Electronic Colloquium on Computational Complexity, Report No. 52 (2017)

Derandomizing Isolation in Space-Bounded Settings[*]

Dieter van Melkebeek Gautam Prakriya
University of Wisconsin-Madison

March 19, 2017

Abstract

We study the possibility of deterministic and randomness-efficient isolation in space-bounded models of computation: Can one efficiently reduce instances of computational problems to equivalent instances that have at most one solution? 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 individual 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 somewhat more restricted 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.

1. Introduction

Isolation is the process of singling out a solution to a problem that may have many solutions. It is used in algorithms with an algebraic flavor in order to prevent cancellations from happening. Examples include reductions of multivariate to univariate polynomial identity testing [KS01, AGKS15]

[*]Research partially supported by the US National Science Foundation (grant CCF-1319822), the Foundation Sciences Mathématiques de Paris and the French Agence Nationale de la Recherche (grant ANR-13-BS02-0001-01 Compa), and the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant ERC-2014-CoG 648276 AUTAR).

ISSN 1433-8092

and recent approaches to the hamiltonicity problem [Bjö14, FK13, CNP+11, CKN13]. 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 [MVV87] and related problems [KUW86, AAK89, LK89]. A wide range of other algorithmic applications of isolation exist [Tod91, BRS91, BDCGL92, KVVY93, Tar93, WT93, OS93, AH94, MW01, DS08, Tra08, AMS10, EW10, KBB+11, GSW12, BCMR13, KMO+13, Str13, BH14, DHK14, HR14, DKM+15, LP15, HN16]. 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 [CIKP03, and references further in this section].

Becoming more precise, let us define a computational (promise)[1] 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 represent 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 to produce a solution $y \in \Pi(x)$, or report that no solution exists. For example, for the NP-complete problem of Satisfiability, 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 $x \in \Pi$. 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 $\Pi(x) \neq \varnothing$. 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 paths additionally 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 1 (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 requires 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.

[1]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.

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

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

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 by viewing the solutions y for a given instance x as subsets of a finite universe $U = U(x)$, assigning small weights $w(u) \in \mathbb{N}$ to the elements $u \in U$, and defining $\omega(x,y)$ 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 [VV86, MVV87, CRS95] 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 1 (Isolation Lemma [MVV87]). *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 equal $\omega(x)$, 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 NP $\subseteq$ R$\cdot$PromiseUP [VV86].[2] 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

[2]The original argument in [VV86] uses a different randomized isolation for SATISFIABILITY; it has a success probability of $\Omega(1/n)$.

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. 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.

Isolation in Space-Bounded Settings. Gal and Wigderson [GW96] 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 $1/poly(n)$ 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 PromiseUL is the logspace equivalent of PromiseUP. Thus, in the randomized space-bounded setting, having unique solutions does not make instances of NL-complete problems easier.

Reinhardt and Allender [RA00] strengthened this result to $\mathrm{NL} \subseteq \mathrm{R}\cdot(\mathrm{UL}\cap\mathrm{coUL})$. The class UL consists of the problems in 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., $\mathrm{R}\cdot(\mathrm{UL}\cap\mathrm{coUL}) \subseteq (\mathrm{UL}\cap\mathrm{coUL})/\mathrm{poly}$. 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 [RA00] needs a stronger property of the weight assignment w than merely being min-isolating on the given input (G,s,t). It requires w to be min-isolating for G, i.e., min-isolating for (G,s,t) for *all* choices of vertices s and t. 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 still only used to generate random weight assignments. The new ingredients in [RA00] that enable the strengthening from $\mathrm{R}\cdot\mathrm{PromiseUL}$ to $\mathrm{R}\cdot(\mathrm{UL}\cap\mathrm{coUL})$ are unambiguous logspace machines to (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.

Gal and Wigderson [GW96] and Reinhardt and Allender [RA00] developed analogous results for the complexity class LogCFL, where the role of Reachability is taken over by the problem Circuit Certification of finding a certificate that a given Boolean circuit accepts a given input. We refer to Section 3.1 for the definition of a certificate and for background, and defer further discussion of this setting to that section.

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 3 (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 3 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 $[|U|] \doteq \{1, 2, \dots, |U|\}$ 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}.

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 $O(n\log(qn))$, bit length $O(\log n)$, and trivial complexity. By allowing weights that are polynomially larger than in the Isolation Lemma, one can achieve seed length $O(\log(qn) + \log(|\Pi(x)|))$, which is provably optimal for a generic $\Pi(x)$ [CRS95]. 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 [AM08], and vice versa [KvM02]. 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 $p > 2/3$ [DKvMW13].

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 $O(\log n)$ [KvM02, ARZ99]. 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 [BTV09]. Those assignments have been used to construct disambiguations that are logspace computable and logspace recoverable for larger classes of graphs [BTV09, TW15, KV10].

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 [DKR10, DKTV12, AGGT16]. Recently, Fenner, Gurjar, and Thierauf [FGT16] constructed 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.

Main Results. We present positive and negative results regarding the possibility of derandomized isolations for Reachability and for Circuit Certification.

The crux for our positive results are logspace min-isolating weight assignment generators with seed length and bitlength $O((\log n)^{3/2})$. We actually shift the paradigm a bit – we assign weights to the (internal) *vertices* rather than to the edges. This is not an essential difference,[3] but it facilitates a natural iterative/recursive approach towards the construction of the weight assignment, and allows for a cleaner and unified treatment.

Recall that in the context of Reachability we call a weight assignment w min-isolating for an instance (G, s, t) if G has at most one path from s to t of minimum weight under w. For technical reasons we only consider the restriction of Reachability to *layered* digraphs G.

Theorem 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*

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

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

The domain underlying $\Gamma^{(\mathrm{reach})}_{n,d}$ is $[n] \times [\![d]\!] \doteq \{1, 2, \ldots, n\} \times \{0, 1, 2, \ldots, d\}$. We refer to Section 2.1 for more details.

We use Theorem 1 to derive the following isolation result for NL, where the notation UTISP(t, s) stands for the class of languages accepted by unambiguous nondeterministic machines that run in time t and space s.

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

[3]We could alternately assign the weight of a vertex to each of its outgoing edges without affecting the total weight of any solution.

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 should be contrasted with the most space efficient simulation of NL on *deterministic* machines, which is given by Savitch's Theorem [Sav70]: $\mathrm{NL} \subseteq \mathrm{DSPACE}((\log n)^2)$. 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 known space bound for an algorithm that decides Reachability in polynomial time is only slightly sublinear, namely $n/2^{\Theta(\sqrt{\log n})}$ [BBRS98].

On the "negative" side we show:

Theorem 3. *Either one of the following hypotheses implies that* $\mathrm{NL} \subseteq \mathrm{L}/\mathrm{poly}$*:*

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.*[4]

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.

It is not clear to us that Theorem 3 should be viewed as a roadblock towards reducing the seed length and bitlength in Theorem 1 from $O((\log n)^{3/2})$ down to $O(\log n)$, and thereby show that NL = UL. Regarding the first part of Theorem 3, none of the known randomized isolations for Reachability are of the pruning type. This is because they map an instance $x \doteq (G, s, t)$ to an instance $f(x)$ where the underlying graph contains more vertices than G, which makes it impossible to meet the pruning requirement that $\Pi(f(x)) \subseteq \Pi(x)$.

The corresponding results for Circuit Certification and the complexity class LogCFL are stated in Section 3 (positive) and Section 4 (negative).

Techniques. The crux for our positive results is an iterative/recursive construction of a min-isolating weight assignment generator Γ. 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. In the case of Circuit Certification the kth level corresponds to the kth level of AND gates of the given circuit.

We develop several methods to build out of a min-isolating weight assignment w_k at the kth level, a min-isolating weight assignment w_{k+1} at the $(k+1)$st 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

[4]If $\mu(x) = \omega(x)$ on all instances x, we can easily decide Reachability in logspace as there exists a path from s to t in G if and only if $\omega(G, s, t) < \infty$.

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.,

$$\mathrm{NL} \subseteq \mathrm{UTISP}(\mathrm{poly}(n), (\log n)^2). \tag{1}$$

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 1 and its counterpart for Circuit Certification.

