Derandomizing Isolation and Polynomial Identity Testing

By

Subramanya Gautam Prakriya Venkata

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2019

Date of final oral examination: 5/9/2019

The dissertation is approved by the following members of the Final Oral Committee:

Dieter van Melkebeek, Professor, Computer Sciences
Jin-Yi Cai, Professor, Computer Sciences
Eric Bach, Professor, Computer Sciences
Matthew Anderson, Visiting Assistant Professor, Union College

Dedicated to my parents

Acknowledgments

First and foremost I would like to thank my advisor Dieter van Melkebeek for being an ideal mentor and a patient guide through my research journey. Dieter's high standards of scholarship are an ideal I will strive towards throughout my career. I'm particularly indebted to Dieter for his superhuman patience while waiting for, and proofreading numerous poorly written drafts of my work, including this thesis. Dieter has also played a significant role in shaping me as a teacher. First as an exceptional example through his lectures, then as a mentor when I worked with him as a teaching assistant, and finally by attending and providing feedback for my lectures (!!) when I was an instructor. Dieter has been an unyielding source of support and encouragement through the highs and lows of graduate school.

I would like to thank my advisor at the Chennai Mathematical Institute (CMI), Samir Datta for introducing me to the wonderful world of complexity theory, including the NL vs. UL question that is the subject of the first chapter of this thesis.

I would like to thank my committee members Matt Anderson, Eric Bach and Jin-Yi Cai for making time for my defense. I'm particularly indebted to Matt Anderson for his careful proofreading of this thesis and his valuable feedback.

I would also like to thank my colleague in graduate school, Andrew Morgan, for many insightful discussions on research and for his many suggestions that have played a significant role in making this document better. Drew's passion for research and work ethic are inspirational.

I would like to thank my partner Aayushi Uberoi. Aayushi's optimism and

support have been sources of strength throughout my graduate career.

Finally, I am indebted to my parents for their patient support and encouragement.

Contents

Abstract

We study the possibility of obtaining deterministic isolations and polynomial identity tests in restricted settings.

- Isolations: Isolation refers to the problem of singling out a solution to a problem that may have many solutions. We present results for the NL-complete problem of reachability on digraphs, and for the LogCFL-complete problem of certifying acceptance on shallow semi-unbounded circuits.

  A common approach employs small weight assignments that make the solution of minimum weight unique. The Isolation Lemma and other known procedures use $\Omega(n)$ random bits to generate weights of bitlength $O(\log n)$. We develop a derandomized version for both settings that uses $O((\log n)^{3/2})$ random bits and produces weights of bitlength $O((\log n)^{3/2})$ in logarithmic space. The construction allows us to show that every language in NL can be accepted by a nondeterministic machine that runs in polynomial time and $O((\log n)^{3/2})$ space, and has at most one accepting computation path on every input. Similarly, every language in LogCFL can be accepted by a nondeterministic machine equipped with a stack that does not count towards the space bound, that runs in polynomial time and $O((\log n)^{3/2})$ space, and has at most one accepting computation path on every input.

  We also show that the existence of stronger notions of isolations for reachability on digraphs implies that NL can be decided in logspace with polynomial advice. A similar result holds for certifying acceptance on shallow semi-unbounded circuits and LogCFL.

- Polynomial identity testing: We report progress towards constructing efficient deterministic algorithms to determine whether constant-read arithmetic formulas are identically zero. We give a polynomial time whitebox and $n^{O(\log n)}$ time blackbox reduction from testing whether read-$(k+1)$ formulas are identically zero to testing whether the sum of constantly many read-k formulas is identically zero. Using the blackbox reduction, we construct an $n^{O(\log n)}$ time blackbox identity test for read-3 formulas. Prior to our work no subexponential time deterministic blackbox identity tests were known for this model.

  We also report progress towards constructing a reduction from testing sums of read-k formulas to testing a single read-k formula. Combined with the first reduction this would be sufficient to obtain identity tests for read-k formulas for all constants k. We establish a so-called hardness of representation result for sums of read-k formulas that generalizes a similar result for read-once formulas.

1 Introduction

Randomness is a powerful computational resource. Many natural computational problems have simple and efficient randomized algorithms, but are not known to have comparable deterministic algorithms. A fundamental question is whether we can simulate such randomized algorithms deterministically without a significant loss in efficiency. Besides its evident applications in algorithm design, this question is also significant due to its close connections to circuit lower bounds.

Proving circuit lower bounds is one of the central goals in complexity theory. Perhaps the most famous example of a conjectured circuit lower bound is the following.

Conjecture 1.1. SATISFIABILITY *is not computable by any polynomial sized family of boolean circuits.*

This conjecture implies $\mathrm{P} \neq \mathrm{NP}$. Conjecture 1.1 has an arithmetic analogue which is usually stated with respect to the Permanent. The Permanent is the unsigned version of the Determinant. For $n \in \mathbb{N}$, the Permanent, $\textsc{Perm}_n$, over variables $\{X_{ij}\}_{i,j\in[n]}$ is defined as follows.

$$\mathrm{Perm}_n \doteq \sum_{\sigma\in S_n} \prod_{i\in[n]} X_{i\sigma(i)}.$$

Conjecture 1.2. *The family of polynomials* $\{\mathrm{Perm}_n\}_{n\in\mathbb{N}}$ *is not computable by any polynomial sized family of arithmetic circuits.*

Resolving these conjectures is an extremely challenging task, and there are barrier results that show that current techniques are likely insufficient [1, 14, 75].

A long line of research shows that strong circuit lower bounds imply efficient derandomizations for large classes of randomized algorithms [13, 21, 47–49, 61, 73, 81, 85, 92, 98]. In the other direction, such derandomizations are known to imply circuit lower bounds [50, 51]. For example, it is known that if all languages in BPP, the class of languages computable by bounded error polynomial time randomized algorithms, have subexponential time deterministic (or even nondeterministic) algorithms, then either Conjecture 1.2 holds or a weaker version of Conjecture 1.1 holds [51]. In fact, the same conclusion follows from subexponential time deterministic algorithms for specific computational problems that are known to have efficient randomized algorithms [11, 51]. In this thesis we consider two such problems: The *min-isolation problem* and *polynomial identity testing* (PIT). We study the possibility of obtaining deterministic and randomness-efficient algorithms for these problems in restricted settings.

Isolation refers to the process of singling out a solution to a problem that may have many solutions. In most settings the computational problem underlying the isolation takes the following form.

Min-isolation problem: Given a finite universe U, and a family of subsets $\mathcal{S} \subseteq 2^U$, specified succinctly, construct a min-isolating weight assignment $w : U \to [\mathrm{poly}(|U|)]$ for the set system $\mathcal{S}$, i.e., construct a weight assignment w such that there is a unique min-weight set in $\mathcal{S}$ under w. Typically, the family of subsets $\mathcal{S}$ is specified by associating the characteristic vectors of sets in $\mathcal{S}$ with accepting assignments of a boolean circuit.

An efficient randomized algorithm for the min-isolation problem follows from the Mulmuley, Vazirani and Vazirani Isolation Lemma [70], which shows that a random weight assignment, that assigns weights to elements in U from $\{1, \ldots, k|U|\}$, isolates a unique min-weight set with probability at least $1 - 1/k$.

Isolations play an important role in algorithms with an algebraic flavor in order to prevent cancellations from happening. Examples include reductions of multivariate to univariate PIT [5, 60] and recent approaches to the hamiltonicity

problem [18, 27, 28, 38]. The process also plays an important role in the design of parallel algorithms, where it ensures that the various parallel processes all work towards a single global solution rather than towards individual solutions that may not be compatible with one another. Both uses culminate in the asymptotically best known parallel algorithms for finding perfect matchings in graphs [70] and related problems [2, 56, 65]. A wide range of other algorithmic applications of isolation exist [6, 12, 16, 17, 19, 23, 29, 32, 33, 36, 41, 44, 45, 53, 54, 59, 66, 68, 74, 84, 88, 90, 91, 96].

In complexity theory isolation constitutes a key tool to show that in some computational models, hard problems are no easier to solve on instances with unique solutions than on general instances. This happens by establishing efficient simulations on *unambiguous* machines, i.e., nondeterministic machines with at most one accepting computation path on every input.

In Chapter 2, we present our results on derandomizing isolations in the space bounded setting. Our main result is an efficient construction of a min-isolating weight assignment for set systems $\mathcal{S} \subseteq 2^{|U|}$ defined by paths between pairs of vertices in directed acyclic graphs over the vertex set U. Our weight assignment construction requires $O((\log |U|)^{3/2})$ random bits and assigns weights of size at most $2^{O((\log |U|)^{3/2})}$. Using our construction, we show that every non-deterministic logspace machine can be simulated by an unambiguous machine that runs in polynomial time and $O((\log n)^{3/2})$ space.

Polynomial identity testing is the problem of determining whether a given multivariate polynomial is identically zero, where the input polynomial is specified succinctly by an arithmetic circuit.

A simple randomized algorithm follows from the Schwartz-Zippel Lemma [35, 80, 99]. The lemma states that a random assignment to the variables picked from some set S of field elements is a non-zero of the input polynomial, P, with probability at least $d/|S|$, where d is the total degree of P.

The resulting randomized algorithm is a *blackbox* identity test, i.e., the al-

gorithm only evaluates the arithmetic circuit at certain points, as opposed to a *whitebox* test, which might use more information about the input circuit.

PIT has many algorithmic applications, including primality testing [3], exact algorithms for NP-complete problems [18, 20, 97], and parallel algorithms for perfect matching. PIT also plays a role in the proof of $\mathrm{IP} = \mathrm{PSPACE}$ [82].

Given the fundamental nature of PIT and its close connections to circuit lower bounds, a substantial amount of research has focused on constructing efficient deterministic identity tests for restricted arithmetic circuit families. These include bounded-depth circuits [57, 58, 60, 78], bounded-read multilinear formulas [8, 69, 83], and bounded-read algebraic branching programs [5, 9, 42, 43].

In Chapter 3, we consider the model of bounded-read arithmetic formulas *without* the multilinear restriction. We present the first blackbox identity tests for read-2 and read-3 arithmetic formulas. We also present progress towards constructing efficient tests for read-k formulas for all constants k.

2 Isolation in Space-Bounded Settings

2.1 Introduction

In this chapter we focus on isolation and unambiguity in two space-bounded settings, namely the NL-complete setting of reachability on digraphs (denoted as REACHABILITY), and the LogCFL-complete setting of certifying acceptance on shallow semi-unbounded circuits (denoted as CIRCUIT CERTIFICATION). Gal and Wigderson [40] developed *randomized* isolations for those settings, and Reinhardt and Allender [76] established efficient *randomized* unambiguous simulations. We investigate the possibility of *deterministic* isolations and unambiguous simulations, and present both positive and negative results for those settings.

Randomness enters these works [40, 76] via the Isolation Lemma: For any non-empty set system over a finite universe U, a random assignment of small integer weights to the elements of U likely makes the set of minimum weight unique. In the context of REACHABILITY, [76] applies the Isolation Lemma to construct a weight assignment to the edges of a digraph G such that the following property holds with high probability: For all vertices s and t, there is at most one path of minimum weight from s to t in G. We call such a weight assignment *min-isolating* for G. The process uses $\Omega(n)$ random bits, produces weights of bitlength $O(\log n)$, and runs in space $O(\log n)$, where n denotes the number of vertices of G. In fact, the process only depends on the number of vertices; we refer to it as a *weight assignment generator*.

The crux of our positive results is a logspace weight assignment generator for the specific settings considered that uses significantly fewer random bits at the expense of slightly higher bitlengths. For technical reasons we restrict to layered digraphs and assign weights to the (internal) *vertices* rather than to the edges. These are not essential differences[1] but they facilitate a natural iterative/recursive approach towards the construction of the weight assignment, and allow for a cleaner and unified treatment. We state an informal version of the result here, and refer to Section 2.3 for the formal statement.

Theorem 2.1 (informal)**.** *There exists a min-isolating weight assignment generator for layered digraphs that uses $O((\log n)^{3/2})$ random bits, produces weights of bitlength $O((\log n)^{3/2})$, and runs in space $O(\log n)$, where n denotes the number of vertices.*

We use Theorem 2.1 to derive the following isolation result for NL, where the notation $\mathrm{UTISP}(t, s)$ stands for the class of languages accepted by unambiguous nondeterministic machines that run in time t and space s.

Theorem 2.2. $\mathrm{NL} \subseteq \mathrm{UTISP}(\mathrm{poly}(n), (\log n)^{3/2})$.

In words: Every language in NL can be accepted by a nondeterministic machine that runs in polynomial time and $O((\log n)^{3/2})$ space, and has at most one accepting computation path on every input.

Theorem 2.2 should be contrasted with the current most space efficient simulation of NL on *deterministic* machines, which is given by Savitch's Theorem [79]: $\mathrm{NL} \subseteq \mathrm{DSPACE}((\log n)^2)$. That simulation does not run in polynomial time. In fact, the best upper bound on the running time is the one for generic computations in $\mathrm{DSPACE}((\log n)^2)$, namely $n^{O(\log n)}$. REACHABILITY can be solved in linear time and space using depth-first search or breadth-first search. The smallest

[1] The restriction of REACHABILITY to layered digraphs remains NL-complete. We can reassign the weight of a vertex to each of its outgoing edges without affecting the total weight of any solution.

known space bound for a deterministic algorithm that decides REACHABILITY in polynomial time is only slightly sublinear, namely $n/2^{\Theta(\sqrt{\log n})}$ [15].

On the "negative" side, we give evidence that certain restricted types of isolations for REACHABILITY will be hard to find (if they exist at all):

- When viewed as reductions from REACHABILITY to itself, the isolations from [40, 76] as well as ours map an instance $x \doteq (G, s, t)$ to an instance $f(x)$ where the underlying graph contains more vertices than G. As such, solutions to the reduced instance $f(x)$ are not necessarily solutions to the original instance x. One can ask for isolations f with the additional property that the solution to $f(x)$ is also a solution to x. We refer to such isolations as *prunings*.

- Suppose that in addition to computing a min-isolating weight assignment, we can also compute the minimum weight of a solution in logspace. Then we can trivially decide REACHABILITY in logspace: There exists a path from s to t in G if and only if the min-weight of the instance (G, s, t) is finite. What if we have a logspace function that is only known to agree with the min-weight when the latter is finite?

We show that the existence of either restricted type of isolation implies an inclusion of complexity classes that is considered unlikely.

Theorem 2.3. *Either one of the following hypotheses implies that* $\mathrm{NL} \subseteq \mathrm{L/poly}$*:*

1. REACHABILITY *on layered digraphs has a logspace pruning.*

2. REACHABILITY *on layered digraphs has a logspace weight function* ω *that is min-isolating, and there exists a logspace function* μ *such that* $\mu(x)$ *equals the min-weight* $\omega(x)$ *of* x *under* ω *on positive instances* x*.*

In fact, the conclusion holds even if the algorithms are randomized, as long as the probability of success exceeds $\frac{2}{3} + \frac{1}{\mathrm{poly}(n)}$ *and the algorithms run in logspace when given two-way access to the random bits.*

Two-way access to the random bits means that the random bits are provided on a dedicated tape to which the machine has two-way read access.

It is not clear to us that Theorem 2.3 should be viewed as a roadblock towards reducing the number of random bits and the bitlength in Theorem 2.1 from $O((\log n)^{3/2})$ down to $O(\log n)$, and thereby show that $\mathrm{NL} = \mathrm{UL}$.

The corresponding results for CIRCUIT CERTIFICATION and the complexity class LogCFL are stated in Section 2.4 (positive) and Section 2.5 (negative).

Techniques The crux for our positive results is an iterative/recursive construction of a min-isolating weight assignment generator. In both settings there are $\Theta(\log n)$ levels of recursion. In the case of REACHABILITY the subproblems at the kth level correspond to the subgraphs induced by blocks of 2^k successive layers of G.

We develop several methods to build a min-isolating weight assignment w_{k+1} at the $(k+1)$st level out of a min-isolating weight assignment w_k at the kth level. The methods represent different trade-offs between the seed length and the bitlength. Our starting point is two simple constructions, namely one based on shifting, and one based on universal families of hash functions. The shifting approach does not need any randomness at all but yields bitlength $\Theta((\log n)^2)$. Hashing yields the smaller bitlength $O(\log n)$ but needs $\Theta((\log n)^2)$ random bits. Either one of those simple approaches on its own is sufficient to establish weaker versions of our positive results, namely where the randomness or space bound is increased from $O((\log n)^{3/2})$ to $O((\log n)^2)$, i.e.,

$$\mathrm{NL} \subseteq \mathrm{UTISP}(\mathrm{poly}(n), (\log n)^2). \tag{2.1}$$

The $\Theta((\log n)^2)$ bits of randomness in the hashing-based approach are composed of $\Theta(\log n)$ bits to describe a fresh hash function at each of the $\Theta(\log n)$ levels of recursion. The reason one needs a fresh hash function at each level is to avoid potential stochastic dependencies. We show how to use shifting to preclude the existence of such dependencies, allowing us to reuse the same hash

function at $\Theta(\sqrt{\log n})$ levels. This combination of shifting and hashing balances the seed length and bitlength to $\Theta((\log n)^{3/2})$ each, and yields Theorem 2.1 and its counterpart for CIRCUIT CERTIFICATION.

For Theorem 2.2 and its counterpart for LogCFL we need to get rid of the randomness completely. We could do so by exhaustively trying all random seeds, and employing an unambiguous logspace machine of [76] to select one that yields a min-isolating weight assignment. However, given that the number of random bits is $\Theta((\log n)^{3/2})$, an exhaustive search would require time $n^{\Theta(\sqrt{\log n})}$. In order to do better, we exploit the structure of the randomness – it consists of $\Theta(\sqrt{\log n})$ hash functions requiring $\Theta(\log n)$ random bits each. Using unambiguous logspace machines from [76] this allows us to pick the hash functions one by one, maintaining the invariant that the resulting weight assignments are min-isolating for the corresponding levels, and then use the final assignment to decide reachability unambiguously. As we can cycle through all possibilities for a hash function at a given level in polynomial time, this yields a full derandomization running in polynomial time and space $O((\log n)^{3/2})$.

The "negative" results, Theorem 2.3 and its counterpart for CIRCUIT CERTIFICATION, follow along the lines of the argument for a similar result from [34] in the time-bounded setting. The first part is the space-bounded equivalent of the main result in [34]; it suffices to verify that the argument from the time-bounded setting carries over to the space-bounded setting. The second part does not have a counterpart in [34] but follows from a similar argument and some additional observations.

Related Papers There is a remarkable correspondence in terms of statements and high-level approach between Theorem 2.2 and the result by Saks and Zhou [77] that $\mathrm{BPL} \subseteq \mathrm{DSPACE}((\log n)^{3/2})$. Both have a recursive structure, use hashing,[2] need to get rid of stochastic dependencies so as to enable the reuse of the same hash function at multiple levels of recursion, exploit the leeway created

[2] [77] does so via Nisan's pseudorandom generator [71].

by the discrepancy between the randomness and processing space (bitlength), and ultimately balance them to $\Theta((\log n)^{3/2})$ bits each. In contrast to [77], we do obtain the equivalent of a pseudorandom generator. As another contrast we are able to improve the running time to polynomial, which remains open in the case of BPL [24]. Our high-level approach for the improvement is similar to the one for the improvement from $\mathrm{BPL} \subseteq \mathrm{DSPACE}((\log n)^2)$ in [71] to $\mathrm{BPL} \subseteq \mathrm{DTISP}(\mathrm{poly}(n), (\log n)^2)$ in [72].

The recent derandomization results for PERFECTMATCHING on bipartite graphs [37] and for polynomial identity testing (PIT) for read-once arithmetic branching programs [5] also employ a combination of hashing and shifting but no balancing. Their construction requires $O((\log n)^2)$ random bits as opposed to our $O((\log n)^{3/2})$. It is an open question whether our approach can be used to reduce the number of random bits in those settings. The application to PIT in [5] is via a reduction from multivariate (multilinear) PIT to univariate PIT based on isolation: If $w : [n] \mapsto \mathbb{N}$ is a weight assignment to the variables that is min-isolating for the monomials that occur in a nonzero n-variate polynomial $P(x_1, x_2, \ldots, x_n)$, then the univariate polynomial $P(t^{w(1)}, t^{w(2)}, \ldots, t^{w(n)})$ is nonzero.

Kallampally and Tewari [52] independently proved the weaker inclusion (2.1) that follows from either of our starting points – the pure shifting approach that needs no randomness and bitlength $\Theta((\log n)^2)$, and the pure hashing approach that needs $\Theta((\log n)^2)$ random bits and yields bitlength $O(\log n)$. In their construction both quantities are $\Theta((\log n)^2)$.

Krishan and Limaye [63] independently proved the first part of our "negative" results (Theorem 2.3 and its counterpart for LogCFL), which follow from a space-bounded rendering of the main argument in [34].[3] They verify in detail that the argument from [34] carries over to the settings of REACHABILITY and CIRCUIT CERTIFICATION. Our approach is to present some generic conditions for the argument from [34] to apply, and show that the conditions hold in the settings of REACHABILITY and CIRCUIT CERTIFICATION.

[3]The current version of the paper claims that the arguments also rely on [76], but the authors agree that [76] is not needed there (personal communication).

Organization In Section 2.2 we introduce our terminology and survey prior work. In Section 2.3 we derive our positive results for REACHABILITY and NL. The results essentially also follow as corollaries to the corresponding results for CIRCUIT CERTIFICATION and LogCFL, which we prove from scratch in Section 2.4. This organization allows us to develop our ideas in the more familiar setting of REACHABILITY and NL in a gradual and somewhat informal way, and suffice with a formal proof without much intuition in the more general setting of CIRCUIT CERTIFICATION and LogCFL. We spell out the connection in Section 2.4.3. In Section 2.5 we present our negative results for both settings. In Section 2.6 we review results from [76] and present stronger variants that we need for our results.

2.2 Preliminaries

We introduce our notation and terminology regarding isolation and unambiguity, and provide background on REACHABILITY, CIRCUIT CERTIFICATION, and randomized isolations for those problems. We also present a formal definition of the notion of a weight assignment generator, and survey prior work on derandomizing isolation.

2.2.1 Isolation and Unambiguity

Let us define a computational (promise)[4] problem as a mapping $\Pi : X \mapsto 2^Y$ from an instance $x \in X$ to a set $\Pi(x)$ of solutions $y \in Y$, where x and y are strings that typically describe other types of objects. Given an instance $x \in X$, the decision version of Π asks to determine whether $\Pi(x)$ is nonempty. We denote by $L(\Pi)$ the set (language) of all instances $x \in X$ for which the decision is positive. The search version of Π asks to produce a solution $y \in \Pi(x)$, or report that no solution exists. For example, for the NP-complete problem of SATISFIABILITY,

[4] We use the prefix "promise" when we want to make it clear that the domain X of Π may be restricted, i.e., may not equal the set of all strings.

x represents a Boolean formula, and $\Pi(x)$ its satisfying assignments. For the NL-complete problem of REACHABILITY, x represents a triple (G, s, t) consisting of a directed graph G, a start vertex s, and a target vertex t, and $\Pi(x)$ is the set of paths from s to t in G.

A nondeterministic machine M is said to *accept* Π (or $L(\Pi)$) if for every $x \in X$, M on input x has an accepting computation path if and only if $x \in L(\Pi)$. We say that the machine M *decides* Π (or $L(\Pi)$) if M has an accepting computation path on every $x \in X$, and on each such path M outputs a bit indicating whether $\Pi(x) \neq \emptyset$. Note that the existence of a nondeterministic machine M that decides $L(\Pi)$ is equivalent to the existence of nondeterministic machines M_+ and M_- of the same complexity that accept $L(\Pi)$ and the complement of $L(\Pi)$, respectively. We say that M *computes* Π if it decides Π and on each accepting computation path on an input x with $\Pi(x) \neq \emptyset$ also outputs some $y \in \Pi(x)$ (which can depend on the path).

Within this framework we formalize the notion of isolation and distinguish between two types.

Definition 2.4 (Notions of isolation)**.** *An isolation for a computational problem $\Pi : X \mapsto 2^Y$ is a mapping reduction f that transforms $x \in X$ into an equivalent instance $f(x) \in X$ with $|\Pi(f(x))| \leq 1$. A disambiguation is an isolation where "equivalence" means that $\Pi(x)$ is empty if and only if $\Pi(f(x))$ is. A pruning is a disambiguation where "equivalence" additionally requires that $\Pi(f(x)) \subseteq \Pi(x)$.*

Disambiguations are isolations geared towards decision problems. Prunings are isolations geared towards search problems. Actually, for search problems it suffices to have an intermediate notion, namely a recoverable disambiguation f, i.e, one for which there exists an efficient transformation f' that takes any solution $y \in \Pi(f(x))$ and turns it into a solution $f'(x, f(x), y) \in \Pi(x)$.

A closely related notion in the machine realm is that of unambiguity. A nondeterministic machine M is called *unambiguous* on an input x if it has at most one accepting computation path on input x. The machine is called unambiguous if it is unambiguous on every input x.

A common way to achieve isolation is by introducing a weight function $\omega : X \times Y \mapsto \mathbb{N}$ and restricting the set of solutions to those of minimum weight, in the hope that there is unique solution of minimum weight (or none in the case where there are no solutions). We use the following terminology.

Definition 2.5 (Min-isolation)**.** *Given $\omega : X \times Y \mapsto \mathbb{N}$, the min-weight of $x \in X$ is defined as*

$$\omega(x) = \begin{cases} \min_{y \in \Pi(x)}(\omega(x,y)) & \text{if } \Pi(x) \neq \emptyset \\ \infty & \text{otherwise.} \end{cases}$$

We call ω min-isolating for x if there is at most one $y \in \Pi(x)$ with $\omega(x,y) = \omega(x)$.

In order to construct an actual isolation for Π, we need to express the restricted search on input x for a solution of weight $\mu = \omega(x)$ as an instance $f(x)$ of Π.

In many cases a suitable min-isolating weight function can be obtained as follows: View the solutions y for a given instance x as subsets of a finite universe $U = U(x)$, assign small weights $w(u) \in \mathbb{N}$ to the elements $u \in U$, and define $\omega(x,y)$ as a linear combination of the weights $w(u)$ of the elements $u \in y$. In fact, the trivial linear combination (all coefficients 1) often suffices. If the linear combination is clear from context, we often abuse notation and use w in lieu of ω, e.g., writing $w(y)$ for $\omega(x,y)$, or $w(x)$ for $\omega(x)$, or applying the term "min-isolating" to w.

The known generic isolation procedures [26, 70, 93] are all *randomized*. A randomized isolation with success probability p is a randomized mapping reduction f that, on every instance $x \in X$, satisfies the defining requirements for an isolation on input x with probability at least p. In the min-isolation approach via a weight assignment to the underlying universe, randomness comes into play in the construction of the weight assignment. The following well-known mathematical fact (rephrased using our terminology) forms the basis.

Fact 2.1 (Isolation Lemma [70])**.** *Suppose that $\Pi(x) \subseteq 2^U$ and that $\omega(x,y) = \sum_{u \in y} w(u)$ for $y \in \Pi(x)$. For any positive integer q, if $w : U \mapsto [q \cdot |U|]$ is*

picked uniformly at random then ω is min-isolating for x with probability at least $1 - 1/q$.

An important feature of the Isolation Lemma is that it keeps the range of the min-weight small, namely within $[c \cdot |U|^2]$. Once we have a min-isolating weight assignment of small range, we can further pick an integer μ uniformly at random within that range, and look for a solution $y \in \Pi(x)$ with $\omega(x, y) = \mu$. If μ happens to be equal to $\omega(x)$, then there is a unique such y. The small range of the min-weight guarantees a reasonable probability of success p.

We can apply this process to SATISFIABILITY with U denoting the set of variables of the formula x, and $q = 2$, say. The probability of success is $\Omega(1/n^2)$, where n denotes the number of variables of x. Since the weight restriction can be translated in polynomial time into a Boolean formula on the variables of the original formula, the resulting randomized isolation can be computed in polynomial time and is of the pruning type. The former implies that $\mathrm{NP} \subseteq \mathrm{R} \cdot \mathrm{PromiseUP}$ [93].[5] Intuitively, the result means that, in the randomized time-bounded setting, having unique solutions does not make instances of NP-complete problems easier. Formally, R denotes the one-sided error (no false positives) probabilistic operator on classes $\mathcal{C}$ of languages: $\mathrm{R} \cdot \mathcal{C}$ is the class of languages L for which there exists a constant $c \in \mathbb{N}$ and a language $C \in \mathcal{C}$ such that for all inputs x:

$$\begin{aligned} x \in L &\Rightarrow \Pr_\rho[\langle x, \rho \rangle \in C] \geq 1/n^c \\ x \notin L &\Rightarrow \Pr_\rho[\langle x, \rho \rangle \in C] = 0, \end{aligned}$$

where ρ is picked uniformly at random from $\{0,1\}^{n^c}$, and n denotes the input length $|x|$. The operator extends to classes of promise problems in a natural way. $\mathrm{PromiseUP}$ represents the class of promise decision problems that can be accepted by nondeterministic polynomial-time machines that are unambiguous on every input satisfying the promise.

[5] The original argument in [93] uses a different randomized isolation for SATISFIABILITY; it has a success probability of $\Omega(1/n)$.

2.2.2 Reachability

Gal and Wigderson [40] obtained a randomized isolation for REACHABILITY by applying the Isolation Lemma in a similar fashion with the edges for the graph G as the universe U. Since the weighted reachability problem with polynomially bounded weights is also in NL, one can translate the weight restricted instance into an equivalent instance in logarithmic space, though on a graph with more vertices. This results in a randomized disambiguation with success probability $1/\operatorname{poly}(n)$ that is computable in logarithmic space with two-way access to the random bits. (The disambiguation is recoverable in deterministic logspace, but is not a pruning.) It follows that $\mathrm{NL} \subseteq \mathrm{R} \cdot \mathrm{PromiseUL}$, where $\mathrm{PromiseUL}$ is the logspace equivalent of $\mathrm{PromiseUP}$. Thus, in the randomized space-bounded setting, having unique solutions does not make instances of NL-complete problems easier.

Reinhardt and Allender [76] strengthened this result to $\mathrm{NL} \subseteq \mathrm{R} \cdot (\mathrm{UL} \cap \mathrm{coUL})$. The class UL consists of the problems in $\mathrm{PromiseUL}$ for which the promise holds for all inputs. In other words, UL is the class of languages accepted by unambiguous logspace machines. The significance of the strengthening is that within the class $\mathrm{R} \cdot (\mathrm{UL} \cap \mathrm{coUL})$ the probability of error can be reduced to exponentially small levels, allowing the randomness to be replaced by polynomial advice, i.e., $\mathrm{R} \cdot (\mathrm{UL} \cap \mathrm{coUL}) \subseteq (\mathrm{UL} \cap \mathrm{coUL})/\operatorname{poly}$. It follows that REACHABILITY has a randomized disambiguation with exponentially small error that is computable in logspace with two-way access to the random bits, as well as a disambiguation that is computable in logspace with polynomial advice.

