Electronic Colloquium on Computational Complexity, Report No. 155 (2016)

Isolation Lemma for Directed Reachability and NL vs. L

Vaibhav Krishan* Nutan Limaye†

October 10, 2016

Abstract

In this work we study the problem of efficiently isolating witnesses for the complexity classes NL and LogCFL, which are two well-studied complexity classes contained in P. We prove that if there is a L/poly randomized procedure with success probability at least 2/3 for *isolating* an s-t path in a given directed graph with a source sink pair (s, t) then NL is contained in L/poly. By *isolating a path* we mean outputting a new graph on the same set of nodes such that exactly one s-t path from the original graph survives. Such an isolating procedure will naturally imply a UL/poly algorithm for reachability, but we prove that in fact this implies an L/poly algorithm.

We also prove a similar result for the class LogCFL.

1 Introduction

The notion of nondeterminism and that of *a witness* are very closely related. For example, the class NP, which is one of the most well-studied nondeterministic complexity classes, can be defined both in terms of nondeterministic computations and in terms of the notion of a witness. A witness for the satisfiability of a Boolean formula is simply an assignment to its Boolean variables. Similarly, for a directed acyclic graph, a witness for reachability between two vertices is a directed path between them. *Isolation* is a procedure which given a positive instance to a computation problem, with possibly many witnesses, produces another instance of the problem with a unique witness. A randomized algorithm, which given a positive instance of a problem produces a positive instance with a unique witness with some probability, say p, and given a negative instance never produces a positive instance is termed as *a randomized isolation procedure*.

The study of isolation procedures was spurred by the Isolation Lemma, introduced by Valiant and Vazirani [14]. They used the Isolation Lemma to give a randomized reduction from satisfiability to unique-satisfiability[1]. This gave an evidence that the complexity of NP-hard problems is not in the number of witnesses, at least in the randomized setting. Subsequently it was reformulated by Mulmuley, Vazirani, and Vazirani [10] and used to give the first randomized parallel algorithm for the perfect matching problem. Since then the lemma has found a wide array of applications ranging over algorithmic and complexity theoretic problems.

The lemma and its versions are of fundamental importance mainly because they improve our understanding of the solution sets of many important computational problems such as the satisfiability problem, the perfect matching problem, and the graph reachability problem. For instance,

*Independent Researcher vaibhkrishan@gmail.com

†Indian Institute of Technology, Bombay, nutan@cse.iitb.ac.in

[1] satisfiability instances with a unique accepting assignment.

ISSN 1433-8092

the Valiant-Vazirani isolation lemma states that there is a randomized polynomial time algorithm which given any propositional formula ϕ produces another formula ψ such that ψ has no satisfying assignment if the given formula ϕ is not satisfiable, whereas with probability $\Omega\left(\frac{1}{n}\right)$ it has exactly one satisfying assignment if ϕ is satisfiable, where n is the number of variables in the formula ϕ.

The question of derandomizing the isolation lemma and its versions has also received a lot of attention (see for instance [4, 2, 6]). It is known that successful derandomization of the isolation lemma, among many other things, will imply an efficient deterministic parallel algorithm for the perfect matching problem. Recently, in a surprising and elegant result, an almost efficient derandomization of the Mulmuley-Vazirani-Vazirani isolation lemma was given by Fenner et al. [6]. Aligned with the study of derandomization of the isolation lemma, Dell et al. [5] considered the question of improving the success probability of the isolation lemma. In particular, they showed that improving the success probability of the randomized isolation procedure beyond $\frac{2}{3}$ is equivalent to a complexity collapse, namely $\mathsf{NP} \subseteq \mathsf{P/poly}$.

In this work we raise complexity theoretic questions about isolating a unique witness for some problems in P. A very interesting variant of the isolation lemma was first given by Gál and Wigderson [7] in the context of complexity classes contained in P, namely NL and LogCFL. Allender and Reinhardt [12] using a clever combination of ideas of [10, 7] and the notion of *double inductive counting* proved unconditional complexity containments such as $\mathsf{NL} \subseteq \mathsf{UL/poly}$ and $\mathsf{LogCFL} \subseteq \mathsf{UAuxPDA/poly}$. In the process they gave a version of the isolation lemma from [10] which they termed as *min-unique* isolation[2].

1.1 Isolation for NL

We consider three different types of witnesses for the directed reachability in graphs, Reach. We call them *reachability witness, lex-min witness*, and *minimal length witness*. For a DAG G with two designated vertices 0 and 1, a *reachability witness* is merely a directed path from 0 to 1. For a labeled DAG (vertices are labeled with elements from some alphabet) a *lex-min witness* is the lexicographically smallest reachability witness[3]. A *minimal length witness* is one of the possibly many reachability witnesses which has the minimum length.

A witness isolation algorithm for Reach takes as input a DAG G and two designated vertices 0 and 1 and outputs another DAG G' on the *same set of vertices* with certain properties. The properties the graph G' satisfies, depend on the type of the isolation algorithm. Here we consider three types of isolation algorithms: A *randomized witness isolation* algorithm with success probability p ensures that

- [p-complete] if G has a valid witness then G' has a unique valid witness with probability $\geq p$,
- [sound] and every valid witness of G' is also a valid witness for G (with probability 1).

A *deterministic witness isolation* algorithm is the one where no randomness is used. A *satisfiability preserving isolation* algorithm is a randomized algorithm which ensures that G' has a valid witness if and only if G does.

An isolation algorithm can be either randomized or deterministic. A randomized isolation algorithm can further be satisfiability preserving or not. The witness thus isolated by an isolation

[2] min-unique isolation guarantees that the minimum weight path between any pair of vertices is unique w.r.t. some weight function on the graph edges.

[3] For example, a path labeled as 0-2-n-1 is lexicographically smaller than a path labeled 0-n-1.

algorithm can be a reachability witness, a lex-min witness, or a minimal length witness. We study these three types of isolation algorithms and their performance with respect to the three different types of witnesses defined above.

For the reachability problem our two main theorems are as follows:

Theorem 4. *The following are equivalent to* $\mathsf{NL} = \mathsf{L}$.

