^\star}}}$.*
3. *The network has size $\text{poly}(n, s, d, \log(q))$, degree $n^{O(1)} d^{O(\log(q))}$, and error degree $\text{poly}(d)$. If we are additionally provided with a representation of $\mathbb{F}_{q(n)}$ as an extension of $\mathbb{F}_{p(n)}$, where $p(n)$ is the characteristic of $\mathbb{F}_{q(n)}$, then the size of the network becomes $\text{poly}(n, s, d, p)$ and the degree becomes $\text{poly}(n, d, p)$.*

**Proof.** By Corollary 3.4, if $C(\vec{z}, g(\vec{z})) = 0$, then the polynomial $y - g(\vec{z})$ is an irreducible factor of $C(\vec{z}, y)$. Using the arithmetic network of Proposition A.20, we can compute the standard descriptions of a set of

circuits $C_1, \ldots, C_t$ and natural numbers $e_1, \ldots, e_t \in \mathbb{N}$ and $\beta_1, \ldots, \beta_t \in \mathbb{N}$ such that the circuit $C_i$ computes a $p^{\beta_i}$-th power of an irreducible factor of $C(\vec{z}, y)$. In particular, one of the circuits $C_i$ must compute $(y - g(\vec{z}))^{p^{\beta_i}} = y^{p^{\beta_i}} - g(\vec{z})^{p^{\beta_i}}$. Subtracting $y^{p^{\beta_i}}$ and negating the output yields a circuit that computes $g(\vec{z})^{p^{\beta_i}}$. Since we do not know which circuit $C_i$ computes the desired irreducible factor, we simultaneously modify all circuits output by Kaltofen's algorithm and output these modified circuits. □

## A.5 Uniformity-preserving depth reduction

In this final section, we show that the depth reduction of Valiant, Skyum, Berkowitz, and Rackoff [VSBR83] can be carried out in a uniformity-preserving way. Specifically, we show that if $\{C_n\}_{n\in\mathbb{N}}$ is a $\log^c$-uniform family of arithmetic circuits that compute polynomials of low degree, then there is a family of low-depth arithmetic circuits $\{C'_n\}_{n\in\mathbb{N}}$ that computes the same family of polynomials and is $\log^{c'}$-uniform for some constant $c' = c + O(1)$. To do this, we first need a lemma that says circuits can be homogenized in a uniformity-preserving manner.

**Lemma A.22** (uniformity-preserving homogenization)**.** *Let $\{C_n\}$ be a $\log^c$-uniform arithmetic circuit family of size $T(n)$ that computes a family of polynomials $\{f_n\}$. For any polynomial-time computable $d : \mathbb{N} \to \mathbb{N}$, there is a $\log^{c+O(1)}$-uniform family of homogeneous multi-output arithmetic circuits $\{C'_n\}$ of size $T'(n) = O(T(n)d(n)^2)$ such that $C'_n$ computes the homogeneous components of the polynomial $f_n$ up to degree $d(n)$.*

**Proof.** We follow the standard gate-simulation argument that arithmetic circuits can be homogenized efficiently (as seen in the proof of Lemma A.6), verifying that this argument can be carried out in a uniformity-preserving manner.

Each gate $u$ in $C_n$ will correspond to a collection of $d+1$ gates $(u, 0), \ldots, (u, d)$ in $C'_n$. The intention is that the gate $(u, a)$ in $C'_n$ computes the degree-$a$ homogeneous component of the polynomial computed by the gate $u$ in $C_n$. The wiring of the gates in $C'_n$ depends on the type of the gate $u$.

- If $u$ is an input gate labeled by a field constant $\alpha \in \mathbb{F}$, then we set $(u, 0)$ to be an input gate labeled by $\alpha$. For $i \in \{1, 2, \ldots, d\}$, we set $(u, i)$ to be an input gate labeled by zero.
- If $u$ is an input gate labeled by the variable $x_i$, we set $(u, 1)$ to be an input gate labeled by $x_i$. For $i \in \{0, 2, 3, \ldots, d\}$, we set $(u, i)$ to be an input gate labeled by zero.
- If $u$ is an addition gate in $C_n$ with children $v$ and $w$, then for each $i \in \{0, 1, \ldots, d\}$, we set $(u, i)$ to be an addition gate with children $(v, i)$ and $(w, i)$.
- If $u$ is a multiplication gate in $C_n$ with children $v$ and $w$, then for each $a \in \{0, 1, \ldots, d\}$, we set

$$(u, a) = \sum_{b=0}^{a} (v, b) \times (w, a - b),$$

  adding extra gates as necessary to implement the sum of products above as an alternating circuit of fan-in two.

We set the output gates of $C'_n$ to be $(u, 0), (u, 1), \ldots, (u, d)$, where $u$ is the output gate of $C_n$ and $d$ is the degree of the polynomial computed by $C_n$.

The following induction argument shows that the gate $(u, a)$ computes precisely the degree-$a$ homogeneous component of the polynomial computed by the gate $u$.

- If $u$ is an input gate, then it immediately follows from the definition of the gate $(u, a)$ that $(u, a)$ computes the degree-$a$ homogeneous component of $u$.
- If $u$ an addition gate with children $v$ and $w$, then the degree-$a$ component of $u$ is the sum of the degree-$a$ components of $v$ and $w$. By induction, the gates $(v, a)$ and $(w, a)$ compute the degree-$a$ components of $v$ and $w$, respectively, so $(u, a)$ correctly computes the degree-$a$ component of $u$.