For Theorem 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 the unambiguous logspace machine of [RA00] 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 the unambiguous logspace machines from [RA00] 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 3 and its counterpart for Circuit Certification, follow along the lines of the argument for a similar result from [DKvMW13] in the time-bounded setting. The first part is the space-bounded equivalent of the main result in [DKvMW13]; 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 [DKvMW13] 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 and the result by Saks and Zhou [SZ99] that $\mathrm{BPL} \subseteq \mathrm{DSPACE}((\log n)^{3/2})$. Both have a recursive structure, use hashing,[5] 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 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 [SZ99], we do obtain the equivalent of a pseudorandom generator. As another contrast we are able improve the running time to polynomial, which remains open in the case of BPL [CCvM06]. 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 [Nis92a] to $\mathrm{BPL} \subseteq \mathrm{DTISP}(\mathrm{poly}(n), (\log n)^2)$ in [Nis92b].

The recent derandomization results for PerfectMatching on bipartite graphs [FGT16] and for polynomial identity testing (PIT) for read-once arithmetic branching programs [AGKS15] also

[5][SZ99] does so via Nisan's pseudorandom generator [Nis92a].

employ a combination of hashing and shifting but no balancing. They need $O((\log n)^2)$ 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. This question is related to the 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 substituted polynomial $Q(t) \doteq P(t^{w(1)}, t^{w(2)}, \ldots, t^{w(n)})$ remains nonzero.

Recently, Kalampalli and Tewari [KT16] independently proved the weaker inclusion (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 $\Theta((\log n)^2)$.

Very recently, Krishnan and Limaye [KL16] posted a report on ECCC in which they independently prove part 1 of our "negative" results (Theorem 3 and its counterpart for LogCFL), which follow from a space-bounded rendering of the main argument in [DKvMW13].[6]

Organization. In Section 2 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 3. 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. See Section 3.4 for more about the connection. In Section 4 we present our negative results for both settings. In Appendix A we review results from [RA00] and present a variation that we need for our LogCFL results. Appendix B contains the proofs of some folklore propositions that we include for completeness.

2. Reachability and NL

In this section we develop our min-isolating weight assignment generator for Reachability (Theorem 1), and derive our positive isolation result for NL (Theorem 2).

2.1. Weight Assignment Generator

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

Definition 4 (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 $(G, V \times V)$.*

[6]The current version of the report claims that the arguments also rely on [RA00], but the authors agree that [RA00] is not needed there (personal communication).

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 $E \subseteq \cup_{i \in [d]} (V_{i-1} \times V_i)$. We denote by $V_i \doteq V \cap [n] \times \{i\}$ the ith layer of G.

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., $\cup_{j=(i-1)\cdot 2^k}^{i \cdot 2^k} V_j$.

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 $V \setminus \cup_{i=0}^{2^{\ell-k}} V_{i \cdot 2^k}$.

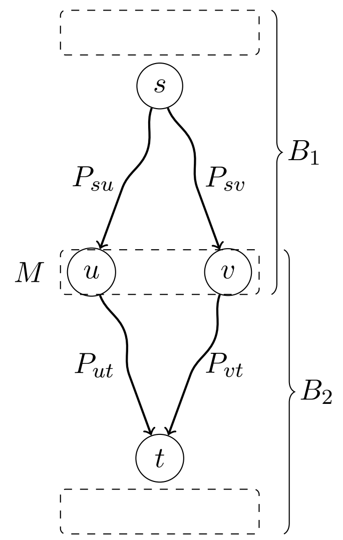

Figure 1

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 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.,

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

The new weights are assigned so as the 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,

$$\mu_k(s,u) + \mu_k(u,t) + w_{k+1}(u) \neq \mu_k(s,v) + \mu_k(v,t) + w_{k+1}(v), \tag{3}$$

  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 (3) as a *disambiguation requirement*. See Figure 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 2.2, 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 NL $\subseteq$ UL.

Shifting. For $v \in M \subseteq L_{k+1}$ we set $w_{k+1}(v) = \text{index}(u) \cdot b$, 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 (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 (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 (Universal hashing [CW79]). *There exists a logspace computable weight assignment generator* $(\Gamma^{(\text{hashing})}_{m,r})_{m,r\in\mathbb{N}}$ *with seed length* $s(m,r) = O(\log(mr))$ *such that* $\Gamma^{(\text{hashing})}_{m,r}$ *produces functions* $h : [m] \mapsto [r]$ *with the following property: For every* $u, v \in [m]$ *with* $u \neq v$*, and every* $a, b \in \mathbb{N}$

$$\Pr_h[a + h(u) = b + h(v)] \leq 1/r, \tag{4}$$

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

We identify $D \doteq [\![d]\!] \times [n]$ with $[m] = [d \cdot n]$ in a natural way. If we pick $h : D \mapsto [r]$ uniformly at random from $\Gamma^{(\text{hashing})}_{m,r}$ and set $w_{k+1} = h$ on L_{k+1}, (4) guarantees that each individual disambiguation requirement (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 charateristics of the family of hash functions $\Gamma^{(\text{hashing})}$ from Fact 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 (3) for iterations above $k+1$ depend on h, and we cannot guarantee the bound (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}

$$\mu_k(s,t) \leq \mu_{k+1}(s,t) \leq \mu_k(s,t) + r. \tag{5}$$

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 (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, (4) in Fact 2 shows that the disambiguations requirements with respect to μ'_{k+1}, i.e.,

$$\mu'_{k+1}(s,u) + \mu'_{k+1}(u,t) + h(u) \neq \mu'_{k+1}(s,v) + \mu'_{k+1}(v,t) + h(v), \tag{6}$$

hold with high probability. Undoing the truncation, (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 (5) becomes

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

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

$$\lfloor \mu_k(s,u)/b^{i-1}\rfloor + \lfloor \mu_k(u,t)/b^{i-1}\rfloor + h(u) \neq \lfloor \mu_k(s,v)/b^{i-1}\rfloor + \lfloor \mu_k(v,t)/b^{i-1}\rfloor + h(v) \tag{7}$$

for all appropriate choices of s,t,u,v. Equation (4) in Fact 2 and a union bound show that the requirements (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.

Alternately, $b \geq 2r$ is enough to ensure that the carries are no larger than 1. We can handle such carries by strengthening the disambiguation requirements (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 (4) in the union bound, and guarantees that the values remain distinct after undoing the truncation. In fact, it suffices to require for all $i \in [k'-k]$ that

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

for some $b \geq 4r$. We refer to the formal proof of Lemma 2 in Section 3.2 for more details (in the setting of Circuit Certification instead of Reachability), in particular to the argument for Claim 1.

We obtain the following characteristics: $R_{k'} = R_k + O(\log(dnr)) = R_k + O(\log n)$ 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 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 $R = O(\Delta/\ell \cdot \log n)$ and $W = O(\Delta/\ell \cdot n^\Delta)$, so $R + \log(W) = O((\Delta/\ell + \Delta)\log n)$. The latter expression is minimized up to constant factors when $\Delta/\ell = \ell$, 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 $\Gamma^{(\text{reach})}$ that is indexed by the number of vertices n and the depth d, with $D_{n,d} \doteq [n] \times [\![d]\!]$ as the domain for the weight functions given by $\Gamma^{(\text{reach})}_{n,d}$. The construction works for any d that is a power of 2 and any $n \in \mathbb{N}$, and has the properties stated in Theorem 1. 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., [Vol99]).

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 concludes a somewhat informal proof of Theorem 1. Section 3.2 contains a more formal proof (in the setting of Circuit Certification instead of Reachability).

2.2. Isolation

We now establish Theorem 2. The following proposition[8] shows that it suffices to construct a Turing machine that accepts Reachability unambiguously on layered digraphs in time poly(n) and space $O((\log n)^{3/2})$. For completeness we include a brief argument in Appendix B.

Proposition 1. Reachability *on layered digraphs is hard for* NL *under logspace mapping reductions that preserve the number of solutions.*

Given our weight assignment generator $\Gamma^{(\text{reach})}$, 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 $\Gamma^{(\text{reach})}$, 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 $\in \mathrm{R} \cdot \mathrm{PromiseUL}$ [GW96] to Reachability $\in \mathrm{R} \cdot (\mathrm{UL} \cap \mathrm{coUL})$ [RA00] can be viewed as following the same approach. Instead of the list of weight assignments obtained from $\Gamma^{(\text{reach})}$ (which is guaranteed to contain a min-isolating one), [RA00] 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.

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

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

Lemma 1 ([RA00]). *There exist unambiguous nondeterministic machines* $\textsc{WeightCheck}^{(\text{reach})}$ *and* $\textsc{WeightEval}^{(\text{reach})}$ *such that for every digraph* $G = (V, E)$ *on* n *vertices, weight assignment* $w : V \mapsto \mathbb{N}$*, and* $s, t \in V$*:*

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

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

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

Note that the machine $\textsc{WeightEval}^{(\text{reach})}$ 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, whereas the machine $\textsc{WeightEval}^{(\text{reach})}$ from Lemma 1 runs in time $\text{poly}(\log(W), n)$. We refer to Appendix A for the workings of the machines $\textsc{WeightEval}^{(\text{reach})}$ and $\textsc{WeightCheck}^{(\text{reach})}$.