The construction in [76] needs a stronger property of the weight assignment w than being min-isolating for the given input (G, s, t). It requires w to be min-isolating for (G, s, t) for *all* choices of vertices s and t. In that case we call w min-isolating for G. By setting $q = 2n^2$ in the Isolation Lemma, a union bound guarantees that with probability at least 50%, a random weight assignment $w : E \mapsto [2n^2m]$ is min-isolating for any given graph $G = (V, E)$ with n vertices and m edges. The randomness in $\mathrm{NL} \subseteq \mathrm{R} \cdot (\mathrm{UL} \cap \mathrm{coUL})$ is only used

to generate random weight assignments. The new ingredients in [76] that enable the strengthening from $\mathrm{R} \cdot \mathrm{PromiseUL}$ to $\mathrm{R} \cdot (\mathrm{UL} \cap \mathrm{coUL})$ are unambiguous logspace machines that (i) decide whether or not a given weight assignment is min-isolating for a given graph G, and (ii) compute the min-weight $w(G, s, t)$ under a given min-isolating weight assignment w.

2.2.3 Circuit Certification

Gal and Wigderson [40] applied their approach for isolating REACHABILITY also to the following computational problem. Recall that a *certificate* for a gate g in a Boolean circuit C on an input z is a minimal[6] subcircuit F of C with output gate g that accepts z, written $F(z) = 1$.

Definition 2.6 (Circuit certification)**.** CIRCUIT CERTIFICATION *is the computational problem that maps an input $x \doteq (C, z, g)$ composed of a Boolean circuit C, an input z for C, and a gate g of C, to the set of certificates for g in C on input z.*

Based on De Morgan's laws, one can always push the negations in a circuit to the inputs without changing the input/output behavior or the depth of the circuit, while at most doubling its size. On any given input z, there is a simple bijection between the certificates for the transformed circuit and for the original one. Thus, it suffices to consider circuits where negations appear on the inputs only. In such a circuit C on input z, a certificate for a gate g satisfying $g(z) = 1$ can be constructed in the following recursive fashion, starting from the subcircuit of C rooted at g: If g is an AND gate, keep each incoming wire but replace its originating gate by a certificate for that gate. If g is an OR gate, keep a single incoming wire from a gate v satisfying $v(z) = 1$, and replace v by a certificate for v. If g is a leaf (necessarily evaluating to 1), keep it.

[40] assigns random weights w to the wires E of C. In order to facilitate the translation of the search for a certificate F for $x \doteq (C, z, g)$ of a given weight τ

[6] The restriction of minimality is imposed in some references (e.g., [76]) but not in others (e.g., [40]). We impose it as it allows for a bijection between certificates and accepting computation paths in the machine characterization of LogCFL.

into an equivalent instance $f(x)$ of CIRCUIT CERTIFICATION, the certificate is conceptually first expanded into an equivalent formula in the standard way by duplicating gates, wires, and their weights. The weight of the certificate F is then defined as the weight of this formula seen as a weighted tree. Equivalently, along the lines of the above process for constructing a certificate, the weight of a certificate F for g can be defined recursively as the sum of the weights of the wires feeding into g and the weights of the certificates that F induces for their originating gates. Thus, the weight of a certificate is not merely the sum of the weights of the edges in the certificate, but a linear combination of those weights with nonnegative integer coefficients. The Isolation Lemma can be extended to this setting, namely to families of multisets over the universe E, and guarantees with probability at least $1 - 1/q$ that g has a unique certificate of minimum weight when $w : E \mapsto [q \cdot |E|]$ is chosen uniformly at random. The number of times a wire can appear in the multiset (the coefficient in the linear combination) can be as large as the maximum product of the fan-ins of the AND gates on a path in C from the inputs to g. As a consequence, only circuits of low depth in which the fan-in of the AND gates is small can be handled efficiently. More specifically, [40] considers *shallow semi-unbounded circuits*. "Shallow" means that the depth is bounded by $\log_2(n)$, where n denotes the number of gates. "Semi-unbounded" means that the fan-in of the AND gates is bounded by two (and that negations appear on the inputs only).

Shallow semi-unbounded circuits are intimately connected to the complexity class LogCFL of languages that reduce to a context-free language under logspace mapping reductions. The class can be defined equivalently as the languages accepted by logspace-uniform families of shallow semi-unbounded circuits of polynomial size, the non-uniform version of which is denoted as SAC^1 [94]. The class LogCFL can also be characterized as the languages accepted by nondeterministic machines that run in polynomial time and logarithmic space, and are equipped with an auxiliary stack that does not count towards the space bound [86]. Such machines are sometimes called auxiliary pushdown automata, and the class

of languages accepted by such machines running in time t and space s is denoted as $\mathrm{AuxPDA\text{-}TISP}(t, s)$. The corresponding subclass for unambiguous machines is written as $\mathrm{UAuxPDA\text{-}TISP}(t, s)$. For any given problem in LogCFL and any input x, there is a logspace computable and logspace invertible bijection between the certificates for the circuits underlying the logspace-uniform SAC^1 characterization, and the accepting computation paths of the machine underlying the $\mathrm{AuxPDA\text{-}TISP}(\mathrm{poly}(n), O(\log n))$ characterization. It follows that the restriction of CIRCUIT CERTIFICATION to shallow semi-unbounded circuits is complete for LogCFL under logspace mapping reductions, and that logspace computable and recoverable disambiguations for that problem and for the entire class are equivalent.

Using the Isolation Lemma, Gal and Wigderson obtained a randomized disambiguation for CIRCUIT CERTIFICATION on shallow semi-bounded circuits that has success probability $1/\mathrm{poly}(n)$, is computable in logspace with two-way access to the random bits, and is recoverable in logspace. This implies the inclusion $\mathrm{LogCFL} \subseteq \mathrm{R} \cdot \mathrm{Promise}\mathcal{C}$ where we use the shorthand $\mathcal{C}$ for the class $\mathrm{UAuxPDA\text{-}TISP}(\mathrm{poly}(n), O(\log n))$. Reinhardt and Allender [76] strengthened this result to $\mathrm{LogCFL} \subseteq \mathrm{R} \cdot (\mathcal{C} \cap \mathrm{co}\mathcal{C})$, replacing the condition on the weight assignment w by the requirement that w is min-isolating for *every* gate of C on input z (not just the specified gate g). This implies that $\mathrm{LogCFL} \subseteq (\mathcal{C} \cap \mathrm{co}\mathcal{C})/\mathrm{poly}$ and that a disambiguation for CIRCUIT CERTIFICATION on shallow semi-unbounded circuits can be computed in logspace with polynomial advice.

2.2.4 Derandomizing Isolation

The number of random bits needed for an application of the Isolation Lemma as stated is $\Theta(n \log(qn))$, namely $\Theta(\log(qn))$ bits for each of the $n \doteq |U|$ elements of the universe U. In order to develop variants that require fewer random bits, we introduce the notion of a *weight assignment generator*, which can be viewed as a structured form of a pseudorandom generator geared towards the setting of the Isolation Lemma. Whereas a pseudorandom generator is parameterized by the

desired length of the pseudorandom sequence, a weight assignment generator is parameterized by the desired domain D of the weight assignments.

Definition 2.7 (Weight assignment generator)**.** *A weight assignment generator Γ for a family of domains $\mathcal{D}$ is a family of mappings $(\Gamma_D)_{D\in\mathcal{D}}$ such that Γ_D takes a string $\sigma \in \{0,1\}^{s(D)}$ for some function $s : \mathcal{D} \mapsto \mathbb{N}$, and maps it to a weight assignment $w : D \mapsto \mathbb{N}$. We say that w is chosen uniformly at random from Γ_D if it is obtained as $w = \Gamma_D(\sigma)$ where σ is chosen uniformly at random from $\{0,1\}^{s(D)}$.*

The family of domains $\mathcal{D}$ in Definition 2.7 is usually indexed by one or more integer parameters, in which case we also index Γ that way. For example, for a derandomization of the Isolation Lemma we can equate the universe U with $[|U|] \doteq \{1, 2, \ldots, |U|\}$ by ordering the elements of U in some way, e.g., lexicographically. We can then choose $\mathcal{D} = (D_n)_{n\in\mathbb{N}}$ with $D_n \doteq [n]$, and write Γ_n for Γ_{D_n}. This vanilla set-up does not allow the weight assignments to distinguish between the elements of the universe U (other than by their ordering). Definition 2.7 enables us to provide the weight assignments with more information about their arguments, namely by switching to more structured domains. For example, if the elements of the universe can be colored red or blue, we can provide that information to the weight assignments by using the domain $D = [n] \times \{\text{red}, \text{blue}\}$ for a universe of n elements. As another example, in the context of REACHABILITY we will use the domain $D_{n,d} = [n] \times [\![d]\!] \doteq \{1, 2, \ldots, n\} \times \{0, 1, 2, \ldots, d\}$ to handle layered graphs with n vertices and depth d; here the second component enables the weight assignment to take into account the layer of the vertex.

The relevant characteristics of a weight assignment generator are the following:

- The seed length $s(D)$, which is the number of random bits we need when we pick a weight assignment from Γ_D uniformly at random.

- The maximum weight assigned by Γ_D, the logarithm of the maximum weight is called the *bitlength of the generator*. A bound on the weights is sometimes also used as a parameter indexing the generator (in addition to the domain D).

- The computational complexity of Γ, by which we mean the complexity of the deciding, on input the parameters p, $\sigma \in \{0,1\}^s$, $z \in D$, $i \in \mathbb{N}$, and $b \in \{0,1\}$, whether the ith bit of $w(z)$ for $w = \Gamma_p(\sigma)$ is b.

The Isolation Lemma can be viewed as a generic weight assignment generator (for the family of domains $([n])_{n\in\mathbb{N}}$) that has seed length $O(n \log(qn))$, bitlength $O(\log n)$, and trivial complexity. By allowing weights that are polynomially larger than in the Isolation Lemma, one can achieve a seed length of $O(\log(qn) + \log(|\Pi(x)|))$ bits, which is provably optimal for a generic $\Pi(x)$ [26]. In our setting this yields seed length $O(n)$ and bitlength $O(\log n)$. In order to do better, one needs to exploit the specifics of the set systems. Doing so generically in the time-bounded setting seems difficult. There are implications from derandomizing the Isolation Lemma for generic $\Pi(x)$ of small circuit complexity to circuit lower bounds of various sorts [11], and vice versa [61]. The circuit lower bounds are arguably reasonable but have been open for a long time. There may be ways to obtain deterministic or derandomized isolations other than by derandomizing the Isolation Lemma, but for SATISFIABILITY the existence of a deterministic polynomial-time pruning implies that $\mathrm{NP} \subseteq \mathrm{P/poly}$. In fact, the collapse follows from the existence of a randomized polynomial-time pruning that has success probability $p > 2/3$ [34].

In the space-bounded setting there is more hope to obtain unconditional derandomizations. An implication from lower bounds to derandomization still holds: If there exists a problem in $\mathrm{DSPACE}(n)$ that requires Boolean circuits of linear-exponential size, then there exists a logspace computable weight assignment generator with seed length and bitlength $O(\log n)$ [7, 61]. There is no known result showing that deterministic isolations in the space-bounded setting imply circuit (or branching program) lower bounds that are open. Moreover,

unconditional results already exist for certain restricted classes of digraphs. For REACHABILITY on directed planar grid graphs, min-isolating weight assignments of bitlength $O(\log n)$ are known to be computable in deterministic logspace [22]. Those assignments have been used to construct disambiguations that are logspace computable and logspace recoverable for larger classes of graphs [22, 64, 89].

There have also been related successes for isolating PERFECTMATCHING, the problem of deciding/finding perfect matchings in graphs, restricted to certain special types of graphs [10, 30, 31]. Fenner, Gurjar, and Thierauf [37] construct a weight assignment generator with seed length and bitlength $O((\log n)^2)$ that is computable in logspace and that produces a min-isolating weight assignment for PERFECTMATCHING on a given bipartite graph with probability at least $1 - \log(n)/n$. This allowed them to prove that PERFECTMATCHING on bipartite graphs has logspace-uniform circuits of polylogarithmic depth and quasi-polynomial size.

2.3 Reachability and NL

In this section we develop our min-isolating weight assignment generator for REACHABILITY (Theorem 2.1), and derive our positive isolation result for NL (Theorem 2.2).

2.3.1 Weight Assignment Generator

Recall the notion of min-isolation in the context of REACHABILITY:

Definition 2.8 (Min-isolating weight assignment for REACHABILITY)**.** *Let $G = (V, E)$ be a digraph. A weight assignment for G is a mapping $w : V \mapsto \mathbb{N}$. The weight $w(P)$ of a path P in G is the sum of $w(v)$ over all vertices v on the path. For $s, t \in V$, $w(G, s, t)$ denotes the minimum of $w(P)$ over all paths from s to t, or ∞ if no such path exists. The weight assignment w is min-isolating for (G, s, t) if there is at most one path P from s to t with $w(P) = w(G, s, t)$. For $A \subseteq V \times V$,*

w is min-isolating for (G, A) if w is min-isolating for (G, s, t) for each $(s, t) \in A$. We call w min-isolating for G if w is min-isolating for $(G, V \times V)$.

We restrict attention to *layered* digraphs. A layered digraph $G = (V, E)$ of depth d consists of $d+1$ layers of vertices such that edges only go from one layer to the next. More formally, with $n \doteq |V|$ we have that $V \subseteq [n] \times [\![d]\!] \doteq \{1, 2, \ldots, n\} \times \{0, 1, 2, \ldots, d\}$ and $E \subseteq \dot{\bigcup}_{i \in [d]}(V_{i-1} \times V_i)$. We denote by $V_i \doteq V \cap [n] \times \{i\}$ the ith layer of G.

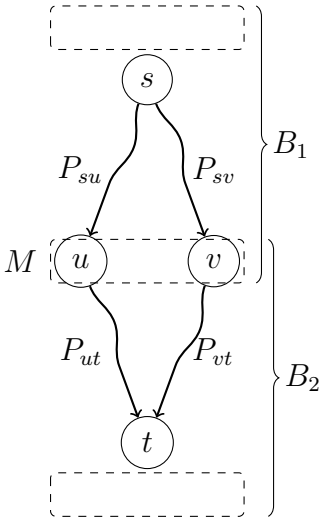

Figure 2.1:

In fact, we only need to consider layered digraphs of depths that are powers of two. For $d = 2^\ell$ with $\ell \in \mathbb{N}$, and $k \in [\![\ell]\!]$, such a digraph can be viewed as consisting of $d/2^k = 2^{\ell-k}$ consecutive blocks of depth 2^k, where the ith block is the subgraph induced by the vertices in layers $(i-1)2^k$ through $i2^k$, i.e., $\bigcup_{j=(i-1)\cdot 2^k}^{i \cdot 2^k} V_j$.

We need to design a randomness efficient process that, given $d = 2^\ell$ and n, generates small weight assignments $w : [n] \times [\![d]\!] \mapsto \mathbb{N}$ that are min-isolating for any layered digraph $G = (V, E)$ of depth d on n vertices with high probability. Note that the use of the domain $[n] \times [\![d]\!]$ rather than merely $[n]$ enables the weight assignment to depend on the layer a vertex is in.

Iterative Approach Given the recursive nesting structure of the blocks, there is a natural iterative/recursive approach towards the construction of w, based on the following simple observation:

> A min-weight path from s to t that passes through a vertex u is the concatenation of a min-weight path from s to u and a min-weight path from u to t.

We present an iterative (i.e., bottom-up) version, where in the kth iteration we try to construct a weight assignment w_k that is min-isolating for each block of depth 2^k and only assigns nonzero weights to the vertices that are internal to those blocks, i.e., to $V \setminus \bigcup_{i=0}^{2^{\ell-k}} V_{i \cdot 2^k}$.

We start with $w_0 \equiv 0$, and end with $w = w_\ell$. Here is how we move from w_k to w_{k+1} in iteration $k+1$ for $k \in [\![\ell - 1]\!]$. Consider a block B of depth 2^{k+1}. It consists of two consecutive blocks B_1 and B_2 of depth 2^k that have the middle layer M of B in common (see Figure 2.1). The assignment w_k gives weights to all vertices of B except the initial layer, the middle layer M, and the final layer. We construct the assignment w_{k+1} by extending w_k, i.e., w_{k+1} keeps the values of w_k on the layers internal to B_1 or B_2, and additionally assigns weights to the vertices in M. We refer to the union of the middle layers M over all blocks of depth 2^{k+1} as the set L_{k+1} of vertices at level $k+1$, i.e.,

$$L_{k+1} \doteq \bigcup_{\text{odd } i \in [2^{\ell-k}]} V_{i \cdot 2^k}. \tag{2.2}$$

The new weights are assigned so as to maintain the invariant – Assuming that w_k is min-isolating for B_1 and B_2 individually, we want to make sure that w_{k+1} is min-isolating for all of B. Consider two vertices s and t in B such that s appears in an earlier layer than t.

- If t is internal to B_1 then w_{k+1} is min-isolating for (B, s, t) no matter how w_{k+1} assigns weights to M. This follows from the hypothesis and the fact that w_{k+1} and w_k agree on the vertices of B_1 other than M. The case where s is internal to B_2 is similar.

- Otherwise, s belongs to B_1 and t belongs to B_2. In that case every path from s to t has to cross layer M. We claim that among the paths (if any) that cross M in a fixed vertex v, there is a unique one of minimum weight with respect to w_{k+1}, say P_v. This follows from the above observation, the hypothesis, and the fact that w_{k+1} and w_k agree on the vertices of

B other than M. Indeed, any such path P_v is the concatenation of a path P_{sv} in B_1 from s to v, and a path P_{vt} in B_2 from v to t. Since $w_{k+1}(P_v) = w_k(P_v) + w_{k+1}(v) = w_k(P_{sv}) + w_k(P_{vt}) + w_{k+1}(v)$, both P_{sv} and P_{vt} need to be min-weight with respect to w_k. By hypothesis, both those min-weight paths are uniquely determined, whence so is P_v.

Thus, in order to guarantee that w_{k+1} is min-isolating for (B, s, t), it suffices to ensure that for all vertices $u, v \in M$ that are on a path from s to t,

$$\mu_k(s,u) + \mu_k(u,t) + w_{k+1}(u) \neq \mu_k(s,v) + \mu_k(v,t) + w_{k+1}(v), \quad (2.3)$$

where $\mu_k(s,t) \doteq w_k(G,s,t)$ denotes the minimum weight of a path from s to t under w_k, or ∞ if no such path exists. We refer to condition (2.3) as a *disambiguation requirement*. See Figure 2.1 for an illustration.

We now consider three ways to meet the disambiguation requirements: shifting, hashing, and a combination of both. For each construction we track:

- the number R_k of random bits that w_k needs, and
- the maximum weight W_k of paths in G under w_k.

The quantity $R \doteq R_\ell$ corresponds to the seed length of the weight assignment generator Γ. The logarithm of the quantity $W \doteq W_\ell$ equals the bitlength of Γ up to an additive term of $O(\log d)$. As we will see in Section 2.3.2, the simulations of NL on unambiguous machines that we obtain via Γ run in space $O(R + \log(W) + \log(n))$. Thus, our aim is to minimize the quantity $R + \log(W)$ up to constant factors. We will ultimately succeed in making it as small as $O((\log n)^{3/2})$. Ideally, we would like to reduce it further to $O(\log n)$ so as to establish $\mathrm{NL} \subseteq \mathrm{UL}$.

Shifting For $v \in M \subseteq L_{k+1}$ we set $w_{k+1}(v) = \mathrm{index}(v) \cdot b$, where b is an integer that exceeds W_k, and index is an injective function from M to $\mathbb{N}$. As the vertices in V are represented as pairs $(i,j) \in [\![d]\!] \times [n]$ and all vertices in M have

the same first component, we can simply use the projection $(i, j) \mapsto j$ as the index function. This guarantees distinct values for the two sides of (2.3) for different u and v, irrespective of the values of $\mu_k(s, u) + \mu_k(u, t)$ and $\mu_k(s, v) + \mu_k(v, t)$. In terms of binary representations, if b is a power of 2, this construction can be interpreted as shifting the index function into a region of the binary representation that has not been used before.

We have that $R_{k+1} = R_k$ and $W_{k+1} \leq W_k + 2^{\ell-k-1} \cdot n \cdot b \leq (dn + 1)(W_k + 1) - 1$. When we use shifting at all levels, we end up with $R = 0$ and $W \leq (dn + 1)^\ell = n^{O(\log n)}$, so $R + \log(W) = O((\log n)^2)$.

Hashing When $w_{k+1}(u)$ and $w_{k+1}(v)$ are picked uniformly at random from a sufficiently large range, independently from each other and from the values $\mu_k(s, u) + \mu_k(u, t)$ and $\mu_k(s, v) + \mu_k(v, t)$, the disambiguation requirement (2.3) holds with high probability. We make use of *universal hashing* to obtain the random values we need using few random bits, and in particular of the following well-known family and property. We cast the notion in terms of a weight assignment generator with a bound on the weights as an additional parameter.

Fact 2.2 (Universal hashing [25])**.** *There exists a logspace computable weight assignment generator* $(\Gamma^{(\mathrm{hashing})}_{m,r})_{m,r\in\mathbb{N}}$ *with seed length* $s(m, r) = O(\log(mr))$ *such that* $\Gamma^{(\mathrm{hashing})}_{m,r}$ *produces functions* $h : [m] \mapsto [r]$ *with the following property: For every* $u, v \in [m]$ *with* $u \neq v$*, and every* $a, b \in \mathbb{N}$

$$\Pr_h[a + h(u) = b + h(v)] \leq 1/r, \tag{2.4}$$

where h *is chosen uniformly at random from* $\Gamma^{(\mathrm{hashing})}_{m,r}$.

We identify $D \doteq [\![d]\!] \times [n]$ with $[m] = [(d + 1) \cdot n]$ in a natural way. If we pick $h : D \mapsto [r]$ uniformly at random from $\Gamma^{(\mathrm{hashing})}_{m,r}$ and set $w_{k+1} = h$ on L_{k+1}, (2.4) guarantees that each individual disambiguation requirement (2.3) holds with probability at least $1 - 1/r$. As there are at most n^4 choices for (s, t, u, v), a union bound shows that all disambiguation conditions are met simultaneously

with probability at least $1 - n^4/r$. It suffices to pick r as a sufficiently large polynomial in n in order to guarantee high success probability. In particular, $r = n^6$ suffices for probability of success at least $1 - 1/n^2$.

Based on the characteristics of the family of hash functions $\Gamma^{(\text{hashing})}$ from Fact 2.2 we have that $R_{k+1} = R_k + O(\log(dnr)) = R_k + O(\log n)$ and $W_{k+1} \leq W_k + 2^{\ell-k-1} \cdot r \leq W_k + dr = W_k + n^{O(1)}$. When we use a fresh uniform sample $h = h_k$ from $\Gamma^{(\text{hashing})}$ for each iteration $k \in [\ell]$, we end up with $R = O(\ell \log(n)) = O((\log n)^2)$, and $W = \ell \cdot n^{O(1)} = n^{O(1)}$, so $R + \log(W) = O((\log n)^2)$ again.

Combined Approach The shifting approach is ideal in terms of the amount of randomness R but leads to weights that are too large. The hashing approach is ideal in terms of the bound W on the path weights but requires too many random bits. We now combine the two approaches so as to balance R and $\log(W)$. The construction can be viewed as incorporating shifting into the hashing approach, or vice versa. Our presentation follows the former perspective.

In order to reduce the number of random bits in the hashing approach, we attempt to employ the same hash function h in multiple successive iterations, say iterations $k + 1$ through k', going from w_k to $w_{k'}$. This does not work as such because the minimum path weights in the disambiguation requirements (2.3) for iterations above $k + 1$ depend on h, and we cannot guarantee the bound (2.4) if a or b depend on h. However, the influence of the choice of h on those minimum path weights is limited. More specifically, in iteration $k + 2$ we have that for any s and t that belong to the same block of depth 2^{k+1}

$$\mu_k(s, t) \leq \mu_{k+1}(s, t) \leq \mu_k(s, t) + r. \tag{2.5}$$

The first inequality follows because $w_{k+1} \geq w_k$. The second one follows by considering a minimum-weight path P from s to t under w_k and realizing that

$$\mu_{k+1}(s, t) \leq w_{k+1}(P) = w_k(P) + h(v) = \mu_k(s, t) + h(v) \leq \mu_k(s, t) + r,$$

where v is the unique vertex in $P \cap L_{k+1}$.

Let b be a power of two to be determined later. Equation (2.5) implies that $\mu_k(s,t)$ and $\mu_{k+1}(s,t)$ are the same after truncating the $\log b$ lowest-order bits, i.e., $\lfloor \mu_k(s,t)/b \rfloor = \lfloor \mu_{k+1}(s,t)/b \rfloor$, unless adding r to $\mu_k(s,t)$ results in a carry into bit position $\log b$ (the position corresponding to the power $2^{\log b} = b$). Suppose we can prevent such carries from happening. Conceptually, in iteration $k+2$ we can then apply the hashing approach with the *same* hash function h as in iteration $k+1$ provided we use the *truncated* values $\mu'_{k+1}(s,t) \doteq \lfloor \mu_k(s,t)/b \rfloor = \lfloor \mu_{k+1}(s,t)/b \rfloor$ as the minimum path weights. Indeed, since the values μ'_{k+1} are independent of h, (2.4) in Fact 2.2 shows that the disambiguations requirements with respect to μ'_{k+1}, i.e.,

$$\mu'_{k+1}(s,u) + \mu'_{k+1}(u,t) + h(u) \neq \mu'_{k+1}(s,v) + \mu'_{k+1}(v,t) + h(v), \qquad (2.6)$$

hold with high probability. Undoing the truncation, (2.6) implies that

$$\mu_{k+1}(s,u) + \mu_{k+1}(u,t) + h(u) \cdot b \neq \mu_{k+1}(s,v) + \mu_{k+1}(v,t) + h(v) \cdot b.$$

Thus, by setting $w_{k+2}(v) = h(v) \cdot b$ for $v \in L_{k+2}$ we realize the actual disambiguation requirements for iteration $k+2$ with high probability *in conjunction* with the disambiguation requirements for iteration $k+1$. The setting of w_{k+2} on L_{k+2} can be interpreted as using a shifted version of the same hash function h instead of h itself.

We can repeat the process for iterations $k+3$ through k'. In iteration $k+i$, the bound (2.5) becomes

$$\begin{aligned} \mu_k(s,t) \leq \mu_{k+i-1}(s,t) \leq \mu_k(s,t) + r \cdot (b^{i-2} + 2b^{i-3} + \ldots + 2^{i-2}) \\ \leq \mu_k(s,t) + 2rb^{i-2}, \end{aligned}$$

where the last inequality assumes that $b \geq 4$. We set $w_{k+i}(v) = h(v) \cdot b^{i-1}$ for

$v \in L_{k+i}$, and achieve our goal if h satisfies the disambiguation requirements

$$\lfloor \mu_k(s,u)/b^{i-1} \rfloor + \lfloor \mu_k(u,t)/b^{i-1} \rfloor + h(u) \neq \lfloor \mu_k(s,v)/b^{i-1} \rfloor + \lfloor \mu_k(v,t)/b^{i-1} \rfloor + h(v) \quad (2.7)$$

for all appropriate choices of s, t, u, v. Equation (2.4) in Fact 2.2 and a union bound show that the requirements (2.7) are all met simultaneously by the same hash function h for all iterations $k+1$ through k' with probability at least $1-\Delta/n^2$ for $r = n^6$, where $\Delta \doteq k' - k$.

In iteration $k+2$ we made the assumption that there are no carries into position $\log b$ when adding r to the values μ_k. More generally, in iteration $k+i$, we assumed there are no carries into position $(i-1) \cdot \log b$ when adding $2rb^{i-2}$ to the values μ_k. The assumption holds if $b \geq 4r$ and the values μ_k have a 0 in the position right before each of the positions $(i-1) \cdot \log b$. We can maintain the latter condition as an invariant throughout the construction by setting $b = O(r)$ sufficiently large.

This completes the combined construction and its correctness argument. For future reference, we mention the following alternate way of handling the carries in the correctness argument. Setting $b \geq 2r$ is enough to ensure that the carries are no larger than 1. We can handle such carries by strengthening the disambiguation requirements (2.7) and impose that the left-hand side and right-hand side are not just distinct but are separated by a small constant. This only involves a constant factor more of applications of (2.4) in the union bound, and guarantees that the values remain distinct after undoing the truncation. In fact, it suffices that for all $i \in [k' - k]$

$$\lfloor (\mu_k(s,u) + \mu_k(u,t))/b^{i-1} \rfloor + h(u) \notin \lfloor (\mu_k(s,v) + \mu_k(v,t))/b^{i-1} \rfloor + h(v) + \{-1, 0, 1\}$$

for some $b \geq 4r$. This is the approach we use in the formal proof of Lemma 2.14

in Section 2.4.1 (in the setting of CIRCUIT CERTIFICATION instead of REACHABILITY). We refer to the argument for Claim 2.15 on page 37 for more details.

The combined construction obeys: $R_{k'} = R_k + O(\log(dnr)) = R_k + O(\log n)$ and $W_{k'} \leq W_k + 2^{\ell-k'} \cdot 2rb^{\Delta-1} \leq W_k + drb^{\Delta-1} = W_k + O(n^{\Delta})$, where $\Delta \doteq k' - k$.

Final Construction Starting from $w_0 \equiv 0$, for any $\Delta \in [\ell]$ we can apply the combined construction ℓ/Δ times consecutively to obtain $w = w_\ell$. Each application uses a fresh hash function to bridge the next Δ levels. The setting $\Delta = 1$ corresponds to the pure hashing approach, and the setting $\Delta = \ell$ essentially corresponds to the pure shifting approach.[7] We can interpolate between the parameters of the pure shifting and pure hashing approaches by considering intermediate values of Δ. We have $R = O(\ell/\Delta \cdot \log n)$ and $W = O(\ell/\Delta \cdot n^{\Delta})$, so $R + \log(W) = O((\ell/\Delta + \Delta) \log n)$. The latter expression is minimized up to constant factors when $\ell/\Delta = \Delta$, i.e., when $\Delta = \sqrt{\ell}$, yielding a value of $R + \log(W) = O(\sqrt{\ell} \log n) = O(\sqrt{\log d} \log n) = O((\log n)^{3/2})$.