1. *There exists a deterministic logspace algorithm isolating a lex-min witness.*
2. *There exists a deterministic logspace algorithm isolating a minimal length witness.*

Theorem 8. *The following are equivalent to* $\mathsf{NL} \subseteq \mathsf{L/poly}$*: there is a randomized* $\mathsf{L/poly}$ *algorithm for the reachability problem on directed graphs on* n *vertices that isolates*

1. *minimal length witnesses with success probability* $p > \frac{2}{3} + \frac{1}{poly(n)}$.
2. *lex-min witnesses with success probability* $p > \frac{2}{3} + \frac{1}{poly(n)}$.
3. *reachability witnesses with success probability* $p > \frac{2}{3} + \frac{1}{poly(n)}$.
4. *reachability witnesses with success probability* $p > \frac{1}{poly(n)}$ *and moreover, the algorithm is also satisfiability preserving.*

The theorem above states that improving the success probability of the isolation algorithm to strictly greater than $\frac{2}{3}$ will imply a containment, namely NL is contained in $\mathsf{L/poly}$. Theorem 8 is one of our main contributions. The proof technique combines the idea of [5] with that of [12]. Our proof non-trivially uses the $\mathsf{UL/poly}$ algorithm developed in [12].

1.2 Isolation for LogCFL

Here too we start by defining three notions of witnesses (similar to lex-min and min-weight) and prove results similar to Theorem 4 and Theorem 8, but now in the context of a LogCFL complete problem instead of an NL complete problem. Here too, we non-trivially use the non-uniform $\mathsf{UAuxPDA}$ algorithm from [12]. The results appear in Section 4.

1.3 Other constributions

While proving these results we also give L and $\mathsf{LogDCFL}$ variants of two crucial lemmas, namely Ko's Lemma and Adleman's Lemma, which were stated and used in [5]. Though the proofs of these variants are not very difficult, we believe that the statements may be of independent interest. These variants along with some preliminaries appear in Section 2.

We also state some results regarding isolation for CNF formulas.

2 Preliminaries and Definitions

In this section we will introduce basic terminologies used in the rest of the paper and some useful lemmas. We assume familiarity with basic complexity classes such as $\mathsf{L}, \mathsf{NL}, \mathsf{P}, \mathsf{NP}$, boolean circuits, CNF formulas and uniformity for circuits. (See for instance [3],[13, 15].)

2.1 Directed reachability

Reachability in directed acyclic graphs (DAGs) is a well known NL-complete problem, denoted by Reach. W.l.o.g., we rename designated source and target vertices as 0 and 1 respectively. A DAG is said to be *satisfiable* if it has a valid witness, *unsatisfiable* if it has no valid witness and *uniquely satisfiable* if it has a unique valid witness. A witness for satisfiability is simply a 0-1 path.

We consider a promise version of Reach which we call prUReach which is a restricted version of Reach wherein there is a promise on the input instances that each input graph has either a unique valid witness or none at all.

We also consider a distributional version of the randomized isolation algorithm. A *p-distributional isolation algorithm* for the reachability problem is a deterministic algorithm that on input G of size n outputs a list $\langle G_1, \ldots, G_{f(n)} \rangle$, ($f(\cdot)$ polynomially bounded) on the same set of vertices as G such that

- [p-complete] if G is satisfiable then at least p-fraction of G_is are uniquely satisfiable,
- [sound] and every valid witness of any G_i is also a valid witness of G.

2.2 LogCFL or SAC^1 circuits

Recall that a Boolean semi-unbounded circuit is a circuit in which every AND gate has $O(1)$ fan-in and every OR gate has unbounded fan-in (i.e. as much as the size of the circuit). The class SAC^1 is a class of semi-unbounded circuits of polynomial size and logarithmic depth. The complexity class LogCFL and SAC^1 are known to be equal (see for instance [15]). LogCFL can also be characterized as $\mathsf{AuxPDA}(log(n), n^{O(1)})$, which is the set of languages accepted by auxiliary PDAs with log amount of auxiliary workspace and polynomial runtime, from Sudborough [13]. An auxiliary PDA is a PDA with an auxiliary read/write work-tape, like a read/write tape in a Turing machine. $LogDCFL$ and $DAuxPDA$ are deterministic versions of $LogCFL$ and $AuxPDA$ respectively.

Let SAC^1-EVAL be a problem in which given a logspace-uniform semi-unbounded circuit C of log depth and an assignment a for its input variables, one needs to check whether the circuit evaluates to 1 on the assignment a or not. As all negations can be pushed to the top, we assume without loss of generality that all the variables and their negations are provided as inputs, eliminating the need for NOT gates.

Proposition 1. SAC^1-EVAL *is complete for* LogCFL *under logspace reductions. This follows directly from the characterization of* LogCFL *as* SAC^1 *circuits by Venkateswaran [15].*

Here, the notion of a witness is a *proof tree*. Given a circuit C, create a tree C' (of possibly exponential size) from C by duplicating gates with a fan-out of more than 1 in a bottom up fashion. A proof tree T is a subtree of C' such that: T must contain the output gate of C. For every AND gate in T, all its inputs must be in T. For every OR gate in T, exactly one of its inputs must be in T. Starting from the output gate, exactly the nodes added in this way in a bottom up fashion can be in T. It is easy to see that the circuit C evaluates to 1 on an input if and only if there is a proof tree of C' with leaves labelled by 1s. It is known that SAC^1 circuits have polynomial sized proof trees [15].

Here too we consider three different types of witnesses which we call *evaluation witness, lex-min witness, minimal weight witness*. Given an SAC^1 circuit C and an assignment a, an *evaluation witness* is merely a proof tree of the circuit which certifies that C evaluates to true on a. For

a labeled circuit (wires are labeled with elements from some alphabet) a *lex-min witness* is the lexicographically smallest evaluation witness. For a weighted circuit (wires are weighted) a *minimal weight witness* is one of the possible many evaluation witnesses which has the minimum weight. Here the weight of the witness is the sum of the weights of the wires in the witness.

The promise problem version of SAC^1-EVAL we consider is prUSAC^1-EVAL, evaluating a logspace-uniform log depth semi-unbounded circuit on an input with the promise that it will either accept with a unique proof tree or reject[4].

Finally, we define randomized, deterministic, p-distributional isolation as well as satisfiability preserving isolation procedures for the SAC^1-EVAL problem as we did for Reach. Except that now the witnesses are proof trees of the SAC^1 circuit rather than paths in DAGs, and the isolation procedure should output circuits on the *same set of gates* as the input circuit.

2.3 Isolation for CNF formulas

A witness for satisfiability of a CNF formula is simply a satisfying assignment, which we call a *satisfiability witness*. A *lex-min witness* is simply the lexicographically smallest (on binary alphabet) satisfying assignment. The promise problem version of SAT, namely prUSAT deals with checking the satisfiability of the given formulas under the promise that formulas either have unique satisfying assignments or none.