- If $u$ is instead a product gate with children $v$ and $w$, then the degree-$a$ component of $u$ is given by $\sum_{b=0}^{a} v_b w_{a-b}$, where $v_b$ is the degree-$b$ component of $v$ and likewise $w_{a-b}$ is the degree-$(a-b)$ component of $w$. By induction, the gates $(v, b)$ and $(w, a-b)$ correctly compute these polynomials, so $(u, a)$ correctly computes the degree-$a$ component of $u$.

Because each gate of the circuit $C'_n$ computes a homogeneous polynomial, the circuit $C'_n$ is homogeneous. To bound the size of $C'_n$, observe that $C'_n$ contains $d(n)+1$ copies of every gate in $C_n$, and that each copy $(u, a)$ of a gate from $C_n$ uses at most $O(d(n))$ additional gates in its implementation. This results in the claimed bound of $O(T(n)d(n)^2)$ on the number of gates of $C'_n$.

It remains to bound the uniformity of the circuit family $\{C'_n\}$. To do this, we slightly modify the construction of $C'_n$ described above. Whenever $u$ is a multiplication gate in $C_n$ with children $v$ and $w$, we implement each convolution $(u, a) = \sum_{b=0}^{a}(v, b) \times (w, a-b)$ as a balanced full binary tree of depth $\log d$ by implementing a subcircuit that computes $(u, a)$ as

$$(u, a) = \sum_{b=0}^{d} (v, b) \times (w, a-b),$$

with the convention that $(w, a-b)$ is labeled by the constant zero whenever $a < b$. If $u$ is a product gate in $C_n$, we label the gates in the subcircuit computing $(u, a)$ as $(u, a, x)$, where $x \in \{0,1\}^{\leq 2\log d}$ is a bitstring of length at most $2\log d$. The intended meaning is that $(u, a, x)$ is either the sum or product of $(u, a, x0)$ and $(u, a, x1)$, depending on the parity of the length of $x$. Building $C'_n$ in this manner allows us to more efficiently decide the gate adjacency relation. We can quickly decide if the gates in the $i$-th layer compute (1) $(u, a)$ where $u$ is an addition gate, (2) $(u, a)$ where $u$ is a product gate, or (3) a subgate used in the computation of $(u, a)$.

We now describe a $\mathsf{P}$-uniform family of formulas $\{\Phi'_n\}$ that decides the gate adjacency relation in the new circuit family $C'_n$. If the original circuit $C_n$ has depth $\Delta(n)$, then $C'_n$ has depth $\Delta'(n) \coloneqq O(\Delta(n)\log d)$. Given $i \in [\Delta'(n)]$ and gate names $\hat{u}, \hat{v}, \hat{w} \in [T'(n)]$, we first inspect the parity of $i$ to determine the type of gates $\hat{u}, \hat{v}$, and $\hat{w}$ should be. If any of $\hat{u}$, $\hat{v}$, or $\hat{w}$ are not of the correct type (which we can inspect from the binary encoding of the gate names), then $\Phi'_n(i, \hat{u}, \hat{w}, \hat{v}) = 0$. Otherwise, we branch based on the type of gate $\hat{u}$.

1. Suppose $\hat{u} = (u, a)$, $\hat{v} = (v, b)$, and $\hat{w} = (w, c)$, where $u$ is an addition gate at layer $i'$ of the circuit $C_n$. In this case, we set $\Phi'_n(i, \hat{u}, \hat{v}, \hat{w}) = 1$ iff $a = b = c$ and $\Phi_n(i', u, v, w) = 1$.

2. Suppose instead that $\hat{u} = (u, a, x)$ is a gate inside the multiplication tree used to compute $(u, a)$. If $x$ is not of maximal length, then we output 1 iff $\hat{v} = (u, a, x0)$ and $\hat{w} = (u, a, x1)$. Otherwise, if $x$ is of maximal length, we output 1 iff $\hat{v} = (v, b)$ and $\hat{w} = (w, a-b)$ for the correct values of $b$ and $a-b$ (determined by the bits of $x$) and if $\Phi(i', u, v, w) = 1$.

It is clear that if the formulas $\{\Phi_n\}$ are $\mathsf{P}$-uniform, then the resulting formulas $\{\Phi'_n\}$ describing the circuit family $\{C'_n\}$ are also $\mathsf{P}$-uniform. Furthermore, if the formulas $\{\Phi_n\}$ have size $(\log T(n))^c$, then the formulas $\Phi'_n$ have size $(\log T(n))^c (\log T'(n))^{O(1)}$, so the formulas $\{\Phi'_n\}$ are $\log^{c+O(1)}$-uniform. □

With Lemma A.22, we now proceed to show that the depth reduction of Valiant, Skyum, Berkowitz, and Rackoff [VSBR83] preserves $\log^c$-uniformity.

**Proposition A.23** (Uniformity-preserving depth reduction)**.** *Let $\{C_n\}$ be a $\log^c$-uniform arithmetic circuit family of size $T(n)$ and degree $d(n)$ that computes a family of polynomials $\{f_n\}$. Then, there is a $\log^{c+O(1)}$-uniform family of arithmetic circuits $\{C'_n\}$ of size $T'(n) = \mathrm{poly}(T(n), d(n))$ and depth $\Delta(n) = O(\log T(n) \cdot \log d(n))$ that computes the same family of polynomials $\{f_n\}$.*

**Proof.** At a high level, the circuit family $\{C'_n\}$ will be obtained by applying the depth reduction of Valiant, Skyum, Berkowitz, and Rackoff [VSBR83] to the circuit family $\{C_n\}$. The depth reduction guarantees that the circuit family $\{C'_n\}$ has size $T'(n) = \mathrm{poly}(T(n))$ and depth $\Delta(n) = O(\log T(n) \log d(n))$. We need to verify that this depth reduction can be performed in a uniformity-preserving manner.