The above construction yields a weight assignment generator $\Gamma^{(\mathrm{reach})}$ that is indexed by the number of vertices n and the depth d, with $D_{n,d} \doteq [n] \times [\![d]\!] \doteq \{1, 2, \ldots, n\} \times \{0, 1, 2, \ldots, d\}$ as the domain for the weight functions given by $\Gamma^{(\mathrm{reach})}_{n,d}$. We conclude:

Theorem 2.9 (formal version of Theorem 2.1)**.** *There exists a weight assignment generator $\Gamma^{(\mathrm{reach})} = (\Gamma^{(\mathrm{reach})}_{n,d})_{n,d \in \mathbb{N}}$ that is computable in space $O(\log n)$ and has seed length and bitlength $O(\sqrt{\log d} \log n)$ such that for every layered digraph G of depth d with n vertices*

$$\Pr_w[w \textit{ is min-isolating for } G] \geq 1 - 1/n,$$

where w is chosen uniformly at random from $\Gamma^{(\mathrm{reach})}_{n,d}$.

[7] In this setting the hash function h from the "combined" approach can be replaced by an index function.

The construction we developed works for any d that is a power of 2 and any $n \in \mathbb{N}$, and has the properties stated in Theorem 2.9. We already analyzed the seed length and bitlength. For any given layered digraph of depth d on n vertices, the failure probability at each level of the construction is at most $1/n^2$. As there are $\ell \doteq \log d \leq \log n$ levels, the overall failure probability is at most $\log(n)/n^2 \leq 1/n$. The logspace computability follows from the logspace computability of the underlying universal family of hash functions and the fact that iterated addition is in logspace (see, e.g., [95]).

Values of d that are not powers of 2 can be handled by first extending the given layered digraph G with identity matchings (for each i connect the ith gate in the next layer with the ith gate in the previous layer) until the depth reaches a power of 2, and then applying the above construction.

This finishes a somewhat informal proof of Theorem 2.9. Section 2.4.1 contains a more formal proof (in the setting of CIRCUIT CERTIFICATION instead of REACHABILITY).

2.3.2 Isolation

We now establish Theorem 2.2. The following proposition[8] shows that it suffices to construct a Turing machine that accepts REACHABILITY unambiguously on layered digraphs in time $\mathrm{poly}(n)$ and space $O((\log n)^{3/2})$.

Proposition 2.10. REACHABILITY *on layered digraphs is hard for* NL *under logspace mapping reductions that preserve the number of solutions.*

Given our weight assignment generator $\Gamma^{(\mathrm{reach})}$, a natural approach towards computing REACHABILITY unambiguously on a given layered instance (G, s, t) is to go over the list of all weight assignments w produced by $\Gamma^{(\mathrm{reach})}$, pick the first one that is min-isolating for G, and use it to decide the given instance (G, s, t). In fact, the earlier improvement from REACHABILITY $\in \mathrm{R} \cdot \mathrm{PromiseUL}$ [40] to

[8] The property that the mapping reductions preserve the number of solutions is not needed here but will be used later.

REACHABILITY $\in \mathrm{R} \cdot (\mathrm{UL} \cap \mathrm{coUL})$ [76] can be viewed as following the same approach. Instead of the list of weight assignments obtained from $\Gamma^{(\mathrm{reach})}$ (which is guaranteed to contain a min-isolating one), [76] uses a list of $2n^2$ random weight assignments of bitlength $O(\log n)$ (which contains a min-isolating one with probability at least 50%). The following ingredients are essential to get the approach to work.

Lemma 2.11. *There exist unambiguous machines,* WEIGHTCHECK$^{(\mathrm{reach})}$ *and* WEIGHTEVAL$^{(\mathrm{reach})}$*, such that for every digraph $G = (V, E)$ on n vertices, weight assignment $w : V \mapsto \mathbb{N}$, and $s, t \in V$:*

(i) WEIGHTCHECK$^{(\mathrm{reach})}(G, w)$ *decides whether or not w is min-isolating for G, and*

(ii) WEIGHTEVAL$^{(\mathrm{reach})}(G, w, s, t)$ *computes $w(G, s, t)$ provided w is min-isolating for G.*

Both machines run in time $\mathrm{poly}(\log(W), n)$ *and space* $O(\log(W) + \log(n))$*, where W denotes an upper bound on the finite values of $w(G, u, v)$ for $u, v \in V$.*

Lemma 2.11 is an improvement of a result in [76]. It follows along the same lines but has a better dependency of the running time on W, namely polynomial in $\log(W)$ instead of polynomial in W. As our weight assignment generator yields values of $W = n^{\Theta(\sqrt{\log n})}$, the improvement is necessary to make sure that our unambiguous machine for REACHABILITY runs in polynomial time. Note that the machine WEIGHTEVAL$^{(\mathrm{reach})}$ does not simply go over all integers μ from 0 to W and check whether a path from s to t of weight μ exists (knowing that it is unique if it exists) until the first success or the weight range is exhausted. That process would take at least W steps in the worst case. We refer to Section 2.6 for the workings of the machines WEIGHTEVAL$^{(\mathrm{reach})}$ and WEIGHTCHECK$^{(\mathrm{reach})}$ and for further discussion.

Like [76], we call WEIGHTCHECK$^{(\mathrm{reach})}(G, w)$ for each w from the list up and until the first success, and then call WEIGHTEVAL$^{(\mathrm{reach})}(G, w, s, t)$ with that

first successful w. This describes a deterministic machine for REACHABILITY on layered digraphs that makes calls to the unambiguous nondeterministic machines WEIGHTCHECK$^{(\mathrm{reach})}$ and WEIGHTEVAL$^{(\mathrm{reach})}$. The result is an unambiguous nondeterministic machine assuming the following general convention regarding the behavior of a machine M making a call to a nondeterministic machine N: On any computation path on which N rejects, M halts and rejects; on any accepting computation path of N, M continues the path assuming the output of N as the result of the call.

In order to try all weight assignments produced by $\Gamma^{(\mathrm{reach})}$, we go over all seeds σ, and produce the required bits of $w = \Gamma^{(\mathrm{reach})}(\sigma)$ from σ on the fly whenever they are needed, without storing them. Given the logspace computability of $\Gamma^{(\mathrm{reach})}$, the resulting unambiguous machine for REACHABILITY on layered digraphs runs in time $2^R \cdot \mathrm{poly}(\log(W), n)$ and space $O(R + \log(W) + \log n)$, where R denotes the seed length of $\Gamma^{(\mathrm{reach})}$, and W the maximum path length under a weight assignment that $\Gamma^{(\mathrm{reach})}$ produces. With the parameters of $\Gamma^{(\mathrm{reach})}$ stated in Theorem 2.9 this gives time $n^{O(\sqrt{\log n})}$ and space $O((\log n)^{3/2})$.

In order to reduce the running time to $n^{O(1)}$ while keeping the space bound to $O((\log n)^{3/2})$, we improve over the exhaustive search over all seeds of $\Gamma^{(\mathrm{reach})}$ by exploiting the internal structure of $\Gamma^{(\mathrm{reach})}$. We use the same technique as in [72]. Recall from the final construction in Section 2.3.1 that the seed σ consists of $\Delta = O(\sqrt{\log n})$ parts of $O(\log n)$ bits, each describing a hash function h_i from the family $\Gamma^{(\mathrm{hashing})}$ from Fact 2.2. The hash functions $h_1, \ldots, h_i$ define a weight assignment $w_{i \cdot \Delta}$ that is intended to have the following property: $w_{i \cdot \Delta}$ is min-isolating for each block of depth $2^{i \cdot \Delta}$ of G. We construct (the seeds σ_i for) the hash functions h_i one by one, maintaining the intended property as an invariant for $i = 0, 1, \ldots, \Delta$. The invariant trivially holds for $i = 0$. In the step from $i - 1$ to i for $i \in [\Delta]$, we go over all possible seeds σ_i for $\Gamma^{(\mathrm{hashing})}_{m,r}$, consider $h_i \doteq \Gamma^{(\mathrm{hashing})}_{m,r}(\sigma_i)$, check whether or not the weight assignment $w_{i \cdot \Delta}$ defined by the already determined $h_1, \ldots, h_{i-1}$ and the current choice for h_i maintains the invariant, and select the first σ_i for which it does. Each check

is performed by running WEIGHTCHECK$^{(\mathrm{reach})}(B, w)$ for each of the blocks B of depth 2^i, passing if and only if all of them pass. The correctness argument from Section 2.3.1 guarantees that the search always succeeds. Once we arrive at $w = w_\Delta$, we run WEIGHTEVAL$^{(\mathrm{reach})}(G, w, s, t)$ as before. Note that the number of choices for σ_i that need to be examined for each $i \in [\Delta]$ is $n^{O(1)}$. It follows that the resulting machine runs in time $n^{O(1)}$ and space $O((\log n)^{3/2})$, and unambiguously decides REACHABILITY on layered digraphs.

This finishes the proof of Theorem 2.2. A more formal proof in the setting of CIRCUIT CERTIFICATION and LogCFL is given in Section 2.4.2.

2.4 Circuit Certification and LogCFL

In this section we state and formally prove our positive results for CIRCUIT CERTIFICATION and LogCFL. In the last part we explain how these results imply the results for REACHABILITY and NL from Section 2.3.

2.4.1 Weight Assignment Generator

Analogous to the setting of REACHABILITY and NL, our isolations for CIRCUIT CERTIFICATION and LogCFL hinge on an efficient min-isolating weight assignment generator. Although not essential, it is more convenient for us to assign weights to the *gates* rather than the wires.

Let us formally define what min-isolation means in the context of CIRCUIT CERTIFICATION. We view a Boolean circuit C as an acyclic digraph $C = (V, E)$, where V represents the gates of the circuit, and E the wires. Each leaf (vertex of indegree zero) is labeled with a literal (input variable or its negation) or a Boolean constant (0 or 1); each other vertex is labeled with AND or OR. We consider circuits with and without a single designated output gate.

Definition 2.12 (Min-isolating weight assignments for CIRCUIT CERTIFICATION)**.** *Let $C = (V, E)$ be a circuit. A weight assignment for C is a mapping*

$w : V \mapsto \mathbb{N}$. *The weight* $w(F)$ *of a certificate* F *with output* v *equals* $w(v)$ *plus the sum over all gates* u *that feed into* v *in* F*, of the weight of the certificate with output* u *induced by* F*. For an input* z *for* C*, and* $g \in V$*,* $w(C, z, g)$ *denotes the minimum of* $w(F)$ *over all certificates* F *for* (C, z, g)*, or* ∞ *if no certificate exists. The weight assignment* w *is min-isolating for* (C, z, g) *if there is at most one certificate* F *for* (C, z, g) *with* $w(F) = w(C, z, g)$*. For* $U \subseteq V$*,* w *is min-isolating for* (C, z, U) *if* w *is min-isolating for* (C, z, u) *for each* $u \in U$*. We call* w *min-isolating for* (C, z) *if* w *is min-isolating for* (C, z, V)*.*

Note that the weight $w(F)$ of a certificate F for a gate g is a linear combination of the weights $w(v)$ for $v \in V$ with coefficients that are natural numbers. The sum of the coefficients in any given layer below g is at most 2^ℓ, where ℓ denotes the number of AND layers between that layer and g (inclusive).

We restrict attention to semi-unbounded circuits that are *layered* and *alternating*. A circuit is layered if the underlying digraph is layered and all leaves appear in the same layer. A circuit is alternating if on every path the non-leaves alternate between AND and OR. More formally, for a circuit $C = (V, E)$ of depth d with n gates we have that $V = \dot{\bigcup}_{i \in [\![d]\!]} V_i$ where $V_i \subseteq [n] \times \{i\}$ and $E \subseteq \dot{\bigcup}_{i \in [d]} (V_{i-1} \times V_i)$. Vertices in V_0 are labeled with literals and constants only. Every other layer V_i contains only AND gates or only OR gates, depending on the parity of i.

With the above conventions we can view weight assignments to the gates as mappings $w : [n] \times [\![d]\!] \mapsto \mathbb{N}$. We construct such assignments inside the following weight assignment generator $\Gamma^{(\mathrm{cert})} = (\Gamma^{(\mathrm{cert})}_{n,d})_{n,d \in \mathbb{N}}$, which is indexed by the number of gates n and the depth d. The domain of the weight assignments given by $\Gamma^{(\mathrm{cert})}_{n,d}$ is $D_{n,d} \doteq [n] \times [\![d]\!]$, enabling the weight assignment of a gate to depend on the layer the gate belongs to.

Theorem 2.13. *There exists a weight assignment generator* $\Gamma^{(\mathrm{cert})} = (\Gamma^{(\mathrm{cert})}_{n,d})_{n,d \in \mathbb{N}}$ *that is computable in space* $O(\log n)$ *and has seed length and bitlength* $O(\sqrt{d} \log n)$ *such that for every layered alternating semi-unbounded*

Boolean circuit C of depth d with n gates and any input z for C,

$$\Pr_w[w \textit{ is min-isolating for } (C, z)] \geq 1 - 1/n, \tag{2.8}$$

where w is chosen uniformly at random from $\Gamma^{(\mathrm{cert})}_{n,d}$.

The essential ingredient in the proof of Theorem 2.13 is the following formalization of the combined approach from Section 2.3.1 for the setting of CIRCUIT CERTIFICATION. It turns a weight assignment that is min-isolating for all gates up to some layer into one that is min-isolating for all gates up to some higher layer, and only assigns new weights to the AND gates of the layers in between. For ease of notation, we assume that the depth is even (say $d = 2\ell$ for some $\ell \in \mathbb{N}$), that we jump from an even layer $2k$ to some higher even layer $2k'$, and that the layer V_1 next to the leaves consists of ANDs. Thus, odd layers consist of AND gates, and positive even layers of OR gates.

Lemma 2.14. *There exists a weight assignment generator $\Gamma^{(\mathrm{cert,step})} = (\Gamma^{(\mathrm{cert,step})}_{n,\ell,k,k'})$ for $n, \ell, k, k' \in \mathbb{N}$ with $k \leq k' \leq \ell$ and domain $D_{n,\ell,k,k'} \doteq [n] \times [\![2\ell]\!]$ that is computable in space $O(\log n)$, has seed length $O(\log n)$ and bitlength $O((k' - k) \log n)$, and has the following property for every layered alternating semi-unbounded Boolean circuit $C = (V, E)$ of depth $d \doteq 2\ell$ with n gates and layers $V_0, V_1, \ldots, V_d$ where layer V_1 consists of AND gates, and for every input z for C: If $w : V \mapsto \mathbb{N}$ is a weight assignment that is min-isolating for $(C, z, V_{\leq 2k})$, where $V_{\leq i} \doteq \bigcup_{j \leq i} V_j$, then*

$$\Pr_\sigma[w + \Gamma^{(\mathrm{cert,step})}_{n,\ell,k,k'}(\sigma) \textit{ is min-isolating for } (C, z, V_{\leq 2k'})] \geq 1 - 1/n^2,$$

where the seed σ is chosen uniformly at random. Moreover, $\Gamma^{(\mathrm{cert,step})}_{n,\ell,k,k'}(\sigma)$ assigns nonzero weights only to $\bigcup_{j \in [k+1,k']} L_j$, where $L_j \doteq V_{2j-1}$ denotes the jth AND layer.

Proof. Let C be a circuit as in the statement of the lemma, z an input for C, and $w : V \mapsto \mathbb{N}$ a weight assignment that is min-isolating for $(C, z, V_{\leq 2k})$.

Pick $h : D \mapsto [r]$ with $D = D_{n,\ell,k,k'} \doteq [n] \times [\![2\ell]\!]$ uniformly at random from $\Gamma^{(\text{hashing})}_{n(2\ell+1),r}$, identifying $[n(2\ell+1)]$ and $[n] \times [\![2\ell]\!]$ in a natural way. For a given h, we define a sequence of weight assignments $w_j : V \mapsto \mathbb{N}$ for $j = k, k+1, \ldots, k'$ as follows: $w_k = w$, and for $i \in [k'-k]$ and $g \in V$:

$$w_{k+i}(g) = \begin{cases} w_{k+i-1}(g) + h(g) \cdot b^{i-1} & \text{if } g \in L_{k+i} \\ w_{k+i-1}(g) & \text{otherwise,} \end{cases}$$

where b is a positive integer to be determined.

For $g \in V$, we denote by $\mu_j(g) \doteq w_j(C, z, g)$ the minimum weight of a certificate for (C, z, g) with respect to w_j, or ∞ if no certificate exists. We show that if b and r are sufficiently large polynomials in n, then with probability at least $1 - 1/n^2$ the following invariant holds for $i \in [\![k'-k]\!]$:

$$w_{k+i} \text{ is min-isolating for } (C, z, V_{\leq 2(k+i)}). \tag{2.9}$$

We make the following observations:

- By the hypothesis on w the invariant holds for $i = 0$.
- For $i \in [k'-k]$, the invariant for $i-1$ implies that w_{k+i} is min-isolating for $(C, z, V_{\leq 2(k+i-1)})$. The reason is that for gates $g \in V_{\leq 2(k+i-1)}$, whether a weight assignment is min-isolating for (C, z, g) only depends on the weights of the gates in $V_{\leq 2(k+i-1)}$. As w_{k+i-1} and w_{k+i} agree on that set, the invariant for $i-1$ implies that w_{k+i} is min-isolating for (C, z, g).
- For $i \in [k'-k]$, the invariant for $i-1$ implies that w_{k+i} is min-isolating for $(C, z, V_{2(k+i)-1})$. This follows because $V_{2(k+i)-1}$ is an AND layer. A certificate for an AND gate $g \in V_{2(k+i)-1}$ is the AND of certificates for gates $u, v \in V_{2(k+i-1)}$ feeding into g, and $w_{k+i}(C, z, g) = w_{k+i}(g) + w_{k+i}(C, z, u) + w_{k+i}(C, z, v)$. Since w_{k+i} and w_{k+i-1} agree on $V_{2(k+i-1)}$, the invariant for $i-1$ implies that w_{k+i} is min-isolating for (C, z, g).

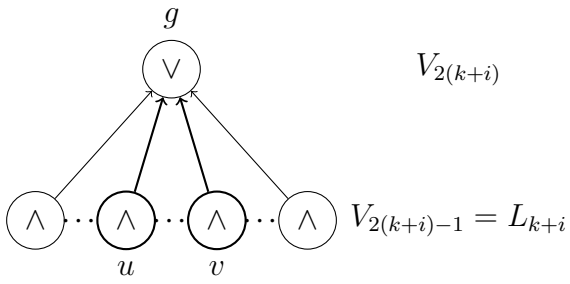

Figure 2.2:

Thus, in order to show that the invariant is maintained from $i-1$ to i for $i \in [k'-k]$, it suffices to show that w_{k+i} is min-isolating for $(C, z, V_{2(k+i)})$ assuming the invariant holds for $i-1$. The following claim provides a sufficient condition.

Claim 2.15. *Let $i \in [k'-k]$, and $g \in V_{2(k+i)}$ with $g(z) = 1$. Suppose that $b \geq 4r$ and that w_{k+i-1} is min-isolating for $(C, z, V_{\leq 2(k+i-1)})$. If for all distinct $u, v \in L_{k+i} \doteq V_{2(k+i)-1}$ that feed into g*

$$\left\lfloor \frac{\mu_k(u)}{b^{i-1}} \right\rfloor + h(u) \notin \left\lfloor \frac{\mu_k(v)}{b^{i-1}} \right\rfloor + h(v) + \{-1, 0, 1\}, \tag{2.10}$$

then w_{k+i} is min-isolating for (C, z, g).

See Figure 2.2 for an illustration.

Proof of Claim 2.15. Since g is an OR gate, a certificate F_g for (C, z, g) consists of an edge from g to one of its inputs v for which $v(z) = 1$, and a certificate F_v for v. As $w_{k+i}(F_v) = w_{k+i-1}(F_v) + h(v) \cdot b^{i-1}$, it follows that the min-weight certificates for v under w_{k+i-1} and under w_{k+i} are the same. Thus, v has a unique min-weight certificate under w_{k+i},

$$\mu_{k+i}(v) = \mu_{k+i-1}(v) + h(v) \cdot b^{i-1}, \tag{2.11}$$

and the following condition is sufficient to guarantee that w_{k+i} is min-isolating for (C, z, g): For all distinct gates $u, v \in L_{k+i} \doteq V_{2(k+i)-1}$ that feed into g

$$\mu_{k+i}(u) \neq \mu_{k+i}(v). \tag{2.12}$$

We argue that (2.12) follows from (2.10) as long as $b \geq 4r$.

For $v \in L_{k+i}$ with $v(z) = 1$, let F_v denote a min-weight certificate for v under w_k. We have that

$$\mu_k(v) \leq \mu_{k+i-1}(v) \leq w_{k+i-1}(F_v) \leq w_k(F_v) + 4r \cdot b^{i-2} = \mu_k(v) + 4r \cdot b^{i-2}. \tag{2.13}$$

The first inequality follows because $w_{k+i-1} \geq w_k$, and the second one and the last one from the definition of μ. For the third inequality, note that w_{k+i-1} is obtained from w_k by adding weights to the vertices in the AND layers below L_{k+i}. In particular, for a vertex $u \in L_{k+i-j}$ we have that $w_{k+i-1}(u) = w_k(u) + h(v) \cdot b^{i-1-j} \leq w_k(u) + r \cdot b^{i-1-j}$. Recall that the weight $w_{k+i-1}(F_v)$ is a linear combination of vertex weights $w_{k+i-1}(\cdot)$ with nonnegative integral coefficients. The sum of the coefficients that the weights of the vertices in L_{k+i-j} receive in $w_{k+i-1}(F_v)$ is at most 2^j. Summing over all such layers with $j > 0$ we have that

$$w_{k+i-1}(F_v) \leq w_k(F_v) + r \cdot \sum_{j=1}^{i-1} 2^j b^{i-1-j} \leq w_k(F_v) + 4r \cdot b^{i-2}$$

for $b \geq 4$.

After division by b^{i-1}, (2.13) shows that

$$\frac{\mu_k(v)}{b^{i-1}} \leq \frac{\mu_{k+i-1}(v)}{b^{i-1}} \leq \frac{\mu_k(v)}{b^{i-1}} + \frac{4r}{b},$$

which implies that

$$\left\lfloor \frac{\mu_k(v)}{b^{i-1}} \right\rfloor \leq \left\lfloor \frac{\mu_{k+i-1}(v)}{b^{i-1}} \right\rfloor \leq \left\lfloor \frac{\mu_k(v)}{b^{i-1}} \right\rfloor + 1 \tag{2.14}$$

for $b \geq 4r$. In combination with the hypothesis (2.10), (2.14) implies that

$$\left\lfloor \frac{\mu_{k+i-1}(u)}{b^{i-1}} \right\rfloor + h(u) \neq \left\lfloor \frac{\mu_{k+i-1}(v)}{b^{i-1}} \right\rfloor + h(v),$$

which by (2.11) in turn implies (2.12) after undoing the division. This finishes the proof of Claim 2.15. □

Each individual disambiguation requirement (2.10) can be written as three conditions of the form (2.4). By Fact 2.2, each of these three conditions individually holds with probability at least $1 - 1/r$. There are at most n^3 disambiguation requirements over all $i \in [k' - k]$, namely n choices for each of g, u, and v. A union bound shows that they all hold simultaneously with probability at least $1 - 3n^3/r$, which is at least $1 - 1/n^2$ for $r \geq 3n^5$. Whenever they hold, we know that the invariant (2.9) holds for each $i \in [\![k' - k]\!]$, and in particular that $w_{k'}$ is min-isolating for $(C, z, V_{\leq 2k'})$.

This leads to the following definition of $\Gamma^{(\mathrm{cert,step})}$: $\Gamma^{(\mathrm{cert,step})}_{n,\ell,k,k'}$ takes a seed σ for $\Gamma^{(\mathrm{hashing})}_{n(2\ell+1),r}$, considers $h = \Gamma^{(\mathrm{hashing})}_{n(2\ell+1),r}(\sigma)$ as a function $h : D \mapsto [r]$ with $D = D_{n,\ell,k,k'} \doteq [n] \times [\![2\ell]\!]$, and for $g \in [n] \times \{j\}$ sets

$$(\Gamma^{(\mathrm{cert,step})}_{n,\ell,k,k'}(\sigma))(g) = \begin{cases} h(g) \cdot b^{(j-1)/2-k} & \text{for odd } j \in [2k+1, 2k'-1] \\ 0 & \text{otherwise.} \end{cases}$$

The above analysis shows that $\Gamma^{(\mathrm{cert,step})}$ has the required min-isolating property. By setting b to the first power of 2 that is at least $4r$ with $r = 3n^5$, the bitlength becomes $O((k' - k) \log b) = O((k' - k) \log n)$. The other required properties follow from the properties of the universal family $\Gamma^{(\mathrm{hashing})}_{m,r}$ given in Fact 2.2. They imply that $\Gamma^{(\mathrm{cert,step})}_{n,\ell,k.k'}$ has seed length $O(\log(|D_{n,\ell,k,k'}| \cdot r)) = O(\log n)$. As each bit of $(\Gamma^{(\mathrm{cert,step})}(\sigma))(g)$ equals an easily determined bit of $h(g)$, the logspace computability of the universal family of hash functions implies the logspace computability of $\Gamma^{(\mathrm{cert,step})}_{n,\ell,k,k'}$. This completes the proof of Lemma 2.14. □

We now turn to the proof of the theorem.

Proof of Theorem 2.13. Let C be a circuit as in the statement of the theorem with layers $V_j \subseteq [n] \times \{j\}$ for $j \in [\![d]\!]$, and let z be an input for C. Consider first the case where the layer V_1 of C next to the leaves consists of ANDs.

If d is even and of the form $d = 2\ell$ with $\ell = \Delta^2$ for some $\Delta \in \mathbb{N}$, we can apply Lemma 2.14 Δ times successively, starting from an arbitrary weight assignment w_0. The ith application sets $k = k_i \doteq (i-1) \cdot \Delta$ and $k' = k'_i \doteq i \cdot \Delta$, uses a fresh seed σ_i for $\Gamma^{(\mathrm{cert,step})}_{n,\ell,k_i,k'_i}$, sets $w_{i\cdot\Delta} = w_{(i-1)\cdot\Delta} + \Gamma^{(\mathrm{cert,step})}_{n,\ell,k_i,k'_i}(\sigma_i)$, and tries to maintain the invariant that $w_{i\cdot\Delta}$ is min-isolating for $(C, z, V_{\leq 2i\cdot\Delta})$. We end up with $w_\ell = w_0 + \Gamma^{(\mathrm{cert,odd})}_{n,d}(\sigma_1, \sigma_2, \ldots, \sigma_\Delta)$, where

$$\Gamma^{(\mathrm{cert,odd})}_{n,d}(\sigma_1, \sigma_2, \ldots, \sigma_\Delta) \doteq \sum_{i \in [\Delta]} \Gamma^{(\mathrm{cert,step})}_{n,\ell,k_i,k'_i}(\sigma_i). \tag{2.15}$$

The superscript "odd" in $\Gamma^{(\mathrm{cert,odd})}$ refers to the fact that only odd layers receive nonzero values under weight assignments generated by $\Gamma^{(\mathrm{cert,odd})}$. The probability that the ith application breaks the invariant is at most $1/n^2$. By a union bound, the probability that the invariant fails at the end is at most $\Delta/n^2 \leq 1/n$. Thus, for any fixed $w_0 : [n] \times [\![d]\!] \mapsto \mathbb{N}$, $w_0 + \Gamma^{(\mathrm{cert,odd})}_{n,d}$ is min-isolating for (C, z) with probability at least $1 - 1/n$. The seed length of $\Gamma^{(\mathrm{cert,odd})}_{n,d}$ is Δ times the one of $\Gamma^{(\mathrm{cert,step})}_{n,d,\cdot,\cdot}$, i.e., $O(\Delta \log n) = O(\sqrt{d} \log n)$. The maximum weight assigned by $\Gamma^{(\mathrm{cert,odd})}_{n,d}$ is at most Δ times the one assigned by $\Gamma^{(\mathrm{cert,step})}_{n,d,\cdot,\cdot}$, so the bitlength of $\Gamma^{(\mathrm{cert,odd})}_{n,d}$ is $O(\log(\Delta) + \Delta \cdot \log n) = O(\sqrt{d} \log n)$. The logspace computability of $\Gamma^{(\mathrm{cert,step})}$ and the fact that iterated addition can be computed in logspace (see, e.g., [95]) imply that $\Gamma^{(\mathrm{cert,odd})}_{n,d}$ is computable in space $O(\log n)$.

Other values of d can be handled by conceptually extending the circuit with successive matchings until the depth is of the form $2\Delta^2$, applying the above construction, and then only using the part needed. As the smallest such Δ still satisfies $\Delta = \Theta(\sqrt{d})$, the parameters remain the same up to constant factors. Thus, we have a weight assignment generator $\Gamma^{(\mathrm{cert,odd})}$ with all the properties required of $\Gamma^{(\mathrm{cert})}$ in the case where the layer V_1 of C consists of ANDs.

To handle the case where V_1 consists of ORs, we can conceptually split every

wire (u, v) from a leaf u to $v \in V_1$ into two by inserting a fresh AND gate g and replacing (u, v) by (u, g) and (g, v). We then apply the construction for the case where V_1 consists of ANDs, and finally undo the splitting again, transferring the weight of each fresh AND gate g to the leaf u that feeds into it. This results in a weight assignment generator $\Gamma^{(\mathrm{cert,even})}$ that only assigns nonzero weights to the even layers, and has all the properties required of $\Gamma^{(\mathrm{cert})}$ for circuits C where the layer V_1 next to the leaves consists of ORs. For any such circuit C, input z for C, and any fixed $w'_0 : [n] \times [\![d]\!] \mapsto \mathbb{N}$, we have that $w'_0 + \Gamma^{(\mathrm{cert,even})}_{n,d}$ is min-isolating for (C, z) with probability at least $1 - 1/n$.

We claim that

$$\Gamma^{(\mathrm{cert})}_{n,d} \doteq \Gamma^{(\mathrm{cert,odd})}_{n,d} + \Gamma^{(\mathrm{cert,even})}_{n,d}$$

satisfies all requirements irrespective of the type of V_1, provided that we pick the seeds for $\Gamma^{(\mathrm{cert,odd})}_{n,d}$ and $\Gamma^{(\mathrm{cert,even})}_{n,d}$ independently. This follows from the above analysis by setting $w_0 = \Gamma^{(\mathrm{cert,even})}_{n,d}$ and $w'_0 = \Gamma^{(\mathrm{cert,odd})}_{n,d}$. More specifically, in the case where V_1 consists of ANDs, the above analysis shows that (2.8) holds for any fixed choice of the seed for $\Gamma^{(\mathrm{cert,even})}_{n,d}$, and thus holds overall by averaging. The case where V_1 consists of ORs is similar. The parameters of $\Gamma^{(\mathrm{cert})}$ follow from those of $\Gamma^{(\mathrm{cert,odd})}$ and $\Gamma^{(\mathrm{cert,even})}$. This finishes the proof of Theorem 2.13. □

2.4.2 Isolation

We use the weight assignment generator from Theorem 2.13 to establish the following result

Theorem 2.16. $\mathrm{LogCFL} \subseteq \mathrm{UAuxPDA\text{-}TISP}(\mathrm{poly}(n), (\log n)^{3/2})$.