Different types of isolation procedures (deterministic, randomized, p-distributional, satisfiability preserving) can be defined for the CNF satisfiability problem as we did for Reach.

2.4 Useful Lemmas

We start by proving that the existence of a randomized isolation algorithm with success probability p implies the existence of a p'-distributional isolation algorithm, p' being slightly smaller than p. Here the emphasis is on obtaining complexity efficient algorithm for the latter assuming a similar algorithm for the prior. Adleman (see [1]) proved such a lemma for algorithms working in P/poly. We modify their original lemma to obtain a similar statement for algorithms in the class L/poly and LogDCFL/poly. Formally, we prove the following:

Lemma 2 (Adleman's)**.** *If there is a randomized* L/poly *(or* LogDCFL/poly*) algorithm A, which maps input x to output y, and two probability functions, $p_1, p_2 : \mathbb{N} \to [0,1]$ such that two properties $P_1(x,y), P_2(x,y)$ hold with probability at least $p_1(n), p_2(n)$ respectively for inputs of size n, then for every $c > 0$ there is a deterministic* L/poly *(*LogDCFL/poly*) algorithm B, which on input x produces a list, $y_1, y_2, \ldots, y_t$, where $t = poly(n)$ and such that (i) $P_1(x, y_i)$ holds for at least $p'_1(n)t$ many $i \in [t]$ (ii) $P_2(x, y_i)$ holds for at least $p'_2(n)t$ many $i \in [t]$, where $p'_j(n) = 1$ whenever $p_j(n) = 1$, otherwise $p'_j(n) = p_j(n) - \frac{1}{cn^c}$.*

Proof. Let us say we run A, t number of times, where t for now is a parameter which we will show can be $poly(n)$. Each time we run A with fresh randomness and output the list of outputs produced. If $p_1 = 1 (p_2 = 1)$ clearly all of them satisfy $P_1 (P_2)$. For $j \in \{1, 2\}$ and $p_j < 1$, the expected number of runs satisfying P_j is $p_j t$.

We can set $p'_j = p_j - \epsilon$ with $\epsilon = \frac{1}{cn^c}$. For $t = O(\frac{n}{\epsilon^2})$, by applying Hoeffding's bound [8], the probability that less than $p'_j t$ runs of A satisfy P_j is smaller than 2^{-n-1}. By union bound, the

[4]Hereafter, when we say semi-unbounded circuits, we implicitly mean semi-unbounded circuits that are logspace-uniform and of log depth.

probability that fewer than $p_1't$ satisfy P_1 or fewer than $p_2't$ satisfy P_2 is smaller than 2^{-n}. Thus there must be some random string for A which works for every input of length n. We give this string and t as the advice. □

The notion of a p-selector was first defined by Ko [9]. They proved that the existence of a p-selector for a promise problem[5] implies a non-uniform polynomial time algorithm for the promise problem. In the following lemma we observe that Ko's lemma can be proved in L and LogDCFL settings.

Lemma 3 (Ko's)**.** *Given a promise problem $\Pi = (Yes, No)$ and a binary relation R over $Yes \cup No$ such that:*

(K1) $(x \in Yes \ \& \ R(x,y)) \Rightarrow y \in Yes$.

(K2) $(x, y \in Yes \ \& \ |x| = |y|) \Rightarrow (R(x,y) \text{ or } R(y,x))$.

Let $Yes_n = Yes \cap \{0,1\}^n$ and $No_n = No \cap \{0,1\}^n$. We can output for each $n \in \mathbb{N}$ a list of $x_i \in Yes_n$, $x_1, x_2, \ldots, x_{n+1}$ such that for every $y \in Yes_n$, there is some i such that $R(x_i, y)$ holds, and for every $y \in No_n$, there is no i such that $R(x_i, y)$ holds.

Thus, if there is a L algorithm which for each $n \in \mathbb{N}$ and each $x \in Yes_n$, takes a polynomial amount of advice and decides on input $y \in (Yes_n \cup No_n)$, whether $R(x,y)$ holds or not, then there is an algorithm for Π in L/poly.

If machine for $R(x,y)$ is a DAuxPDA$(log(n), n^{O(1)})$ with polynomial advice, then there is an algorithm for Π in LogDCFL/poly.

Proof. Fix an $n \in \mathbb{N}$. Say we have a list $x_1, x_2, \ldots, x_j$ for some $j \geq 0$. Let's define $S_j = \{y \in Yes_n \mid R(x_i, y)$ does not hold for any $i \in \{1, 2, \ldots, j\}\}$. Clearly $S_0 = Yes_n$. Also, if $S_j = \phi$, we are done. Otherwise we choose x_{j+1} as follows. Property (K2) implies that for all $x, y \in S_j$, we have $R(x,y)$ or $R(y,x)$. Thus, the average out-degree of the directed graph induced by R is at least $\frac{|S_j|}{2}$. This implies there exists a $y \in S_j$ with out-degree at least $\frac{|S_j|}{2}$. By choosing this y as the x_{j+1} we can ensure that $|S_{j+1}| \leq \frac{|S_j|}{2} \leq \frac{|S_0|}{2^{j+1}}$. Since $|S_0| \leq 2^n$, $S_{n+1} = \phi$.

Let $y \in No_n$. Property (K1) ensures that $R(x_i, y)$ cannot hold for any i as for each i, $x_i \in Yes$. If $y \in Yes_n$, our choice of x_is ensures an i such that $R(x_i, y)$ holds.

The complete advice comprises of the advice required by each x_i. It is now easy to see that this advice suffices and thus if the algorithm for $R(x,y)$ is a logspace machine or a deterministic auxiliary PDA with log workspace and polynomial time, then there is an algorithm for Π in L/poly or LogDCFL/poly respectively. □

3 Directed acyclic graphs

3.1 Uniform isolation for DAGs

We argue that isolating lex-min witnesses and minimal length witnesses for DAGs are very strong notions. In particular, the existence of a deterministic logspace procedure isolating the lex-min witness or the minimal length witness is equivalent to NL = L.

[5] A promise problem Π is a problem in which the inputs are promised to come from a restricted subset $Yes \cup No$, Yes and No being disjoint. An algorithm must decide which among the two sets does a given input belong to. The behavior of the algorithm on inputs outside this subset is ignored.

Theorem 4. *The following are equivalent:*

1. *There exists a deterministic isolation for lex-min witnesses in* L.
2. *There exists a deterministic isolation for minimal length witnesses in* L.
3. $\mathsf{NL} = \mathsf{L}$.