By applying Lemma A.22, we obtain a $\log^{c+O(1)}$-uniform family of arithmetic circuits $\{C_n^{\hom}\}$ such that $C_n^{\hom}$ is a homogeneous arithmetic circuit of size $O(T(n)d(n)^2)$ that computes the homogeneous components of the polynomial $f_n$. We apply the depth reduction to the circuit family $\{C_n^{\hom}\}$.

We now present the depth reduction, following the construction and proof given by Saptharishi [Sap15, §5.3].

We first define gate quotients, which are intermediate polynomials that will be used in wiring the depth-reduced circuit. For a pair of gates $u$ and $v$ in $C_n^{\hom}$, we define the gate quotient $[u : v](\vec{x})$ to be the polynomial defined inductively as follows.

1. If $u$ and $v$ are the same game, then we define $[u : v](\vec{x}) \coloneqq 1$.
2. If $u$ and $v$ are different gates, then the definition of $[u : v](\vec{x})$ depends on the gate type of $u$.
   (a) If $u$ is a leaf, then we define $[u : v](\vec{x}) \coloneqq 0$.
   (b) If $u$ is an addition gate with children $u_\ell$ and $u_r$, then we define $[u : v](\vec{x}) \coloneqq [u_\ell : v](\vec{x}) + [u_r : v](\vec{x})$.
   (c) If $u$ is a multiplication gate with children $u_\ell$ and $u_r$ where $\deg(u_\ell) \leq \deg(u_r)$, then we define $[u : v](\vec{x}) \coloneqq [u_\ell](\vec{x}) \times [u_r : v](\vec{x})$, where $[u_\ell](\vec{x})$ is the polynomial computed by the gate $u_\ell$.

Because the circuit $C_n^{\hom}$ is homogeneous, each gate quotient $[u : v](\vec{x})$ is a homogeneous polynomial, and if $[u : v](\vec{x}) \neq 0$, then $\deg([u : v](\vec{x})) = \deg(u) - \deg(v)$.

For a parameter $m \in \mathbb{N}$, we define the frontier at degree $m$, denoted by $\mathcal{F}_m$, to be the set of gates of degree $\geq m$ whose children have degree strictly less than $m$. Formally, we have

$$\mathcal{F}_m \coloneqq \{v : \deg(v) \geq m, \deg(v_\ell) < m, \deg(v_r) < m\},$$

where $v_\ell$ and $v_r$ are the left and right children, respectively, of $v$. Note that if $v$ appears in a frontier $\mathcal{F}_m$, then because the circuit $C_n^{\hom}$ is homogeneous, it must be the case that $v$ is a multiplication gate. By induction, one can establish the following identities.

**Subclaim A.24** ([Sap15, Lemma 5.12])**.** *Let $u$ and $v$ be gates in $C_n^{hom}$ and let $m \in \mathbb{N}$ be a parameter such that $\deg(u) \geq m > \deg(v)$. Then*

$$[u](\vec{x}) = \sum_{w \in \mathcal{F}_m} [u : w](\vec{x}) \cdot [w](\vec{x})$$
$$[u : v](\vec{x}) = \sum_{w \in \mathcal{F}_m} [u : w](\vec{x}) \cdot [w : v](\vec{x}).$$

For our purposes, we only need the identities of Subclaim A.24 to argue about the correctness of our depth-reduced circuit. As our main concern is the uniformity of the depth reduction procedure, not its correctness, we encourage readers who are interested in further details about the correctness of the depth reduction to consult Saptharishi [Sap15, Section 5.3].

We now construct the depth reduced circuit $\{C'_n\}$ in $\log_2 d$ rounds, where in the $i$-th round we compute polynomials of the form $[u](\vec{x})$ or $[u : v](\vec{x})$ of degree $t$, where $2^{i-1} < t \leq 2^i$. For $i = 1$, each linear polynomial of the form $[u](\vec{x})$ or $[u : v](\vec{x})$ can be computed directly by an arithmetic circuit of size $O(s)$ and depth $O(\log s)$.

For $i \geq 2$, we compute these polynomials as follows.

- We first describe how to compute polynomials of the form $[u](\vec{x})$ for a gate $u$. Let $m = \deg(u)/2$. Then it follows from Subclaim A.24 that

  $$[u](\vec{x}) = \sum_{w \in \mathcal{F}_m} [u : w](\vec{x}) \cdot [w](\vec{x}) = \sum_{w \in \mathcal{F}_m} [u : w](\vec{x}) \cdot [w_\ell](\vec{x}) \cdot [w_r](\vec{x}),$$

  where $w_\ell$ and $w_r$ are the children of gate $w$. By definition of the frontier $\mathcal{F}_m$, each polynomial on the right-hand side above has degree at most $\deg(u)/2$, and so has been computed in a previous round of the depth reduction. Thus, we can compute $[u](\vec{x})$ by adding an additional $O(T)$ gates and $O(\log T)$ depth.

- Next, we describe how to compute a polynomial of the form $[u : v](\vec{x})$ for gates $u$ and $v$. Let $m = (\deg(u) + \deg(v))/2$. After two applications of Subclaim A.24, we obtain the identity

$$[u : v](\vec{x}) = \sum_{w \in \mathcal{F}_m} \sum_{p \in \mathcal{F}(w_\ell)} [u : w](\vec{x}) \cdot [w_\ell : p](\vec{x}) \cdot [p_\ell](\vec{x}) \cdot [p_r](\vec{x}) \cdot [w_r : v](\vec{x}),$$

  where the gates $w_\ell$ and $w_r$ are the children of the gate $w$ with $\deg(w_\ell) \leq \deg(w_r)$, the gates $p_\ell$ and $p_r$ are likewise the children of $p$, and the set $\mathcal{F}(w_\ell)$ is the frontier at degree $\deg(w_\ell)/2$. As a consequence of the choice of the frontiers in the above sum, each of the five terms in the innermost product are of degree at most $(\deg(u) - \deg(v))/2 = \deg([u : v])/2$. By induction, each of these terms has already been computed in a previous round of the construction. This allows us to compute the gate quotient $[u : v](\vec{x})$ by adding $O(T^2)$ gates and $O(\log T)$ depth to the circuit.