In words: Every language in the class LogCFL can be accepted by a nondeterministic machine equipped with a stack that does not count towards the space bound, that runs in polynomial time and $O((\log n)^{3/2})$ space, and has at most one accepting computation path on every input.

By the following proposition, it suffices to construct such a machine for CIRCUIT CERTIFICATION on shallow layered alternating semi-unbounded circuits.

Proposition 2.17. CIRCUIT CERTIFICATION *on shallow layered alternating semi-unbounded Boolean circuits is hard for* LogCFL *under logspace mapping reductions that preserve the number of solutions.*

The unambiguous machine for CIRCUIT CERTIFICATION uses our weight assignment generator for the problem as well as the following unambiguous machines. They represent improvements of machines from [76], similar to those given by Lemma 2.11 in the context of REACHABILITY. See Section 2.6 for a proof and further discussion.

Lemma 2.18. *There exist unambiguous machines,* $\text{WEIGHTCHECK}^{(\text{cert})}$ *and* $\text{WEIGHTEVAL}^{(\text{cert})}$*, each equipped with a stack that does not count towards the space bound, such that for every layered semi-unbounded Boolean circuit* $C = (V, E)$ *of depth* d *with* n *gates, every input* z *for* C*, weight assignment* $w : V \mapsto \mathbb{N}$*, and* $g \in V$*:*

(i) $\text{WEIGHTCHECK}^{(\text{cert})}(C, z, w)$ *decides whether or not* w *is min-isolating for* (C, z)*, and*

(ii) $\text{WEIGHTEVAL}^{(\text{cert})}(C, z, w, g)$ *computes* $w(C, z, g)$ *provided* w *is min-isolating for* (C, z)*.*

Both machines run in time $\mathrm{poly}(2^d, \log(W), n)$ *and space* $O(d + \log(W) + \log(n))$*, where* W *denotes an upper bound on the finite values* $w(C, z, g)$ *for* $g \in V$*.*

We now have all the ingredients to establish our efficient unambiguous machines for LogCFL.

Proof of Theorem 2.16. By way of Proposition 2.17, it suffices to construct an unambiguous machine that decides[9] CIRCUIT CERTIFICATION on layered alternating semi-unbounded Boolean circuits $C = (V, E)$ of size n and depth

[9] In fact, we only need to construct a machine that *accepts* the language, but we naturally get the stronger notion of one that *decides* the language.

$d \leq \log_2(n)$, and runs in time $n^{O(1)}$ and space $O((\log n)^{3/2})$ when equipped with a stack that does not count towards the space bound. In fact, thanks to simple manipulations described earlier, it suffices to consider the case where the depth d is of the form $d = 2\Delta^2$ for $\Delta \in \mathbb{N}$, and where the layer next to the leaves consists of ANDs. We claim that the machine CIRCUITEVAL described in Algorithm 2.3 does the job.

Algorithm 2.3: CIRCUITEVAL(C, z, g)

Input : $C = (V, E)$: layered semi-unbounded circuits of depth d with layers $V_0, V_1, \ldots, V_d$
z: input for C
$g \in V$

Promise : $d = 2\Delta^2$ for $\Delta \in \mathbb{N}$ and V_1 consists of ANDs

Output : $g(z)$

for $i \leftarrow 1$ **to** Δ **do**
  foreach $\sigma_i \in \{0,1\}^{s(n,d)}$ *in lex order* **do**
    $isolating \leftarrow$ true;
    foreach $v \in V_{\leq 2i\Delta}$ *in lex order* **do**
      if *not* WEIGHTCHECK$^{(\mathrm{cert})}(C_v, z, \sum_{j=1}^{i} \Gamma^{(\mathrm{cert,step})}_{n,d,(j-1)\cdot\Delta,j\cdot\Delta}(\sigma_j))$ **then**
        $isolating \leftarrow$ false;
        exit the for loop over v;
    if $isolating$ **then** store the current σ_i and exit the loop over σ_i;
if WEIGHTEVAL$^{(\mathrm{cert})}(C, z, \sum_{j=1}^{\Delta} \Gamma^{(\mathrm{cert,step})}_{n,d,(j-1)\cdot\Delta,j\cdot\Delta}(\sigma_j), g) < \infty$ **then**
  accept and **return** 1
else accept and **return** 0;

Consider the version of our weight assignment generator $\Gamma^{(\mathrm{cert})}$ from Theorem 2.13 that is geared towards such circuits, namely $\Gamma^{(\mathrm{cert,odd})}$ given by (2.15). We know that on most seeds $\Gamma^{(\mathrm{cert,odd})}_{n,d}$ produces a weight assignment w that is min-isolating for (C, z). The machine CIRCUITEVAL in Algorithm 2.3 constructs such a seed. In fact, it constructs the lexicographically first such seed.

Recall that the seed σ consists of Δ parts $\sigma_i \in \{0,1\}^{s(n,d)}$ for $i \in [\Delta]$, where $s(n,d)$ denotes the seed length of $\Gamma^{(\text{cert,step})}_{n,d,\cdot,\cdot}$. The weight assignment $w \doteq \Gamma^{(\text{cert,odd})}_{n,d}(\sigma_1, \ldots, \sigma_\Delta)$ is min-isolating for (C, z) if and only if

$$w_{i\cdot\Delta} \doteq \sum_{j=1}^{i} \Gamma^{(\text{cert,step})}_{n,d,(j-1)\Delta,j\Delta}(\sigma_j) \text{ is min-isolating for } (C, V_{\leq 2i\Delta}) \tag{2.16}$$

for each $i \in [\Delta]$. This enables a prefix search for the lexicographically first σ for which w is min-isolating for (C, z). The first part of Algorithm 2.3 implements this search. In the ith iteration it finds the lexicographically first σ_i satisfying the invariant (2.16), given values for $\sigma_1, \ldots, \sigma_{i-1}$ from prior iterations. In order to check whether a given candidate σ_i works, it runs the machine $\text{WEIGHTCHECK}^{(\text{cert})}(C_v, z, w_{i\cdot\Delta})$ for each $v \in V_{\leq 2i\Delta}$, where C_v denotes the subcircuit of C rooted at v.

Once σ is determined, CIRCUITEVAL calls $\text{WEIGHTEVAL}^{(\text{cert})}(C, z, w, g)$ to compute $w(C, z, g)$, which is finite if and only if $g(z) = 1$.

The correctness of CIRCUITEVAL follows from maintaining the invariant (2.16) and the specifications of $\text{WEIGHTCHECK}^{(\text{cert})}$ and $\text{WEIGHTEVAL}^{(\text{cert})}$. The unambiguity of CIRCUITEVAL follows from the unambiguity of the machines $\text{WEIGHTCHECK}^{(\text{cert})}$ and $\text{WEIGHTEVAL}^{(\text{cert})}$ (and the usual conventions regarding composing unambiguous machines).

We end with a time and space analysis of CIRCUITEVAL. Each run of line 5 takes time $\text{poly}(2^d, \log(W), n)$ and space $O(d + \log(W) + \log(n))$, where W is a bound on the path weights under w. This follows from the complexities of $\Gamma^{(\text{cert,odd})}$ and $\text{WEIGHTCHECK}^{(\text{cert})}$, and the fact that iterated addition is in logspace (see, e.g., [95]). The three loops add a multiplicative term of $\Delta \cdot 2^{s(n,d)} \cdot n$ to the running time, and an additive term of $\log(\Delta) + s(n,d) + \log(n) + \log(W)$ to the space bound. The time and space needed for the call to $\text{WEIGHTEVAL}^{(\text{cert})}$ at the end is dominated by the rest of the computation. Since $\Delta = \Theta(\sqrt{d}) \leq \sqrt{\log n}$, $s(n,d) = O(\log n)$, and $W = 2^{O(\Delta \cdot \log(n))}$, the overall running time is $\text{poly}(2^d, n)$ and the space is $O(\sqrt{d}\log(n))$. This yields the stated complexities in the case of

shallow circuits, for which $d \leq \log_2(n)$. □

2.4.3 Reachability through Circuit Certification

We now explain how our results for CIRCUIT CERTIFICATION and LogCFL essentially imply the results for REACHABILITY and NL from Section 2.3. We review the reduction from REACHABILITY to CIRCUIT CERTIFICATION given by Savitch's Theorem, and show how it yields our weight assignment generator $\Gamma^{(\mathrm{reach})}$ for REACHABILITY from a slight modification $\tilde{\Gamma}^{(\mathrm{cert,odd})}$ of our weight assignment generator $\Gamma^{(\mathrm{cert,odd})}$ for CIRCUIT CERTIFICATION, and that min-isolation of $\tilde{\Gamma}^{(\mathrm{cert,odd})}$ on a reduced instance is equivalent to a restricted version of min-isolation of $\Gamma^{(\mathrm{reach})}$ on the original instance. The reduction also allows us to obtain alternate unambiguous machines for REACHABILITY and NL meeting the requirements of Theorem 2.2 from the unambiguous machines for CIRCUIT CERTIFICATION and LogCFL of Theorem 2.16.

Reduction Savitch's Theorem transforms nondeterministic logspace computations into equivalent logspace-uniform polynomial-size families of shallow alternating semi-unbounded circuits. This is one way to see that $\mathrm{NL} \subseteq \mathrm{LogCFL}$, and is equivalent to the following statement.

Proposition 2.19. *There exists a logspace mapping reduction from* REACHABILITY *to* CIRCUIT CERTIFICATION *on shallow alternating semi-unbounded Boolean circuits.*

We sketch the reduction as we need to analyze some of its properties.

Proof sketch. Let $G = (V, E)$ be a digraph of depth d on n vertices. For $k \in \mathbb{N}$ with $k \geq 2$, and $s, t \in V$, we can express the predicate $\mathrm{Reach}_k(s, t)$ of whether there exists a path of at most k edges from s to t in G as

$$\bigvee_{v \in V} \mathrm{Reach}_k(s, v, t), \tag{2.17}$$

where

$$\text{Reach}_k(s, v, t) \doteq \text{Reach}_{\lceil k/2 \rceil}(s, v) \wedge \text{Reach}_{\lfloor k/2 \rfloor}(v, t).$$

Recursive application starting from $k = d$ yields an alternating semi-unbounded circuit of even depth $\tilde{d} \leq 2 \log d$. The OR gates are labeled $\text{Reach}_k(s, t)$ for $k \in [2, d]$ and $s, t \in V$, indicating their meaning. The AND gates are labeled $\text{Reach}_k(s, v, u)$ for $k \in [2, d]$ and $s, v, t \in V$, also indicating their meaning, namely whether there exists a path of at most k edges from s to t consisting of a path of at most $\lceil k/2 \rceil$ edges from s to v followed by a path of at most $\lfloor k/2 \rfloor$ edges from v to t. The gates with fan-in zero correspond to $\text{Reach}_k(s, t)$ with $k = 1$, which we replace with an input variable indicating whether $(s, t) \in E$. Let C denote the resulting circuit (without a designated output gate). The reduction maps the REACHABILITY instance $x \doteq (G, s, t)$ to the CIRCUIT CERTIFICATION instance $\tilde{x} = (C, z, g)$ where z encodes the graph G, and g denotes the gate $\text{Reach}_d(s, t)$. □

For future reference we introduce the following terminology.

Definition 2.20 (REACHABILITY instances of CIRCUIT CERTIFICATION)**.** *The instances of* CIRCUIT CERTIFICATION *that result from the reduction in Proposition 2.19 are called* REACHABILITY *instances of* CIRCUIT CERTIFICATION.

Like before, we restrict attention to REACHABILITY instances $x \doteq (G, s^*, t^*)$ where $G = (V, E)$ is a layered digraph of depth $d = 2^\ell$ for some $\ell \in \mathbb{N}$. Let $V_0, V_1, \ldots, V_d$ denote the layers of G. We further restrict x such that $s^* \in V_0$ and $t^* \in V_d$. For such instances x the reduction in Proposition 2.19 yields a CIRCUIT CERTIFICATION instance $\tilde{x} \doteq (C, z, g)$ where C has depth $\tilde{d} = 2\ell$, is layered, alternating, and semi-unbounded, and has an AND layer next to the leaves. We denote the successive layers of C by $\tilde{V}_0, \tilde{V}_1, \ldots, \tilde{V}_{\tilde{d}}$, and the kth layer of ANDs by $\tilde{L}_k \doteq \tilde{V}_{2k-1}$. We can also restrict the range of v in (2.17) from all of V to the layer in the middle between the layers of s^* and t^*.

The following connections exist between x and its solutions (paths P), and $\tilde{x}$ and its solutions (certificates F).

- There is a bijection between the OR gates in C and pairs of vertices (s, t) such that s belong to first layer of some block and t to the last layer of the same block – in symbols, pairs (s, t) in $A_{\leq \ell} \doteq \bigcup_{k \leq \ell} A_k$ where

$$A_k \doteq \bigcup_{i \in [d/2^k]} V_{(i-1)\cdot 2^k} \times V_{i \cdot 2^k}. \tag{2.18}$$

  For any fixed such pair (s, t) and the corresponding OR gate g, there is a bijection between the solutions to the REACHABILITY instance (G, s, t) (paths P from s to t), and the solutions to the CIRCUIT CERTIFICATION instance (C, z, g) (certificates F in C witnessing that $g(z) = 1$).

- There is a bijection between the AND gates in C and triples of vertices (s, v, t) where $(s, t) \in A_{\leq \ell}$ as above, and v belongs to the middle layer between s and t. We can view $\tilde{L}_k$ as the subset of those triples (s, v, t) where $(s, t) \in A_k$. The projection of $\tilde{L}_k$ onto its middle component equals the set L_k given by (2.2).

  For any fixed such triple (s, v, t) and the corresponding AND gate g, there is a bijection between the paths in G from s to t that pass through v, and the certificates in C witnessing that $g(z) = 1$.

Consider a weight assignment $\tilde{w}$ to the gates of C that only assigns nonzero weights to the AND gates, i.e., to the gates in $\tilde{L}_{\leq \ell}$. ***Suppose*** that $\tilde{w}$ has the additional property that the value of $\tilde{w}(s, v, t)$ only depends on v, i.e., there exists a weight assignment w to V such that $\tilde{w}(s, v, t) = w(v)$. Then for any OR gate g and solution F for (C, z, g), and for the corresponding (s, t) and solution P for (G, s, t), it is the case that $\tilde{w}(F) = w(P)$. In particular, we have that $\tilde{w}$ is min-isolating for (C, z, g) if and only if w is min-isolating for (G, s, t). It follows that

$$\tilde{w} \text{ is min-isolating for } (C, z) \Leftrightarrow w \text{ is min-isolating for } (G, A_{\leq \ell}), \tag{2.19}$$

i.e., $\tilde{w}$ is min-isolating for (C, z, g) for all gates g if and only if w is min-isolating for (G, s, t) for all pairs s and t such that s belongs to the first layer of some block and t to the last layer of the same block.

Weight Assignment Generator Consider the version of our weight assignment generator $\Gamma^{(\mathrm{cert})}$ that is geared towards circuits (like C) with an AND layer next to the leaves, namely $\Gamma^{(\mathrm{cert,odd})}$ given by (2.15). $\Gamma^{(\mathrm{cert,odd})}$ has the property that the weight assignments $\tilde{w}$ it produces only assign nonzero weights to AND gates. It does not have the property that the weights of the AND gates (s, v, t) in C only depend on v. However, we can easily tweak the construction of $\Gamma^{(\mathrm{cert,odd})}$ so that it does.

The critical part in our analysis of the weight assignment generator $\Gamma^{(\mathrm{cert,odd})}$ is the disambiguation requirements (2.10). There is one such requirement for each choice of an OR gate g and two of the AND gates $\tilde{u}$ and $\tilde{v}$ that feed into g.[10] In the case of the circuit C, each OR gate g is of the form $g = (s, t)$, and the ANDs feeding into it are of the form $\tilde{u} = (s, u, t)$ and $\tilde{v} = (s, v, t)$. Thus, in each of the disambiguation requirements the gates $\tilde{u}$ and $\tilde{v}$ necessarily share their first and last components. This allows us to relax the requirement (2.4), i.e.,

$$\Pr_{\tilde{h}}[a + \tilde{h}(\tilde{u}) = b + \tilde{h}(\tilde{v})] \leq 1/r$$

where $\tilde{h}$ is chosen uniformly at random from $\Gamma^{(\mathrm{hashing})}_{\tilde{m},\tilde{r}}$, from holding for *all* pairs $(\tilde{u}, \tilde{v})$ with $\tilde{u} \neq \tilde{v}$, to holding for all pairs of the form $((s, u, t), (s, v, t))$ with $u \neq v$. This in turn allows us to replace the family $\Gamma^{(\mathrm{hashing})}_{\tilde{m},\tilde{r}}$ by a family of functions of the form $\tilde{h}(s, v, t) \doteq h(v)$ for h from another (smaller) universal family of hash functions $\Gamma^{(\mathrm{hashing})}_{m,r}$, without affecting the analysis.

The result can be interpreted as the following modification $\tilde{\Gamma}^{(\mathrm{cert,odd})}$ of our weight assignment generator $\Gamma^{(\mathrm{cert,odd})}$. In order to formally express the rela-

[10] We consistently introduce tildes for elements of the reduced CIRCUIT CERTIFICATION instance if there is a need to – for lack of a better word – disambiguate them from corresponding elements of the original REACHABILITY instance.

tionship, we view both as taking hash functions as inputs rather than seeds for $\Gamma^{(\text{hashing})}$ that produce those hash functions. We use square brackets to make the distinction clear. We have that $\tilde{\Gamma}^{(\text{cert,odd})}[h_1, \ldots, h_\ell] \doteq \Gamma^{(\text{cert,odd})}[\tilde{h_1}, \ldots, \tilde{h_\ell}]$ where $\tilde{h_k}(s, v, t) \doteq h_k(v)$ for $k \in [\ell]$. The new generator produces a weight assignment $\tilde{w} = \tilde{\Gamma}^{(\text{cert,odd})}[h_1, \ldots, h_\ell]$ to the gates of C that is min-isolating for (C, z) with probability at least $1 - 1/n$. It only assigns nonzero weights to the AND gates of C, i.e., the gates in $\tilde{L}_{\leq \ell}$. It has the above additional property – there exists a weight assignment w on V such that $\tilde{w}(s, v, t) = w(v)$ for all AND gates (s, v, t) in C. The weight assignment w only gives nonzero weights to $L_{\leq \ell} = V \setminus (V_0 \bigcup V_d)$. Moreover, the construction of $\tilde{\Gamma}^{(\text{cert,odd})}$ mimics the one of $\Gamma^{(\text{reach})}$, and we have that $w = \Gamma^{(\text{reach})}[h_1, \ldots, h_\ell]$.

In conclusion, we have exhibited an alternate way to obtain the weight assignment generator $\Gamma^{(\text{reach})}$ from Section 2.3.1 for REACHABILITY: Start with the slight modification $\tilde{\Gamma}^{(\text{cert,odd})}$ of our weight assignment generator $\Gamma^{(\text{cert,odd})}$ for CIRCUIT CERTIFICATION and apply the reduction from REACHABILITY to CIRCUIT CERTIFICATION given in Proposition 2.19. Moreover, by (2.19) we have the following equivalence for every seed σ: $\tilde{\Gamma}^{(\text{cert,odd})}(\sigma)$ is min-isolating for (C, z) if and only if $\Gamma^{(\text{reach})}(\sigma)$ is min-isolating for $(G, A_{\leq \ell})$. In other words, $\tilde{\Gamma}^{(\text{cert,odd})}(\sigma)$ is min-isolating for (C, z, g) for *all* gates g of C if and only if $\Gamma^{(\text{reach})}(\sigma)$ is min-isolating for (G, s, t) for all s and t such that s belongs to the first layer of some block and t to the last layer of the same block.

Isolation The min-isolation property that we obtain for $\Gamma^{(\text{reach})}$ via the alternate route is weaker than via the direct route – the weight assignment generator is only min-isolating for $(G, A_{\leq \ell})$ rather than for $(G, V \times V)$. Nevertheless, the weaker property is sufficient to derive the unambiguous machines for REACHABILITY and NL from Theorem 2.2. This is because the weaker property is compatible with the constructions in Lemma 2.11 – see Lemma 2.31 in Section 2.6. In fact, Lemma 2.31 can be obtained from the corresponding result in the setting of CIRCUIT CERTIFICATION and LogCFL, namely Lemma 2.18. Applying

the reduction from REACHABILITY to CIRCUIT CERTIFICATION as above to Lemma 2.18 yields Lemma 2.31 except that the resulting unambiguous machines make use of a stack that is not counted towards the space bound. However, one can argue that, in the case of REACHABILITY instances, the stack is not needed. The only reason the stack is used in Lemma 2.18 is for guessing and checking certificates in a space efficient manner. In the setting of REACHABILITY the role of the certificates is taken over by paths, which can be guessed and checked space efficiently without access to a stack. See the proofs in Section 2.6 for more details.

For the same reason the unambiguous machine CIRCUITEVAL in the proof of Theorem 2.16 does not need access to its stack on REACHABILITY instances. In combination with Proposition 2.10, this observation yields Theorem 2.2 as a corollary to Theorem 2.16.

2.5 Limitations

In this section we prove our "negative result" for isolating REACHABILITY (Theorem 2.3) and a corresponding result for CIRCUIT CERTIFICATION.

Recall that we view a computational problem as a mapping $\Pi : X \mapsto 2^Y$, where $\Pi(x)$ for $x \in X$ represents the set of solutions on input x. One can also think of Π as defining a relation $\pi : X \times Y \mapsto \{0, 1\}$, where $\pi(x, y)$ indicates whether $y \in \Pi(x)$. We use the notation $L(\Pi)$ to denote the set (language) of instances $x \in X$ for which $\Pi(x) \neq \emptyset$.

The first part of Theorem 2.3 follows by verifying that the main result of Dell, Kabanets, Van Melkebeek, and Watanable [34] carries over to the space-bounded setting: If Π has an efficient pruning and π is efficiently computable, then $L(\Pi)$ can be decided efficiently. The prunings in this statement are deterministic or, more generally, randomized with probability of success at least $\frac{2}{3} + \frac{1}{\text{poly}(n)}$. [34] showed that the above statement holds when "efficient" means polynomial-time for any Π that satisfies certain additional properties, which all the classical

problems like SATISFIABILITY do. We observe that the argument in [34] also works when "efficient" means logspace, and that both REACHABILITY and CIRCUIT CERTIFICATION have the required additional properties. This yields the first part of Theorem 2.3 and its counterpart for CIRCUIT CERTIFICATION. The second part follows from a slight modification of the argument.

The proof in [34] relies on a proposition of Ko's [62].

Proposition 2.21 ([62])**.** *Suppose that there exists a predicate $T : D \times D \mapsto \{0,1\}$ for some $D \subseteq X$ with the following properties:*

$$(\forall x, z \in D \cap L(\Pi))\ T(x,z) \vee T(z,x) \tag{2.20}$$

$$(\forall x, z \in D)\ z \in L(\Pi) \wedge T(z,x) \Rightarrow x \in L(\Pi) \tag{2.21}$$

Then for some $\ell \in [\![\lceil \log(|D|+1) \rceil]\!]$ there exists a sequence $z_1^*, \ldots, z_\ell^* \in D \cap L(\Pi)$ such that for every $x \in D$

$$x \in L(\Pi) \Leftrightarrow (\exists i \in [\ell])\, T(z_i^*, x). \tag{2.22}$$

If the $\vee$ in (2.20) were replaced by an exclusive or, T would induce a tournament over the vertex set $D \cap L(\Pi)$, where $T(z,x)$ (an edge from z to x) means that x wins the duel between z and x. Equation (2.20) requires the digraph T over D to contain a tournament over $D \cap L(\Pi)$ (and have a selfloop at every vertex in $D \cap L(\Pi)$). Equation (2.21) can be interpreted as saying that winners of duels are more likely be to in $L(\Pi)$ in the following sense: If at least one of x or z is in $L(\Pi)$, then any winner of the duel between x and z is.

Proposition 2.21 follows from the fact that every tournament graph has a dominating set of logarithmic size. In the case where D represents all instances of a given size n (of which there are at most 2^n), Proposition 2.21 shows us via (2.22) how to decide $L(\Pi)$ efficiently on D with the help of the $\ell \cdot n \leq n^2$ bits of advice z_i^* for $i \in [\ell]$, provided $T(z_i^*, x)$ is efficiently computable for $i \in [\ell]$ and $x \in D$.

[34] constructs such a predicate T satisfying (2.20) and (2.21) assuming the existence of an efficient deterministic pruning f for Π, that π is efficiently computable, and that Π allows an efficient disjoint union operator.

Definition 2.22 (Disjoint union of computational problems)**.** *Let $\Pi : X \mapsto 2^Y$ be a computational problem. A disjoint union operator for Π consists of a mapping $\sqcup : X \times X \mapsto X$ and a mapping $\tau : X \times X \times [2] \times Y \mapsto Y$ such that for all $x_1, x_2 \in X$, $|\Pi(x_1 \sqcup x_2)| = |\Pi(x_1)| + |\Pi(x_2)|$ and $\Pi(x_1 \sqcup x_2) = \dot{\bigcup}_{i\in[2]}\tau(x_1, x_2, i, \Pi(x_i))$, where $\tau(x_1, x_2, i, W) \doteq \dot{\bigcup}_{y\in W}\{\tau(x_1, x_2, i, y)\}$ for any $W \subseteq Y$.*

$\sqcup$ maps a pair of instances (x_1, x_2) to an instance $x_1 \sqcup x_2$ whose solutions can be viewed as the disjoint union of the solutions of x_1 and of x_2, where $\tau(x_1, x_2, i, y_i)$ describes the translation of the solution $y_i \in \Pi(x_i)$ into the corresponding solution in $\Pi(x_1 \sqcup x_2)$.

Several of the classical computational problems Π allow a simple disjoint union operator that is computable in logspace, meaning that both $\sqcup$ and τ in Definition 2.22 are computable in logspace. Often times the underlying predicate π is computable in logspace as well. This is the case, among others, for SATISFIABILITY, REACHABILITY, and CIRCUIT CERTIFICATION.

Proposition 2.23. REACHABILITY *and* CIRCUIT CERTIFICATION *on shallow semi-unbounded circuits have disjoint union operators as well as underlying predicates that are computable in logspace. The same holds for their restrictions to layered digraphs, and to layered alternating circuits, respectively.*

The key insight in [34] is that a pruning f applied to the disjoint union $x_1 \sqcup x_2$ implicitly selects an instance among x_1 and x_2 that is more likely to be positive – the unique solution of $f(x_1 \sqcup x_2)$ (if there is one) corresponds to a solution of exactly one of the two instances x_1 and x_2. A predicate T satisfying Ko's requirements (2.20) and (2.21) can be defined as follows on all pairs of instances $(z, x) \in X \times X$:

$$T(z,x) \Leftrightarrow \begin{cases} \tau(z,x,1,\Pi(z)) \cap \Pi(f(z \sqcup x)) = \emptyset & \text{for } z \leq_{lex} x \\ \tau(x,z,2,\Pi(z)) \cap \Pi(f(x \sqcup z)) = \emptyset & \text{for } x \leq_{lex} z, \end{cases}$$

where $\leq_{lex}$ denotes the lexicographic ordering. In other words, $T(z,x)$ holds if the unique solution of $f(\min(x,z) \sqcup \max(x,z))$ does not correspond to a solution of z, where $\min$ and $\max$ refer to the lexicographic ordering $\leq_{lex}$. The ordering of the arguments of the disjoint union is necessary to ensure that we work with the same disjoint union instance while determining $T(z,x)$ and $T(x,z)$. The isolation property $|\Pi(f(\cdot))| \leq 1$ ensures that T satisfies condition (2.20). The pruning property $\Pi(f(\cdot)) \subseteq \Pi(\cdot)$ implies condition (2.21).

In order to obtain an efficient algorithm for Π, [34] apply Ko's proposition on the set D of instances of size n with at most one solution. In the case of an instance z^* with a unique solution, say $\Pi(z^*) = \{y^*\}$, we can evaluate $T(z^*,x)$ as

$$T(z^*,x) \Leftrightarrow \begin{cases} \neg\pi(f(z^* \sqcup x), \tau(z^*,x,1,y^*)) & \text{for } z^* \leq_{lex} x \\ \neg\pi(f(x \sqcup z^*), \tau(x,z^*,2,y^*)) & \text{for } x \leq_{lex} z^*. \end{cases} \tag{2.23}$$

Given z^* and y^*, $T(z^*,x)$ can be computed efficiently when all of π, f, $\sqcup$, and τ can. This leads to an efficient algorithm with advice for deciding $L(\Pi)$ on the instances with at most one solution, where the advice consists of the strings (z_i^*, y_i^*) for $i \in [\ell]$. In order to decide $L(\Pi)$ on *any* instance $x \in X$, we first apply the pruning f, and then run the algorithm for instances with at most one solution on $f(x)$. This results in an efficient algorithm with polynomial advice for deciding $L(\Pi)$.

The above argument works for polynomial-time efficiency as well as for logspace efficiency. The polynomial-time incarnation yields the main result of [34] regarding the existence of deterministic polynomial-time prunings for SATISFIABILITY. The logspace incarnation yields the first part of Theorem 2.3 regarding the existence of deterministic logspace prunings for REACHABILITY

as well as a corresponding result for CIRCUIT CERTIFICATION.

As for the second part of Theorem 2.3 and its counterpart for CIRCUIT CERTIFICATION, a min-isolating weight assignment $\omega(x, y)$ applied to the disjoint union $x_1 \sqcup x_2$ selects between x_1 and x_2 in a similar way as a pruning does – the unique min-weight solution (if one exists) of the disjoint union corresponds to a solution of exactly one of x_1 and x_2. Given a function $\mu(x)$ that agrees with the min-weight $\omega(x)$ on positive instances x, this leads to the following predicate T satisfying the requirements (2.20) and (2.21) on instances (z^*, x) where z^* has a unique solution y^*:

$$T(z^*, x) \Leftrightarrow \begin{cases} \omega(z^* \sqcup x, \tau(z^*, x, 1, y^*)) \neq \mu(z^* \sqcup x) & \text{for } z^* \leq_{lex} x \\ \omega(x \sqcup z^*, \tau(x, z^*, 2, y^*)) \neq \mu(x \sqcup z^*) & \text{for } x \leq_{lex} z^*. \end{cases} \tag{2.24}$$

As in the setting of part 1, we obtain an efficient algorithm with polynomial advice for deciding $L(\Pi)$ on instances with at most one solution. To handle all inputs, we no longer have access to a pruning as we did in the case of part 1 of the Theorem. Access to the functions ω and μ does yield an efficient disambiguation provided the search for a solution of a given weight can be efficiently reduced to Π under a reduction that preserves the number of solutions. This is the case for polynomially-bounded ω and each of SATISFIABILITY, REACHABILITY, and CIRCUIT CERTIFICATION on shallow semi-unbounded circuits, both for polynomial-time efficiency and for logspace efficiency. In order to handle larger weight functions ω, we can alternately make use of a randomized disambiguation with success probability $1/\operatorname{poly}(n)$, which exists for all of these problems by virtue of the Isolation Lemma, and hardwire good random bit strings in the advice.