Proof. 1 $\implies$ 3. We need to solve s-t reachability for DAGs in L.

Consider any graph G on n vertices. We first check for s-t edge. If yes, trivial to output *yes*. Otherwise we label them arbitrarily from 0 to $n-1$ except for s as 0 and t as 1. We add a *new* vertex with label n such that it has an edge from 0 and an edge to 1, and no other edges. Let the added path be called w_{new}. We call this modified graph G'. Let the procedure achieving this be P_1.

Let the logspace lex-min witness isolation procedure be P_2. We run P_2 on G' to get G''. We reject G if G'' has both 0-n and 1-n edges and accept otherwise.

Clearly, the overall procedure is in logspace as both P_1 and P_2 are in logspace. For correctness, we simply observe that the path w_{new} has label 0-n-1 which is the lexicographically largest label among all possible paths in G'. If G has no s-t path, w_{new} has to be kept in G'' (as it is the only one). If G does have an s-t path, then w_{new} cannot be in G'' as it necessarily larger than any s-t path in G when ordered lexicographically. This completes the proof.

2 $\implies$ 3. To prove this implication we show how to solve s-t reachability for DAGs in L. Consider any graph G on n vertices. We first check whether there exists an s-t edge. If yes, trivial to output *yes*. If not, we construct a DAG G' which is same as G except we add an n length path from s to t. All the vertices used in the new path are new i.e. distinct from vertices of G. Let the newly added path be w_{new} and the modified graph be G'. Let the procedure achieving this be P_1.

Let the minimal length witness isolation be P_2. We run P_2 on G' to get a G''. We accept G if and only if G'' does not have the new path w_{new} from s to t, using the new vertices.

Clearly, the overall procedure is in logspace as both P_1 and P_2 are in logspace. For correctness, we need to observe that any path from s to t in G can be of length at most $n-1$. Also, from the definition of the isolation algorithm, any path from G'' must also be in G'. Therefore, G'' cannot have any new paths, which did not appear in G'. Thus, if G had a path from s to t, then the minimum witness isolation will have to remove w_{new} from G''. If G had no path from s to t, then the only s-t path in G' is w_{new} thus it has to be kept in G''. Thus, G'' can have w_{new} if and only if G did not have any s-t path. This completes the proof.

The proofs of 3 $\implies$ 1, 3 $\implies$ 2 are straightforward. For 3 $\implies$ 1, guess a path and check if it is the smallest lexicographically. For 3 $\implies$ 2, an NL algorithm starts with $i = 1$. It tries to guess a path of length i. If it succeeds, it outputs the path itself as the isolation. If it fails, it increments i till it reaches n. If $\mathsf{NL} = \mathsf{L}$, we can do the above in L itself. □

3.2 Non-uniform isolation for DAGs

We show that the existence of an $\mathsf{L/poly}$ algorithm for isolating a witness (any of the three types of witnesses defined earlier) for Reach is equivalent to $\mathsf{NL} \subseteq \mathsf{L/poly}$.

First we prove that there exists a p-distributional isolation algorithm for Reach, with $p = 1/poly(n)$. We also show an $\mathsf{L/poly}$ reduction from Reach to $\mathsf{prUReach}$. We then use Ko's lemma (Lemma 3) to prove conditionally that $\mathsf{prUReach} \in \mathsf{L/poly}$ and thereby prove the main result

(Theorem 8). We start by stating the following lemma which is at the core of the rest of the argument.

Lemma 5. *There is a non-deterministic logspace algorithm for* Reach *with a property that if the input graph G is* min-unique*, that is, the smallest length path between any pair of connected vertices in the graph is unique, then the algorithm accepts on a single path.*

The algorithm required to prove the above lemma is a small modification of the UL algorithm for a restricted version of Reach as presented in [12]. In the original algorithm in [12], they add additional checks for whether the input graph is min-unique or not. We remove this check. The full algorithm and its correctness are presented in Section A.1.

Lemma 6 (Isolation for DAGs)**.** *There is a logspace-computable function f and a sequence of advice strings $\{\alpha(n)|n \in \mathbb{N}\}$ (where $|\alpha(n)|$ is bounded by a polynomial in n) such that for any DAG on n vertices:*

- $f(G, \alpha(n)) = \langle G_1, G_2, \ldots, G_{n^2} \rangle$.
- *For each i, the DAG G_i is satisfiable if and only if G is satisfiable.*
- *If G was satisfiable, then for some i, G_i is uniquely satisfiable.*

Proof. As argued by Reinhardt and Allender [12], given G, we have a procedure to output a list $\langle G_1, G_2, \ldots, G_{n^2} \rangle$, such that at least one of them is min-unique. Additionally, we note that each of these graphs is satisfiable if and only if G is too. It is possible that these graphs may not have the same number of vertices as G, but we can remedy that by adding *dummy* vertices which are not connected to any other. Let this procedure be called P_1.

Consider the configuration graph of Algorithm 1 which was designed to prove Lemma 5 (and which appears in the Appendix) on n length input obtained using the logspace parsimonious reduction from an NL algorithm to a DAG (see [3]). Also rename *start* as 0 and *accept* as 1. Let us call the graph obtained as above when G_i is input to the algorithm as C_{G_i}. We output $\langle C_{G_1}, C_{G_2}, \ldots, C_{G_{n^2}} \rangle$. Let this procedure be P_2.

Both P_1 and P_2 are clearly in L/poly. The construction ensures that each C_{G_i} is satisfiable if and only if G_i is satisfiable if and only if G is satisfiable. Finally, if G is satisfiable, one of the G_i has to be min-unique, implying that Algorithm 1 accepts on a unique path, thus ensuring C_{G_i} is uniquely satisfiable, due to parsimony of reduction. This completes the proof. □

The isolation procedure obtained from the preceding lemma can be used to prove an L/poly reduction from Reach to prUReach, which is our next lemma.

Lemma 7. prUReach $\in$ L/poly *if and only if* NL $\subseteq$ L/poly.

Proof. Backward implication is fairly straightforward, as any algorithm solving Reach in L/poly, also solves prUReach in L/poly.

For the other direction, assume M is an L/poly-algorithm solving prUReach. Let the procedure from Lemma 6 be B. On input a DAG G, B outputs a list of graphs $\langle G_1, G_2, \ldots, G_{n^2} \rangle$ such that if G is satisfiable, at least one of the G_i is uniquely satisfiable and if G is unsatisfiable, then each G_i is unsatisfiable. If M accepts at least one G_i, accept; otherwise reject.