Like [RA00], we call $\textsc{WeightCheck}^{(\text{reach})}(G, w)$ for each w from the list up and until the first success, and then call $\textsc{WeightEval}^{(\text{reach})}(G, w, s, t)$ with that first successful w. This describes a deterministic machine for $\textsc{Reachability}$ on layered digraphs that makes calls to the unambiguous nondeterministic machines $\textsc{WeightCheck}^{(\text{reach})}$ and $\textsc{WeightEval}^{(\text{reach})}$. 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.

Going over the list of all weight assignments produced by $\Gamma^{(\text{reach})}$ is done by going over all seeds σ, and producing the required bits of $w = \Gamma^{(\text{reach})}(\sigma)$ from σ on the fly whenever they are needed, without storing them. Given the logspace computability of $\Gamma^{(\text{reach})}$, the resulting unambiguous machine for $\textsc{Reachability}$ on layered digraphs runs in time $2^R \cdot \text{poly}(\log(W), n)$ and space $O(R + \log(W) + \log n)$, where R denotes the seed length of $\Gamma^{(\text{reach})}$, and W the maximum path length under a weight assignment that $\Gamma^{(\text{reach})}$ produces. With the parameters of $\Gamma^{(\text{reach})}$ stated in Theorem 1 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 $\Gamma^{(\text{reach})}$ by exploiting the internal structure of $\Gamma^{(\text{reach})}$. Recall from the final construction in Section 2.1 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 $\Gamma^{(\text{hashing})}$ from Fact 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 $\Gamma^{(\text{hashing})}_{m,r}$, consider $h_i \doteq \Gamma^{(\text{hashing})}_{m,r}(\sigma_i)$, 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 $\textsc{WeightCheck}^{(\text{reach})}(B, w)$ for each of the blocks B of depth 2^i, passing if and only if all of them pass. The correctness argument from Section 2.1 guarantees that the search always succeeds. Once we arrive at $w = w_\Delta$, we run $\textsc{WeightEval}^{(\text{reach})}(G, w, s, t)$ 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. A more formal proof in the setting of CIRCUIT CERTIFICATION and LogCFL is given in Section 3.3.

3. Circuit Certification and LogCFL

We start this section with some background on CIRCUIT CERTIFICATION and LogCFL, including known isolation results. We then state and formally prove our positive results for this setting. We also explain how they essentially imply the results for REACHABILITY and NL from Section 2.

3.1. Background

Gal and Wigderson [GW96] applied their approach for isolating REACHABILITY also to the following computational problem.

Definition 5 (Circuit certification). CIRCUIT CERTIFICATION *denotes 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.*

A *certificate* for a gate g in a Boolean circuit C on an input z is a minimal[9] subcircuit F of C with output gate g that accepts z, written $F(z) = 1$. 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.

[GW96] 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 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

[9]The restriction of minimality is imposed in some references (e.g., [RA00]) but not in others (e.g., [GW96]). We impose it as it allows for a bijection between certificates and accepting computation paths in the machine characterization of LogCFL.

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, [GW96] 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 [Ven91]. 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 [Sud78]. 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 $\text{AuxPDA-TISP}(t, s)$. The corresponding subclass for unambiguous machines is written as $\text{UAuxPDA-TISP}(t, s)$. 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 $\text{AuxPDA-TISP}(\text{poly}(n), O(\log n))$ 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.

Gal and Wigderson obtained a randomized disambiguation for Circuit Certification on shallow semi-bounded circuits that has success probability $1/\text{poly}(n)$, is computable in logspace with two-way access to the random bits, and is recoverable in logspace. This implies that $\text{LogCFL} \subseteq \text{R} \cdot \text{Promise}\mathcal{C}$ where $\mathcal{C} \doteq \text{UAuxPDA-TISP}(\text{poly}(n), O(\log n))$. Reinhardt and Allender [RA00] strengthened this result to $\text{LogCFL} \subseteq \text{R} \cdot (\mathcal{C} \cap \text{co}\mathcal{C})$, 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 $\text{LogCFL} \subseteq (\mathcal{C} \cap \text{co}\mathcal{C})/\text{poly}$ and that a disambiguation for Circuit Certification on shallow semi-unbounded circuits can be computed in logspace with polynomial advice.

3.2. 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 6 (Min-isolating weight assignment 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 $V = \uplus_{i \in [\![d]\!]} V_i$ where $V_i \subseteq [n] \times \{i\}$ and $E \subseteq \uplus_{i \in [d]} (V_{i-1} \times V_i)$. 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 $\Gamma^{(\text{cert})} = (\Gamma^{(\text{cert})}_{n,d})_{n,d \in \mathbb{N}}$, which is indexed by the number of gates n and the depth d. The domain of the weight assignments given by $\Gamma^{(\text{cert})}_{n,d}$ 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 4. *There exists a weight assignment generator $\Gamma^{(\text{cert})} = (\Gamma^{(\text{cert})}_{n,d})_{n,d \in \mathbb{N}}$ 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,*

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

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

The essential ingredient in the proof of Theorem 4 is the following formalization of the combined approach from Section 2.1 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.[10]

[10]The transformation claimed in the proof of Proposition 2 yields circuits of this type.

Lemma 2. *There exists a weight assignment generator $\Gamma^{(\text{cert,step})} = (\Gamma^{(\text{cert,step})}_{n,\ell,k,k'})$ 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 $V_{\leq i} \doteq \cup_{j \leq i} V_j$, then*

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

where the seed σ is chosen uniformly at random. Moreover, $\Gamma^{(\text{cert,step})}_{n,\ell,k,k'}(\sigma)$ assigns nonzero weights only to $\cup_{j \in [k+1,k']} L_j$, 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 $i \in [\![k'-k]\!]$:

$$w_{k+i} \text{ is min-isolating for } (C, z, V_{\leq 2(k+i)}). \tag{8}$$

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).

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.

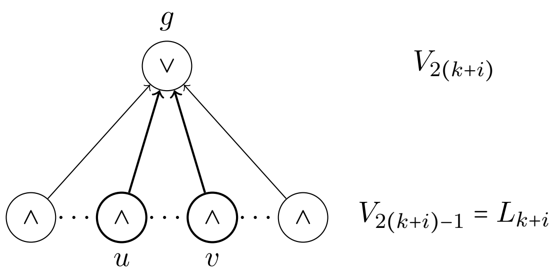

Figure 2

Claim 1. *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*

$$\left\lfloor \frac{\mu_k(u)}{b^{i-1}} \right\rfloor + h(u) \notin \left\lfloor \frac{\mu_k(v)}{b^{i-1}} \right\rfloor + h(v) + \{-1, 0, 1\}, \tag{9}$$

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

See Figure 2 for an illustration.

Proof (of Claim 1). 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},

$$\mu_{k+i}(v) = \mu_{k+i-1}(v) + h(v) \cdot b^{i-1}, \tag{10}$$

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

$$\mu_{k+i}(u) \neq \mu_{k+i}(v). \tag{11}$$

We argue that (11) follows from (9) 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

$$\mu_k(v) \leq \mu_{k+i-1}(v) \leq w_{k+i-1}(F_v) \leq w_k(F_v) + 4r \cdot b^{i-2} = \mu_k(v) + 4r \cdot b^{i-2}. \tag{12}$$

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 $u \in L_{k+i-j}$ we have that $w_{k+i-1}(u) = w_k(u) + h(v) \cdot b^{i-1-j} \leq w_k(u) + r \cdot b^{i-1-j}$. 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}, (12) 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

$$\left\lfloor \frac{\mu_k(v)}{b^{i-1}} \right\rfloor \leq \left\lfloor \frac{\mu_{k+i-1}(v)}{b^{i-1}} \right\rfloor \leq \left\lfloor \frac{\mu_k(v)}{b^{i-1}} \right\rfloor + 1 \tag{13}$$

for $b \geq 4r$. In combination with the hypothesis (9), (13) 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 (10) in turn implies (11) after undoing the division. This finishes the proof of Claim 1. ∎

Each individual disambiguation requirement (9) can be written as three conditions of the form (4). By Fact 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 (8) 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 $\Gamma^{(\text{cert,step})}$: $\Gamma^{(\text{cert,step})}_{n,\ell,k,k'}$ takes a seed σ for $\Gamma^{(\text{hashing})}_{n(2\ell+1),r}$, considers $h = \Gamma^{(\text{hashing})}_{n(2\ell+1),r}(\sigma)$ 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

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

The above analysis shows that $\Gamma^{(\text{cert,step})}$ 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 $\Gamma^{(\text{hashing})}_{m,r}$ given in Fact 2. They imply that $\Gamma^{(\text{cert,step})}_{n,\ell,k.k'}$ has seed length $O(\log(|D_{n,\ell,k,k'}| \cdot r) = O(\log n)$. As each bit of $(\Gamma^{(\text{cert,step})}(\sigma))(g)$ equals an easily determined bit of $h(g)$, the logspace computability of the universal family of hash functions implies the logspace computability of $\Gamma^{(\text{cert,step})}_{n,\ell,k,k'}$. This completes the proof of Lemma 2. ∎

We now turn to the proof of the theorem.