This completes the proof outlines for parts 1 and 2 of Theorem 2.3 (as well as their counterparts for CIRCUIT CERTIFICATION) in the case where the pruning f and the functions ω and μ are deterministic. For the more general case where they can be randomized and have probability of success at least $\frac{2}{3} + \frac{1}{\operatorname{poly}(n)}$, the predicate T needs to be generalized in an appropriate way. The following lemma formalizes the general case. We view a randomized mapping as a deterministic

one that gets a random bit string $\rho \in \{0,1\}^r$ as an additional input, and often write ρ as a subscript to the name of the procedure.

We state the lemma for logspace efficiency for concreteness, but the proof only requires mild properties of the underlying notion of efficiency. In particular, it also applies to polynomial-time efficiency.

Lemma 2.24. *Let $\Pi : X \mapsto 2^Y$ be a computational problem with an underlying predicate π that is computable in logspace and has the following additional properties:*

- *Π has a disjoint union operator given by $\sqcup$ and τ in Definition 2.22 where $\sqcup$ and τ are computable in logspace.*
- *Π has a randomized disambiguation g that is computable in logspace and satisfies the following for all inputs x:*

$$\begin{aligned} x \in L(\Pi) &\Rightarrow \Pr[g(x) \in L(\Pi)] \geq 1/\operatorname{poly}(|x|) \\ x \notin L(\Pi) &\Rightarrow \Pr[g(x) \in L(\Pi)] = 0, \end{aligned}$$

  where the probabilities are over the internal coin flips of g.

- *There exists a logspace mapping reduction h from the following decision problem to Π: On input an instance $x \in X$ and an index $i \in \mathbb{N}$, decide whether there exists $y \in \Pi(x)$ such that the ith bit of y is 1. Furthermore, the instances $h(x,i)$ have at most one solution if the instance x does.*

For any $p = \frac{2}{3} + \frac{1}{\operatorname{poly}(n)}$ either of the following hypotheses imply that $L(\Pi)$ can be decided in logspace with polynomial advice, where ρ is chosen uniformly at random from $\{0,1\}^r$ for some $r = \operatorname{poly}(n)$:

1. *There exists a randomized mapping $f : X \mapsto X$ computable in logspace such that for every input $x \in X$:*

$$\Pr_\rho[\ f_\rho \text{ satisfies the pruning requirement on input } x\ ] \geq p. \qquad (2.25)$$

2. *There exist randomized mappings* $\omega : X \times Y \times \mapsto \mathbb{N}$ *and* $\mu : X \mapsto \mathbb{N}$ *that are computable in logspace such that for every* $x \in L(\Pi)$

$$\Pr_{\rho}[\ \omega_\rho(x, \cdot) \textit{ is min-isolating for } x \textit{ and } \mu_\rho(x) = \omega_\rho(x)\ ] \geq p. \tag{2.26}$$

Proof. Let us first focus on the instances of Π that have at most one solution. Consider the predicate T defined as follows on input (z^*, x) where $\Pi(z^*) = \{y^*\}$ and q denotes a fraction to be set:

$$T(z^*, x) \Leftrightarrow \begin{cases} \Pr_\rho[\text{ right-hand side of (2.23) holds }] > q & \text{for part 1} \\ \Pr_\rho[\text{ right-hand side of (2.24) holds }] > q & \text{for part 2,} \end{cases} \tag{2.27}$$

where $\rho \in \{0,1\}^r$ is chosen uniformly at random for some $r = \mathrm{poly}(n)$, and is used as the randomness for all randomized mappings involved.

Claim 2.25. *Both* (2.20) *and* (2.21) *hold for* $q = 1/3$ *as long as* $p > 2/3$*, where* D *represents the set of all instances of* Π *with at most one solution.*

Proof. We argue by contradiction that T satisfies condition (2.20). Consider part 1 first, and suppose that neither $T(x, z^*)$ nor $T(z^*, x)$ hold for some $x, z^* \in D \cap L(\Pi)$. Then with probability at most $2q$ the translation of the unique solution for at least one of x or z^* is not a solution for $f(x^*)$, where $x^* \doteq \min(x, z^*) \sqcup \max(x, z^*)$. By complementing, with probability at least $1 - 2q$ it is the case that both translations are solutions for $f(x^*)$, which therefore has at least two distinct solutions. Thus, f fails the pruning condition on input x^* with probability at least $1 - 2q$, which contradicts the hypothesis that f has success probability p as long as $q < p/2$. In the case of part 2, a similar argument by contradiction leads to the conclusion that with probability at least $1 - 2q$ two distinct solutions for x^* achieve the value $\mu(x^*)$ under ω, which contradicts the hypothesis (2.26) as long as $q < p/2$.

We argue condition (2.21) by contradiction also. For part 1, consider $z^* \in D \cap L(\Pi)$ and $x \in D \setminus L(\Pi)$, and let $x^* \doteq \min(x, z^*) \sqcup \max(x, z^*)$. Suppose

for contradiction that $T(z^*, x)$ holds. Then the translation of the unique solution of z^* is not a solution of $f(x^*)$ with probability more than q. Since $x \notin L(\Pi)$, this tells us that f fails the pruning property on x^* with probability more than q, which contradicts the hypothesis (2.25) as long as $q \geq 1 - p$. For part 2, a similar argument leads to a contradiction with the hypothesis (2.26) as long as $q \geq 1 - p$.

The conditions $q < p/2$ and $q \geq 1 - p$ imply that $p > 2/3$, which is where the bound of $2/3$ in the statement of the lemma comes from. Setting $q = 1/3$ satisfies both requirements when $p > 2/3$. This finishes the proof of Claim 2.25. □

Note that the statement of the lemma entails some leeway in that p does not just exceed $2/3$ but does so with some margin, namely $p \geq \frac{2}{3} + \frac{1}{\mathrm{poly}(n)}$. We now exploit this leeway to replace the randomness in the definition of T by advice. More specifically, an application of the Chernoff bound shows that a subset R of a sufficiently large polynomial number of random strings $\rho \in \{0,1\}^r$ has the following property with high probability: All of the conditions (2.25) (in the case of part 1) or (2.26) (in the case of part 2) hold for all inputs x of length n simultaneously when the uniform distribution of ρ over $\{0,1\}^r$ is replaced by the uniform distribution over R, and p is replaced by $\tilde{p}$ for some $\tilde{p} = \frac{2}{3} + \frac{1}{\mathrm{poly}(n)}$. By fixing a good set R and giving it as advice, the predicates (2.27) become computable in logspace.

This shows the existence of an algorithm A that runs in logspace with polynomial advice and correctly decides $L(\Pi)$ on instances $x \in X$ with at most one solution. In order to handle *all* instances $x \in X$ we employ the randomized disambiguation g to reduce to the case of at most one solution.

Denoting by σ the random bit string of the randomized disambiguation g, another application of the Chernoff bound shows that for every size n there exists a set S of $\mathrm{poly}(n)$ strings of length $\mathrm{poly}(n)$ each such that for every instance $x \in L(\Pi)$ of size n there exists at least one $\sigma \in S$ such that $x_\sigma \doteq g_\sigma(x)$ has a unique solution, and for instances $x \notin L(\Pi)$, $x_\sigma \notin L(\Pi)$ for every $\sigma \in S$.

Consider a positive instance $x \in L(\Pi)$ of size n. We do not know which σ works but we do know that there is at least one and that for anyone that does,

the unique solution for the instance x_σ of Π is given by $(\chi[h(x_\sigma, i) \in L(\Pi)])_{i=1}^{n^c}$. Here χ denotes the indicator function and n^c for some constant c the bitlength of solutions to instance of Π of size n. This follows because if x_σ has a unique solution then the ith bit of that solution is 1 if and only if there exists a solution whose ith bit is 1, which is equivalent to $h(x_\sigma, i) \in L(\Pi)$ by the definition of h. Moreover, the instances $h(x_\sigma, i)$ of Π each have at most one solution themselves, so we can use our algorithm A to decide $L(\Pi)$ on those instances and retrieve the unique solution for x_σ as

$$y_\sigma \doteq (A(h(x_\sigma, i)))_{i=1}^{n^c}.$$

Finally, we try all possible $\sigma \in S$, and check whether y_σ is a valid solution for x_σ. More formally, we evaluate the predicate

$$\bigvee_{\sigma \in S} \pi(x_\sigma, y_\sigma). \tag{2.28}$$

If $x \in L(\Pi)$ then we know that for at least one $\sigma \in S$, y_σ is the unique solution to x_σ, so (2.28) evaluates to true. If $x \notin L(\Pi)$, then for all $\sigma \in S$, $x_\sigma \notin L(\Pi)$ and (2.28) evaluates to false no matter what. Thus, (2.28) correctly decides $L(\Pi)$ on all instances $x \in X$. As all the algorithms involved run in logspace with access to their random bit strings, which are given as advice, it follows that the predicate (2.28) can be evaluated in logspace with polynomial advice. This concludes the proof of Lemma 2.24. □

Theorem 2.3 follows from an instantiation of Lemma 2.24 with REACHABILITY on layered digraphs as the computational problem Π.

Proof of Theorem 2.3. Since REACHABILITY on layered digraphs is hard for NL under logspace mapping reductions (see Proposition 2.10), it suffices to verify that REACHABILITY on layered digraphs has all the properties required of the computational problem Π in Lemma 2.24.

- The properties regarding the predicate π and the disjoint union operator follow from Proposition 2.23.

- The existence of the required randomized disambiguation g with one-sided error follows from the Isolation Lemma (as described in the Introduction on page 15.)

- Finally, here is how we can compute the required retrieving predicate $h(x, i)$ for $x \doteq (G, s, t)$. The index i corresponds to a bit position, say the jth one, of the label of an edge in some layer, say the ℓth one, of G. The instance $h(x, i)$ is obtained by removing from G all edges in layer ℓ whose jth bit is not 1. This operation can be performed in logspace.

□

A similar argument for CIRCUIT CERTIFICATION on shallow layered alternating semi-unbounded circuits yields the following equivalent to Theorem 2.3.

Theorem 2.26. *Either of the following hypotheses imply* $\mathrm{LogCFL} \subseteq \mathrm{L}/\,\mathrm{poly}$*:*

1. CIRCUIT CERTIFICATION *on shallow semi-unbounded circuits that are layered and alternating has a logspace pruning.*

2. CIRCUIT CERTIFICATION *on shallow semi-unbounded circuits that are layered and alternating has a logspace weight function ω that is min-isolating, and there exists a logspace function μ such that $\mu(x)$ equals the min-weight $\omega(x)$ of x under ω on positive instances x.*

In fact, the conclusion holds even if the algorithms are randomized, as long as the probability of success exceeds $\frac{2}{3} + \frac{1}{\mathrm{poly}(n)}$, and the algorithms run in logspace when given two-way access to the random bits.

2.6 Checking Min-Isolation and Computing Min-Weights

This section presents the unambiguous logspace machines from Lemma 2.11 and Lemma 2.18 for computing whether a given weight assignment is min-isolating, and for computing min-weights in the case of min-isolation. The machines are used in our unambiguous simulations: Lemma 2.11 in the proof of Theorem 2.2 in the setting of REACHABILITY and NL, and Lemma 2.18 in the proof of Theorem 2.16 in the setting of CIRCUIT CERTIFICATION and LogCFL.

Both lemmas follow from the results and techniques of Reinhardt and Allender [76]. The underlying machines are slight variations with improved running times, which are necessary for Theorems 2.2 and 2.16. We present a full proof of Lemma 2.18, and sketch the proof of Lemma 2.11. In fact, we establish the following extension of Lemma 2.18, which will aid us with the proof sketch for Lemma 2.11.

Lemma 2.27. *There exist unambiguous machines,* WEIGHTCHECK$^{(\mathrm{cert})}$ *and* WEIGHTEVAL$^{(\mathrm{cert})}$*, each equipped with a stack that does not count towards the space bound, such that for every layered semi-unbounded Boolean circuit $C = (V, E)$ of depth d with n gates, every input z for C, weight assignment $w : V \mapsto \mathbb{N}$, and $g \in V$:*

(i) WEIGHTCHECK$^{(\mathrm{cert})}(C, z, w)$ *decides whether or not w is min-isolating for (C, z), and*

(ii) WEIGHTEVAL$^{(\mathrm{cert})}(C, z, w, g)$ *computes $w(C, z, g)$ provided w is min-isolating for (C, z).*

Both machines run in time $\mathrm{poly}(2^d, \log(W), n)$ *and space* $O(d + \log(W) + \log(n))$*, where W denotes on upper bound on the finite values $w(C, z, v)$ for $v \in V$. Moreover, on* REACHABILITY *instances the machines do not need the stack.*

The "moreover" clause, which refers to "REACHABILITY instances" from Definition 2.20, enables us to formally derive a variant of Lemma 2.11 that is sufficiently strong for the proof of Theorem 2.2. We explain this after the proof of Lemma 2.27.

The key tool in [76] is a modification of the inductive counting technique of Immerman and Szelepcsenyi [46, 87] where besides computing the values

$$n_k \doteq |\{g \in V_k \,:\, g(z) = 1\}| \tag{2.29}$$

for $k = 0, 1, \ldots$, we also compute the values

$$s_k \doteq \sum_{g \in V_k \colon g(z)=1} w(C, z, g) \tag{2.30}$$

in sync. As in the proofs in [46, 87] that NL is closed under complementation, the knowledge of n_k allows us to cycle through all of V_k in nondeterministic logspace (without missing anyone). The additional knowledge of s_k enables us to modify that nondeterministic process so that it rejects if it ever guesses a certificate that is not of minimum weight. This makes the process unambiguous provided the weight assignment is min-isolating.

Inspired by the reduction from the search for a certificate of a given weight to CIRCUIT CERTIFICATION from [40], Reinhardt and Allender [76] define the sets V_k based on the min-weight. More precisely, their set V_k consists of all gates g for which $w(C, z, g) \leq k$. This approach necessarily involves a number of steps that is at least W, which is superpolynomial in n for our weight assignment generator. Instead, we use the layers of the circuit C as the sets V_k (as our notation suggests). This reduces the number of steps down to $d \leq n$.

Proof of Lemma 2.27. Let C, z, g, and w be as in the statement of the lemma. Let $V_0, V_1, \ldots, V_d$ be the layers of C. We will maintain the invariants (2.29) and (2.30) for each $k \in [\![d]\!]$.

We first show how the knowledge of n_k and s_k allows us to efficiently and

unambiguously compute $w(C, z, g)$ for $g \in V_k$ provided w is min-isolating for (C, z, V_k).

Claim 2.28. *The nondeterministic machine* PROMISEWEIGHTEVAL *given in Algorithm 2.4 computes the problem of its specification, and is unambiguous on all inputs satisfying the promise. Equipped with a stack that does not count towards the space bound, the machine runs in time* $\mathrm{poly}(2^d, \log(W), n)$ *and space* $O(\log(W) + \log(n))$. PROMISEWEIGHTEVAL *does not need the stack on* REACHABILITY *instances.*

Proof of Claim 2.28. We first argue correctness and unambiguity. Consider an input as in the specification, satisfying the promise. The machine returns an output if and only if $n = n_k$ and $s = s_k$ at the end of the loop in line 11. We argue that $n = n_k$ holds in that line if and only if all non-deterministic guesses in line 3 were correct and the algorithm guessed a valid certificate every time it executed line 5. In the case of a false positive, i.e., an incorrect guess in line 3 for a gate v with $v(z) = 0$, the machine will fail to find a certificate in line 5 as no certificate exists. In the case of a false negative, i.e., an incorrect guess in line 3 for a gate v with $v(z) = 1$, it has to be the case that $n < n_k$ in line 11 if that line is reached at all. Therefore, the machine reaches the end of the loop with $n = n_k$ if and only if all guesses in line 3 were correct.

Assuming the machine reaches line 11 with $n = n_k$, $s = s_k$ holds at that point in time if and only if each time the machine guessed a certificate in line 5, the certificate had minimum weight. Therefore, if w is min-isolating for (C, z, V_k), there is a unique computational path on which it reaches the end of the loop with $n = n_k$ and $s = s_k$. On that computation path, the machine has checked g against every gate $v \in V_k$ for which $v(z) = 1$. Thus, if it has not encountered g, $g(z) = 0$ and the machine correctly returns $\omega = \infty$. Otherwise, in the iteration with $v = g$ the machine guessed the unique min-weight certificate F for g, computed its weight $w(F)$, and set $\omega = w(F)$, which it correctly returns at the end.

Algorithm 2.4: PROMISEWEIGHTEVAL$(C, z, w, k, n_k, s_k, g)$

Input : $C = (V, E)$: layered semi-unbounded circuits of depth d with layers $V_0, V_1, \ldots, V_d$
z: input for C
$w : V \mapsto \mathbb{N}$
$k \in [\![d]\!]$
$n_k, s_k \in \mathbb{N}$
$g \in V_k$

Promise : w is min-isolating for (C, z, V_k)
n_k and s_k satisfy (2.29) and (2.30)

Output : $w(C, z, g)$

$n \leftarrow 0$, $s \leftarrow 0$, $\omega \leftarrow \infty$;
foreach $v \in V_k$ *in lex order* **do**
  guess whether $v(z) = 1$;
  if *the guess is "yes"* **then**
    guess a candidate certificate F for (C, z, v), check its validity, and compute $w(F)$;
    if F *is valid* **then**
      $n \leftarrow n + 1$;
      $s \leftarrow s + w(F)$;
      if $v = g$ **then** $\omega \leftarrow w(F)$;
    else reject;
if $n = n_k$ *and* $s = s_k$ **then**
  accept and **return** ω
else reject;

This argues that the machine satisfies its specification, and behaves unambiguously on every input satisfying the promise.

To analyze the complexity, we need to be more specific about the implementation of line 5. We implement it as a space efficient depth-first search with the help of the stack, while keeping track of a variable to compute $w(F)$ on the fly. We start by pushing v on the stack, and initialize the variable to zero. We then repeat the following process: Pop a gate u from the stack and add $w(u)$ to the variable. If u is an OR, nondeterministically guess a gate that feeds into u, and push it onto the stack. If u is an AND, then push the gates that feed into u onto the stack in lex order. If u is a leaf, then check whether it evaluates to 1 on z; if not, we know that F is not a valid certificate; otherwise, continue.

The space used by the machine, other than the stack, is dominated by the space required to keep track of the variables n and s, which is $O(\log(W)+\log(n))$. The running time is dominated by the depth-first searches in line 5. They essentially explore an expansion of the certificate as a formula, which can have size 2^d. Each step involves elementary graph operations (time $\mathrm{poly}(n)$) and the addition of numbers bounded by W (time $\mathrm{poly}(\log(W))$). Thus, the overall running time is bounded by $\mathrm{poly}(2^d, \log(W), n)$.

On REACHABILITY instances, we can implement line 5 without using the stack, namely by guessing a path of length k and computing its weight on the fly. This concludes the proof of Claim 2.28. □

Next we build on PROMISEWEIGHTEVAL to efficiently and unambiguously check whether w is min-isolating for (C, z, V_{k+1}), using the values (n_k, s_k), and computing the values (n_{k+1}, s_{k+1}) along the way in case w is indeed min-isolating for (C, z, V_{k+1}).

Claim 2.29. *The nondeterministic machine* PROMISEWEIGHTCHECK *in Algorithm 2.5 computes the problem of its specification, and is unambiguous on all inputs satisfying the promise. Equipped with a stack that does not count towards the space bound,* PROMISEWEIGHTCHECK *runs in time* $\mathrm{poly}(2^d, \log(W), n)$

and space $O(\log(W) + \log(n))$. PROMISEWEIGHTCHECK *does not need the stack on* REACHABILITY *instances.*

Proof of Claim 2.29. We first argue correctness. Consider an input satisfying the promise. Note that all the calls that PROMISEWEIGHTCHECK makes to PROMISEWEIGHTEVAL satisfy the promise that PROMISEWEIGHTEVAL requires. Thus, all these calls return the correct values on any accepting computation path, of which there exists at least one.

For an AND gate $g \in V_{k+1}$ with u and v as the gates feeding into it, we have that $w(C, z, g) = w(g) + w(C, z, u) + w(C, z, v)$, and w is min-isolating for (C, z, g) no matter what, since the promise guarantees that it is min-isolating for both (C, z, u) and (C, z, v).

What PROMISEWEIGHTCHECK does in the case where $g \in V_{k+1}$ is an AND layer, is to conceptually compute $w(C, z, g)$ as $w(g) + w(C, z, u) + w(C, z, v)$, and if that value is finite, increase n by one and add the value $w(C, z, g)$ to s. As $w(C, z, g)$ is finite if and only if $g(z) = 1$, this shows that the correct contributions of g to the quantities n_{k+1} and s_{k+1} are added to n and s, respectively.

For an OR gate g, $w(C, z, g)$ equals $w(g)$ plus the minimum of $w(C, z, v)$ over all gates v feeding into g. As the promise guarantees that w is min-isolating for (C, z, v) for each of those gates v, it follows that w is min-isolating for g unless $w(C, z, g)$ is finite and there are two distinct gates, say u and v, that feed into g and satisfy $w(C, z, g) = w(g) + w(C, z, u) = w(g) + w(C, z, v)$.

In the case where $g \in V_{k+1}$ is an OR gate, PROMISEWEIGHTCHECK does the following: Go over all gates v that feed into g and keep track of two quantities – $currmin$ is the minimum value of $w(C, z, v)$ seen thus far, and $prevmin$ is the next smallest value of $w(C, z, v)$ seen thus far, where duplicate values are taken into account, and both quantities are initialized to ∞. After having processed all gates v, there are three cases:

- $currmin < prevmin$: As $currmin$ is finite, we know that $w(C, z, g) = w(g) + currmin$ is finite as well. We also know that $w(C, z, g) = w(g) +$

Algorithm 2.5: PROMISEWEIGHTCHECK(C, z, w, k, n_k, s_k)

Input : $C = (V, E)$: layered semi-unbounded circuits of depth d with layers $V_0, V_1, \ldots, V_d$
z: input for C
$w : V \mapsto \mathbb{N}$
$k \in [\![d-1]\!]$
$n_k, s_k \in \mathbb{N}$

Promise : w is min-isolating for (C, z, V_k)
n_k and s_k satisfy (2.29) and (2.30)

Output : (true, n_{k+1}, s_{k+1}) satisfying (2.29) and (2.30) if w is min-isolating for (C, z, V_{k+1})
(false, $-, -$) otherwise

```
(n, s) ← (0, 0);
foreach g ∈ V_{k+1} in lex order do
    if g is an AND gate then
        let u and v be the gates feeding into g in lex order;
        μ ← PROMISEWEIGHTEVAL(C, z, w, k, n_k, s_k, u);
        ν ← PROMISEWEIGHTEVAL(C, z, w, k, n_k, s_k, v);
        if μ < ∞ and ν < ∞ then
            (n, s) ← (n + 1, s + w(g) + μ + ν);
    else { g is an OR gate }
        (currmin, prevmin) ← (∞, ∞);
        foreach gate v that feeds into g in lex order do
            ν ← PROMISEWEIGHTEVAL(C, z, w, k, n_k, s_k, v);
            if ν ≤ currmin then
                prevmin ← currmin;
                currmin ← ν ;
        if currmin < prevmin then
            (n, s) ← (n + 1, s + w(g) + currmin)
        if currmin < ∞ then
            accept and return (false, −, −)
accept and return (true, n, s);
```

$w(C, z, v)$ for only one of the gates v feeding into g. Thus, $g(z) = 1$ and w is min-isolating for (C, z, g). In this case PROMISEWEIGHTCHECK increases the variable n by 1, and adds $w(C, z, g) = w(g) + currmin$ to the variable s.

- $currmin = prevmin < \infty$: This means that $g(z) = 1$ and that there are two distinct gates, say u and v, that feed into g and satisfy $w(C, z, g) = w(g) + w(C, z, u) = w(g) + w(C, z, v)$. Hence, w is not min-isolating for (C, z, g). In this case, PROMISEWEIGHTCHECK ends the loop over g, and correctly returns $(\text{false}, -, -)$.

- $currmin = \infty$: This means that $g(z) = 0$, and w is vacuously min-isolating for (C, z, g). In this case PROMISEWEIGHTCHECK leaves the variables n and s as are.

If the end of the loop over g is reached, we know that w is min-isolating for (C, z, V_{k+1}). The variable n has been increased with the number of $g \in V_{k+1}$ for which $g(z) = 1$, and s by $w(C, z, g)$ for each such g. As both n and s are initialized to 0, this shows that the value (true, n, s) that PROMISEWEIGHTCHECK returns at the end is correct.

Regarding unambiguity, note that PROMISEWEIGHTCHECK is deterministic modulo the runs of the calls to PROMISEWEIGHTEVAL. As the argument of each call satisfies the promise of PROMISEWEIGHTEVAL, each of those runs is unambiguous. It follows that PROMISEWEIGHTCHECK is unambiguous (because of the understanding that PROMISEWEIGHTCHECK rejects on any computation path on which a call to PROMISEWEIGHTEVAL rejects).

Excluding the calls to PROMISEWEIGHTEVAL, PROMISEWEIGHTCHECK runs in time $\text{poly}(\log(W), n)$ and in space $O(\log(W) + \log(n))$, and does not need access to the stack. Each PROMISEWEIGHTEVAL call takes time $\text{poly}(2^d, \log(W), n)$ and space $O(\log(W) + \log(n))$ with the use of a stack. It follows that PROMISEWEIGHTCHECK runs in time $\text{poly}(2^d, \log(W), n)$ and

space $O(\log(W) + \log(n))$ with the use of a stack (as the space needed for subsequent class to PROMISEWEIGHTEVAL can be reused).

Since the calls to PROMISEWEIGHTEVAL do not need access to the stack on REACHABILITY instances, PROMISEWEIGHTCHECK does not need access to the stack on those instances either. This concludes the proof of Claim 2.29. □

Finally, we use PROMISEWEIGHTCHECK and PROMISEWEIGHTEVAL to unambiguously decide whether or not w is min-isolating for (C, z) and, if so, compute $w(C, z, g)$. We call the machine PROMISEWEIGHTCHECK iteratively to bootstrap and construct the sequence of values (n_k, n_k) for $k = 1, 2, \ldots, d$ while ascertaining the invariant that w is min-isolating for $(C, z, V_{\leq k})$, aborting the construction as soon as a violation of the invariant is detected. When we arrive at the (n_k, s_k) values for the layer V_k of a given gate g for which we want to compute $w(C, z, g)$, we call PROMISEWEIGHTEVAL to evaluate $w(C, z, g)$.

Claim 2.30. *The nondeterministic machine* WEIGHTCHECKEVAL *given in Algorithm 2.6 unambiguously computes the problem in its specification. Equipped with a stack that does not count towards the space bound,* WEIGHTCHECKEVAL *runs in time* $\mathrm{poly}(2^d, \log(W), n)$ *and space* $O(\log(W) + \log(n))$. *Moreover,* WEIGHTCHECKEVAL *does not need the stack on* REACHABILITY *instances.*

Proof of Claim 2.30. Consider an input as in the specification. The following loop invariants hold each time the loop condition in line 7 of WEIGHTCHECKEVAL is checked:

1. $isolating$ indicates whether w is min-isolating for $(C, z, V_{\leq k})$.
2. If $isolating$ is true, then $(n, s) = (n_k, s_k)$ where n_k and s_k are given by (2.29) and (2.30).
3. If $isolating$ is true and $g \in V_{\leq k}$ then $\omega = w(C, z, g)$.

The invariants are set up in the first 7 lines for $k = 0$. The loop body calls the machine PROMISEWEIGHTCHECK on the input (C, z, w, k, n_k, s_k), knowing that

Algorithm 2.6: WEIGHTCHECKEVAL(C, z, w, g)

Input : $C = (V, E)$: layered semi-unbounded circuits of depth d with layers $V_0, V_1, \ldots, V_d$
z: input for C
$w : V \mapsto \mathbb{N}$
$g \in V$

Output : (true, $w(C, z, g)$) if w is min-isolating for (C, z); (false, $-$) otherwise

```
(n, s) ← (0, 0);
foreach v ∈ V_0 in lex order do
    if v(z) = 1 then (n, s) ← (n + 1, s + w(v));
(isolating, k) ← (true, 0);
if g ∈ V_0 then
    if g(z) = 1 then ω ← w(g) else ω ← ∞;
while isolating and k < d do
    (isolating, n, s) ← PROMISEWEIGHTCHECK(C, z, w, k, n, s);
    if isolating and g ∈ V_{k+1} then
        ω ← PROMISEWEIGHTEVAL(C, z, w, k + 1, n, s, g);
    k ← k + 1;
if isolating then
    accept and return (true, ω)
else accept and return (false, −);
```

w is min-isolating for $(C, z, V_{\leq k})$. The call to PROMISEWEIGHTCHECK returns the values for $isolating$, n and s that satisfy the first two invariants for $k + 1$. Moreover, if the call indicates that w is min-isolating for $V_{\leq k+1}$ and $g \in V_{k+1}$, then PROMISEWEIGHTEVAL is called on the input $(C, z, w, k+1, n_{k+1}, s_{k+1}, g)$, and ω is set to its return value. By the specification of PROMISEWEIGHTEVAL, this means that the third invariant holds for $k + 1$. Thus, the body maintains all three invariants.

The loop either halts because $isolating$ becomes false, in which case the call to WEIGHTCHECKEVAL correctly returns (false, $-$) by the first invariant; or else it halts with $isolating =$ true and $k = d$, in which case it correctly returns

(true, $w(C, g, z)$) by the first and the third invariant. Thus, WEIGHTCHECKEVAL is correct.

As for unambiguity, note that WEIGHTCHECKEVAL is deterministic modulo the runs of the calls to PROMISEWEIGHTCHECK and PROMISEWEIGHTEVAL. As the argument of each call satisfies the respective promises, and both PROMISEWEIGHTCHECK and PROMISEWEIGHTEVAL are unambiguous on inputs that satisfy their promise, all of those runs are unambiguous. It follows that WEIGHTCHECKEVAL is unambiguous (because of the convention that any computation path on which a call rejects WEIGHTCHECKEVAL also rejects).

Excluding subroutine calls, the machine WEIGHTCHECKEVAL runs in time $\mathrm{poly}(\log(W), n)$ and in space $O(\log(W) + \log(n))$, and does not need access to the stack. Each call to the machine PROMISEWEIGHTCHECK takes time $\mathrm{poly}(2^d, \log(W), n)$ and space $O(\log(W) + \log(n))$ with the use of a stack, as does the call to PROMISEWEIGHTEVAL. It follows that WEIGHTCHECKEVAL runs in time $\mathrm{poly}(2^d, \log(W), n)$ and space $O(\log(W) + \log(n))$ with the use of a stack (as the space needed for subsequent calls can be reused).