The algorithm described is clearly in L/poly. For correctness, if G is unsatisfiable, then each G_i is unsatisfiable and must be rejected by M. If on the other hand, G is satisfiable, at least one of the G_i is uniquely satisfiable and thus must be accepted by M. □

Theorem 8. *The following statements are equivalent: there is a randomized* L/poly *algorithm for the reachability problem on directed graphs on n vertices that isolates*

(i) minimal length witnesses with success probability $p > \frac{2}{3} + \frac{1}{poly(n)}$.

(ii) lex-min witnesses with success probability $p > \frac{2}{3} + \frac{1}{poly(n)}$.

(iii) reachability witnesses with success probability $p > \frac{2}{3} + \frac{1}{poly(n)}$.

(iv) reachability witnesses with success probability $p > \frac{1}{poly(n)}$ and moreover, the algorithm is also satisfiability preserving.

(v) NL $\subseteq$ L/poly.

Proof. (i) $\implies$ (iii), (i) $\implies$ (iv), (ii) $\implies$ (iii), (ii) $\implies$ (iv) are immediate. For (v) $\implies$ (i), (v) $\implies$ (ii), we use techniques similar to that in the proof for Theorem 4.

For (iii) $\implies$ (v), it suffices to prove prUReach $\in$ L/poly as per Lemma 7. Let A be the assumed procedure for isolation. Using Lemma 2 we know that we can come up with a procedure B which on input G, produces $\langle G'_1, G'_2, \ldots, G'_t \rangle$, $t = poly(n)$ such that if G is satisfiable, at least $p' = p - \frac{1}{cn^c}$ fraction of them are uniquely satisfiable. We choose c such that for every n, $p' > \frac{2}{3}$. Also, all satisfying assignments of each G'_i satisfy G.

We first define an operator which mimics the OR operator from circuits for graphs. Given G_1 with $0_1, 1_1$ and G_2 with $0_2, 1_2$, we define $G_1 \vee G_2$ as follows. We create two vertices, named $0_G, 1_G$. We add an edge from 0_G to 0_1 and 0_2. We add an edge from 1_1 and 1_2 to 1_G. G_1 and G_2 keep their structure, not sharing any edges or vertices. This is an OR in the sense that any $0-1$ path in G_1 (or G_2) gives a $0-1$ path in $G_1 \vee G_2$. When we say a path w_i of G_i satisfies $G_1 \vee G_2$ for either $i = 1, 2$, we mean that we take the edge from 0_G to 0_i, follow the path as per w_i and then the edge from 1_i to 1_G thus we get a $0-1$ path in $G_1 \vee G_2$.

Given G_1 and G_2 we define new graph G as follows: $G = \min\{G_1, G_2\} \vee \max\{G_1, G_2\}$, where min and max are defined based on lexicographical ordering on the vertices of these graphs. Let R be a relation such that, given $G_1 \in Yes$ with w_1 as its unique certificate, $G_2 \in (Yes \cup No)$, $R(G_1, G_2)$ holds if and only if w_1 satisfies less than p' fraction of the list $B(G)$. We show that R satisfies the properties required for Lemma 3.

For (K1), as there are at least p fraction of successful isolations and w_1 satisfies less than p' fraction of them, there must be some isolation not admitting w_1 implying it must be an isolation of G_2 thus $G_2 \in Yes$.

For (K2), we assume for contradiction $G_1, G_2 \in Yes$ and neither of $R(G_1, G_2), R(G_2, G_1)$ is true. This of course means both w_1 and w_2 must satisfy at least p' fraction of $B(G)$ and $w_1 \neq w_2$ by design. Easy to see there are at least $2p' - 1$ G'_is which admit both w_1, w_2. $2p' - 1 > \frac{1}{3} > 1 - p'$ contradicting that at least p'-fraction of $B(G)$ are successful isolations.

$R(G_1, G_2)$ can be determined by a logspace machine using G_1, w_1, pt as advice and calling B as a subroutine once. B is in L/poly and checking whether a path satisfies a graph can also be done in logspace. Using Lemma 3 we get prUReach $\in$ L/poly; hence NL $\subseteq$ L/poly.

(iv) $\implies$ (v). Let A be the assumed procedure for isolation. Using Lemma 2 we know that we can come up with a procedure B which on input G, produces $\langle G'_1, G'_2, \ldots, G'_t \rangle$, $t = poly(n)$ such that if G is satisfiable, at least $p' = p - \frac{1}{cn^c}$ fraction of them are uniquely satisfiable. We choose c

such that for every n, $p' > 0$. We modify the relation R as follows. Given G_1, w_1, G_2, $R(G_1, G_2)$ holds if and only if w_1 does not satisfy at least one of G'_i from $B(G)$.

For (K1), if $R(G_1, G_2)$ holds and $G_1 \in Yes$, it means w_1 does not satisfy some G'_i from $B(G)$. As the isolation is satisfiability preserving, it must be the case that the graph not satisfied by w_1 is satisfied by a witness of G_2 thus $G_2 \in Yes$.

For (K2), assume for contradiction $G_1, G_2 \in Yes$ but neither $R(G_1, G_2)$ nor $R(G_2, G_1)$ holds. It means both w_1 and w_2 (again $w_1 \neq w_2$ by design) must satisfy all of $B(G)$ contradicting the fact that one of them is a successful isolation. It is easy to see again that $R(G_1, G_2)$ can be checked in logspace given G_1, w_1 thus leading to $\mathsf{NL} \subseteq \mathsf{L/poly}$. □

4 Semi-unbounded circuits

4.1 Uniform isolation for semi-unbounded circuits

We show that isolating lex-min witnesses or minimal weight witnesses for uniform SAC^1 circuits is a very strong notion. In fact, the existence of $\mathsf{LogDCFL}$ computable isolation algorithm for either lex-min or minimal weight witnesses is equivalent to $\mathsf{LogCFL} = \mathsf{LogDCFL}$. Similarly, the existence of L computable isolation algorithm for either lex-min or minimal weight witnesses is equivalent to $\mathsf{LogCFL} = \mathsf{L}$.