In all, the preceding construction produces a circuit of size $T' = \mathrm{poly}(T, d)$ and depth $O(\log T \cdot \log d)$ that computes the same outputs as the circuit $C_n^{\mathrm{hom}}$. We obtain the depth-reduced circuit for $f_n$ by summing over the gates that correspond to the outputs of $C_n^{\mathrm{hom}}$, each of which computes one of the homogeneous components of $f_n$.

It remains to describe a $\mathsf{P}$-uniform family of formulas $\{\Phi'_n\}$ that decides the gate adjacency relation in the depth-reduced circuit family $\{C'_n\}$. We describe a high-level implementation; the low-level details to properly implement each operation as a full binary tree are similar to those appearing in the proof of Lemma A.22, so we leave these out for the sake of clarity. Let $\{\Phi_n\}$ be the $\mathsf{P}$-uniform family of formulas that decide the gate adjacency relation in the circuit family $\{C_n^{\mathrm{hom}}\}$. Recall that $C'_n$ has depth $\Delta'(n) = O(\log T(n) \log d(n))$ and size $T'(n) = \mathrm{poly}(T(n), d(n))$. Given $i \in [\Delta'(n)]$ and gate names $x, y, z \in [T'(n)]$, we first inspect the parity of $i$ to determine the intended gate types of $x, y, z$. We can determine the types of $x$, $y$, and $z$ from their binary expansion. If any gate is not of the correct type, we output a 0. Otherwise, we branch based on the type of gate $x$.

We first describe a formula $\Psi_n$ that takes two gates $x, y \in [T'(n)]$ as input and returns 1 if and only if $x$ is a parent of $y$.

1. Suppose $x$ is a gate meant to compute $[u]$ for some gate $u$ in the circuit $C_n^{\mathrm{hom}}$. To check if $y$ is a child of $x$, we need to check that $y$ is of the form $[u : w]$, $[w_\ell]$ or $[w_r]$ for some gate $w \in \mathcal{F}_{\deg(u)/2}$. This is straightforward to implement as an OR over all such gate names. (Because the underlying circuit is obtained from the homogenization procedure of Lemma A.22, we can assume without loss of generality that each gate is also labeled by its degree. This makes it easy to determine which gates are in the frontier $\mathcal{F}_{\deg(u)/2}$.)

2. Suppose $x$ is a gate meant to compute $[u : v]$ for some gate $u$ in the circuit $C_n^{\mathrm{hom}}$. To check if $y$ is a child of $x$, we need to check that $y$ is one of the five terms appearing in the sum used to compute $[u : v]$. As in the previous case, this is easy to implement as an OR over the appropriate gates, using the fact that each gate is labeled by its degree.

Given $\Psi_n$, it is easy to construct $\Phi'_n$: define

$$\Phi'_n(i, x, y, z) \coloneqq \Psi_n(i, x, y) \wedge \Psi_n(i, x, z).$$

If the formulas $\{\Phi_n\}$ are $\mathsf{P}$-uniform, then the same is true for $\{\Phi'_n\}$. Furthermore, if the formulas $\{\Phi_n\}$ have size $(\log T(n))^c$, then the formulas $\Phi'_n$ have size $(\log T(n))^c (\log T'(n))^{O(1)}$, so the formulas $\Phi'_n$ are $\log^{c+O(1)}$-uniform as claimed.

The preceding description of $C'_n$ and the formulas $\Phi'_n$ allow the circuit $C'_n$ to have arbitrary fan-in. We can replace $C'_n$ with an equivalent circuit where every gate has fan-in two and the gate types alternate by replacing each large fan-in gate in $C'_n$ with a full alternating binary tree of gates. The details of the wiring for this circuit and the appropriate modifications to the formula $\Phi'_n$ are the same as in Lemma A.22. □

# Appendix B An alternative proof for a special case of Theorem 1.4

In this appendix we describe what is essentially an alternative proof for a special case of Theorem 1.4. The special case when the field's characteristic is bounded by a fixed polynomial, say $n$,[43] and when there is a polynomial-time algorithm that finds a representation of the field (i.e., a suitable irreducible polynomial). In the alternative proof, the conclusion we get differs from the one in Theorem 1.4 in two aspects: the deduced PIT algorithm runs in polynomial time (rather than in time $n^{\log^{(c)}(n)}$), but this algorithm is a Boolean computation (rather than an arithmetic network). We view these differences as relatively minor, and the main reason for presenting the alternative proof is that we find the proof itself interesting.

**The main observation.** The proof is based on the observation that in low-characteristic fields, the hardness assumption in Theorem 1.4 actually implies hardness for Boolean circuits. Hence, we can use results from Boolean hardness-to-randomness to deduce derandomization of a Boolean class that contains PIT.

**The alternative proof.** Recall that the hardness assumption in Theorem 1.4 is determined by the complexity of the reconstruction procedure from Theorem 6.1. The reconstruction procedure is a network of degree larger than $q = |\mathbb{F}|$, and with this relatively high degree, one may hope that the network will be able to perform not only arithmetic computation, but also simulate Boolean circuits. Indeed, when such simulation is possible, the hardness hypothesis implies hardness for Boolean circuits.