Since the calls do not need access to the stack on REACHABILITY instances, neither does the machine WEIGHTCHECKEVAL on those instances. This concludes the proof of Claim 2.30. □

The machines $\text{WEIGHTCHECK}^{(\mathrm{cert})}$ and $\text{WEIGHTEVAL}^{(\mathrm{cert})}$ in the statement of Lemma 2.27 immediately follow from the machine WEIGHTCHECKEVAL in the statement of Claim 2.30. □

Lemma 2.27 trivially implies Lemma 2.18. Lemma 2.27 also yields the following variant of Lemma 2.11 in the setting of REACHABILITY and NL.

Lemma 2.31. *There exist unambiguous machines* $\text{WEIGHTCHECK}^{(\mathrm{reach,block})}$ *and* $\text{WEIGHTEVAL}^{(\mathrm{reach,block})}$ *such that for every layered digraph* $G = (V, E)$ *of depth* $d = 2^\ell$ *for for some* $\ell \in \mathbb{N}$ *and on* n *vertices, for every weight assignment* $w : V \mapsto \mathbb{N}$*, and* $s, t \in V$*:*

(i) WEIGHTCHECK$^{(\mathrm{reach},\mathrm{block})}(G, w)$ *decides whether* w *is min-isolating for* $(G, A_{\leq \ell})$ *where* $A_{\leq \ell} \doteq \bigcup_{k \leq \ell} A_k$ *and* A_k *is given by* (2.18)*.*

(ii) WEIGHTEVAL$^{(\mathrm{reach},\mathrm{block})}(G, w, s, t)$ *computes* $w(G, s, t)$ *provided that* w *is min-isolating for* $(G, A_{\leq \ell})$*.*

Both machines run in time $\mathrm{poly}(\log(W), n)$ *and space* $O(\log(W) + \log(n))$*, where* W *denotes an upper bound on the finite values of* $w(G, u, v)$ *for* $u, v \in V$*.*

Proof sketch. The lemma follows from Lemma 2.27 via the connection between REACHABILITY and CIRCUIT CERTIFICATION developed in Section 2.4.3. The "moreover" part of Lemma 2.27 is what allows us to eliminate the need for a stack in this setting. □

As mentioned in Section 2.4.3, Lemma 2.31 is sufficient for deriving the unambiguous simulations of NL given by Theorem 2.2, for which we used Lemma 2.11 in Section 2.3.2. Lemma 2.11 deals with min-isolation for G, i.e., for $(G, V \times V)$ instead of for $(G, A_{\leq \ell})$. It can be proved in a similar way as Lemma 2.18 – the same type of similarity as exists between the weight assignment generator constructions for REACHABILITY and for CIRCUIT CERTIFICATION.

In fact, the constructions in [76] carry through with the weaker requirement of min-isolation for $(G, \{s\} \times V)$. Due to the recursive nature of our weight assignment generator (Theorem 2.9) and the step-wise selection of the seed σ in our unambiguous simulations (Theorem 2.2), the requirement of min-isolation for $(G, \{s\} \times V)$ does not seem compatible with our approach. The stronger requirement of min-isolation for $(G, V \times V)$ is, as is the incomparable requirement of min-isolation for $(G, A_{\leq \ell})$.

3

Polynomial Identity Testing of Bounded-Read Arithmetic Formulas

3.1 Introduction

In this chapter we present our results on polynomial identity testing of constant-read arithmetic formulas. The model of read-restricted formulas was first investigated by Shpilka and Volkovich [83]. They constructed a $\mathrm{poly}(s)$-time whitebox identity test and a $s^{O(\log s)}$-time blackbox identity test for size s read-once formulas. Using their results for a single read-once formula, [83] constructed identity tests with similar running times for sums of constantly many read-once formulas. Recently, Minahan and Volkovich [69] gave a $\mathrm{poly}(s)$-time blackbox test for read-once formulas. Combined with the techniques in [83], their result yields a $\mathrm{poly}(s)$-time blackbox test for sums of constantly many read-once formulas.

Anderson et al. [8] constructed identity tests for constant-read multilinear formulas. Constant-read multilinear formulas are a strictly more powerful model than sums of read-once formulas [8]. Anderson et al. give a $\mathrm{poly}(s)$-time whitebox test, and a $s^{O(\log s)}$-time blackbox test for constant-read multilinear formulas. Their identity tests are obtained by iteratively applying the following steps:

- Step 1: Reduce testing read-$(k+1)$ multilinear formulas to testing the sum of two read-k multilinear formulas, and
- Step 2: reduce testing the sum of two read-k multilinear formulas to testing

a single read-k multilinear formula.

We attempt to follow the same blueprint to construct deterministic identity tests for constant-read formulas without the multilinear restriction. Building on ideas in [8], we show that an analogue of Step 1 holds in the non-multilinear setting. In particular, we show that there is a polynomial time whitebox reduction and a quasi-polynomial time blackbox reduction from testing read-$(k+1)$ formulas to testing $\Sigma^{(4)}$-read-k formulas, where a $\Sigma^{(m)}$-read-k formula is a formula that is the sum of m read-k formulas.

Theorem 3.1 (read-$(k+1) \leq \Sigma^{(4)}$-read-k)**.** *Let k be a positive integer.*

(a) If there is a $T(n)$-time deterministic blackbox identity test for $\Sigma^{(4)}$-read-k formulas over n variables, then there is a $\mathrm{poly}(T(n)) \cdot n^{O(\log n)}$-time deterministic blackbox identity test for read-$(k+1)$ formulas over n variables.

(b) Given a deterministic identity test for size-s Σ^4-read-k formulas that runs in time $T(s)$, there is an identity test for size-s read-$(k+1)$ formulas that runs in time $\mathrm{poly}(s) \cdot T(s)$.

Using Theorem 3.1, and identity tests for sums of read-once formulas [69, 83] and multilinear read-3 formulas [8], we construct identity tests for read-2 and read-3 formulas.

Theorem 3.2. *There is a $n^{O(\log n)}$-time deterministic blackbox identity test and a $\mathrm{poly}(s, n)$-time deterministic whitebox identity test for size-s read-3 formulas over n variables.*

Prior to this work, no non-trivial blackbox identity tests were known for read-2 and read-3 formulas. A polynomial time whitebox identity test for read-2 and read-3 formulas was previously constructed by Mahajan et al. [67]. Anderson et al. [9], constructed a $2^{O(n^{1-1/2^{k-1}})}$-time whitebox test for polynomial width read-k algebraic branching programs. Since every size-s read-k formula is

computable by a read-k algebraic branching program of width $\mathrm{poly}(s)$, [9] gives a subexponential time whitebox test for read-k formulas.

We also report progress towards obtaining an analogue of Step 2 *without* the multilinear restriction for all constants k. The test for sums of constantly many read-once formulas in [83] hinges on a so-called *hardness of representation* result for sums of read-once formulas. The hardness of representation result shows that for any formula, $F \in \mathbb{F}[x_1, \ldots, x_n]$, that is the sum of constantly many read-once formulas, there is a shift $\sigma \in \mathbb{F}^n$ such that $F(\boldsymbol{x} + \boldsymbol{\sigma})$ is not equivalent to the monomial $M_{[n]} \doteq \prod_{i \in [n]} x_i$. [8] prove a similar result for sums of constantly many read-k multilinear formulas.

Implicit in [83] and [8] is that their hardness of representation results imply that the shifted polynomial $F(\boldsymbol{x} + \boldsymbol{\sigma})$ must contain a monomial of small *support*, i.e., the monomial expansion of $F(\boldsymbol{x} + \boldsymbol{\sigma})$ contains a monomial with non-zero coefficient that depends on a small number of variables. Polynomials that contain a monomial of support ℓ are said to have ℓ-support concentration. There are known $n^{O(\ell)}$-time blackbox identity tests for such polynomials [4, 83].

Thus, the hardness of representation results in [83] and [8] imply identity tests for sums of read-once and sums of read-k multilinear formulas, provided the shift $\boldsymbol{\sigma}$ is efficiently computable. In [83] constructing $\boldsymbol{\sigma}$ reduces to testing a single read-once formula, whereas in [8] constructing the shift reduces to testing a single read-k multilinear formula, giving us the reduction in Step 2. This technique of constructing a shift $\boldsymbol{\sigma}$ for polynomials in a class $\mathcal{P}$, so that the shifted polynomials in $\mathcal{P}$ have low-support concentration has been used to construct identity tests for a variety of arithmetic models [4, 39, 43].

We present two results about low-support concentration for sums of read-k formulas without the multilinear restriction, for arbitrary constants k. Our first result generalizes the hardness of representation result of [83]. Let $D_x^k F$ denote the k-th order partial derivative of F with respect to the variable x.

Theorem 3.3. *Let $k \in \mathbb{N}$. Let $F = \sum_{r=1}^{m} F_r \in \mathbb{F}[x_1, \ldots, x_n]$ be a non-zero sum of m read-k formulas $F_1, \ldots, F_m$. Let $\boldsymbol{\sigma} \in \mathbb{F}^n$ be a common non-zero of the*

non-zero formulas in the set $\{D^k_{x_j} F_r \mid r \in [m], x_j \in [n]\}$). *If* $k(k+2)m < n$, *then* $M^k_{[n]} \nmid F(\boldsymbol{x} + \boldsymbol{\sigma})$.

The proof of Theorem 3.3 builds on ideas developed in [83] for the $k = 1$ case. We note that the main technical lemma in [83], underlying their proof of Theorem 3.3 for the $k = 1$ case, contains an error. We fix this error and generalize their ideas to prove Theorem 3.3 for all $k \geq 1$. This requires a rather delicate argument (see Lemma 3.24 and the discussion that follows it).

For $k = 1$, the theorem implies that $F(\boldsymbol{x} + \boldsymbol{\sigma})$ has $(3m + 1)$-support concentration. It is unclear to us whether Theorem 3.3 can be used to establish low-support concentration for larger values of k.

Our second result shows that a portion of the proof of the hardness of representation result in [8] continues to hold in the non-multilinear setting. To prove their hardness of representation result, [8] use partial derivatives and variable substitutions to reduce the sum of read-k multilinear formulas to an α-split multilinear formula with similar top fan-in. A formula $F = \sum_{r \in [m]} F_r$ is an α-split formula if each F_r is a product of subformulas that depend on at most an α fraction of the variables. [8] prove a hardness of representation result for multilinear α-split formulas, and lift the result to sums of multilinear read-k formulas. Their proof of the hardness of representation result for multilinear α-split formulas builds on ideas in [55].

We prove a hardness of representation result for bounded degree α-split formulas. This result may be proved using techniques in [55] and [8], but our proof is more direct and does not rely on the notion of 'structural witnesses' for depth-3 circuits that was central to the proofs in [55] and [8].

Theorem 3.4. *Let* $k, m \in \mathbb{N}$ *and* $\alpha = \frac{1}{km^2}$. *Let* $F = \sum_{r \in [m]} F_r$ *be a non-zero* α*-split formula with* $\cup_{r \in [m]} \mathrm{var}(F_r) = [n]$, *such that for every* $i \in [n]$, $r \in [m]$, *the degree of* x_i *in* F_r *is at most* k *and* $x_i \nmid F_r$. *Then,* $M_{[n]} \nmid F$.

Organization In Section 3.2 we introduce some notation and preliminaries including relevant results from [83] and [8]. In Section 3.3 we prove Theorem 3.1

and Theorem 3.2. In Section 3.4 we prove Theorem 3.3, and in Section 3.5 we prove Theorem 3.4.

3.2 Preliminaries

3.2.1 Notation

Let $[n]$ denote the set of variables $\{x_1, \ldots, x_n\}$. We often associate indices in $[n]$ with the corresponding variable, and sets of indices $V \subseteq [n]$ with the corresponding set of variables.

An arithmetic formula $F \in \mathbb{F}[x_1, \ldots, x_n]$ is a binary tree in which every internal node has two children. The leaves of the tree are labeled by variables $x \in [n]$ or field constants in $\mathbb{F}$, and internal nodes are labeled by either addition or multiplication.

The polynomial computed at a node/gate g in F is defined in the canonical fashion. The polynomial computed by F is the polynomial computed at the root of F. We often overload notation and let F also refer to the polynomial computed by the formula F. Similarly a gate g in F will often refer to the polynomial computed at g, as well as the subformula rooted at g. The size of the formula F is the number of nodes in F.

We say a polynomial $F \in \mathbb{F}[x_1, \ldots, x_n]$ depends on a variable x_i if there are two assignments $\sigma_1, \sigma_2 \in \mathbb{F}^n$ such that σ_1 and σ_2 differ only at the i-th coordinate, and $F(\sigma_1) \neq F(\sigma_2)$. We let $\mathrm{var}(F)$ denote the set of variables that F depends on. For a formula F and an integer $k \in \mathbb{N}$, $\mathrm{var}_k(F)$ denotes the set of variables in $\mathrm{var}(F)$ that are read exactly k times by F.

Let $r_i(F)$ denote the number of times the variable x_i is read by the formula F, and $\boldsymbol{r}(F)$ the vector $(r_1(F), \ldots, r_n(F))$. For $k \in \mathbb{N}$, we let $\boldsymbol{k}$ denote the vector $(k, \ldots, k) \in \mathbb{N}^n$.

For a subset $V \subseteq [n]$, we let M_V denote the monomial $\prod_{i \in V} x_i$.

3.2.2 Partial Derivatives

A critical difference between the multilinear and non-multilinear settings is that partial derivatives of multilinear read-k formulas are computable by multilinear read-k formulas. The analogue no longer holds in the non-multilinear setting, and is one of the key obstacles in transferring the reduction in Step 2 of [8] to the non-multilinear setting.

Nevertheless, the k-th order derivatives of a read-k formula are computable by read-k formulas (Proposition 3.6). We use this observation crucially to prove Theorem 3.3.

In order to prove Theorem 3.3 over finite fields, we work with the following variant of a partial derivative. Let P be a polynomial in $\mathbb{F}[x_1, \ldots, x_n]$, and let $\boldsymbol{r}$ be a vector in $\mathbb{F}^n$. For a subset $U \subseteq [n]$, we denote by $D^{\boldsymbol{r}}_U P$ the coefficient of the monomial $\prod_{i \in U} x_i^{r_i}$ in P. For example, if P is the polynomial $x_1^3 x_2^2 + x_1^2 x_2^2$, then $D^{(2,2)}_{\{x_1,x_2\}} = x_1 + 1$. For univariate monomials, we write $D^{r_i}_{x_i} P$ instead of $D^{\boldsymbol{r}}_{\{x_i\}}$, where r_i is the i-th coordinate in $\boldsymbol{r}$. We refer to $D^{\boldsymbol{r}}_U P$ as the partial derivative of P with respect to the monomial $\prod_{i \in U} x_i^{r_i}$.

Note that

$$D^{r_i}_{x_i} P = \overbrace{D_{x_i} \cdots (D_{x_i}}^{r \text{ times}} P),$$

and

$$D^{\boldsymbol{r}}_U P = D^{r_{i_1}}_{x_{i_1}} \cdots D^{r_{i_{|U|}}}_{x_{i_{|U|}}} P,$$

where $U = \{i_1, \ldots, i_{|U|}\}$.

For a variable x_i, and field constant $\alpha \in \mathbb{F}$, we denote by $P|_{x_i=\alpha}$ the polynomial obtained from P by substituting α for the variable x_i. Observe that if $x_i \notin U$, then $D^{\boldsymbol{r}}_U(P|_{x_i=\alpha}) = (D^{\boldsymbol{r}}_U P)|_{x_i=\alpha}$.

We will need the following basic facts about partial derivatives of arithmetic formulas.

Proposition 3.5. *Let $F \in \mathbb{F}[x_1, \ldots, x_n]$ be an arithmetic formula, and let $U \subseteq [n]$.*

(Product Rule) If $F = F_1 \times F_2$*,*

$$D_U^{\boldsymbol{r}(F)} F = D_U^{\boldsymbol{r}(F_1)} F_1 \cdot D_U^{\boldsymbol{r}(F_2)} F_2.$$

(Chain Rule) Let g *be a gate in the formula* F *that reads at least one variable in* U*, and let* Q *denote the formula obtained from* F *by replacing* g *with a fresh variable* y*. Then,*

$$D_U^{\boldsymbol{r}(F)} F = D_U^{\boldsymbol{r}(g)} g \cdot D_y(D_U^{\boldsymbol{r}(Q)} Q).$$

The following proposition shows that if $F \in \mathbb{F}[x_1, \ldots, x_n]$ is a read-k formula, and $U \subseteq [n]$, then there is a read-k formula F' that computes the polynomial $D_U^{\boldsymbol{k}} F$.

Proposition 3.6. *Let* $F \in \mathbb{F}[x_1, \ldots, x_n]$ *be an arithmetic formula of size* s*. For any subset* $U \subseteq [n]$*, there is a arithmetic formula* F' *of size at most* s *that computes the polynomial* $D_U^{\boldsymbol{r}(F)} F$*, and satisfies* $r_i(F') \leq r_i(F)$*, for all* $x_i \in [n]$*.*

Proof. The proof is by induction on the size of F, $s(F)$.

If $s(F) = 1$, $D_U^{\boldsymbol{r}(F)} F$ is a constant polynomial and the statement holds.

Now, suppose $s(F) > 1$. Assume that there is a variable in U that is read by F, otherwise $D_U^{\boldsymbol{r}(F)} F = F$. Let g denote the top-gate in F, and let g_1, g_2 denote the children of g. By the induction hypothesis, for $j \in \{1, 2\}$, there is a formula g_j' computing the polynomial $D_U^{\boldsymbol{r}(g_j)} g_j$, such that $r_i(g_j') \leq r_i(g_j)$ for all $x_i \in [n]$. We consider two cases based on whether g is a sum gate or product gate.

Suppose g is a sum gate. If both g_1 and g_2 witness reads of variables in U, $D_U^{\boldsymbol{r}(F)} F$ is the zero polynomial, and the statement holds. Otherwise, there is a $j \in \{1, 2\}$ such that g_j witnesses all reads of variables in U. Then $D_U^{\boldsymbol{r}(F)} F = D_U^{\boldsymbol{r}(g_j)} g_j$, and $F' \doteq g_j'$ is the required formula.

If g is a product gate, then by Proposition 3.5, $D_U^{\boldsymbol{r}(F)} F = D_U^{\boldsymbol{r}(g_1)} g_1 \cdot D_U^{\boldsymbol{r}(g_2)} g_2$. So, $F' = g_1' \cdot g_2'$ computes $D_U^{\boldsymbol{r}(F)} F$. Note that for all $x_i \in [n]$, $r_i(F') = r_i(g_1') + r_i(g_2') \leq r_i(g_1) + r_i(g_2) = r_i(F)$. This completes the proof. □

3.2.3 Hitting Set Generators

A hitting set generator $\mathcal{G} : \mathbb{F}^{s(n)} \to \mathbb{F}^n$ for a class $\mathcal{P}$ of polynomials is a family of polynomial maps $\{G_n\}_{n\in\mathbb{N}}$, where each G_n is an element of $(\mathbb{F}[y_1, \ldots, y_{s(n)}])^n$, such that for every non-zero n-variate polynomial $P \in \mathcal{P}$, $P(G_n) \not\equiv 0$.

We say a hitting set generator has degree $d(n)$ if for every $n \in \mathbb{N}$, the entries in the n-tuple G_n are polynomials of degree at most $d(n)$. We say a hitting set generator is $t(n)$-explicit if there is an algorithm that takes as input a point $\alpha \in \mathbb{F}^{s(n)}$, and returns $G_n(\alpha)$ in time $t(n)$[1].

Explicit hitting set generators are equivalent to blackbox identity tests (see [83]).

Proposition 3.7. *Let $\mathcal{P}$ be a class of polynomials such that for all $n \in \mathbb{N}$, n-variate polynomials in P have total degree at most $D(n)$. If $\mathcal{G} : \mathbb{F}^{s(n)} \to \mathbb{F}^n$ is a $t(n)$-explicit degree $d(n)$ hitting set generator for $\mathcal{P}$, there is a deterministic blackbox identity test for $\mathcal{P}$ that runs in time $t(n) \cdot (D(n) \cdot d(n))^{s(n)}$, and queries points in a field extension of size $O(D(n) \cdot d(n))$.*

Conversely, if there is a $T(n)$-time deterministic blackbox identity test for $\mathcal{P}$, then there is a $\mathrm{poly}(n, T(n))$-explicit degree $n - 1$ hitting set generator $\mathcal{G} : \mathbb{F}^{\lceil \log T(n) \rceil} \to \mathbb{F}^n$ for $\mathcal{P}$.

The SV Generator Shpilka and Volkovich defined a polynomial map $G_{n,w} : \mathbb{F}^{2w} \to \mathbb{F}^n$ that contains in its image the set of all points in $\mathbb{F}^n$ of hamming weight at most w.

Definition 3.8 (SV generator [83])**.** *Let $a_1, \ldots, a_n$ be distinct points in $\mathbb{F}$, and for $i \in [n]$ let $L_i(y) \doteq \prod_{j\in[n], j\neq i} \frac{(y-a_j)}{(a_i-a_j)}$ denote the i-th Lagrange interpolant. For $w \in \mathbb{N}$, the polynomial map $G_{n,w} : \mathbb{F}^{2w} \to \mathbb{F}^n$ is defined as*

$$G_{n,w}(y_1, \ldots, y_w, z_1, \ldots, z_w) \doteq (\sum_{j\in[w]} z_j L_1(y_j), \ldots, \sum_{j\in[w]} z_j L_n(y_j)).$$

[1] Each field operation counts as a single step

Shpilka and Volkovich [83] showed that $\{G_{n,\lceil\log(n+1)\rceil}\}_{n\in\mathbb{N}}$ is a hitting set generator for read-once formulas, and using their hardness of representation result concluded that $\{G_{n,3m+\lceil\log(n+1)\rceil}\}_{n\in\mathbb{N}}$ is a hitting set generator for $\Sigma^{(m)}$-read-once formulas. Minahan and Volkovich [69] improved on the result of Shpilka and Volkovich and showed that $\{G_{n,1}\}_{n\in\mathbb{N}}$ is a hitting set generator for read-once formulas. Combined with the hardness of representation result of [83], this implies following.

Theorem 3.9 ([69, 83])**.** *The family of polynomial maps $\{G_{n,3m+1}\}_{n\in\mathbb{N}}$ is a hitting set generator for $\Sigma^{(m)}$-read-once formulas.*

[8] used the polynomial map $G_{n,w}$ to construct hitting set generators for constant-read multilinear formulas.

Theorem 3.10 ([8])**.** *There exists a function $w_k = k^{O(k)}$ such that the family of polynomial maps $\{G_{n,w_k+k\lceil\log(n+1)\rceil}\}_{n\in\mathbb{N}}$ is a hitting set generator for multilinear read-k formulas.*

We will need the following simple property of $G_{n,w}$.

Proposition 3.11 ([8])**.** *For $i \in [n]$, let $P = \sum_{j=0}^{d} x_i^j P_j$, where each P_j is a polynomial in $\mathbb{F}[x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n]$. If there is a polynomial map $\mathcal{G}_n$ with range in $\mathbb{F}^n$ such that $P_j(\mathcal{G}_n) \not\equiv 0$, for some $j \in [d]$, then $\mathcal{G}_n + G_{n,1}$ hits the polynomial P.*

3.3 Reducing testing read-$(k+1)$ formulas to testing $\Sigma^{(4)}$-read-k formulas

In this section we prove Theorems 3.1 and 3.2. We develop our results in the blackbox and whitebox settings separately. In Section 3.3.1, we prove Part (a) of Theorem 3.1 and give the blackbox identity test for read-3 formulas, and in Section 3.3.2 we establish the whitebox counterparts.

3.3.1 Blackbox setting

In [8] the blackbox version of the reduction from testing read-$(k+1)$ multilinear formulas to testing $\Sigma^{(2)}$-read-k multilinear formulas is based on a structural result called Fragmentation. The Fragmentation Lemma in [8] states that for any read-$(k+1)$ multilinear formula F over n variables, there exists a variable x such that $D_x F$ is a product of at most one $\Sigma^{(2)}$-read-k multilinear formula and read-$(k+1)$ multilinear formulas that depend on fewer than $n/2$ variables. While this result does not hold without the multilinear restriction, a weaker statement holds.

Lemma 3.12 (Fragmentation)**.** *Let $k \in \mathbb{N}$, and let F be a non-zero read-$(k+1)$ formula over n variables with $|\mathrm{var}_{(k+1)}|(F) = \ell > 0$. F can be written as*

$$H \cdot Q_1 \cdots Q_m + R,$$

where

(i) H is a minimal subformula of F with $\mathrm{var}_{(k+1)}(H) \cap \mathrm{var}_{(k+1)}(F) \neq \emptyset$.

(ii) $Q_1, \ldots, Q_m$ are non-zero read-$(k+1)$ subformulas of F such that for all $i \in [m]$, $\mathrm{var}(Q_i) \cap \mathrm{var}_{(k+1)}(H) = \emptyset$.

(iii) For all $i \in [m]$, $|\mathrm{var}_{(k+1)}(Q_i)| \leq \ell/2$.

(iv) R is the read-$(k+1)$ formula obtained from F by replacing H with the zero polynomial.

Proof. The proof is constructive. Start at the root of F and trace a path down the formula by iteratively picking the child node that witnesses all $k+1$ occurrences of a larger number of variables in the set $\mathrm{var}_{(k+1)}(F)$, while breaking ties arbitrarily. This path terminates at a gate g that is either the sum or product of two read-k formulas. Let H denote the subformula rooted at g. F can be written as $H \cdot Q_1 \cdots Q_m + R$, where $Q_1, \ldots, Q_m$ are the subformulas rooted at off-path children of product gates on the path from the root to g, and R is the formula

obtained from F by replacing g with zero. By construction, H is a minimal subformula of F with $\mathrm{var}_{(k+1)}(H) \cap \mathrm{var}_{(k+1)}(F) \neq \emptyset$, and as a result the Q_i's must be non-zero. Furthermore, for all $i \in [m]$, $\mathrm{var}_{(k+1)}(H) \cap \mathrm{var}(Q_i) = \emptyset$ and $|\mathrm{var}_{(k+1)}(Q_i)| \leq \ell/2$. □

The theorem below is a restatement of Part (a) of Theorem 3.1 in terms of hitting set generators. By Proposition 3.7, Theorem 3.13 implies Part (a) of Theorem 3.1.

Theorem 3.13 (read-$(k+1) \leq \Sigma^{(4)}$-read-k, blackbox setting)**.** *For an integer $k \in \mathbb{N}$, let $\mathcal{G}$ be a hitting set generator for non-zero $\Sigma^{(4)}$-read-k formulas. Then $\{\mathcal{G}_n + G_{n,\lceil \log(n+1) \rceil}\}_{n \in \mathbb{N}}$ is a hitting set generator for non-zero read-$(k+1)$ formulas.*

Proof. Fix $n \in \mathbb{N}$. Let F be a non-zero read-$(k+1)$ formula over n variables. We show that for every $\ell \in \mathbb{N} \cup \{0\}$, if $\mathrm{var}_{(k+1)}(F) \leq \ell$ then $\mathcal{G}_n + G_{n,\lceil \log(\ell+1) \rceil}$ hits F. The proof is by induction on ℓ.

Note that if $\ell = 0$, then F can be computed by a read-k formula, and by definition, $\mathcal{G}_n$ hits F. This is our base case. For the inductive step, suppose that $\ell > 0$. F can be written as $H \cdot Q_1 \cdots Q_m + R$, where the Q_i's, H, and R are as in the statement of Lemma 3.12. By the induction hypothesis, $\mathcal{G}_n + G_{n,\lceil \log(\lfloor \ell/2 \rfloor + 1) \rceil}$ hits each Q_i, and as a result the product $Q_1 \cdots Q_m$.

Fix a variable $x \in \mathrm{var}_{(k+1)}(H)$. H can be written as $\sum_{i=0}^{k+1} x^i P_i$, where the P_i's do not depend on x. Plugging this into the expression for F, we get

$$F = (\sum_{i=1}^{k+1} x^i P_i) Q_1 \cdots Q_m + R + P_0 Q_1 \cdots Q_m.$$

By Proposition 3.11 and the fact that $\mathcal{G}_n + G_{n,\lceil \log(\lfloor \ell/2 \rfloor + 1) \rceil}$ hits $Q_1 \cdots Q_m$, if $\mathcal{G}_n + G_{n,\lceil \log(\lfloor \ell/2 \rfloor + 1) \rceil}$ hits P_i, for some $i > 0$, then $\mathcal{G}_n + G_{n,\lceil \log(\ell+1) \rceil}$ hits F.

Thus, we only need to show that there is an $i \in [k+1]$ such that P_i is hit by $\mathcal{G}_n + G_{n,\lceil \log(\lfloor \ell/2 \rfloor + 1) \rceil}$. This is equivalent to showing that $x \in \mathrm{var}(H')$, where

H' is the formula obtained from H by substituting $\mathcal{G}_n + G_{n,\lceil \log(\lfloor \ell/2 \rfloor + 1)\rceil}$ for all variables except x. We consider two cases:

Case 1: H is the sum of two read-k formulas. Since $x \in \mathrm{var}(H)$, there exist two values $\alpha, \beta \in \mathbb{F}$ such that $H|_{x \leftarrow \alpha} - H|_{x \leftarrow \beta} \not\equiv 0$. By the definition of $\mathcal{G}_n$, we have that $\mathcal{G}_n + G_{n,\lceil \log(\lfloor \ell/2 \rfloor + 1)\rceil}$ hits the $\Sigma^{(4)}$-read-k formula $H|_{x \leftarrow \alpha} - H|_{x \leftarrow \beta}$. Thus, $H'|_{x \leftarrow \alpha} - H'|_{x \leftarrow \beta} \not\equiv 0$, and this implies $x \in \mathrm{var}(H')$.

Case 2: H is the product of two read-k formulas H_1, H_2. Since $x \in \mathrm{var}(H)$, at least one of H_1, H_2 depends on x and the other must be non-zero. Let H_1' and H_2' denote the polynomials obtained from H_1 and H_2 by substituting $\mathcal{G}_n + G_{n,\lceil \log(\lfloor \ell/2 \rfloor + 1)\rceil}$ for all variables other than x. If $x \in \mathrm{var}(H_i)$, then there exist two values $\alpha, \beta \in \mathbb{F}$ such that the $\Sigma^{(2)}$-read-k formula $H_i|_{x=\alpha} - H_i|_{x=\beta}$ is not identically zero. By definition of $\mathcal{G}_n$, this tells us that $H_i'|_{x=\alpha} - H_i'|_{x=\beta} \not\equiv 0$, or equivalently $x \in \mathrm{var}(H_i')$. If $x \notin \mathrm{var}(H_i)$, then H_i is a non-zero read-k formula and is hit by $\mathcal{G}_n + G_{n,\lceil \log(\lfloor \ell/2 \rfloor + 1)\rceil}$, i.e., H_i' is non-zero. Thus, at least one of H_1' and H_2' depends on x and the other is non-zero. It follows that $x \in \mathrm{var}(H')$.