Theorem 9. *Let A be either L or $\mathsf{LogDCFL}$. The following are equivalent: There is a deterministic algorithm for SAC^1-EVAL such that it isolates:*

1. *lex-min witnesses in the complexity class A.*
2. *minimal weight witnesses in the complexity class A*
3. *$\mathsf{LogCFL} = A$.*

Proof. 1 $\implies$ 3. Say we have an SAC^1 circuit C on n variables of size s which we need to evaluate on $x \in \{0,1\}^n$. We label the wires of C from 1 to s^2 arbitrarily and call it C. We design an SAC circuit C' of depth $log(n)$ and size $s' = O(n)$ on n inputs which only accepts x. The circuit we design looks like a tree of AND gates with x as its input. We label the wires of this circuit with labels from $s^2 + 1$ to $s^2 + s'$ and call it C'. We note that C' can be obtained in logspace.

We run the isolation procedure on $C'' = C \vee C'$ to get C'''. We accept C if and only if the proof tree in C''' has no gates from C'. We show that this can be checked in logspace.

To see this we note that C''' is a DAG obtained by taking the union of the DAGs underlying C and C' with possibly a few edges dropped. As C' is a tree, when it appears in C''', it continues to appear as a tree (or a forest). To check whether a gate in C' is connected to the output of the proof tree of C''', say T''', we need to check whether there is a path from that gate to the output gate of T'''. As reachability in a tree can be done in logspace, we get the logspace upper bound.

For correctness, we note that C''' is the circuit we get after running the lex-min witness isolation procedure on $C'' = C \vee C'$. So C''' is such that the proof tree of acceptance of x in C''', i.e. T''' is the lexicographically smallest proof tree from $C'' = C \vee C'$. If C has no proof tree for acceptance for x, T''' will have to be the proof tree from C' (as it is the only one) and if C has a proof tree T for acceptance for x, the proof tree from C' cannot appear in C''' as it is necessarily larger than any proof tree T from C when ordered lexicographically.

As all the steps except for the isolation can be done in L, if the isolation is in LogDCFL, we have proven LogCFL = LogDCFL. If the isolation is in L, we get LogCFL = L.

2 $\implies$ 3. Say we have an SAC^1 circuit C on n variables of size s which we need to evaluate on $x \in \{0,1\}^n$. We assign weight 1 to each wire of C and call it C. We use the semi-unbounded circuit C' of depth $log(n)$ and size $O(n)$ on n inputs which only accepts x from the previous proof. We need to note again that it only uses AND circuits. We assign weight s to each wire of C' and call it C'. This can be done in logspace.

We run the isolation procedure on $C'' = C \vee C'$ to get C'''. We accept C if and only if there is no proof tree in C''' which uses the gates of C'. We can check this just like in the previous case. This can also be done in logspace.

For correctness, we note that any proof tree of C' will have weight at least s while any proof tree of C can only have a total weight of $s-1$. This means that an proof tree of C' can be minimal for C'' if and only if C does not have any proof tree.

As all the steps except for the isolation can be done in L, if the isolation is in LogDCFL, we have proven LogCFL = LogDCFL. If the isolation is in L, we get LogCFL = L.

3 $\implies$ 1, 2. As LogCFL = LogDCFL(L) we can find the isolated witness and thus do the isolation in LogDCFL (respectively L). □

4.2 Non-uniform isolation for semi-unbounded circuits

First we prove that there exists a p-distributional isolation algorithm for SAC^1-EVAL, with $p = 1/poly(n)$. We also show an L/poly reduction from SAC^1-EVAL to prUSAC^1-EVAL. We then use Ko's lemma (Lemma 3) to prove that prUSAC^1-EVAL $\in$ L/poly and thereby proving the main result, similarly for LogDCFL/poly. Firstly, we give the following lemma.

Lemma 10. *There is a non-deterministic $AuxPDA(log(n), n^{O(1)})$ for SAC^1-EVAL such that if the input circuit C is* min-unique[6]*, then the algorithm accepts on a unique path.*

The algorithm thus promised as per the previous lemma is a small modification of the inductive counting algorithm presented in [12] for SAC^1-EVAL. The full algorithm and its correctness are presented in the appendix, Section A.2.

Lemma 11 (Isolation for semi-unbounded circuits)**.** *There is a logspace computable function f and a sequence of advice strings $\{\alpha(n)|n \in \mathbb{N}\}$ (where $|\alpha(n)|$ is bounded by a polynomial in n) such that for any semi-unbounded C on n inputs and size polynomial in n:*

- $f(C, \alpha(n)) = \langle C_1, C_2, \ldots, C_n \rangle$.
- $\forall x \in \{0,1\}^n, i \in [n], C_i(x) = 1 \iff C(x) = 1$.
- $\forall x \in \{0,1\}^n, C(x) = 1 \implies \exists i, \langle C_i, x \rangle \in \mathsf{prUSAC}^1\text{-EVAL}$.

Proof. From [12], given C on n inputs and of size n^l, we have a procedure to output a list of circuits $\langle C_1, C_2, \ldots, C_n \rangle$, such that at least one of them is min-unique for every $x \in \{0,1\}^n$. Additionally,

[6] A circuit C is called min-unique on an input x with respect to a weight function if every gate g that evaluates to 1 on x has a unique minimum weight proof tree. If there are no weights then we assume that every edge has weight 1.

we note that each C_i accepts x if and only if C does, as they do not change the structure of the circuit but simply assign weights to the wires. Let this procedure be P_1.

Now we use the parsimonious reduction as presented by Neidermeier and Rossmanith [11] from $AuxPDA(log(n), n^{O(1)})$ to SAC^1 on algorithm 4 for SAC^1-EVAL. Let us call the circuits we get from this reduction when the input is C_i as C'_i. We output $\langle C'_1, C'_2, \ldots, C'_n \rangle$. Let this procedure be P_2.

P_1 is in logspace taking polynomial advice and P_2 is in logspace. The construction ensures that C'_i accepts x if and only if C_i accepts x if and only if C accepts x. Also, for each $x \in \{0,1\}^n$ one of the C_i is min-unique which means the algorithm runs with unique accepting path which means unique proof tree in C'_i. This completes the proof. □

The isolation procedure obtained from the preceding lemma can be used to prove an $\mathsf{L/poly}$ reduction from SAC^1-EVAL to prUSAC^1-EVAL, which is our next lemma.

Lemma 12. *Let A be either L or $\mathsf{LogDCFL}$, prUSAC^1-EVAL $\in A/poly$ if and only if $\mathsf{LogCFL} \subseteq A/poly$.*

Proof. Backwards implication is fairly straightforward as any algorithm for SAC^1-EVAL also solves the promise version.