Proof (of Theorem 4). 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.[11]

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 Δ times successively, starting from an arbitrary weight assignment w_0. The ith application sets $k = k_i \doteq$

[11]This is the only case we need for the proof of Theorem 5 as the transformation described in Claim 6 in the proof of Proposition 2 yields such circuits.

$(i-1)\cdot\Delta$ and $k' = k'_i \doteq i\cdot\Delta$, uses a fresh seed σ_i for $\Gamma^{(\text{cert,step})}_{n,\ell,k_i,k'_i}$, sets $w_{i\cdot\Delta} = w_{(i-1)\cdot\Delta} + \Gamma^{(\text{cert,step})}_{n,\ell,k_i,k'_i}(\sigma_i)$, and tries to maintain the invariant that $w_{i\cdot\Delta}$ is min-isolating for $(C, z, V_{\le 2i\cdot\Delta})$. We end up with $w_\ell = w_0 + \Gamma^{(\text{cert,odd})}_{n,d}(\sigma_1, \sigma_2, \ldots, \sigma_\Delta)$, where

$$\Gamma^{(\text{cert,odd})}_{n,d}(\sigma_1, \sigma_2, \ldots, \sigma_\Delta) \doteq \sum_{i\in[\Delta]} \Gamma^{(\text{cert,step})}_{n,\ell,k_i,k'_i}(\sigma_i). \tag{14}$$

The superscript "odd" in $\Gamma^{(\text{cert,odd})}$ refers to the fact that only odd layers receive nonzero values under weight assignments generated by $\Gamma^{(\text{cert,odd})}$. 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 $\Delta/n^2 \le 1/n$. Thus, for any fixed $w_0 : [n]\times[\![d]\!] \mapsto \mathbb{N}$, $w_0 + \Gamma^{(\text{cert,odd})}_{n,d}$ is min-isolating for (C, z) with probability at least $1 - 1/n$. The seed length of $\Gamma^{(\text{cert,odd})}_{n,d}$ is Δ times the one of $\Gamma^{(\text{cert,step})}_{n,d,\cdot,\cdot}$, i.e., $O(\Delta \log n) = O(\sqrt{d}\log n)$. The maximum weight assigned by $\Gamma^{(\text{cert,odd})}_{n,d}$ is at most Δ times the one assigned by $\Gamma^{(\text{cert,step})}_{n,d,\cdot,\cdot}$, so the bitlength of $\Gamma^{(\text{cert,odd})}_{n,d}$ is $O(\log(\Delta) + \Delta\cdot\log n) = O(\sqrt{d}\log n)$. The logspace computability of $\Gamma^{(\text{cert,step})}$ and the fact that iterated addition can be computed in logspace (see, e.g., [Vol99] imply that $\Gamma^{(\text{cert,odd})}_{n,d}$ 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 $\Gamma^{(\text{cert,odd})}$ with all the properties required of $\Gamma^{(\text{cert})}$ 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, transfering the weight of each fresh AND gate g to the leaf u that feeds into it. This results in a weight assignment generator $\Gamma^{(\text{cert,even})}$ that only assigns nonzero weights to the even layers, and has all the properties required of $\Gamma^{(\text{cert})}$ 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 $w'_0 + \Gamma^{(\text{cert,even})}_{n,d}$ is min-isolating for (C, z) with probability at least $1 - 1/n$.

We claim that

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

satisfies all requirements irrespective of the type of V_1, provided that we pick the seeds for $\Gamma^{(\text{cert,odd})}_{n,d}$ and $\Gamma^{(\text{cert,even})}_{n,d}$ independently. This follows from the above analysis by setting $w_0 = \Gamma^{(\text{cert,even})}_{n,d}$ and $w'_0 = \Gamma^{(\text{cert,odd})}_{n,d}$, and finishes the proof of Theorem 4. ∎

3.3. Isolation

We use the weight assignment generator from Theorem 4 to establish the following result

Theorem 5. LogCFL $\subseteq$ UAuxPDA-TISP$(\text{poly}(n), (\log n)^{3/2})$.

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. For completeness we include a proof of the proposition in Appendix B.

Proposition 2. Circuit Certification *on shallow layered alternating semi-unbounded Boolean circuits is hard for* LogCFL *under logspace mapping reductions that preserve the number of solutions.*

Our unambiguous machine for Circuit Certification hinges on our weight assignment generator for the problem as well as the following unambiguous machines.

Lemma 3. *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 $\text{poly}(2^d, \log(W), n)$ *and space* $O(d+\log(W)+\log(n))$*, where W denotes an upper bound on the finite values $w(C, z, g)$ for $g \in V$.*

Lemma 3 is an improvement of a result in [RA00] that 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 Circuit Certification on shallow layered alternating semi-unbounded circuits run in polynomial time. We refer Appendix A for a proof and further discussion.

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

Proof (of Theorem 5). By way of Proposition 2, it suffices to construct an unambiguous machine that decides[12] Circuit Certification on layered alternating semi-unbounded Boolean circuits $C = (V, E)$ of size n and depth $d \le \log(n)$, 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 1 does the job.

Consider the version of our weight assignment generator $\Gamma^{(\text{cert})}$ from Theorem 4 that is geared towards such circuits, namely $\Gamma^{(\text{cert,odd})}$ given by (14). We know that on most seeds $\Gamma^{(\text{cert,odd})}_{n,d}$ produces a weight assgnment w that is min-isolating for (C, z). The machine CircuitEval in Algorithm 1 constructs such a seed. In fact, it constructs the lexicographically first such seed.

[12]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.

Algorithm 1: CircuitEval(C, z, g)

```
Input    : $C = (V, E)$: layered semi-unbounded circuits of depth $d$ with layers $V_0, V_1, \dots, V_d$
           $z$: input for $C$
           $g \in V$
Promise: $d = 2\Delta^2$ for $\Delta \in \mathbb{N}$ and $V_1$ consists of ANDs
Output  : $g(z)$
1  for $i \leftarrow 1$ to $\Delta$ do
2      foreach $\sigma_i \in \{0,1\}^{s(n,d)}$ in lex order do
3          isolating $\leftarrow$ true;
4          foreach $v \in V_{\le 2i\Delta}$ in lex order do
5              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
6                  isolating $\leftarrow$ false;
7                  exit the for loop over $v$;
8          end
9          if isolating then exit the loop over $\sigma_i$;
10     end
11 end
12 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
13     accept and return 1
14 else accept and return 0;
```

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 $\Gamma^{(\text{cert,step})}_{n,d,\cdot,\cdot}$. Note that $w \doteq \Gamma^{(\text{cert,odd})}_{n,d}(\sigma_1, \ldots, \sigma_\Delta)$ is min-isolating for (C, z) if and only if

$$w_{i\cdot\Delta} \doteq \sum_{j=1}^{i} \Gamma^{(\text{cert,step})}_{n,d,(j-1)\Delta,j\Delta}(\sigma_j) \text{ is min-isolating for } (C, V_{\leq 2i\Delta}) \tag{15}$$

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 1 implements this search. In the ith iteration it finds the lexicographically first σ_i satisfying the invariant (15), 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 $\textsc{WeightCheck}^{(\text{cert})}(C_v, z, w_{i\cdot\Delta})$ 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 $\textsc{WeightEval}^{(\text{cert})}(C, z, w, g)$ 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 (15) and the specifications of $\textsc{WeightCheck}^{(\text{cert})}$ and $\textsc{WeightEval}^{(\text{cert})}$. The unambiguity of CircuitEval follows from the unambiguity of $\textsc{WeightCheck}^{(\text{cert})}$ and $\textsc{WeightEval}^{(\text{cert})}$ (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 $\textsc{WeightCheck}^{(\text{cert})}$, and the fact that iterated addition is in logspace (see, e.g., [Vol99]). 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 $\log(\Delta) + s(n,d) + \log(W)$ to the space bound. The time and space needed for the call to $\textsc{WeightEval}^{(\text{cert})}$ 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 $d \leq n$. ∎

3.4. Reachability through Circuit Certification

We now explain how our results for Circuit Certification and LogCFL essentially imply the ones for Reachability and NL from Section 2. We review the reduction from Reachability to Circuit Certification given by Savitch's Theorem, and show how it yields our weight assignment generator $\Gamma^{(\text{reach})}$ for Reachability from a slight modification $\tilde{\Gamma}^{(\text{cert,odd})}$ of our weight assignment generator $\Gamma^{(\text{cert,odd})}$ for Circuit Certification, and that min-isolation of $\tilde{\Gamma}^{(\text{cert,odd})}$ on a reduced instance is equivalent to a restricted version of min-isolation of $\Gamma^{(\text{reach})}$ on the original instance. The reduction also allows us to obtain alternate unambiguous machines for Reachability and NL meeting the requirements of Theorem 2 from the unambiguous machines for Circuit Certification and LogCFL of Theorem 5.