Thus, we now ask if it is possible to simulate any Boolean circuit computing a function $\mathbb{F}^n \to \mathbb{F}$ (when the elements are represented as bit-strings) by an arithmetic circuit of similar complexity and degree larger than $q$. We stress that this question refers to functional computation, rather than to computing a polynomial syntactically. As far as we are aware, such simulation is not known in general, but it is known over fields of low characteristics, and it yields the following result:

**Proposition B.1** (simulating Boolean computation by degree-$q$ arithmetic procedures, over fields of low characteristic)**.** *Let $\mathbb{F} = \{\mathbb{F}_n\}_{n\in\mathbb{N}}$ be a field family where $\mathbb{F}_n$ is of characteristic at most $p(n) \leq \text{poly}(n)$, and there is a $\text{poly}(n)$-time algorithm that maps $1^n$ to a representation of $\mathbb{F}_n$. Then, any P-uniform Boolean $\mathcal{NC}$ circuit family of size $n^k$ computing a function $\mathbb{F}_n^n \to \mathbb{F}_n$ can be (functionally) simulated by a P-uniform arithmetic circuit family of size $O(n^{k+1} \cdot p^3)$, degree $p \cdot 2^{\text{polylog}(n)}$, and $p^{th}$ root gates.*

*Proof sketch.* For ease of notation, we refer to the $n^{th}$ field $\mathbb{F}_n$ as $\mathbb{F}_q = \mathbb{F}_{p^r}$ (i.e., the field of size $p^r$). Let $C_n \colon \mathbb{F}_q^n \to \mathbb{F}_q$ be a Boolean $\mathcal{NC}$ circuit. The simulating arithmetic circuit $A$ works as follows:

1. Given input elements in $\mathbb{F}_q$, use the procedure in [LSZ82, Proof of Lemma 2] to transform each element into its representation as an $r$-dimensional vector over $\mathbb{F}_p$. (This requires a trace-dual basis for the standard basis of $\mathbb{F}_q$, and an explicit formula for the trace-dual basis is given in [LN94, Exercise 2.40].)

   The arithmetic complexity of this step is essentially that of computing the field trace function on each element, and computing the trace reduces to computing $r$-many $p^{th}$ roots and adding them.

2. Transform each $\mathbb{F}_p$ element into its binary representation, by brute-force. This can be done using $O(p^3)$ operations per element $x$: for each $i \in [p]$ we map $x$ to $\sum_{c\in\mathbb{F}_p : c_i = 1} \delta_c(x)$, where $c_i$ is the $i^{th}$ bit in the binary representation of $c$ and $\delta_c(x) = \prod_{a\in\mathbb{F}_p\setminus\{c\}} \frac{x-a}{c-a}$. (This is the only part in the proof using the fact that the characteristic is bounded.)

3. Simulate the $\mathcal{NC}$ circuit $C$ on the bits, using arithmetic $+, \times$ operations. Since there are at most $\text{polylog}(n)$ layers of $C$ to simulate, the degree will be at most $2^{\text{polylog}(n)}$.

The size of the arithmetic circuit above is $O\left(n \cdot (p^3 + r \cdot n \cdot \log(p)) + n^k\right)$, and its degree (excluding $p^{th}$ root gates) is at most $p \cdot 2^{\text{polylog}(n)}$. The circuit $A$ is P-uniform because $C$ is P-uniform, and due to our assumption that there is a $\text{poly}(n)$-time algorithm finding a representation of the field (which means that a trace-dual basis can be efficiently hard-wired into $A$). □

[43] This can be generalized to $n^c$ for any fixed constant $c > 1$, at the cost of increasing the constant $k$ in the hypothesis.

Thus, in the special case we are considering, the hardness assumption in Theorem 1.4 (even when using the refined degree bounds in Theorem 6.1) implies that there is a Boolean function computable in fixed polylogarithmic depth that is hard for Boolean circuits of fixed polynomial size $n^{5k}$ and larger polylogarithmic depth on all but finitely many inputs. Specifically, the function treats each block of bits in its input as a field element, and simulates the depth-reduced circuit computing the hard polynomial $\{C_n\}$ using Boolean operations. The hard function is computable by circuits of depth $O(\log^2(n) \cdot \text{polylog}(q))$, where the $\text{polylog}(q)$ term comes from simulating field arithmetic; and the depth of the Boolean circuits of size $n^{5k}$ that fail to compute the hard function is determined by the degree bound $2^{\text{polylog}(n)}$ in our hardness assumption, and thus may be an arbitrarily large polylogarithm.

Using [CT21, Theorem 1.5], it follows that uniform-$\mathcal{BPNC} \subseteq$ uniform-$\mathcal{NC}$, where the uniformity here refers to logspace-uniformity. Finally, Miller, Ramachandran, and Kaltofen [MRK88] proved that PIT can be solved in uniform-$\mathcal{BPNC}$, and hence it can also be solved in uniform-$\mathcal{NC} \subseteq \mathsf{P}$.

# References


[AGS18] Manindra Agrawal, Sumanta Ghosh, and Nitin Saxena. "Bootstrapping variables in algebraic circuits". In: *STOC'18—Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*. ACM, New York, 2018, pp. 1166–1179. DOI: 10.1145/3188745.3188762 (cit. on pp. 1, 2, 6, 9, 56).

[All89] Eric W. Allender. "P-uniform circuit complexity". In: *J. Assoc. Comput. Mach.* 36.4 (1989), pp. 912–928. DOI: 10.1145/76359.76370 (cit. on p. 2).