For forward implication, let M be an algorithm solving prUSAC^1-EVAL. Let the procedure from Lemma 11 be B. B on input a circuit C on n inputs, outputs a list of circuits $\langle C_1, C_2, \ldots, C_n \rangle$, such that for each input x to C, at least one C_i accepts x on a unique proof tree if C accepts x, otherwise none of them accept x. We accept if and only if M accepts at least one C_i.

B is clearly in $\mathsf{L/poly}$. If M is in $\mathsf{L/poly}$, the whole procedure is in $\mathsf{L/poly}$. If M is in $\mathsf{LogDCFL/poly}$, the whole procedure is in $\mathsf{LogDCFL/poly}$. For correctness, if C does not accept x then none of the C_i accept x and thus M must reject all of them. If C accepts x then at least one C_i accepts x with a unique proof tree thus M must accept it. □

Theorem 13. *Let A be either L or $\mathsf{LogDCFL}$. The following are equivalent: there is a randomized $A/poly$ algorithm for the SAC^1-EVAL problem for circuits on n length inputs such that it isolates*

(i) lex-min witnesses with success probability $p > \frac{2}{3} + \frac{1}{poly(n)}$.

(ii) minimal weight witnesses circuits with success probability $p > \frac{2}{3} + \frac{1}{poly(n)}$.

(iii) evaluation witnesses with success probability $p > \frac{2}{3} + \frac{1}{poly(n)}$.

(iv) evaluation witnesses with success probability $p > \frac{1}{poly(n)}$ and moreover, the algorithm is also satisfiability preserving.

(v) $\mathsf{LogCFL} \subseteq A/poly$.

Proof. $(i) \implies (iii), (i) \implies (iv), (ii) \implies (iii), (ii) \implies (iv)$ are immediate.

For $(v) \implies (i), (v) \implies (ii)$. We use techniques similar to the ones used in proving Theorem 9.

$(iii) \implies (v)$. It suffices to prove that prUSAC^1-EVAL $\in \mathsf{L/poly}$ (or $\mathsf{LogDCFL/poly}$) as per Lemma 12. Let us for the sake of simplicity of argument, fix an $n \in \mathbb{N}$ and an $x \in \{0,1\}^n$. Let the assumed procedure be A. Using Lemma 2 we can come up with a procedure B, which when given as an input a semi-unbounded circuit C on n variables, produces a list $\langle C'_1, C'_2, \ldots, C'_t \rangle, t = poly(n)$

such that each proof tree in any C'_i is in C and if C accepts x, at least $p' = p - \frac{1}{cn^c}$ fraction of them accept x on a unique proof tree. We choose c such that for every n, $p' > \frac{2}{3}$.

Let R be a relation such that, given C_1 with T_1 as its unique proof tree, $C_2 \in (Yes \cup No)$, $R(C_1, C_2)$ holds if and only if T_1 is a valid proof tree for less than p' fraction of $B(C)$. Here, C is constructed by formally taking the OR of $\max\{C_1, C_2\}$ and $\min\{C_1, C_2\}$. By T_1 satisfying $C'_i \in B(C)$, we mean the proof tree induced by T_1 in C satisfies C'_i. We show that R has the properties required for Lemma 3.

For (K1), if $C_1 \in Yes$ and $R(C_1, C_2)$ then T_1 is valid for less than p' of $B(C)$. As T_1 is valid for less than p' fraction of $B(C)$ and at least p' fraction of $B(C)$ are successful isolations, the proof tree of C_2 must be valid for the ones for which T_1 was not valid. Thus $C_2 \in Yes$.

For (K2), let us assume for contradiction $C_1, C_2 \in Yes$ and neither $R(C_1, C_2)$ not $R(C_2, C_1)$. This means that both T_1 and T_2 are valid for at least p' fraction of $B(C)$ and $T_1 \neq T_2$ by design. This means that at least $2p'-1$ of $B(C)$ are satisfied by both. $2p'-1 > \frac{1}{3} > 1-p'$ thus contradicting the fact that at least p' many of $B(C)$ are successful isolations.

Easy to see that given C_1, T_1, pt as advice, one can check for $R(C_1, C_2)$ in logspace. Thus if the assumed isolation is in $\mathsf{L/poly}$, we get $\mathsf{prUSAC^1\text{-}EVAL} \in \mathsf{L/poly}$. If the isolation is in $\mathsf{LogDCFL/poly}$, we get $\mathsf{prUSAC^1\text{-}EVAL} \in \mathsf{LogDCFL/poly}$. This then leads to $\mathsf{LogCFL} \subseteq \mathsf{L/poly}$ and $\mathsf{LogCFL} \subseteq \mathsf{LogDCFL/poly}$ respectively.

(iv) $\implies$ *(v)*. Let the assumed procedure be A. Using Lemma 2 we can come up with a procedure B, which when given as an input a semi-unbounded circuit C on n variables, produces a list $\langle C'_1, C'_2, \ldots, C'_t \rangle, t = poly(n)$ such that if C accepts x, at least $p' = p - \frac{1}{cn^c}$ fraction of C'_i accept x on a unique proof tree. We choose c such that for every l, $p' > 0$. We modify the relation R as follows. Given $C_1 \in Yes$, T_1 as its unique proof tree and $C_2 \in Yes \cup No$, we say $R(C_1, C_2)$ holds if and only if T_1 is not a valid proof tree for at least one of $B(C)$.

For (K1), if $C_1 \in Yes$ and $R(C_1, C_2)$ then T_1 is not valid for at least one of $B(C)$. As the isolation is satisfiability preserving, the circuit for which T_1 is not valid must have the proof tree of C_2. Thus, $C_2 \in Yes$.

For (K2), let us assume for contradiction $C_1, C_2 \in Yes$ and neither $R(C_1, C_2)$ not $R(C_2, C_1)$. This means both T_1 and T_2 are valid for all of $B(C)$ (again $T_1 \neq T_2$ by design). This contradicts the fact that at least one of them is a successful isolation.

Easy to see that given C_1, T_1 as advice, one can check for $R(C_1, C_2)$ in logspace. Thus if the assumed isolation is in $\mathsf{L/poly}$, we get $\mathsf{prUSAC^1\text{-}EVAL} \in \mathsf{L/poly}$. If the assumed isolation is in $\mathsf{LogDCFL/poly}$, we get $\mathsf{prUSAC^1\text{-}EVAL} \in \mathsf{LogDCFL/poly}$. This then leads to $\mathsf{LogCFL} \subseteq \mathsf{L/poly}$ and $\mathsf{LogCFL} \subseteq \mathsf{LogDCFL/poly}$ respectively. □

5 CNF formulas

We state some results regarding isolation for CNF formulas in logspace.