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 NL $\subseteq$ LogCFL, and is equivalent to the following statement.

Proposition 3. *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 $\text{Reach}_k(s,t)$ of whether there exists a path of at most k edges from s to t in G as

$$\bigvee_{v \in V} \text{Reach}_k(s, v, t), \tag{16}$$

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 7 (Reachability instances of Circuit Certification). *The instances of* CIRCUIT CERTIFICATION *that result from the reduction in Proposition 3 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 3 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 (16) 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 $A_{\leq \ell} \doteq \cup_{k \leq \ell} A_k$ where

  $$A_k \doteq \cup_{i \in [d/2^k]} V_{(i-1)\cdot 2^k} \times V_{i \cdot 2^k}. \tag{17}$$

  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).

  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

$$\tilde{w} \text{ is min-isolating for } (C,z) \Leftrightarrow w \text{ is min-isolating for } (G, A_{\leq \ell}), \tag{18}$$

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 $\Gamma^{(\text{cert})}$ that is geared towards circuits (like C) with an AND layer next to the leaves, namely $\Gamma^{(\text{cert,odd})}$ given by (14). $\Gamma^{(\text{cert,odd})}$ 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 $\Gamma^{(\text{cert,odd})}$ so that it does.

The critical part in our analysis of the weight assignment generator $\Gamma^{(\text{cert,odd})}$ is the disambiguation requirements (9). 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.[13] 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 (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 $\Gamma^{(\text{hashing})}_{\tilde{m},\tilde{r}}$, 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 $\Gamma^{(\text{hashing})}_{\tilde{m},\tilde{r}}$ 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 $\Gamma^{(\text{hashing})}_{m,r}$, without affecting the analysis.

[13]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.

The result can be interpreted as the following modification $\tilde{\Gamma}^{(\text{cert,odd})}$ of our weight assignment generator $\Gamma^{(\text{cert,odd})}$. In order to formally express the relationship, 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 $L_{\leq \ell} = V \setminus (V_0 \cup V_d)$. 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 2.1 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 3. Moreover, by (18) 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. While the min-isolation property that we obtain for $\Gamma^{(\text{reach})}$ via the alternate route is weaker than via the direct route, it is still sufficient to derive the unambiguous machines for Reachability and NL from Theorem 2. This is because the weaker property is compatible with the constructions in Lemma 1 – see Lemma 6 in Appendix A. In fact, Lemma 6 can be obtained from the corresponding result in the setting of Circuit Certification and LogCFL, namely Lemma 3. Applying the reduction from Reachability to Circuit Certification as above to Lemma 3 yields Lemma 6 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 3 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. We refer to the proofs Appendix A for more details.

For the same reason the unambiguous machine CircuitEval in the proof of Theorem 5 does not need access to its stack on Reachability instances. In combination with Proposition 1, this observation yields Theorem 2 as a corollary to Theorem 5.

4. Limitations

In this section we prove our "negative result" for isolating Reachability (Theorem 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 $\Pi(x) \neq \varnothing$.

The first part of Theorem 3 follows by verifying that the main result of Dell, Kabanets, Van Melkebeek, and Watanable [DKvMW13] 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 $\frac{2}{3}+\frac{1}{\mathrm{poly}(n)}$. [DKvMW13] showed that the 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 [DKvMW13] 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 3 and its counterpart for Circuit Certification. The second part follows from a slight modification of the argument.

The proof in [DKvMW13] relies on a proposition of Ko's [Ko83].

Proposition 4 ([Ko83]). *Suppose that there exists a predicate $T : D \times D \mapsto \{0,1\}$ for some $D \subseteq X$ with the following properties:*

$$(\forall x, z \in D \cap L(\Pi))\; T(x,z) \vee T(z,x) \tag{19}$$
$$(\forall x, z \in D)\; z \in L(\Pi) \wedge T(z,x) \Rightarrow x \in L(\Pi) \tag{20}$$

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

$$x \in L(\Pi) \Leftrightarrow (\exists i \in [\ell])\, T(z_i^*, x). \tag{21}$$

If the $\vee$ in (19) were replaced by an exclusive or, T would be a tournament, where $T(z,x)$ (an edge from z to x) means that x wins the duel between z and x. Equation (19) requires the digraph T to contain a tournament (and have a selfloop at every vertex), so every duel has at least one winner and can have two. Equation (20) can be interpreted as saying that winners of duels are more likely be to in $L(\Pi 0$ 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 4 follows from the fact that a tournament on $D \cap \Pi$ 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 4 shows us via (21) how to decide $L(\Pi 0$ efficiently on D with the help of the predicate T and the $\ell \cdot n \leq n^2$ bits of advice z_i^* for $i \in [\ell]$.

[DKvMW13] constructs a (sufficiently) efficient predicate T satisfying (19) and (20) 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 8 (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 $\Pi(x_1 \sqcup x_2) = \uplus_{i \in [2]} \tau(x_1, x_2, i, \Pi(x_i))$, where $\tau(x_1, x_2, i, W) \doteq \uplus_{y \in W} \{\tau(x_1, x_2, i, y)\}$ 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 simple disjoint union operators that are computable in logspace, meaning that both $\sqcup$ and τ in Definition 8 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. For completeness we sketch a construction for the latter two in Appendix B.

Proposition 5. 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 [DKvMW13] is (i) 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 in the above sense, and (ii) that the corresponding predicate T satisfying Ko's requirements (19) and (20) can be decided sufficiently efficiently on instances with few solutions provided that f is efficiently computable and that Π allows an efficient disjoint union operator. The corresponding predicate T can formally be defined as follows on all pairs of instances $(z, x) \in X \times X$:

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

where $\leq_{lex}$ denotes the lexicographic ordering. The isolation property $|\Pi(f(\cdot))| \leq 1$ implies condition (19). The pruning property $\Pi(f(\cdot)) \subseteq \Pi(\cdot)$ implies condition (20). In the case of an instance z^* with a unique solution, say $\Pi(z^*) = \{y^*\}$, we can evaluate $T(z^*, z)$ as

$$T(z^*,x) \Leftrightarrow \begin{cases} \neg\pi(f(z^* \sqcup x), \tau(z^*,x,1,y^*)) & \text{for } z^* \leq_{lex} x \\ \neg\pi(f(x \sqcup z^*), \tau(x,z^*,2,y^*)) & \text{for } x \leq_{lex} z^*. \end{cases} \tag{22}$$

Given x, z^*, and y^*, the latter expression 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 of size n 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 [DKvMW13] regarding the existence of deterministic polynomial-time prunings for Satisfiability. The logspace incarnation yields the first part of Theorem 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 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. 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 (19) and (20) on instances (z^*, x) where z^* has a unique solution y^*:

$$T(z^*,x) \Leftrightarrow \begin{cases} \omega(z^* \sqcup x, \tau(z^*,x,1,y^*)) \neq \mu(z^* \sqcup x) & \text{for } z^* \leq_{lex} x \\ \omega(x \sqcup z^*, \tau(x,z^*,2,y^*)) \neq \mu(x \sqcup z^*) & \text{for } x \leq_{lex} z^*. \end{cases} \tag{23}$$

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. However, whereas access to the functions ω and μ does not immediately yield an efficient pruning, it does yield an efficient disambiguation in case 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 each of SATISFIABILITY, REACHABILITY, and CIRCUIT CERTIFICATION on shallow semi-unbounded circuits, both for polynomial-time efficiency and for logspace efficiency.