[All99] Eric Allender. "The Permanent Requires Large Uniform Threshold Circuits". In: *Chicago Journal of Theoretical Computer Science* 1999.7 (Aug. 1999) (cit. on p. 2).

[And20] Robert Andrews. "Algebraic hardness versus randomness in low characteristic". In: *35th Computational Complexity Conference*. Vol. 169. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2020, Art. No. 37, 32. DOI: 10.4230/LIPIcs.CCC.2020.37 (cit. on p. 4).

[AW24] Robert Andrews and Avi Wigderson. "Constant-Depth Arithmetic Circuits for Linear Algebra Problems". In: *65th IEEE Symposium on Foundations of Computer Science (FOCS) 2024*. 2024, pp. 2367–2386. DOI: 10.1109/FOCS61266.2024.00138 (cit. on pp. 3, 68).

[BCSS98] Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale. *Complexity and real computation*. With a foreword by Richard M. Karp. Springer-Verlag, New York, 1998, pp. xvi+453. DOI: 10.1007/978-1-4612-0701-6 (cit. on pp. 18, 19).

[Ber70] E. R. Berlekamp. "Factoring polynomials over large finite fields". In: *Math. Comp.* 24 (1970), pp. 713–735. DOI: 10.2307/2004849 (cit. on p. 70).

[Ber84] Stuart J. Berkowitz. "On computing the determinant in small parallel time using a small number of processors". In: *Information Processing Letters* 18.3 (1984), pp. 147–150. DOI: https://doi.org/10.1016/0020-0190(84)90018-8 (cit. on pp. 34, 36, 65).

[BSS89] Lenore Blum, Mike Shub, and Steve Smale. "On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines". In: *Bull. Amer. Math. Soc. (N.S.)* 21.1 (1989), pp. 1–46. DOI: 10.1090/S0273-0979-1989-15750-9 (cit. on p. 18).

[BvH82] Allan Borodin, Joachim von zur Gathen, and John Hopcroft. "Fast parallel matrix and GCD computations". In: *Information and Control* 52.3 (1982), pp. 241–256. DOI: https://doi.org/10.1016/S0019-9958(82)90766-5 (cit. on pp. 3, 64).

[CIKK15] Marco L. Carmosino, Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. "Tighter Connections between Derandomization and Circuit Lower Bounds". In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2015)*. Ed. by Naveen Garg, Klaus Jansen, Anup Rao, and José D. P. Rolim. Vol. 40. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015, pp. 645–658. DOI: 10.4230/LIPIcs.APPROX-RANDOM.2015.645 (cit. on pp. 1–3).

[CK12] Ruiwen Chen and Valentine Kabanets. "Lower bounds against weakly uniform circuits". In: *Computing and combinatorics*. Vol. 7434. Lecture Notes in Comput. Sci. Springer, Heidelberg, 2012, pp. 408–419. DOI: 10.1007/978-3-642-32241-9_35 (cit. on p. 2).

[CKK14] Ruiwen Chen, Valentine Kabanets, and Jeff Kinne. "Lower bounds against weakly-uniform threshold circuits". In: *Algorithmica* 70.1 (2014), pp. 47–75. DOI: 10.1007/s00453-013-9823-y (cit. on pp. 2, 57).

[CT21] Lijie Chen and Roei Tell. "Hardness vs Randomness, Revised: Uniform, Non-Black-Box, and Instance-Wise". In: *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*. 2021, pp. 125–136. DOI: 10.1109/FOCS52979.2021.00021 (cit. on pp. 2, 5, 6, 14, 20, 87).

[CT23] Lijie Chen and Roei Tell. "Guest column: New ways of studying the BPP = P conjecture". In: *ACM SIGACT News* 54.2 (2023), pp. 44–69 (cit. on pp. 2, 5).

[CTW23] Lijie Chen, Roei Tell, and Ryan Williams. "Derandomization vs refutation: a unified framework for characterizing derandomization". In: *2023 IEEE 64th Annual Symposium on Foundations of Computer Science—FOCS 2023*. IEEE Computer Soc., Los Alamitos, CA, [2023] ©2023, pp. 1008–1047. DOI: 10.1109/FOCS57990.2023.00062 (cit. on p. 5).

[CZ81] David G. Cantor and Hans Zassenhaus. "A new algorithm for factoring polynomials over finite fields". In: *Math. Comp.* 36.154 (1981), pp. 587–592. DOI: 10.2307/2007663 (cit. on p. 70).

[DPT24] Dean Doron, Edward Pyne, and Roei Tell. "Opening up the distinguisher: a hardness to randomness approach for BPL = L that uses properties of BPL". In: *STOC'24—Proceedings of the 56th Annual ACM Symposium on Theory of Computing*. ACM, New York, [2024] ©2024, pp. 2039–2049. DOI: 10.1145/3618260.3649772 (cit. on p. 2).

[Ebe89] Wayne Eberly. "Very fast parallel polynomial arithmetic". In: *SIAM J. Comput.* 18.5 (1989), pp. 955–976. DOI: 10.1137/0218066 (cit. on p. 2).

[For15] Michael A. Forbes. "Deterministic Divisibility Testing via Shifted Partial Derivatives". In: *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*. 2015, pp. 451–465. DOI: 10.1109/FOCS.2015.35 (cit. on p. 65).

[GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. "Delegating Computation: Interactive Proofs for Muggles". In: *J. ACM* 62.4 (Sept. 2015). DOI: 10.1145/2699436 (cit. on pp. 7, 14).

[GKSS22] Zeyu Guo, Mrinal Kumar, Ramprasad Saptharishi, and Noam Solomon. "Derandomization from Algebraic Hardness". In: *SIAM Journal on Computing* 51.2 (2022), pp. 315–335. DOI: 10.1137/20M1347395 (cit. on pp. 1–3, 6–8, 27, 28, 30, 32, 33, 35, 53, 56, 63).