□

Blackbox identity test for read-2 and read-3 formulas

A generator for read-2 formulas follows from Theorem 3.13 and Theorem 3.9.

Theorem 3.14. *$\{G_{n,13+\lceil \log(n+1)\rceil}\}_{n \in \mathbb{N}}$ is a hitting set generator for read-2 formulas.*

For read-3 formulas we don't have access to an explicit generator for $\Sigma^{(4)}$-read-2 formulas. We get around this by exploiting the fact that a non-constant read-3 formula $H = H_1 + H_2$ is either multilinear or depends on a variable x that has distinct degrees in the subformulas H_1 and H_2.

Theorem 3.15. *There exists a constant $w \in \mathbb{N}$ such that $\{G_{n,w+5\lceil\log(n+1)\rceil}\}_{n\in\mathbb{N}}$ is a hitting set generator for read-3 formulas.*

Proof. Let $\mathcal{G}$ be the sum of the generators in Theorem 3.10 with $k = 3$, and Theorem 3.14. Note that $\mathcal{G}$ is a hitting set generator for both read-2 formulas and multilinear read-3 formulas.

Fix $n \in \mathbb{N}$, and let F be a read-3 formula over n variables. We show that for every $\ell \in \mathbb{N} \cup \{0\}$, if $\mathrm{var}_3(F) \leq \ell$, then $\mathcal{G}_n + G_{n,\lceil\log(\ell+1)\rceil}$ hits F.

The proof is by induction on ℓ. In the base case, where $\ell = 0$, F is a read-2 formula and is hit by $\mathcal{G}_n$. For the inductive step, let $\ell > 0$. We use Lemma 3.12 and a case analysis to show that F can be expressed as $x^d\mathcal{Q} + \mathcal{R}$, where

(i) $\mathcal{Q}$ is non-zero,

(ii) $\mathcal{Q}$ does not depend on x,

(iii) $\mathcal{Q}$ is the product of read-2 formulas, multilinear read-3 formulas, and read 3 formulas that depend on at most $\ell/2$ read-3 variables, and

(iv) the degree of x in $\mathcal{R}$ is smaller than d.

Note that if F can be expressed in this way, then by the induction hypothesis and Proposition 3.11, F is hit by $\mathcal{G}_n + G_{n,\lceil\log(\ell+1)\rceil}$.

Now, F can be written as $H \cdot Q_1 \cdots Q_m + R$, where H, the Q_i's, and R are as in the statement of Lemma 3.12.

Case 1: Suppose H is a multilinear read-3 formula, and let x be a variable in $\mathrm{var}_3(H)$. H can be written as $x \cdot Q' + R'$, where Q' and R' are multilinear read-3 formulas that don't depend on x. So, F can be written as $x\mathcal{Q} + \mathcal{R}$, where $\mathcal{Q}$ and $\mathcal{R}$ satisfy (i)-(iv) with $d = 1$.

Case 2: Suppose H is not multilinear, and that there is a variable x of degree 3 in H. Since H is a minimal read-3 subformula of F and contains a variable of degree 3, it is the product of two read-2 formulas H_1, H_2, and x has degree

2 in one of H_1, H_2 and degree 1 in the other. Assume that x has degree 2 in H_1, then H_1 can be written as $x^2Q_1' + R_1'$, where Q_1' is a read-2 formula that doesn't depend on x, and the degree of x in R_1' is at most 1. H_2 can be written as $xQ_2' + R_2'$, where Q_2', R_2' are read-2 formulas that don't depend on x. So, F can be written as $x^3\mathcal{Q} + \mathcal{R}$, where $\mathcal{Q}$ and $\mathcal{R}$ satisfy (i)-(iv) with $d = 3$.

Case 3: Otherwise, H is not multilinear and there are no variables of degree 3 in H. Let x be a variable of degree 2 in H. We are in one of the following cases:

- H is the sum of two read-2 formulas H_1 and H_2. In this case x has degree 2 in one of H_1, H_2. Suppose x has degree 2 in H_1. Then H_1 can be written as $x^2Q' + R'$, where Q' is a read-2 formula that doesn't depend on x and the degree of x in R' is at most 1. So, H can be written as $x^2Q'' + R''$, where Q'' is a non-zero read-2 formula that doesn't depend on x, and the degree of x in R'' is at most 1.
- H is the product of two read-2 formulas H_1 and H_2, and x has degree 2 in one of them, H_1 say. Then the read-2 formula H_2 doesn't depend on x, and H_1 can be written $x^2Q' + R'$, where Q' is a read-2 formula that doesn't depend on x and the degree of x in R' is at most 1. So, H can be written as $x^2Q'' + R''$, where Q'' is a non-zero product of two read-2 formulas that don't depend on x, and the degree of x in R'' is at most 1.
- H is the product of two read-2 formulas H_1, H_2, and the degree of x in both H_1, H_2 is 1. In this case for $j = 1, 2$, H_j can be written as $xQ_j' + R_j'$, where Q_j' and R_j' are read-2 formulas that don't depend on x. So, H can be written as $x^2Q'' + R''$, where Q'' is the product of two non-zero read-2 formulas that don't depend on x, and the degree of x in R'' is at most 1.

In each of the three cases above, if for all $i \in [m]$, $x \notin \mathrm{var}(Q_i)$, then F can be written as $x^2\mathcal{Q} + \mathcal{R}$, where $\mathcal{Q}$ and $\mathcal{R}$ satisfy (i)-(iv) with $d = 2$.

On the other hand if $x \in \mathrm{var}(Q_i)$ for some $i \in [m]$, then we can rewrite Q_i as $x\tilde{Q}_i + \tilde{R}_i$, where $\tilde{Q}_i$ and $\tilde{R}_i$ are read-3 formulas that depend on at most $\ell/2$ read-3 variables and don't depend on x. Thus, again F can be expressed as $x^3\mathcal{Q} + \mathcal{R}$, where $\mathcal{Q}$ and $\mathcal{R}$ satisfy (i)-(iv) with $d = 3$.

□

3.3.2 Whitebox Setting

In this section we prove Part (b) of Theorem 3.1 and use it to construct a polynomial time identity test for read-3 formulas. The reduction is essentially the same as in the multilinear setting [8], and for this reason we only provide a sketch here.

The reduction uses the algorithm $\textsc{Reduce}_k$ described in Algorithm 3.1 as a subroutine. When given as input a read-$(k+1)$ formula, F, $\textsc{Reduce}_k$ processes

Algorithm 3.1: $\textsc{Reduce}_k(F)$

```
if F = F_1 op F_2 then
    F ← Reduce_k(F_1) op Reduce_k(F_2)
for each variable x that is read by the formula F do
    if x ∉ var(F) then
        F ← F|_{x=0}
    else if x appears k + 1 times in the formula F then
        F ← y, for a fresh variable y
return F
```

the gates of F bottom-up. At each gate g, the variables that g does not depend on are eliminated from the subformula rooted at g, and if g depends on a variable x and all $k+1$ reads of x appear in the subformula rooted at g, then g is replaced by a fresh variable.

Below we list some useful properties of the formula $\textsc{Reduce}_k(F)$.

(i) $\textsc{Reduce}_k(F)$ is a read-k formula.

(ii) If x is a variable that is read by the formula $\textsc{Reduce}_k(F)$, then $x \in \mathrm{var}(\textsc{Reduce}_k(F))$.

(iii) $F \equiv 0$ if and only if $\textsc{Reduce}_k(F) \equiv 0$.

(i) holds because for every variable that is read $k+1$ times, either the variable is eliminated because F doesn't depend on it or the minimal subformula that witnesses all $k+1$ reads of the variable is replaced by a fresh variable. (ii) holds because any variable that F does not depend on is set to zero. Finally, (iii) holds because the only variables that are set to zero are variables that F doesn't depend on, and a gate g in F is replaced by a fresh variable only if there is a variable x such that g depends on x, and x does not appear elsewhere in the formula.

The following lemma shows that there is an efficient test for read-$(k+1)$ formulas, provided Line 4 of the algorithm $\textsc{Reduce}_k$ can be implemented efficiently when 'op' is addition. We say a formula is a $\textsc{Reduce}_k$ formula, if it is the result of running the algorithm $\textsc{Reduce}_k$ on a read-$(k+1)$ formula.

Lemma 3.16. *If* $\mathrm{var}(F)$ *can be computed in time* $T(s)$ *for size-*s *read-*$(k+1)$ *formulas* F *that are the sum of two* $\textsc{Reduce}_k$ *formulas, then there is an identity test for size-*s *read-*$(k+1)$ *formulas that runs in time* $\mathrm{poly}(s) \cdot T(s)$.

Proof. We first show that there is a simple $\mathrm{poly}(s)$-time identity test for the formula $\textsc{Reduce}_k(F)$, where F is a size-s read-$(k+1)$ formula. If there is a variable x that is read by $\textsc{Reduce}_k(F)$ then by (ii), the formula is non-constant, and therefore not identically zero. If no variable is read by $\textsc{Reduce}_k(F)$, then it computes a constant, and we can evaluate $\textsc{Reduce}_k(F)$ to determine whether this constant is zero. Since the size of the formula $\textsc{Reduce}_k(F)$ is at most s, the test described above requires $\mathrm{poly}(s)$ time.

By property (iii) and the test described above, it is sufficient to show that $\textsc{Reduce}_k(F)$ can be computed in $T(s) \cdot \mathrm{poly}(s)$ time.

The main difficulty in computing $\textsc{Reduce}_k(F)$ is evaluating the predicate on Line 4. We have that if F is the sum of two $\textsc{Reduce}_k$ formulas then Line 4 can

be implemented in $T(s)$ time. We now show that for size-s read-$(k+1)$ formulas F that are products of two REDUCE$_k$ formulas, the predicate on Line 4 can be evaluated in $\mathrm{poly}(s)$ time. Let $F = F_1' \times F_2'$, where F_1' and F_2' are REDUCE$_k$ formulas. $x \in \mathrm{var}(F)$ if and only if one of F_1' and F_2' depends on x, and the other is non-zero. For $i \in \{1,2\}$, we can determine whether F_i' depends on a variable x, and whether F_i' is identically zero in $\mathrm{poly}(s)$ time using property (ii) and the identity test for REDUCE$_k$ formulas described above. Thus, the truth value of the predicate $x \notin \mathrm{var}(F)$ can be determined in $\mathrm{poly}(s)$ time. Therefore, for size-s read-k formulas F, REDUCE$_k(F)$ can be computed in $T(s) \cdot \mathrm{poly}(s)$ time. □

By Lemma 3.16, to establish the required reduction, we only need to show that for size-s read-$(k+1)$ formulas F that are sums of two REDUCE$_k$ formulas, $\mathrm{var}(F)$ can be computed in $\mathrm{poly}(s)$ time with access to an oracle for testing Σ^4-read-k polynomials. Let $F = F_1' + F_2'$, where F_1' and F_2' are REDUCE$_k$ formulas. By (i), F_1' and F_2' are both read-k formulas. In particular, the degree of any variable x in $F = F_1' + F_2'$ is at most k. Thus, $x \in \mathrm{var}(F)$ if and only if for any set of $k+1$ distinct field elements $\alpha_1, \ldots, \alpha_{k+1} \in \mathbb{F}$, there exists an index $i \in [k]$ such that $F|_{x=\alpha_i} - F|_{x=\alpha_{k+1}} \not\equiv 0$. For every $i \in [k]$, $F|_{x=\alpha_i} - F|_{x=\alpha_{k+1}}$ is a Σ^4-read-k formula. Therefore, the predicate $x \in \mathrm{var}(F)$ can be evaluated in $\mathrm{poly}(s)$ time with access to an oracle for testing Σ^4-read-k formulas.

Putting everything together we obtain Part (b) of Theorem 3.1.

Whitebox identity tests for read-2 and read-3 formulas

We will need the following result of [8].

Theorem 3.17 ([8])**.** *For $k \in \mathbb{N}$, there is a $\mathrm{poly}(s)$ deterministic polynomial identity test for size-s multilinear read-k formulas that runs in time $s^{k^{O(k)}}$.*

The following theorem is immediate from Part (b) of Theorem 3.1 and Theorem 3.17.

Theorem 3.18. *There is a deterministic* $\mathrm{poly}(s)$*-time identity test for read-*2 *formulas of size* s*.*

We now describe a polynomial time whitebox identity test for read-3 formulas. The key idea is the same as in the blackbox setting.

Theorem 3.19. *There is a deterministic* $\mathrm{poly}(s)$*-time identity test for read-*3 *formulas of size* s*.*

Proof. By Lemma 3.16, it is sufficient to show that for any size-s read-3 formula F, that is the sum of two $\textsc{Reduce}_2$ formulas F_1' and F_2', $\mathrm{var}(F)$ can be computed in $\mathrm{poly}(s)$ time.

First, note that for every variable x we can determine the degree of x in F_1' and F_2' in $\mathrm{poly}(s)$ time. If x doesn't appear in the formula F_i', then by property (ii), the degree of x is zero. If x is read by F_i' and there is a product gate g in F_i' such that both children of g read the variable x, then by item (ii), the degree of x at g is two, and $x \in \mathrm{var}(F_i')$. Since x does not appear elsewhere in the read-2 formula F_i', the degree of x in F_i' is two. If neither of the above cases hold, the degree of x is one.

If x doesn't appear in F, then $x \notin \mathrm{var}(F)$. If x is read by F, but has different degrees in F_1' and F_2', then $x \in \mathrm{var}(F)$. So assume that the degree of x is one in both F_1' and F_2'. For $i \in \{1,2\}$, F_i' can be written as $xQ_i + R_i$, where Q_i is the product of all off-path children Q_{ij}, $1 \leq j \leq \ell_i$, of product gates on the path from the leaf labeled x to the root of F_i', and R_i is $F_i'|_{x=0}$. Thus, determining whether $x \in \mathrm{var}(F)$ is equivalent to determining whether $Q_1 + Q_2 \not\equiv 0$.

We describe a $\mathrm{poly}(s)$ time identity test for $Q_1 + Q_2$. Note that each Q_{ij} is a $\textsc{Reduce}_2$ formula, and as a result satisfies property (ii). Thus, for every $i \in \{1,2\}$ and $j \in [\ell_i]$, we can determine the degree of every variable in the polynomial Q_{ij} in $\mathrm{poly}(s)$ time. For $i \in \{1,2\}$, Q_i contains a variable of degree 2 if and only if there is a $j \in [\ell_i]$ such that Q_{ij} contains a variable of degree 2, or there exist two indices $j, j' \in [\ell_i]$ such that the same variable has degree 1 in both Q_{ij} and $Q_{ij'}$. Therefore, we can determine whether one of Q_1 and Q_2 contains a

variable of degree 2 in $\mathrm{poly}(s)$ time. If one of Q_1 and Q_2 contains a variable of degree 2, then $Q_1 + Q_2 \not\equiv 0$. Otherwise, $Q_1 + Q_2$ is multilinear and we can use the identity test given by Theorem 3.17 to determine whether $Q_1 + Q_2 \not\equiv 0$ in $\mathrm{poly}(s)$ time. □

3.4 Hardness of Representation for Read-k Formulas

In this section we prove Theorem 3.3.

3.4.1 Proof Overview

Observe that by the choice of $\boldsymbol{\sigma}$ in Theorem 3.3, for every summand F_r and variables $x_i, x_j \in [n]$, if $D^k_{x_j} F_r(\boldsymbol{x} + \boldsymbol{\sigma}) \not\equiv 0$, then $D^k_{x_j} F_r(\boldsymbol{x} + \boldsymbol{\sigma})|_{x_i=0} \not\equiv 0$. In Section 3.4.3, we show that the hypothesis of Theorem 3.3 implies that each $F_r(\boldsymbol{x} + \boldsymbol{\sigma})$ satisfies a stronger property that we refer to as k-th order resilient to zero-substitutions [2].

Definition 3.20. *A polynomial $F \in \mathbb{F}[x_1, \ldots, x_n]$ is k-th order resilient to zero-substitutions if for every subset $U \subseteq [n]$, and variable $x_i \in [n] \setminus U$,*

$$D^{\boldsymbol{k}}_U F \not\equiv 0 \implies D^{\boldsymbol{k}}_U F|_{x_i=0} \not\equiv 0.$$

Thus, it suffices to prove the following theorem.

Theorem 3.21. *Let $k \in \mathbb{N}$. Let $F = \sum_{r=1}^{m} F_r \in \mathbb{F}[x_1, \ldots, x_n]$, be a non-zero sum of m read-k formulas $F_1, \ldots F_m$. If each F_r is k-th order resilient to zero-substitutions, and $k(k+2)m < n$, then $M^k_{[n]} \nmid F$.*

Note that proving $M^k_{[n]} \nmid F$ is equivalent to showing that F does not take the form $c \cdot M^k_{[n]}$ for some $c \in \mathbb{F}$. The proof of Theorem 3.21 is by induction

[2] In the read-once setting, [83] use the term $\bar{0}$-justified.

on the top fan-in, m, of the formula F. For $m = 1$, the theorem holds trivially because F remains non-zero after substituting zero for any of the variables. For larger m, we use partial derivatives and variable substitutions to eliminate some of the summands while ensuring that the surviving summands remain k-th order resilient to zero-substitutions, and the invariant $k(k+2)m < n$ continues to hold.

To elucidate, suppose that $D^{\boldsymbol{k}}_{\{x_i,x_j\}}F_r \equiv 0$, for some $r \in [m]$, and $x_i, x_j \in [n]$. Then, since $k(k+2)(m-1) < n-2$, we can employ the induction hypothesis to conclude that $M^k_{[n]\setminus\{i,j\}} \nmid D^{\boldsymbol{k}}_{\{x_i,x_j\}}F$, which implies $M^k_{[n]} \nmid F$.

On the other hand suppose $D^{\boldsymbol{k}}_{\{x_i,x_j\}}F_r \not\equiv 0$, for all F_r, $x_i, x_j \in [n]$. In this case, each F_r is a *product-only* formula.

Definition 3.22. *A formula F is a product-only formula if*

- *every gate g in F with both children computing non-constant polynomials is a product gate, and*
- *F does not contain any product gates with one child computing the zero polynomial and the other a non-constant polynomial.*

We exploit two useful properties of product-only formulas (see Lemma 3.24), to conclude that for each F_r

(i) there exists a subset U_r of size at most k such that $D^{\boldsymbol{k}}_{U_r}F_r$ takes the form $T_r \cdot Q_r$, where T_r is a degree k univariate polynomial in x_n, and Q_r is a polynomial that is k-th order resilient to zero-substitutions, and

(ii) there are at most k values $\beta \in \mathbb{F}$ such that $F_r|_{x_n=\beta}$ is not k-th order resilient to zero-substitutions.

The properties are stated with respect to the variable x_n, but hold with respect to any variable in $[n]$.

For $\ell \in [m]$, substituting a root, α, of T_ℓ for x_n in the formula $D^{\boldsymbol{k}}_{U_\ell}F$ results in the ℓ-th summand being eliminated. The hope then is to employ the induction hypothesis on the formula $D^{\boldsymbol{k}}_{U_\ell}F|_{x_n=\alpha}$. The caveat here is that the substitution

could result in some of the surviving summands no longer being k-th order resilient to zero-substitutions. To get around this we take the k-th order derivative with respect to a larger set of variables so that each of these 'bad' summands also take the form $T_r \cdot Q_r$. The polynomial Q_r is k-th order resilient to zero substitutions and doesn't depend on x_n. As a result setting x_n to α no longer results in any 'bad' summands.

The fix to deal with the 'bad' summands causes a second issue in that we may end up eliminating too many variables relative to the number of eliminated summands. We get around this by using Property (ii) in conjunction with a counting argument to argue that there exists a choice of α with respect to which the number of 'bad' summands is not too large when compared to the number of summands eliminated with the variable substitution.

In Section 3.4.2, we prove some useful properties of product-only formulas, including Lemma 3.24, which allows us to establish properties (i) and (ii) in the proof of Theorem 3.21. In Section 3.4.3, we prove Theorem 3.21 and derive Theorem 3.3 from it.

3.4.2 Product-only formulas

An immediate consequence of Definition 3.22 is that for every product-only formula $F \in \mathbb{F}[x_1, \ldots, x_n]$, and subset $U \subseteq [n]$, the derivative $D^r_U F$ is non-zero. The following proposition shows that the converse holds, i.e., if a formula $F \in \mathbb{F}[x_1, \ldots, x_n]$ is such that for every subset of variables $U \subseteq [n]$, the derivative $D^r_U F$ is non-zero, then F is a product-only formula.

Proposition 3.23. *If $F \in \mathbb{F}[x_1, \ldots, x_n]$ is a formula such that $D^{r(F)}_{\{x_i,x_j\}} F \not\equiv 0$, for every pair of variables $x_i, x_j \in [n]$, then F is a product-only formula.*

Proof. Let g be a non-constant internal gate with children g_1 and g_2. Assume that g_1 is non-constant and let x_i be a variable in $\mathrm{var}(g_1)$. Pick a variable $x_j \neq x_i$ in $\mathrm{var}(g_2)$. If there is no such variable x_j, i.e., if $\mathrm{var}(g_2) \subseteq \{x_i\}$, pick an arbitrary variable $x_j \in [n] \setminus \{x_i\}$.

By Proposition 3.5,

$$D^{r(F)}_{\{x_i,x_j\}}F = (D^{r(g)}_{\{x_i,x_j\}}g) \cdot D_y(D^{r(Q)}_{\{x_i,x_j\}}Q),$$

where Q is the formula obtained from F by replacing g with a fresh variable y. Thus, if g is an addition gate and g_2 is non-constant, $D^{r(F)}_{\{x_i,x_j\}}F \equiv 0$. Similarly, if g is a product-gate and g_2 computes the zero polynomial, then $D^{r(F)}_{\{x_i,x_j\}}F \equiv 0$.

Therefore, F must be a product-only formula. □

Lemma 3.24 establishes the key properties of product-only formulas used in the proof of Theorem 3.21. The Lemma is stated with respect to the variable x_n, but holds for any variable $x_i \in [n]$.

Lemma 3.24 (Key lemma)**.** *Let $F \in \mathbb{F}[x_1, \ldots, x_n]$ be a product-only formula, and let T denote the univariate degree $r_n(F)$ polynomial, $D^{r(F)}_{[n-1]}F$.*

(a) There exists a set of variables $U \subseteq [n-1]$ of size at most $r_n(F)$ such that $D^{r(F)}_U F$ is of the form $T \cdot Q$, where Q is a polynomial over variables in $[n-1] \setminus U$.

(b) Suppose that for every variable $x_i \in [n-1]$ and subset $V \subseteq [n] \setminus \{x_i\}$, $D^{r(F)}_V F|_{x_i=0} \not\equiv 0$. Then, there are at most $r_n(F)$ values $\beta \in \mathbb{F}$ such that for some $x_i \in [n-1]$ and $V \subseteq [n-1] \setminus \{x_i\}$,

$$T(\beta) \neq 0 \text{ and } D^{r(F)}_V F|_{x_n=\beta, x_i=0} \equiv 0.$$

Proof. Assume that x_n is read at least once by the formula F, and $n \geq 2$ otherwise the lemma is trivial. Let $g_1, \ldots, g_\ell$ denote the *maximal* gates in F that compute non-constant univariate polynomials in the variable x_n. Note that $\ell \leq r_n(F)$, and the degree of $\prod_{j\in[\ell]} g_j$ is $r_n(F)$. For every $j \in [\ell]$, let h_j denote the sibling gate of g_j in F. By maximality of the g_j's, each h_j is non-constant.

For a subset of variables, $V \subseteq [n-1]$, let J_V denote the set $\{j \in [\ell] \mid \mathrm{var}(h_j) \cap V \neq \emptyset\}$. We claim that the following is true.

Claim 3.25. *For every $V \subseteq [n-1]$, the polynomial $D_V^{r(F)}F$ is divisible by $\prod_{j\in J_V} g_j$.*

Assuming Claim 3.25 for the moment, we complete the proof of the lemma.

Proof of Part (a): There exists a set $U \subseteq [n-1]$ of size most ℓ such that for every $j \in [\ell]$, h_j reads a variable in U. By Claim 3.25 and the fact that the degree of $\prod_{j\in[\ell]} g_j$ is $r_n(F)$, the polynomial $D_U^{r(F)}F$ is of the form $\prod_{j\in[\ell]} g_j \cdot Q$, where Q is a polynomial over variables in $[n-1] \setminus U$.

By Claim 3.25, $T = D_{[n-1]}^{r(F)}F$ is divisible by $\prod_{j\in[\ell]} g_j$. Since T is a degree $r_n(F)$ univariate polynomial in the variable x_n, $T = c \cdot \prod_{j\in[\ell]} g_i$, for some $c \in \mathbb{F}^*$. This completes the proof of Part (a).

Proof of Part (b): Let x_i be a variable in $[n-1] \setminus U$. For every subset $V \in [n-1] \setminus \{x_i\}$,

$$D_{U\cup V}^{r(F)}F|_{x_i=0} = T \cdot (D_V^{r(F)}Q|_{x_i=0}) \not\equiv 0.$$

Thus, there are no values β such that $T(\beta) \neq 0$ and $D_V^{r(F)}F|_{x_n=\beta,x_i=0} \equiv 0$, for some set $V \subseteq [n-1] \setminus \{x_i\}$.

Now, suppose $x_i \in U$. Let J_i denote the set $\{j \in [\ell] \mid \mathrm{var}(h_j) = \{x_i\}\}$. By Claim 3.25, $D_{[n-1]\setminus\{x_i\}}^{r(F)}F = (\prod_{j\notin J_i} g_j)Q_i$, where Q_i is a bivariate polynomial over the variables x_i, x_n.

Since any root of $\prod_{j\notin J_i} g_j$ is also a root of T, the only values $\beta \in \mathbb{F}$ such that $T(\beta) \neq 0$, and $D_V^{r(F)}F|_{x_n=\beta,x_i=0} \equiv 0$, for some set $V \subseteq [n-1] \setminus \{x_i\}$, must be the roots of $Q_i|_{x_i=0}$. Thus, to show that the number of such values β is at most $r_n(F)$, it is sufficient to argue that $\sum_{x_i\in U} \deg(Q_i|_{x_i=0}) \leq r_n(F)$.

Recall that $\sum_{j\in[\ell]} \deg(g_j) = r_n(F)$. As a result, for any given $x_i \in U$, $\deg(Q_i|_{x_i=0}) \leq \sum_{j\in J_i} \deg(g_j)$. For each $j \in [\ell]$, there is at most one $x_i \in U$ such that $j \in J_i$. Therefore, $\sum_{i\in U} \deg(Q_i|_{x_i=0}) \leq \sum_{j\in[\ell]} \deg(g_j) = r_n(F)$.

All that remains is the proof of Claim 3.25.

Proof of Claim 3.25: The proof is by induction on the size of $J_V = \{j \in [\ell] \mid \mathrm{var}(h_j) \cap V \neq \emptyset\}$. If F and $V \subseteq [n-1]$ are such that $J_V = \emptyset$, then the statement of the claim holds trivially. Suppose that $J_V \neq \emptyset$. For $j \in [\ell]$, let f_j denote the parent of g_j and h_j. Let $i \in J_V$ be such that f_i is a minimal gate in the set $\{f_j \mid j \in J_V\}$. Note that the subformula h_i does not contain any gates in the set $\{g_j \mid j \in J_V\}$. By Proposition 3.5,

$$D_V^{\boldsymbol{r}(F)} F = g_i \cdot D_V^{\boldsymbol{r}(h_i)} h_i \cdot D_y(D_V^{\boldsymbol{r}(Q)} Q), \tag{3.1}$$

where Q is the formula obtained from F by replacing f_i with a fresh variable y. Note that Q is a product-only formula that contains the gate f_j, for all $j \in [\ell] \setminus \{i\}$. By the induction hypothesis applied to the formula Q, $D_V^{\boldsymbol{r}(Q)} Q$ is of the form $(\prod_{j \in J_V \setminus \{i\}} g_j) \cdot Q'$, where Q' is a polynomial over the set of variables $\{y\} \cup ([n] \setminus V)$. Thus, $\prod_{j \in J_V \setminus \{i\}} g_j$ divides the polynomial $D_y(D_V^{\boldsymbol{r}(Q)} Q)$. Therefore, by Equation (3.1), we have that $\prod_{j \in J_V} g_j$ divides $D_V^{\boldsymbol{r}(F)} F$. □

Remark: Part (b) of Lemma 3.24 is a generalization of Part (iii) of Lemma 3.13 in [83]. The proof of Lemma 3.13 in [83] contains an error. In our notation, their proof only shows that for each choice of the subset V, there is at most one value β such that for some $x_i \in [n-1] \setminus V$, $T(\beta) \neq 0$ and $D_V^{\boldsymbol{r}(F)} F|_{x_n = \beta, x_i = 0} \equiv 0$. Their proof can be fixed and generalized via induction to obtain the statement for arbitrary read formulas in Part (b) of Lemma 3.24. However, the resulting proof involves a lengthy and delicate case analysis. The more succinct proof of Part (b) presented here was suggested by Andrew Morgan (personal communication).

3.4.3 Proof of Theorem 3.3

As already mentioned, we first prove Theorem 3.21, and then derive 3.3 as a corollary. We will need the following closure properties of k-th order resilience to zero substitutions.

Proposition 3.26. *Let $F \in \mathbb{F}[x_1, \ldots, x_n]$ be a read-k formula that is k-th order resilient to zero substitutions.*

- *For every subset $V \in [n]$, $D^{\boldsymbol{k}}_V F$ is k-th order resilient to zero-substitutions.*
- *If $F = P \cdot Q$, where $\mathrm{var}(P) \cap \mathrm{var}(Q) = \emptyset$, then P and Q are k-th order resilient to zero substitutions.*

Proof of Theorem 3.21. We may assume that $\mathbb{F}$ is algebraically closed because any polynomial identity over a field continues to hold over its algebraic closure.

The proof is by induction on the top fan-in of F, m. In the base case, where $m = 1$, F is k-th order resilient to zero substitutions. In particular, F remains non-zero after substituting zero for any variable $x_i \in [n]$. Thus, $M^k_{[n]} \nmid F$.

For the inductive step, suppose that $m > 1$. We consider two cases.