This completes the argument for parts 1 and 2 of Theorem 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 $\frac{2}{3} + \frac{1}{\operatorname{poly}(n)}$, some additional properties of Π are needed and the predicate T has to be generalized in the appropriate way. The following lemma captures the general case. We view a randomized mapping as a determinstic 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 4. *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 8 where $\sqcup$ and τ are computable in logspace.*
- *Π has a randomized disambiguation g with probability of success at least $1/\operatorname{poly}(n)$ that is computable and recoverable in logspace.*
- *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, and there exists a constant c such that the solutions to instances of Π of size n are strings of length n^c.*

For any $p = \frac{2}{3} + \frac{1}{\operatorname{poly}(n)}$ either of the following hypotheses imply that $L(\Pi 0$ 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 $x \in X$:*
$$\Pr_{\rho}[\ f_\rho \text{ satisfies the pruning requirement on input } x\ ] \geq p. \tag{24}$$
2. *There exist randomized mappings $\omega : X \times Y \times \mapsto \mathbb{N}$ and $\mu : X \mapsto \mathbb{N}$ that are computable in space $O(\log n)$ such that for every $x \in L(\Pi)$*
$$\Pr_{\rho}[\ \omega_\rho(x,\cdot) \text{ is min-isolating for } x \text{ and } \mu_\rho(x) = \omega_\rho(x)\ ] \geq p. \tag{25}$$

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:

$$T(z^*, x) \Leftrightarrow \begin{cases} \Pr_\rho[\text{ right-hand side of (22) holds }] > q & \text{for part 1} \\ \Pr_\rho[\text{ right-hand side of (23) holds }] > q & \text{for part 2,} \end{cases} \tag{26}$$

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. *Both* (19) *and* (20) *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 (19). 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 $x^* \doteq \min(x, z^*) \sqcup \max(x, z^*)$ and max and min refer to the lexicographic order $\leq_{lex}$. By complementing, with probability at least $1 - 2q$ it is the case that both 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 (25) as long as $q < p/2$.

We argue condition (20) by contradiction also. For part 1, consider $z^* \in D \cap L(\Pi)$ and $x \in D \smallsetminus L(\Pi)$, and let $x^* \doteq \min(x, z^*) \sqcup \max(x, z^*)$. Note that if the right-hand side of (22) holds then f fails the pruning property on input x^*. Thus, if $T(z^*, x)$ holds, then f fails the pruning property on input x^* with probability more than q, which contradicts the hypothesis (24) as long as $q \geq 1-p$. For part 2, a similar argument leads to a contradiction with the hypothesis (25) 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. ∎

Note that the statement of the lemma entails some leeway in that p does not just exceeds $2/3$ 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 (24) (in the case of part 1) or (25) (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 (26) 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, the predicate h to retrieve a solution in case it is unique, and the predicate π to check purported solutions.

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 $x \in X$ of size n there exists at least one $\sigma \in S$ such that g_σ satisfies the disambiguation requirement on input x, i.e., $x_\sigma \doteq g_\sigma(x)$ is an instance of Π that is equivalent with respect to membership to $L(\Pi)$, and has at most one solution. Thus, we can apply our algorithm A to decide $L(\Pi)$ on x_σ.

We do not know which σ works but we do know that there is at least one and that for anyone that does, the only possible solution for the instance x_σ of Π is $(L(\Pi)(h(x_\sigma, i)))_{i=1}^{n^c}$. 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, and by definition $h(x_\sigma, i)$ is an instance of Π whose memberhip to $L(\Pi)$ is equivalent to the latter decision. 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 only candidate 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 $g'(x, x_\sigma, y_\sigma)$ is a valid solution for x, where $g' : X \times X \times Y \mapsto Y$ denotes the logspace recovery algorithm underlying the recoverable disambiguation g. More formally, we evaluate the predicate

$$\bigvee_{\sigma \in S} \pi(x, g'(x, x_\sigma, y_\sigma)). \tag{27}$$

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, and $g'(x, x_\sigma, y_\sigma)$ is a valid solution to x, so (27) evaluates to true. If $x \notin L(\Pi)$, then there is no string y for which $\pi(x, y)$ holds, so (27) evaluates to false no matter what. Thus, (27) 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 (27) can be evaluated in logspace with polynomial advice. This concludes the proof of Lemma 4. ∎

Theorem 3 follows from an instantiation of Lemma 4 with REACHABILITY on layered digraphs as the computational problem Π.

Proof (of Theorem 3). Since REACHABILITY on layered digraphs is hard for NL under logspace mapping reductions (see Proposition 1), it suffices to verify that REACHABILITY on layered digraphs has all the properties required of the computational problem Π in Lemma 4. The properties regarding the predicate π and the disjoint union operator follow from Proposition 5. The existence of the required randomized disambiguation g follows from the Isolation Lemma (as explained in the introduction). 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 3.

Theorem 6. *Either of the following hypotheses imply that* LogCFL $\subseteq$ L/poly*:*

1. Circuit Certification *on shallow layered alternating semi-unbounded circuits has a logspace pruning.*

2. Circuit Certification *on shallow layered alternating semi-unbounded circuits 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 $\frac{2}{3} + \frac{1}{\text{poly}(n)}$ and the algorithms run in logspace when given two-way access to the random bits.

References

[AAK89] 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.

[AGGT16] 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.

[AGKS15] 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.

[AH94] E. Allender and U. Hertrampf. Depth reduction for circuits of unbounded fan-in. *Information and Computation*, 112(2):217–238, 1994.

[AM08] 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.

[AMS10] V. Arvind, P. Mukhopadhyay, and S. Srinivasan. New results on noncommutative and commutative polynomial identity testing. *Computational Complexity*, 19(4):521–558, 2010.

[ARZ99] 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.

[BBRS98] 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.

[BCMR13] 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.

[BDCGL92] 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.

[BH14] 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.

[Bjö14] A. Björklund. Determinant sums for undirected hamiltonicity. *SIAM Journal on Computing*, 43(1):280–299, 2014.

[BRS91] 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.

[BTV09] C. Bourke, R. Tewari, and N. V. Vinodchandran. Directed planar reachability is in unambiguous log-space. *ACM Transactions on Computation Theory*, 1(1), 2009.

[CCvM06] 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.

[CIKP03] C. Calabro, R. Impagliazzo, V. Kabanets, and R. Paturi. The complexity of unique k-SAT: an isolation lemma for k-CNFs. In *Proceedings of the 18th Annual IEEE Conference on Computational Complexity*, pages 135–141, 2003.

[CKN13] 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.

[CNP$^+$11] 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.

[CRS95] 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.

[CW79] J. L. Carter and M. N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979.

[DHK14] 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.

[DKM$^+$15] 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.

[DKR10] S. Datta, R. Kulkarni, and S. Roy. Deterministically isolating a perfect matching in bipartite planar graphs. *Theory Comput. Syst.*, 47(3):737–757, 2010.

[DKTV12] 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.

[DKvMW13] H. Dell, V. Kabanets, D. van Melkebeek, and O. Watanabe. Is Valiant-Vazirani's isolation probability improvable? *Computational Complexity*, 22(2):345–383, 2013.

[DS08] 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.

[EW10] J. Erickson and P. Worah. Computing the shortest essential cycle. *Discrete and Computational Geometry*, 44(4):912–930, 2010.

[FGT16] 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.

[FK13] F. V. Fomin and P. Kaski. Exact exponential algorithms. *Communications of the ACM*, 56(3):80–88, March 2013.

[GSW12] D. Gamarnik, D. Shah, and Y. Wei. Belief propagation for min-cost network flow: Convergence and correctness. *Operations Research*, 60(2):410–428, 2012.

[GW96] A. Gál and A. Wigderson. Boolean complexity classes vs. their arithmetic analogs. *Random Struct. Algorithms*, 9(1-2):99–111, 1996.

[HN16] H. Hirai and H. Namba. Shortest $(A+B)$-path packing via hafnian. *Computing Research Repository*, abs/1603.08073, 2016.

[HR14] 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.

[Imm88] Neil Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17(5):935–938, 1988.

[KBB$^+$11] 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.

[KL16] V. Krishan and N. Limaye. Isolation lemma for directed reachability and NL vs. L. *Electronic Colloquium on Computational Complexity*, 23:155, 2016.

[KMO$^+$13] 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.

[Ko83] K. Ko. On self-reducibility and weak P-selectivity. *Journal of Computer and System Sciences*, 26(2):209–211, 1983.

[KS01] 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.

[KT16] 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.