[Gol11] Oded Goldreich. "In a world of **P** = **BPP**". In: *Studies in complexity and cryptography*. Vol. 6650. Lecture Notes in Comput. Sci. Springer, Heidelberg, 2011, pp. 191–232. DOI: 10.1007/978-3-642-22670-0_20 (cit. on p. 5).

[Gol18] Oded Goldreich. "On doubly-efficient interactive proof systems". In: *Found. Trends Theor. Comput. Sci.* 13.3 (2018), front matter, 1–89. DOI: 10.1561/0400000084 (cit. on p. 14).

[GV17] Andrei Gabrielov and Nicolai Vorobjov. "Topological lower bounds for arithmetic networks". In: *computational complexity* 26.3 (2017), pp. 687–715. DOI: 10.1007/s00037-016-0145-8 (cit. on p. 3).

[IKW02] Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. "In search of an easy witness: exponential time vs. probabilistic polynomial time". In: vol. 65. 4. Special issue on complexity, 2001 (Chicago, IL). 2002, pp. 672–694. DOI: 10.1016/S0022-0000(02)00024-7 (cit. on p. 2).

[JS12] Maurice Jansen and Rahul Santhanam. "Stronger Lower Bounds and Randomness-Hardness Trade-Offs Using Associated Algebraic Complexity Classes". In: *29th International Symposium on Theoretical Aspects of Computer Science (STACS 2012)*. Ed. by Christoph Dürr and Thomas Wilke. Vol. 14. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012, pp. 519–530. DOI: 10.4230/LIPIcs.STACS.2012.519 (cit. on pp. 1, 2).

[JS13] Maurice Jansen and Rahul Santhanam. "Permanent does not have succinct polynomial size arithmetic circuits of constant depth". In: *Information and Computation* 222 (2013). 38th International Colloquium on Automata, Languages and Programming (ICALP 2011), pp. 195–207. DOI: https://doi.org/10.1016/j.ic.2012.10.013 (cit. on pp. 2, 57).

[Kal89] Erich L. Kaltofen. "Factorization of Polynomials Given by Straight-Line Programs". In: *Adv. Comput. Res.* 5 (1989), pp. 375–412 (cit. on pp. 16, 39, 56, 57, 69, 73).

[KI04] Valentine Kabanets and Russell Impagliazzo. "Derandomizing Polynomial Identity Tests Means Proving Circuit Lower Bounds". In: *computational complexity* 13 (1 2004), pp. 1–46. DOI: 10.1007/s00037-004-0182-6 (cit. on pp. 1–4, 6, 8, 40, 54, 56).

[KLV23] Yael Tauman Kalai, Alex Lombardi, and Vinod Vaikuntanathan. "SNARGs and PPAD Hardness from the Decisional Diffie-Hellman Assumption". In: *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part II*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14005. Lecture Notes in Computer Science. Springer, 2023, pp. 470–498. DOI: 10.1007/978-3-031-30617-4_16 (cit. on pp. 10, 21).

[Koz80] Dexter Kozen. "Indexings of subrecursive classes". In: *Theoretical Computer Science* 11.3 (1980), pp. 277–301 (cit. on p. 2).

[KP09] Pascal Koiran and Sylvain Perifel. "A Superpolynomial Lower Bound on the Size of Uniform Nonconstant-depth Threshold Circuits for the Permanent". In: *2009 24th Annual IEEE Conference on Computational Complexity*. 2009, pp. 35–40. DOI: 10.1109/CCC.2009.19 (cit. on pp. 2, 57).

[KP11] Pascal Koiran and Sylvain Perifel. "Interpolation in Valiant's theory". In: *Comput. Complexity* 20.1 (2011), pp. 1–20. DOI: 10.1007/s00037-011-0002-8 (cit. on p. 1).

[KS19] Mrinal Kumar and Ramprasad Saptharishi. "Hardness-randomness tradeoffs for algebraic computation". In: *Bulletin of the European Association for Theoretical Computer Science. EATCS* 129 (2019), pp. 56–87 (cit. on pp. 1, 5).

[KSS15] Swastik Kopparty, Shubhangi Saraf, and Amir Shpilka. "Equivalence of Polynomial Identity Testing and Polynomial Factorization". In: *Computational Complexity* 24.2 (2015), pp. 295–331. DOI: 10.1007/s00037-015-0102-y (cit. on pp. 64, 69, 70, 74, 76, 78–80).

[KST19] Mrinal Kumar, Ramprasad Saptharishi, and Anamay Tengse. "Near-optimal bootstrapping of hitting sets for algebraic circuits". In: *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, Philadelphia, PA, 2019, pp. 639–646. DOI: 10.1137/1.9781611975482.40 (cit. on pp. 1, 2, 6, 9, 56).

[LN94] Rudolf Lidl and Harald Niederreiter. *Introduction to finite fields and their applications*. first. Cambridge University Press, Cambridge, 1994, pp. xii+416. DOI: 10.1017/CBO9781139172769 (cit. on p. 86).

[LSZ82] Abraham Lempel, Gadiel Seroussi, and Jacob Ziv. "On the power of straight-line computations in finite fields". In: *IEEE Transactions on Information Theory* 28.6 (1982), pp. 875–880 (cit. on p. 86).

[MM97] Klaus Meer and Christian Michaux. "A survey on real structural complexity theory". In: *Bulletin of the Belgian Mathematical Society. Simon Stevin* 4.1 (1997), pp. 113–148. DOI: 10.36045/bbms/1105730626 (cit. on p. 19).