Case 1: There exists a pair of variables $x_i, x_j \in [n]$ such that $D^{\boldsymbol{k}}_{\{x_i,x_j\}} F_{r'} \equiv 0$ for some $r' \in [m]$. Note that if $D^{\boldsymbol{k}}_{\{x_i,x_k\}} F \equiv 0$, then $x_i^k x_j^k \nmid F$ and as a result $M^k_{[n]} \nmid F$. Thus, we may assume that $D^{\boldsymbol{k}}_{\{x_i,x_j\}} F$ is the non-zero sum of at most $m-1$ read-k formulas over variables in $[n] \setminus \{i, j\}$. Moreover, each surviving summand $D^{\boldsymbol{k}}_{\{x_i,x_j\}} F_r$, for $r \in [m] \setminus \{r'\}$, remains k-th order resilient to zero substitutions by Proposition 3.26. By the induction hypothesis applied to the formula $D^{\boldsymbol{k}}_{\{x_i,x_j\}} F$, we have that $M^k_{[n]\setminus\{i,j\}} \nmid D^{\boldsymbol{k}}_{\{x_i,x_j\}} F$, provided $k(k+2)(m-1) < n-2$. Note that $k(k+2)(m-1) < n-2$ if $k(k+2)m < n$. Thus, $M^k_{[n]\setminus\{i,j\}} \nmid D^{\boldsymbol{k}}_{\{x_i,x_j\}} F$ and $M^k_{[n]} \nmid F$.

Case 2: If we are not in Case 1, by Proposition 3.23, each F_r is a product-only formula that reads every variable in $[n]$ exactly k times.

By Part (a) of Lemma 3.24, for each $r \in [m]$, there exists a set $U_r \subseteq [n]$ of size at most k, such that $D^{\boldsymbol{k}}_{U_r} F_r$ is of the form $T_r \cdot Q_r$, where $T_r = D^{\boldsymbol{k}}_{[n-1]} F_r$ is a univariate degree k polynomial in the variable x_n, and Q_r is a polynomial over $[n-1] \setminus U_r$. Note that each Q_r is k-th order resilient to zero substitutions by Proposition 3.26.

Let Z denote the set $\{\alpha \in \mathbb{F} \mid \exists r \in [m] : T_r(\alpha) = 0\}$. Since, each F_r is k-th order resilient to zero substitutions, $0 \notin Z$. For $\alpha \in Z$, let G_α denote the set of indices $r \in [m]$ such that $T_r(\alpha) = 0$, and let B_α denote the set of indices $r \in [m]$ such that $T_r(\alpha) \neq 0$ and $F_r|_{x_n=\alpha}$ is not k-th order resilient to zero substitutions. We have the following claim.

Claim 3.27. *There exists a point $\alpha_0 \in Z$ such that $k|G_{\alpha_0}| \geq |B_{\alpha_0}|$.*

Assuming the claim holds for the moment, we complete the proof in Case 2.

Let U denote the set of variables $\cup_{r \in G_{\alpha_0} \cup B_{\alpha_0}} U_r$. Since each U_r contains at most k variables, $|U| \leq k(k+1)|G_{\alpha_0}|$.

For each $r \in [m]$, consider the read-k formula $D^{\boldsymbol{k}}_U(F_r|_{x_n=\alpha_0}) = T_r(\alpha_0) \cdot (D^{\boldsymbol{k}}_{U \setminus U_r} Q_r)$.

- If $r \in G_{\alpha_0}$, then $T_r(\alpha_0) = 0$, and as a result $D^{\boldsymbol{k}}_U(F|_{x_n=\alpha_0}) \equiv 0$.
- If $r \in B_{\alpha_0}$, then $D^{\boldsymbol{k}}_U(F_r|_{x_n=\alpha_0}) = T_r(\alpha_0) \cdot (D^{\boldsymbol{k}}_{U \setminus U_r} Q_r)$ is k-th order resilient to zero substitutions by Proposition 3.26.
- Otherwise, $F_r|_{x_n=\alpha_0}$ is k-th order resilient to zero substitutions, and by Proposition 3.26, so is $D^{\boldsymbol{k}}_U(F_r|_{x_n=\alpha_0})$.

Thus, $D^{\boldsymbol{k}}_U(F|_{x_n=\alpha_0})$ is the sum of at most $m - |G_{\alpha_0}|$ read-k formulas that are k-th order resilient to zero substitutions. Note that we may assume $D^{\boldsymbol{k}}_U(F|_{x_n=\alpha_0}) \not\equiv 0$ because otherwise, $(x_n - \alpha_0) \mid D^{\boldsymbol{k}}_U F$, where $\alpha_0 \neq 0$, which implies $x_n^k \nmid D^{\boldsymbol{k}}_U F$, and as a result $M^k_{[n]} \nmid F$.

Now, if $k(k+2)m < n$, then $k(k+2)(m - |G_{\alpha_0}|) < n - k(k+2)|G_{\alpha_0}|$. Since $G_{|\alpha_0|}$ has at least one element, and $|U| \leq k(k+1)|G_{\alpha_0}|$, we have that $k(k+2)(m-|G_{\alpha_0}|) < n-|U|-1$. Thus, by the induction hypothesis, $M^k_{[n-1]\setminus U} \nmid D^{\boldsymbol{k}}_U(F|_{x_n=\alpha_0})$. Therefore, $M^k_{[n]} \nmid F$.

All that remains is to prove the claim.

Proof of Claim 3.27 Note that

$$\sum_{\alpha \in Z} |G_\alpha| \geq m \tag{3.2}$$

because each T_r is non-constant and $\mathbb{F}$ is algebraically closed. By Lemma 3.24, for each $r \in [m]$, there are at most k values $\alpha \in \mathbb{F}$ such that for some $x_i \in [n-1]$, and subset $V \subseteq [n-1] \setminus \{x_i\}$, $T_r(\alpha) \neq 0$ and $D_V^{\boldsymbol{k}} F_r|_{x_n=\alpha, x_i=0} \equiv 0$. Thus, for each $r \in [m]$, there are at most k values $\alpha \in Z$ such that $T_r(\alpha) \neq 0$ and $F_r|_{x_n=\alpha}$ is not k-th order resilient to zero substitutions. It follows that

$$\sum_{\alpha \in Z} |B_\alpha| \leq km. \tag{3.3}$$

By (3.2) and (3.3), there must exist a value $\alpha_0 \in Z$ such that $k|G_{\alpha_0}| \geq |B_{\alpha_0}|$. □

We now derive Theorem 3.3 as a corollary. The proof boils down to showing that if $F \in \mathbb{F}[x_1, \ldots, x_n]$ is a read-k formula, and $\boldsymbol{\sigma} \in \mathbb{F}^n$ is a common non-zero of the polynomials $F \cup \{D_{x_i^k} F\}_{i \in [n]}$, then $F(\boldsymbol{x} + \boldsymbol{\sigma})$ is k-th order resilient to zero substitutions.

Lemma 3.28. *Let $F \in \mathbb{F}[x_1, \ldots, x_n]$ be an arithmetic formula, and let $V \subseteq [n]$ be such that $D_V^{\boldsymbol{r}(F)} F \not\equiv 0$. If $\boldsymbol{\sigma} \in \mathbb{F}^n$ is a common non-zero of the formulas $D_{x_i}^{r_i(F)} F$, where x_i ranges over V, then $\boldsymbol{\sigma}$ is a non-zero of the formula $D_V^{\boldsymbol{r}(F)} F$.*

Proof. The proof is by induction on $R(F, V) \doteq \sum_{i \in V} r_i(F)$. The statement is vacuously true for $R(F, V) = 0$. If $R(F, V) = 1$, then V contains exactly one variable and the statement holds.

Suppose F, V are such that $R(F, V) > 1$. Let g denote the minimal gate in F that witnesses all reads of the variables in V. Since $D_V^{\boldsymbol{r}(F)} F \not\equiv 0$, g must be a product gate. Let g_1, g_2 denote the children of g in F. Let $Q \in \mathbb{F}[y, x_1, \ldots, x_n]$ be the formula obtained from F by replacing the gate g with a variable y.

By Proposition 3.5, we have that for every variable $x_i \in V$,

$$D_{x_i}^{r_i(F)}F = D_{x_i}^{r_i(g_1)}g_1 \cdot D_{x_i}^{r_i(g_2)}g_2 \cdot D_y Q. \tag{3.4}$$

Since $\boldsymbol{\sigma}$ is a non-zero of $D_{x_i}^{r_i(F)}F$, it is a non-zero of the factors on the right hand side of Equation 3.4.3 as well.

By the induction hypothesis applied to g_1, g_2, we conclude that $\boldsymbol{\sigma}$ is a non-zero of $D_V^{\boldsymbol{r}(g_j)}g_j$ for $j \in \{1, 2\}$. Again, by Proposition 3.5,

$$D_V^{\boldsymbol{r}(F)}F = D_V^{\boldsymbol{r}(g_1)}g_1 \cdot D_V^{\boldsymbol{r}(g_2)}g_2 \cdot D_y Q.$$

$\boldsymbol{\sigma}$ is a non-zero of all of the factors in the right hand side and is therefore a non-zero of $D_V^{\boldsymbol{r}(F)}F$. □

Proof of Theorem 3.3. For $r \in [m]$, let F_r' denote the formula $F_r(\boldsymbol{x} + \boldsymbol{\sigma})$. Fix an $r \in [m]$, and let V be a subset such that $D_V^{\boldsymbol{k}}F_r' \not\equiv 0$. This implies for every $x_i \in V$, $D_{x_i}^k F_r' \not\equiv 0$.

Let x_j be a variable in $[n] \setminus V$. By the definition of $\boldsymbol{\sigma}$, we have that for every $x_i \in V$, $D_{x_i}^k F_r'|_{x_j=0} \not\equiv 0$. By Lemma 3.28 applied to $F_r'|_{x_j=0}$, we conclude that $D_V^{\boldsymbol{k}}F_r'|_{x_j=0} \not\equiv 0$. Thus, each F_r' is k-th order resilient to zero-substitutions.

Therefore, by Theorem 3.21, $M_{[n]}^k \nmid F(\boldsymbol{x} + \boldsymbol{\sigma})$. □

3.5 Hardness of representation for α-split bounded degree formulas

In this section, we prove Theorem 3.4.

Definition 3.29 (α-split)**.** *A polynomial $F = \sum_{r\in[m]} F_r$ with $\cup_{r\in[m]}\mathrm{var}(F_r) = [n]$ is α-split if each F_r is of the form $\prod_\ell F_{r\ell}$, where each $F_{r\ell}$ depends on at most αn variables.*

We will need the following lemma that establishes hardness of representation for sums of products of univariate formulas.

Lemma 3.30. *Let $F = \sum_{r\in[m]} F_r \in \mathbb{F}[x_1, \ldots, x_n]$ be a non-zero formula, where each F_r is a product of univariate polynomials, and no variable divides any F_r. Then $M_{[n]} \nmid F$, provided $m \leq n$.*

We first prove Theorem 3.4 assuming Lemma 3.30.

Proof of Theorem 3.4 assuming Lemma 3.30. For each $r \in [m]$, we can write F_r as $\prod_{\ell\in[m_r]} F_{r\ell}$, where $|\mathrm{var}(F_{r\ell})| \leq \alpha n$. We construct a set $U \subseteq [n]$ of size at least m such that $|U \cap \mathrm{var}(F_{r\ell})| \leq 1$, for every $r \in [m], \ell \in [m_r]$. Start with $U = \emptyset$. While there is a variable x such that all $F_{r\ell}$'s that depend on x, depend on no variable in U, add x to U. For a variable x, and $r \in [m]$, at most k $F_{r\ell}$'s depend on x. Thus, when a variable x is added to U, at most αnkm variables are eliminated from consideration in subsequent iterations. Since $\cup_{r\in[m]}\mathrm{var}(F_r) = [n]$, at least $n/(\alpha nkm) = m$ variables are added to U.

Let $F' = \sum_{r\in[m]} F'_r$ be the formula obtained from F by substituting field elements for the variables outside U. F' is the sum of products of univariate polynomials. Now, for every $r \in [m]$ and $x \in U$, $x \nmid F_r$, i.e., $F_r|_{x=0} \not\equiv 0$. Thus, for a typical assignment to the variables outside U, $F'_r|_{x=0} \not\equiv 0$ for every $x \in U$, $r \in [m]$. By Lemma 3.30 there exists a variable $x \in U$ such that $x \nmid F'$. Stated differently, there exists a variable $x \in U$ such that $F'|_{x=0} \not\equiv 0$, which implies $F|_{x=0} \not\equiv 0$. Therefore, $M_{[n]} \nmid F$. □

We now prove Lemma 3.30.

Proof of Lemma 3.30. The proof is by induction on the top fan-in m. The base case, where $m = 1$, holds because no variable divides F_1.

Suppose that $m > 1$. For each $r \in [m]$, let T_r denote the highest degree monic polynomial in x_n that divides F_r. Note that for every $r \in [m]$, $x_n \nmid T_r$. Let d_n denote the syntactic degree of x_n in F. For each $r \in [m]$, let $\boldsymbol{\tau}_r$ denote the $d_n + 1$-dimension coefficient vector of T_r, i.e., the d-th entry of $\boldsymbol{\tau}_r$ is the coefficient of

x_n^{d-1} in T_r[3]. Let b denote the dimension of $\mathrm{span}(\{\boldsymbol{\tau}_r \mid r \in [m]\})$. After a suitable relabeling, we may assume that $\boldsymbol{\tau}_1, \ldots, \boldsymbol{\tau}_b$, form a basis of $\mathrm{span}(\{\boldsymbol{\tau}_r \mid r \in [m]\})$.

Let $A \in \mathbb{F}^{m \times b}$ be the unique matrix satisfying $\boldsymbol{\tau}_r = \sum_{i=1}^{b} A_{ri}\boldsymbol{\tau}_i$, for every $r \in [m]$. F can be written as

$$F = \sum_{i=1}^{b} T_i\Big(\sum_{r\in[m]} A_{ri}F'_r\Big),$$

where $F'_r = F_r/T_r$. By the definition of T_r, $x_n \notin \mathrm{var}(F'_r)$, for all $r \in [m]$.

We may assume that there exist a pair of indices $r, r' \in [m]$ such that $T_r \neq T'_r$, and therefore $\boldsymbol{\tau}_r \neq \boldsymbol{\tau}'_r$, because otherwise $x_n \nmid F$, and the lemma holds. As a result, $b \geq 2$ and every column of A contains at least one zero. So for every $i \in [b]$, the formula $G_i \doteq \sum_{r\in[m]} A_{ri}F'_r \in \mathbb{F}[x_1, \ldots, x_{n-1}]$ has at most $m-1$ non-zero summands. Furthermore, each summand $A_{ri}F'_r$ is a product of univariate polynomials and is not divisible by any variable. Using the induction hypothesis, we conclude that $M_{n-1} \nmid G_i$, for all $i \in [b]$.

We now show that there exists a variable $x_j \in [n-1]$ such that $x_j \nmid F$. Let $x_j \in [n-1]$ be a variable such that $x_j \nmid G_i$, for some $i \in [b]$. The existence of x_j follows immediately from the fact that $M_{n-1} \nmid G_i$ for all $i \in [b]$. Thus, there exists an assignment to the variables in $[n-1]$, $\boldsymbol{\sigma} \in \mathbb{F}^{n-1}$, such that $\boldsymbol{\sigma}_j = 0$, and $G_i(\boldsymbol{\sigma}) \neq 0$, for some $i \in [b]$. Since $\{\boldsymbol{\tau}_i\}_{i\in[b]}$ are linearly independent coefficient vectors of the polynomials $\{T_i\}_{i\in[b]}$, any non-zero linear combination of the polynomials $\{T_i\}_{i\in[b]}$ is non-zero. Thus, $F(\boldsymbol{\sigma}, x_n) = \sum_{i=1}^{b} T_i G_i(\boldsymbol{\sigma})$ is non-zero. Therefore, $x_j \nmid F$. □

[3]For fields of non-zero characteristic, we work over a large enough extension field so that $\boldsymbol{\tau_r}$ is uniquely defined. This doesn't affect the validity of the lemma because $M_{[n]} \mid F$ in the extension field if and only if $M_{[n]} \mid F$ in the base field.

Bibliography

[1] S. Aaronson and A. Wigderson. Algebrization: A new barrier in complexity theory. *TOCT*, 1(1):2:1–2:54, 2009.

[2] A. Aggarwal, R. J. Anderson, and M.-Y. Kao. Parallel depth-first search in general directed graphs. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 297–308, 1989.

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

[4] M. Agrawal, C. Saha, and N. Saxena. Quasi-polynomial hitting-set for set-depth-Δ formulas. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 321–330, 2013.

[5] M. Agrawal, R. Gurjar, A. Korwar, and N. Saxena. Hitting-sets for ROABP and sum of set-multilinear circuits. *SIAM Journal on Computing*, 44(3): 669–697, 2015.

[6] E. Allender and U. Hertrampf. Depth reduction for circuits of unbounded fan-in. *Information and Computation*, 112(2):217–238, 1994.

[7] E. Allender, K. Reinhardt, and S. Zhou. Isolation, matching, and counting uniform and nonuniform upper bounds. *Journal of Computer and System Sciences*, 59(2):164–181, 1999.

[8] M. Anderson, D. van Melkebeek, and I. Volkovich. Deterministic polynomial identity tests for multilinear bounded-read formulae. *Computational Complexity*, 24(4):695–776, 2015.

[9] M. Anderson, M. A. Forbes, R. Saptharishi, A. Shpilka, and B. Volk. Identity testing and lower bounds for read-k oblivious algebraic branching programs. In *Proceedings of the 31st Conference on Computational Complexity*, pages 30:1–30:25, 2016.

[10] R. Arora, A. Gupta, R. Gurjar, and R. Tewari. Derandomizing Isolation Lemma for $K_{3,3}$-free and K_5-free Bipartite Graphs. In *Proceedings of the 33rd Symposium on Theoretical Aspects of Computer Science*, pages 10:1–10:15, 2016.

[11] V. Arvind and P. Mukhopadhyay. Derandomizing the isolation lemma and lower bounds for circuit size. In *Proceedings of the 12th Intl. Workshop on Randomization and Computation*, pages 276–289, 2008.

[12] V. Arvind, P. Mukhopadhyay, and S. Srinivasan. New results on noncommutative and commutative polynomial identity testing. *Computational Complexity*, 19(4):521–558, 2010.

[13] L. Babai, L. Fortnow, N. Nisan, and A. Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3:307–318, 1993.

[14] T. P. Baker, J. Gill, and R. Solovay. Relativizations of the P = NP question. *SIAM J. Comput.*, 4(4):431–442, 1975.

[15] G. Barnes, J. F. Buss, W. L. Ruzzo, and B. Schieber. A sublinear space, polynomial time algorithm for directed s-t connectivity. *SIAM Journal on Computing*, 27(5):1273–1282, 1998.

[16] R. Beigel, N. Reingold, and D. Spielman. The perceptron strikes back. In *Proceedings of the Sixth Annual Structure in Complexity Theory Conference*, pages 286–291, 1991.

[17] S. Ben-David, B. Chor, O. Goldreich, and M. Luby. On the theory of average case complexity. *Journal of Computer and System Sciences*, 44(2): 193–219, 1992.

[18] A. Björklund. Determinant sums for undirected hamiltonicity. *SIAM Journal on Computing*, 43(1):280–299, 2014.

[19] A. Björklund and T. Husfeldt. Shortest two disjoint paths in polynomial time. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming*, pages 211–222, 2014.

[20] A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Narrow sieves for parameterized paths and packings. *Journal of Computer and System Sciences*, 87:119 – 139, 2017.

[21] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput.*, 13(4):850–864, 1984.

[22] C. Bourke, R. Tewari, and N. V. Vinodchandran. Directed planar reachability is in unambiguous log-space. *ACM Transactions on Computation Theory*, 1(1), 2009.

[23] T. Brunsch, K. Cornelissen, B. Manthey, and H. Röglin. Smoothed analysis of belief propagation for minimum-cost flow and matching. In *Proceedings of the 7th International Workshop on Algorithms and Computation*, pages 182–193, 2013.

[24] J.-Y. Cai, V. T. Chakaravarthy, and D. van Melkebeek. Time-space tradeoff in derandomizing probabilistic logspace. *Theory Comput. Syst.*, 39(1): 189–208, 2006.

[25] J. L. Carter and M. N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979.

[26] S. Chari, P. Rohatgi, and A. Srinivasan. Randomness-optimal unique element isolation with applications to perfect matching and related problems. *SIAM Journal on Computing*, 24(5):1036–1050, 1995.

[27] M. Cygan, J. Nederlof, M. Pilipczuk, M. Pilipczuk, J. M. M. van Rooij, and J. O. Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science*, pages 150–159, 2011.

[28] M. Cygan, S. Kratsch, and J. Nederlof. Fast hamiltonicity checking via bases of perfect matchings. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing*, pages 301–310, 2013.

[29] S. I. Daitch and D. A. Spielman. Faster approximate lossy generalized flow via interior point algorithms. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, pages 451–460, 2008.

[30] S. Datta, R. Kulkarni, and S. Roy. Deterministically isolating a perfect matching in bipartite planar graphs. *Theory Comput. Syst.*, 47(3):737–757, 2010.

[31] S. Datta, R. Kulkarni, R. Tewari, and N.V. Vinodchandran. Space complexity of perfect matching in bounded genus bipartite graphs. *Journal of Computer and System Sciences*, 78(3):765–779, 2012.

[32] S. Datta, W. Hesse, and R. Kulkarni. Dynamic complexity of directed reachability and other problems. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming*, pages 356–367, 2014.

[33] S. Datta, R. Kulkarni, A. Mukherjee, T. Schwentick, and T. Zeume. Reachability is in DynFO. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming*, pages 159–170, 2015.

[34] H. Dell, V. Kabanets, D. van Melkebeek, and O. Watanabe. Is Valiant-Vazirani's isolation probability improvable? *Computational Complexity*, 22 (2):345–383, 2013.

[35] R. A. Demillo and R. J. Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7(4):193 – 195, 1978.

[36] J. Erickson and P. Worah. Computing the shortest essential cycle. *Discrete and Computational Geometry*, 44(4):912–930, 2010.

[37] S. A. Fenner, R. Gurjar, and T. Thierauf. Bipartite perfect matching is in quasi-NC. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing*, pages 754–763, 2016.

[38] F. V. Fomin and P. Kaski. Exact exponential algorithms. *Communications of the ACM*, 56(3):80–88, March 2013.

[39] M. A. Forbes, R. Saptharishi, and A. Shpilka. Hitting sets for multilinear read-once algebraic branching programs, in any order. In *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 867–875, New York, NY, USA, 2014. ACM.

[40] A. Gál and A. Wigderson. Boolean complexity classes vs. their arithmetic analogs. *Random Struct. Algorithms*, 9(1-2):99–111, 1996.

[41] D. Gamarnik, D. Shah, and Y. Wei. Belief propagation for min-cost network flow: Convergence and correctness. *Operations Research*, 60(2):410–428, 2012.

[42] R. Gurjar, A. Korwar, and N. Saxena. Identity testing for constant-width, and any-order, read-once oblivious arithmetic branching programs. *Theory of Computing*, 13(1):1–21, 2017.

[43] R. Gurjar, A. Korwar, N. Saxena, and T. Thierauf. Deterministic identity testing for sum of read-once oblivious arithmetic branching programs. *Computational Complexity*, 26(4):835–880, 2017.

[44] I. Haviv and O. Regev. On the lattice isomorphism problem. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 391–404, 2014.

[45] H. Hirai and H. Namba. Shortest $(A + B)$-path packing via hafnian. *Computing Research Repository*, abs/1603.08073, 2016.

[46] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17(5):935–938, 1988.

[47] R. Impagliazzo and A. Wigderson. $P = BPP$ if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 220–229, 1997.

[48] R. Impagliazzo, R. Shaltiel, and A. Wigderson. Near-optimal conversion of hardness into pseudo-randomness. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 181–190, 1999.

[49] R. Impagliazzo, R. Shaltiel, and A. Wigderson. Extractors and pseudo-random generators with optimal seed length. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 1–10, 2000.

[50] R. Impagliazzo, V. Kabanets, and A. Wigderson. In search of an easy witness: exponential time vs. probabilistic polynomial time. *J. Comput. Syst. Sci.*, 65(4):672–694, 2002.

[51] V. Kabanets and R. Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1-2): 1–46, 2004.

[52] V. A. T. Kallampally and R. Tewari. Trading Determinism for Time in Space Bounded Computations. In *Proceedings of the 41st International Symposium on Mathematical Foundations of Computer Science*, pages 10:1–10:13, 2016.

[53] R. Kannan, H. Venkateswaran, V. Vinay, and A.C. Yao. A circuit-based proof of Toda's theorem. *Information and Computing*, 104(2):271–276, 1993.

[54] Y. Kanoria, M. Bayati, C. Borgs, J. Chayes, and A. Montanari. Fast convergence of natural bargaining dynamics in exchange networks. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1518–1537, 2011.

[55] Z. Karnin, P. Mukhopadhyay, A. Shpilka, and I. Volkovich. Deterministic identity testing of depth-4 multilinear circuits with bounded top fan-in. *SIAM Journal on Computing*, 42(6):2114–2131, 2013.

[56] R. M. Karp, E. Upfal, and A. Wigderson. Constructing a perfect matching is in random NC. *Combinatorica*, 6(1):35–48, 1986.

[57] N. Kayal and S. Saraf. Blackbox polynomial identity testing for depth 3 circuits. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 198–207, 2009.

[58] N. Kayal and N. Saxena. Polynomial identity testing for depth 3 circuits. *Computational Complexity*, 16(2):115–138, 2007.

[59] S. Kiefer, A. S. Murawski, J. Ouaknine, B. Wachter, and J. Worrell. On the complexity of equivalence and minimisation for Q-weighted automata. *Logical Methods in Computer Science*, 9(1), 2013.

[60] A. Klivans and D. A. Spielman. Randomness efficient identity testing of multivariate polynomials. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 216–223, 2001.

[61] A. Klivans and D. van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM Journal on Computing*, 31(5):1501–1526, 2002.

[62] K. Ko. On self-reducibility and weak P-selectivity. *Journal of Computer and System Sciences*, 26(2):209–211, 1983.

[63] V. Krishan and N. Limaye. Isolation lemma for directed reachability and NL vs. L. *Electronic Colloquium on Computational Complexity*, 23:155, 2016.

[64] J. Kynčl and T. Vyskočil. Logspace reduction of directed reachability for bounded genus graphs to the planar case. *ACM Transactions on Computation Theory*, 1(3):8:1–8:11, March 2010.

[65] A. Lingas and M. Karpinski. Subtree isomorphism is NC reducible to bipartite perfect matching. *Information Processing Letters*, 30(1):27 – 32, 1989.

[66] A. Lingas and M. Persson. A fast parallel algorithm for minimum-cost small integral flows. *Algorithmica*, 72(2):607–619, 2015.

[67] M. Mahajan, B. V. R. Rao, and K. Sreenivasaiah. Monomials, multilinearity and identity testing in simple read-restricted circuits. *Theor. Comput. Sci.*, 524:90–102, 2014.

[68] R. Majumdar and J. L. Wong. Watermarking of SAT using combinatorial isolation lemmas. In *Proceedings of the 38th Annual Design Automation Conference*, pages 480–485, 2001.

[69] D. Minahan and I. Volkovich. Complete derandomization of identity testing and reconstruction of read-once formulas. *TOCT*, 10(3):10:1–10:11, 2018.

[70] K. Mulmuley, U. V. Vazirani, and V. V. Vazirani. Matching is as easy as matrix inversion. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 345–354, 1987.

[71] N. Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

[72] N. Nisan. RL $\subseteq$ SC. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 619–623, 1992.

[73] N. Nisan and A. Wigderson. Hardness vs randomness. *J. Comput. Syst. Sci.*, 49(2):149–167, 1994.

[74] J. B. Orlin and C. Stein. Parallel algorithms for the assignment and minimum-cost flow problems. *Operations research letters*, 14(4):181–186, 1993.

[75] A. Razborov and S. Rudich. Natural proofs. *J. Comput. Syst. Sci.*, 55(1): 24–35, 1997.

[76] K. Reinhardt and E. Allender. Making nondeterminism unambiguous. *SIAM Journal on Computing*, 29(4):1118–1131, 2000.

[77] M. E. Saks and S. Zhou. $\text{BP}_H\text{SPACE}(S) \subseteq \text{DSPACE}(S^{3/2})$. *Journal of Computer and System Sciences*, 58(2):376–403, 1999.

[78] S. Saraf and I. Volkovich. Black-box identity testing of depth-4 multilinear circuits. *Combinatorica*, 38(5):1205–1238, Oct 2018.

[79] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

[80] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, October 1980.

[81] R. Shaltiel and C. Umans. Simple extractors for all min-entropies and a new pseudorandom generator. *J. ACM*, 52(2):172–216, 2005.

[82] A. Shamir. $\text{IP} = \text{PSPACE}$. *J. ACM*, 39(4):869–877, October 1992.

[83] A. Shpilka and I. Volkovich. Read-once polynomial identity testing. *Computational Complexity*, 24(3):477–532, 2015.

[84] Y. Strozecki. On enumerating monomials and other combinatorial structures by polynomial interpolation. *Theory of Computing Systems*, 53(4):532–568, 2013.

[85] M. Sudan, L. Trevisan, and S. P. Vadhan. Pseudorandom generators without the XOR lemma. *J. Comput. Syst. Sci.*, 62(2):236–266, 2001.

[86] I. H. Sudborough. On the tape complexity of deterministic context-free languages. *Journal of the ACM*, 25(3):405–414, July 1978.

[87] Robert Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26(3):279–284, 1988.

[88] J. Tarui. Probabilistic polynomials, AC0 functions and the polynomial-time hierarchy. *Theoretical Computer Science*, 113(1):167–183, 1993.

[89] T. Thierauf and F. Wagner. Reachability in $K_{3,3}$-free and K_5-free graphs is in unambiguous logspace. *Chicago Journal of Theoretical Computer Science*, 2015, 2015.

[90] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991.

[91] P. Traxler. The time complexity of constraint satisfaction. In *Proceedings of the 3rd International Workshop on Parameterized and Exact Computation*, pages 190–201, 2008.

[92] C. Umans. Pseudo-random generators for all hardnesses. *J. Comput. Syst. Sci.*, 67(2):419–440, 2003.

[93] L. G. Valiant and V. V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47(3):85–93, 1986.

[94] H. Venkateswaran. Properties that characterize LOGCFL. *Journal of Computer and System Sciences*, 43(2):380–404, October 1991.

[95] H. Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Springer-Verlag New York, 1999.

[96] O. Watanabe and S. Toda. Structural analysis of the complexity of inverse functions. *Mathematical Systems Theory*, 26(2):203–214, 1993.

[97] R. Williams. Finding paths of length k in $O^*(2^k)$ time. *Inf. Process. Lett.*, 109(6):315–318, 2009.

[98] A. C. Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 80–91, 1982.

[99] R. Zippel. Probabilistic algorithms for sparse polynomials. In Edward W. Ng, editor, *Symbolic and Algebraic Computation*, pages 216–226, Berlin, Heidelberg, 1979. Springer Berlin Heidelberg.