[KUW86] R. M. Karp, E. Upfal, and A. Wigderson. Constructing a perfect matching is in random nc. *Combinatorica*, 6(1):35–48, 1986.

[KV10] 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.

[KvM02] 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.

[KVVY93] 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.

[LK89] A. Lingas and M. Karpinski. Subtree isomorphism is NC reducible to bipartite perfect matching. *Information Processing Letters*, 30(1):27 – 32, 1989.

[LP15] A. Lingas and M. Persson. A fast parallel algorithm for minimum-cost small integral flows. *Algorithmica*, 72(2):607–619, 2015.

[MVV87] 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.

[MW01] 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.

[Nis92a] N. Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

[Nis92b] 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.

[OS93] J. B. Orlin and C. Stein. Parallel algorithms for the assignment and minimum-cost flow problems. *Operations research letters*, 14(4):181–186, 1993.

[RA00] K. Reinhardt and E. Allender. Making nondeterminism unambiguous. *SIAM Journal on Computing*, 29(4):1118–1131, 2000.

[Sav70] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

[Str13] Y. Strozecki. On enumerating monomials and other combinatorial structures by polynomial interpolation. *Theory of Computing Systems*, 53(4):532–568, 2013.

[Sud78] I. H. Sudborough. On the tape complexity of deterministic context-free languages. *Journal of the ACM*, 25(3):405–414, July 1978.

[SZ99] M. E. Saks and S. Zhou. $\mathrm{BP}_H\mathrm{SPACE}(S) \subseteq \mathrm{DSPACE}(S^{3/2})$. *Journal of Computer and System Sciences*, 58(2):376–403, 1999.

[Sze88] Robert Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26(3):279–284, 1988.

[Tar93] J. Tarui. Probabilistic polynomials, AC0 functions and the polynomial-time hierarchy. *Theoretical Computer Science*, 113(1):167–183, 1993.

[Tod91] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991.

[Tra08] P. Traxler. The time complexity of constraint satisfaction. In *Proceedings of the 3rd International Workshop on Parameterized and Exact Computation*, pages 190–201, 2008.

[TW15] 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.

[Ven91] H. Venkateswaran. Properties that characterize LOGCFL. *Journal of Computer and System Sciences*, 43(2):380–404, October 1991.

[Vol99] H. Vollmer. *Introduction to Circuit Complexity: A Uniform Approach.* Springer-Verlag New York, 1999.

[VV86] L. G. Valiant and V. V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47(3):85–93, 1986.

[WT93] O. Watanabe and S. Toda. Structural analysis of the complexity of inverse functions. *Mathematical Systems Theory*, 26(2):203–214, 1993.

A. Checking Min-Isolation and Computing Min-Weights under Min-Isolation

This appendix presents the unambiguous logspace machines from Lemma 1 and Lemma 3 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 1 in the proof of Theorem 2 in the setting of REACHABILITY and NL, and Lemma 3 in the proof of Theorem 5 in the setting of CIRCUIT CERTIFICATION and LogCFL.

Both lemmas follow from the results and techniques of Reinhardt and Allender [RA00]. The machines claimed in Lemma 1 appear as are in [RA00]; the machines in Lemma 3 are slight variations with improved running times, which are necessary for the proof of Theorem 5. As such, we present a full proof of Lemma 3, and only sketch the proof of Lemma 1 . In fact, we establish the following extension of Lemma 3, which will aid us with the proof sketch for Lemma 1.

Lemma 5. *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 $\text{poly}(2^d, \log(W), n)$ *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 extension consists of the "moreover" clause, which refers to the "Reachability instances" from Definition 7. The clause enables us to formally derive from Lemma 5 a variant of Lemma 1 that is sufficiently strong for the proof of Theorem 2. We explain this after the proof of Lemma 5.

The key tool in [RA00] is a modification of the inductive counting technique of Immerman and Szelepcsenyi [Imm88, Sze88] where besides computing the values

$$n_k \doteq |\{g \in V_k : g(z) = 1\}| \tag{28}$$

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

$$s_k \doteq \sum_{g \in V_k : g(z)=1} w(C, z, g) \tag{29}$$

in sync. As in the proofs in [Imm88, Sze88] 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 [GW96], Reinhardt and Allender [RA00] 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 5). 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 (28) and (29) 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 3. *The nondeterministic machine* PromiseWeightEval *given in Algorithm 2 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,* PromiseWeightEval *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.*

Algorithm 2: PromiseWeightEval$(C, z, w, k, n_k, s_k, g)$

```
Input    : $C = (V, E)$: layered semi-unbounded circuits of depth $d$ with layers $V_0, V_1, \dots, V_d$
           $z$: 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 (28) and (29)
Output   : $w(C, z, g)$
1  $n \leftarrow 0$, $s \leftarrow 0$, $\omega \leftarrow \infty$;
2  foreach $v \in V_k$ in lex order do
3      guess whether $v(z) = 1$;
4      if the guess is "yes" then
5          guess a candidate certificate $F$ for $(C, z, v)$, check its validity, and compute $w(F)$;
6          if $F$ is valid then
7              $n \leftarrow n + 1$;
8              $s \leftarrow s + w(F)$;
9              if $v = g$ then $\omega \leftarrow w(F)$;
10         else reject;
11 end
12 if $n = n_k$ and $s = s_k$ then
13     accept and return $\omega$
14 else reject;
```

Proof (of Claim 3). 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 12. We argue that $n = n_k$ holds in that line if and only if all nondeterministic 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 12 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 12 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.

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 space efficient 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 $\text{poly}(n)$) and the addition of numbers bounded by W (time $\text{poly}(\log(W))$). Thus, the overall running time is bounded by $\text{poly}(2^d, \log(W), n)$.

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 3. ∎

Next we build on PromiseWeightEval to efficiently and unambiguously check whether w is min-isolating for (C, z, V_{k+1}) assuming that it is for (C, z, V_k), 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 4. *The nondeterministic machine* PromiseWeightCheck *given in Algorithm 3 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* $\text{poly}(2^d, \log(W), n)$ *and space* $O(\log(W) + \log(n))$. PromiseWeightCheck *does not need the stack on* Reachability *instances.*

Proof (of Claim 4). We first argue correctness. Consider an input as in the specification, satisfying the promise. Note that the calls that PromiseWeightCheck makes to PromiseWeightEval all 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

Algorithm 3: PromiseWeightCheck(C, z, w, k, n_k, s_k)

```
Input   : $C = (V, E)$: layered semi-unbounded circuits of depth $d$ with layers $V_0, V_1, \dots, V_d$
          $z$: 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 (28) and (29)
Output  : $(\text{true}, n_{k+1}, s_{k+1})$ satisfying (28) and (29) if $w$ is min-isolating for $(C, z, V_{k+1})$
          $(\text{false}, -, -)$ otherwise

1  $(n, s) \leftarrow (0, 0)$;
2  foreach $g \in V_{k+1}$ in lex order do
3      if $g$ is an AND gate then
4          let $u$ and $v$ be the gates feeding into $g$ in lex order;
5          $\mu \leftarrow$ PromiseWeightEval($C, z, w, k, n_k, s_k, u$);
6          $\nu \leftarrow$ PromiseWeightEval($C, z, w, k, n_k, s_k, v$);
7          if $\mu < \infty$ and $\nu < \infty$ then
8              $(n, s) \leftarrow (n + 1, s + w(g) + \mu + \nu)$;
9      else { $g$ is an OR gate }
10         $(currmin, prevmin) \leftarrow (\infty, \infty)$;
11         foreach gate $v$ that feeds into $g$ in lex order do
12             $\nu \leftarrow$ PromiseWeightEval($C, z, w, k, n_k, s_k, v$);
13             if $\nu \le currmin$ then
14                 $prevmin \leftarrow currmin$;
15                 $currmin \leftarrow \nu$ ;
16         end
17         if $currmin < prevmin$ then
18             $(n, s) \leftarrow (n + 1, s + w(g) + currmin)$
19         if $currmin < \infty$ then
20             accept and return $(\text{false}, -, -)$
21 end
22 accept and return $(\text{true}, n, s)$;
```

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 $w(C,z,g) = w(g)+w(C,z,v)$ 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).