[MMP96] J. L. Montaña, J. E. Morais, and Luis M. Pardo. "Lower bounds for arithmetic networks II: Sum of Betti numbers". In: *Applicable Algebra in Engineering, Communication and Computing* 7.1 (1996), pp. 41–51. DOI: 10.1007/BF01613615 (cit. on p. 3).

[MP93] J. L. Montaña and L. M. Pardo. "Lower bounds for arithmetic networks". In: *Applicable Algebra in Engineering, Communication and Computing* 4.1 (1993), pp. 1–24. DOI: 10.1007/BF01270397 (cit. on p. 3).

[MRK88] Gary L. Miller, Vijaya Ramachandran, and Erich Kaltofen. "Efficient parallel evaluation of straight-line code and arithmetic circuits". In: *SIAM Journal on Computing* 17.4 (1988), pp. 687–695 (cit. on p. 87).

[NIR03] Alan Nash, Russell Impagliazzo, and Jeff Remmel. "Universal Languages and the Power of Diagonalization". In: *Proc. 18th Computational Complexity Conference, CCC*. 2003, p. 337 (cit. on p. 2).

[NIR06] Alan Nash, Russell Impagliazzo, and Jeff Remmel. "Infinitely-Often Universal Languages and Diagonalization". In: *Electronic Colloquium on Computational Complexity* 13 (2006), p. 51 (cit. on p. 2).

[NW94] Noam Nisan and Avi Wigderson. "Hardness vs Randomness". In: *J. Comput. Syst. Sci.* 49.2 (1994), pp. 149–167. DOI: 10.1016/S0022-0000(05)80043-1 (cit. on pp. 2, 9, 40).

[Raz10] Ran Raz. "Elusive Functions and Lower Bounds for Arithmetic Circuits". In: *Theory of Computing* 6.7 (2010), pp. 135–177. DOI: 10.4086/toc.2010.v006a007 (cit. on pp. 6, 20, 56, 58).

[San23] Rahul Santhanam. "An algorithmic approach to uniform lower bounds". In: *38th Computational Complexity Conference*. Vol. 264. LIPIcs. Leibniz Int. Proc. Inform. 2023, Art. No. 35, 26 (cit. on p. 2).

[Sap15] Ramprasad Saptharishi. "A survey of lower bounds in arithmetic circuit complexity". In: *Github Survey* (2015) (cit. on p. 84).

[Sch80] Jacob T. Schwartz. "Fast Probabilistic Algorithms for Verification of Polynomial Identities". In: *J. ACM* 27.4 (1980), pp. 701–717. DOI: 10.1145/322217.322225 (cit. on p. 11).

[Sha13] Igor R. Shafarevich. *Basic algebraic geometry 1: Varieties in projective space*. Third. Springer-Verlag, Berlin, 2013, pp. xviii+310. DOI: 10.1007/978-3-642-37956-7 (cit. on p. 13).

[Str73] Volker Strassen. "Vermeidung von Divisionen". In: *J. Reine Angew. Math.* 264 (1973), pp. 184–202 (cit. on p. 65).

[SW13] Rahul Santhanam and Ryan Williams. "On medium-uniformity and circuit lower bounds". In: *2013 IEEE Conference on Computational Complexity—CCC 2013*. IEEE Computer Soc., Los Alamitos, CA, 2013, pp. 15–23. DOI: 10.1109/CCC.2013.40 (cit. on pp. 2, 57).

[Vad12] Salil P. Vadhan. "Pseudorandomness". In: *Foundations and Trends® in Theoretical Computer Science* 7.1–3 (2012), pp. 1–336. DOI: 10.1561/0400000010 (cit. on p. 40).

[VSBR83] L. G. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff. "Fast Parallel Computation of Polynomials Using Few Processors". In: *SIAM Journal on Computing* 12.4 (1983), pp. 641–644. DOI: 10.1137/0212043. eprint: https://doi.org/10.1137/0212043 (cit. on pp. 5, 6, 39, 57, 82, 83).

[vzGat84] Joachim von zur Gathen. "Parallel Algorithms for Algebraic Problems". In: *SIAM Journal on Computing* 13.4 (1984), pp. 802–824. DOI: 10.1137/0213050 (cit. on pp. 3, 57, 68, 70, 71, 73).

[vzGat86a] Joachim von zur Gathen. "Parallel arithmetic computations: A survey". In: *Mathematical Foundations of Computer Science 1986*. Ed. by Jozef Gruska, Branislav Rovan, and Juraj Wiedermann. Berlin, Heidelberg: Springer Berlin Heidelberg, 1986, pp. 93–112 (cit. on pp. 2, 12, 13).

[vzGat86b] Joachim von zur Gathen. "Representations and Parallel Computations for Rational Functions". In: *SIAM Journal on Computing* 15.2 (1986), pp. 432–452. DOI: 10.1137/0215030 (cit. on p. 3).

[vzGG13] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. 3rd ed. Cambridge University Press, 2013 (cit. on pp. 11, 68–70, 73, 74).

[Yao97] Andrew Chi-Chih Yao. "Decision Tree Complexity and Betti Numbers". In: *Journal of Computer and System Sciences* 55.1 (1997), pp. 36–43. DOI: 10.1006/jcss.1997.1495 (cit. on p. 3).

[Zip79] Richard Zippel. "Probabilistic algorithms for sparse polynomials". In: *Proceedings of the International Symposium on Symbolic and Algebraic Computation, EUROSAM 1979*. 1979, pp. 216–226. DOI: 10.1007/3-540-09519-5_73 (cit. on p. 11).



ECCC ISSN 1433-8092
https://eccc.weizmann.ac.il