Modulo the calls to PromiseWeightEval, the machine 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 4. ∎

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 5. *The nondeterministic machine* WeightCheckEval *given in Algorithm 4 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* $O(\log(W) + \log(n))$. WeightCheckEval *does not need the stack on* Reachability *instances.*

Algorithm 4: 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 : $(\mathrm{true}, w(C, z, g))$ if $w$ is min-isolating for $(C, z)$; $(\mathrm{false}, -)$ otherwise
1  $(n, s) \leftarrow (0, 0)$;
2  foreach $v \in V_0$ in lex order do
3      if $v(z) = 1$ then $(n, s) \leftarrow (n + 1, s + w(v))$;
4  end
5  $(isolating, k) \leftarrow (\mathrm{true}, 0)$;
6  if $g \in V_0$ then
7      if $g(z) = 1$ then $\omega \leftarrow w(g)$ else $\omega \leftarrow \infty$;
8  while isolating and $k < d$ do
9      $(isolating, n, s) \leftarrow$ PromiseWeightCheck$(C, z, w, k, n, s)$;
10     if isolating and $g \in V_{k+1}$ then $\omega \leftarrow$ PromiseWeightEval$(C, z, w, k + 1, n, s, g)$;
11     $k \leftarrow k + 1$;
12 end
13 if isolating then
14     accept and return $(\mathrm{true}, \omega)$
15 else accept and return $(\mathrm{false}, -)$;
```

Proof (of Claim 5). Consider an input as in the specification. WeightCheckEval maintains the following loop invariants (which hold each time the loop condition in line 8 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 (28) and (29).

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 PromiseWeightCheck on the input (C, z, w, k, n_k, s_k), knowing that w is min-isolating for $(C, z, V_{\leq k})$. By the specification of PromiseWeightCheck, this means that the call 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 $V_{\le k+1}$ 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 WeightCheckEval correctly returns (false, −) by the first invariant; or else it halts with *isolating* = true and $k = d$, in which case it correctly returns $(\text{true}, w(C, g, z))$ 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).

Modulo the subroutine calls, WeightCheckEval 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 call to PromiseWeightCheck takes time $\text{poly}(2^d, \log(W), n)$ 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 $\text{poly}(2^d, \log(W), n)$ and space $O(\log(W) + \log(n))$ with the use of a stack (as the space needed for subsequent 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 5. ∎

The machines $\text{WeightCheck}^{(\text{cert})}$ and $\text{WeightEval}^{(\text{cert})}$ in the statement of Lemma 5 immediately follow from the machine WeightCheckEval from Claim 5. ∎

Lemma 5 trivially implies Lemma 3. Lemma 5 also yields the following variant of Lemma 1 in the setting of Reachability and NL.

Lemma 6. *There exist unambiguous nondeterministic machines* $\text{WeightCheck}^{(\text{reach,block})}$ *and* $\text{WeightEval}^{(\text{reach,block})}$ *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) $\text{WeightCheck}^{(\text{reach,block})}(G, w)$ *decides whether or not* w *is min-isolating for* $(G, A_{\le \ell})$ *where* $A_{\le \ell} \doteq \cup_{k \le \ell} A_k$ *and* A_k *is given by* (17).

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

Both machines run in time $\text{poly}(\log(W), n)$ *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 5 via the connection between Reachability and Circuit Certification developed in Section 3.4. The "moreover" part of Lemma 5 is what allows us to eliminate the need for a stack in this setting. ∎

As mentioned in Section 3.4, Lemma 6 is sufficient for deriving the nonambiguous simulations of NL given by Theorem 2, for which we used Lemma 1 in Section 2.2. Lemma 1 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 3 – the same type of similarity as exists between the weight assignment generator constructions for Reachability and for Circuit Certification. Although Lemma 1 is not stated verbatim in [RA00], the proof is implicit in that paper, and we do not repeat it here.

In fact, the constructions in [RA00] 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 1) and the step-wise selection of the seed σ in our unambiguous simulations (Theorem 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 $(G, A_{\leq \ell})$.

B. Proofs of Folklore Propositions

In this appendix we include some proofs and proof sketches of folklore propositions that we use.

Proposition 1. Reachability *on layered digraphs is hard for* NL *under logspace mapping reductions that preserve the number of solutions.*

Proof (sketch). The standard NL-hardness argument for Reachability naturally yields layered instances, namely, computation tableaus of nondeterministic logspace machines on a given input, and has the property that it preserves the number of solutions.

Alternately, there exists a simple logspace mapping reduction that transforms an instance of Reachability on an arbitrary digraph of depth d with n vertices, into an equivalent instance of the same depth d and with at most dn vertices, namely by adding a copy of each original vertex in layers further from the leaves. The number of paths in both instances is the same. ∎

Proposition 2. Circuit Certification *on shallow layered alternating semi-unbounded Boolean circuits is hard for* LogCFL *under logspace mapping reductions that preserve the number of solutions.*

Proof. Since Circuit Certification on shallow semi-unbounded Boolean circuits is complete for LogCFL under logspace mapping reductions that preserve the number of solutions, it suffices to establish the following claim.

Claim 6. *There exists a logspace computable transformation that takes a Boolean circuit C with negations on the inputs only, and transforms it into an equivalent circuit C' that is layered and alternating. On any input z, the number of certificates for (C, z) and (C', z) is the same. If C is semi-unbounded, then so is C'. If C has n gates, m wires, and depth d, then C' has $O(d(n+m))$ gates and depth $O(d)$.*

The transformation consists of two steps.

1. Making the circuit alternating

For each original wire (u, v) where u and v have the same type, apply the following operation: Introduce a fresh gate g of the complementary type of v (AND vs OR), and replace the wire (u, v) by the wires (u, g) and (g, v). This step increases the number of gates as well as the number of wires by one per application, and increases the depth to at most $2d - 1$.

2. Making the circuit layered

   Apply the operation from the previous step for each wire (u, v) where u is a leaf and v is an OR gate. At this point, the circuit has at most $n + m$ gates and depth at most $d' = 2d$.

   Set the layer of each gate v to the depth of the subcircuit rooted at v. Layer 0 consists of all leaves. By the prior operations from this step, layer 1 consists of AND gates, and by the previous step, layers 1 and higher alternate between AND and OR. So, odd layers consist of AND's, and positive even layers of ORs.

   Wires can still cross multiple layers. In order to remedy this, for each gate u we introduce an equivalent gate g_i at each layer i between the layer k of u and the highest layer to which u has a wire. The type of the gate g_i is determined by its layer (AND for odd layers, and OR for even layers), and the only input to g_i is g_{i-1}, where $g_k \doteq u$. Once done, we can replace each wire (u, v) for a gate v at layer ℓ by the wire $(g_{\ell-1}, v)$. This step multiplies the number of gates by at most $d' = 2d$ but does not affect the depth.

The resulting circuit C' has at most $d'(n + m) = 2d(n + m)$ gates and depth at most $d' = 2d$. By construction, C' is equivalent to C, has the same number of certificates as C on any given input z, is layered, and alternating. The transformation preserves semi-unboundedness as the fan-in of no gate increases, and the new gates created all have fan-in one. The transformation can be computed in logspace in a standard way. This argues Claim 6 and finishes the proof of Proposition 2. ■

Proposition 5. 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.*

Proof (sketch). In the case of Reachability, $(G_1, s_1, t_1) \sqcup (G_2, s_2, t_2) = (G, s, t)$, where G consists of the disjoint union of G_1 and G_2, new vertices s and t, and new edges from s to each of s_1 and s_2, and from each of t_1 and t_2 to t. The predicate π underlying Reachability takes an instance $x \doteq (G, s, t)$ and a candidate path P, and checks whether P correctly leads from s to t in G. This can be done in logspace on input x and $y \doteq P$.

In the case of Circuit Certification, $(C_1, z_1, g_1) \sqcup (C_2, z_2, g_2) = (C, z, g)$, where C consists of the disjoint union of C_1 and C_2, a new OR gate g with the gates g_1 and g_2 feeding into it, and $z = (z_1, z_2)$. The predicate π underlying Circuit Certification takes an instance $x \doteq (C, z, g)$ and a certificate F, and checks whether $F(z) = 1$. Every OR gate in F has a single AND gate or input literal feeding into it. Such a circuit F of depth d can be transformed in space $O(d + \log n)$ into an equivalent formula $\tilde{F}$ of depth at most d in which all non-leaf gates are binary ANDs. Checking whether $\tilde{F}(x) = 1$ can be done by keeping track of an edge (g, v) maintaining the invariant that v feeds into g and that all gates u feeding into g that are lexicographically less than v evaluate to 1 on input z. The space needed is for this is $O(\log |\tilde{F}|)$, which is $O(d + \log n)$. As $d \leq \log(n)$ for shallow circuits, the overall space used is $O(\log n)$ when given x and $y \doteq F$ as inputs.

Both constructions can be made to work in the restricted settings using standard tricks. ■