Theorem 14. *There exists a deterministic* L *algorithm for isolating the lex-min witness for CNF formulas if and only if* $\mathsf{NP} = \mathsf{L}$.

Proof. ($\implies$). Given CNF $C(x_1, x_2, \ldots, x_n)$, we construct a formula with an extra variable $D(x_0, x_1, \ldots, x_n) = (\neg x_0 \vee x_1) \wedge (\neg x_0 \vee x_2) \wedge \ldots \wedge (\neg x_0 \vee x_n) \wedge (x_0 \vee C)$. Note that D currently is not a CNF formula due to the last term. For that, we use the fact that C was already in CNF and thus distributing x_0 over the clauses of C will be easy. Say $C = C_1 \wedge C_2 \wedge \ldots \wedge C_m$, each C_i being

a clause. We simply note that, $(x_0 \vee C) = (x_0 \vee C_1) \wedge (x_0 \vee C_2) \wedge \ldots \wedge (x_0 \vee C_m)$. Let P_1 be the procedure that converts C to D.

Now D is such that solutions for $D = 1^{n+1} \cup 0S$, S being the solution set for C. We give D as input to the minimum witness isolation procedure, which outputs another CNF formula D'. We accept C if and only if D' rejects 1^{n+1}. Clearly, 1^{n+1} can be the lexicographically smallest satisfying assignment for D if and only if $S = \phi$. Let P_2 be the procedure that given a CNF formula C and an assignment x, checks whether x satisfies C or not. It is easy to see that both P_1, P_2 are in L which gives us an L algorithm for SAT.

$(\Longleftarrow)$. Just note that $\mathsf{NP} = \mathsf{L} \implies \mathsf{PH} = \mathsf{L}$. We can guess a witness and verify that it is the smallest in Σ_p^2, thus also in L. □

Lemma 15 (Variation of Valiant-Vazirani Isolation)**.** *Given a CNF formula C, there is a randomized logspace algorithm which on input C produces a formula C' such that:*

- *If C is unsatisfiable, C' is unsatisfiable.*
- *If C is satisfiable, C' is uniquely satisfiable with probability at least $\frac{1}{n}$.*

Proof. The procedure presented in [14] is as follows. Let C be a CNF formulas in $x = (x_1, x_2, \ldots, x_n)$. Choose k randomly from 1 to n, randomly choose $w_1, w_2, \ldots, w_k \in \{0,1\}^n$ and output $C' = C \wedge (\langle x, w_1 \rangle = 0) \wedge (\langle x, w_2 \rangle = 0) \wedge \ldots \wedge (\langle x, w_k \rangle = 0)$ where $\langle x, w \rangle$ represents the standard inner product of x and w in the vector space $GF[2]^n$. The proof of probability and correctness remains the same. The non-trivial part is to see this can be achieved in logspace.

First, as was argued by Valiant and Vazirani themselves, $\langle x, w \rangle = 0$ can also be written as $(x_{i_1} \oplus x_{i_2} \oplus \ldots \oplus x_{i_j} \oplus 1)$ where $\oplus$ is the exclusive-or and $i_1, i_2, \ldots, i_j$ are the indices of x that have value 1 in w. The CNF form is

$$(y_1 \Leftrightarrow x_{i_1} \oplus x_{i_2}) \wedge (y_2 \Leftrightarrow y_1 \oplus x_{i_3}) \wedge \ldots \wedge (y_{j-1} \Leftrightarrow y_{j-2} \oplus x_{i_j}) \wedge (y_{j-1} \oplus 1).$$

Now we note that a particular bit of w is required only once, which means we can generate the CNF formula for $\langle x, w \rangle = 0$ without the need to store w. Also, formula for each $\langle x, w_i \rangle = 0$ is independent of other w_js. This means each w_i can be randomly generated and the formula C' output without the need to store all the bits of current w_i or any previous w_j. This completes the proof that the procedure is achievable in logspace. □

Lemma 16. $\mathsf{prUSAT} \in \mathsf{L/poly}$ *if and only if* $\mathsf{NP} \subseteq \mathsf{L/poly}$.

Proof. Backwards implication is fairly straightforward, as any algorithm solving SAT in $\mathsf{L/poly}$, also solves prUSAT in $\mathsf{L/poly}$.

For the forward implication, assume M is an $\mathsf{L/poly}$-algorithm solving prUSAT. We now design an $\mathsf{L/poly}$ algorithm for SAT.

Note that from Lemma 15 we already have a randomized logspace isolation procedure with success probability $p = \Omega\left(\frac{1}{n}\right)$. By Adleman's argument (Lemma 2) we get an $\mathsf{L/poly}$ procedure, let us call it B, which on input a CNF formula C, outputs a list of $t = poly(n)$ CNF formulas, $\langle C'_1, C'_2, \ldots, C'_t \rangle$, such that, if C is unsatisfiable, each C'_i is unsatisfiable and if C is satisfiable then $\Omega\left(\frac{1}{n}\right)$ fraction of the C'_is are successful isolations of C. If M accepts at least one C'_i then accept; otherwise, reject.

The described algorithm is clearly in $\mathsf{L/poly}$. For correctness, if C is unsatisfiable, all of the C'_is are unsatisfiable, thus M must reject all of them. If on the other hand, C is satisfiable then at least one C'_i is uniquely satisfiable, thus M must accept this C'_i. □

Theorem 17. *Each of the following are equivalent: there is a randomized* L/poly *algorithm that isolates*

(i) *a minimum witness for CNF formulas with success probability* $p \geq \frac{2}{3} + \frac{1}{poly(l)}$.

(ii) *a satisfaction witness for CNF formulas with success probability* $p \geq \frac{2}{3} + \frac{1}{poly(l)}$.

(iii) *a satisfaction witness for CNF formulas with success probability* $p \geq \frac{1}{poly(l)}$ *and moreover, the algorithm is satisfiability preserving.*

(iv) NP $\subseteq$ L/poly.

Proof. *(i)* $\implies$ *(ii)*, *(i)* $\implies$ *(iii)* are immediate.

(iv) $\implies$ *(i)*. The proof is similar to the proof for Theorem 14.

(ii) $\implies$ *(iv)*. It suffices to prove that prUSAT $\in$ L/poly as per Lemma 16. Say the promised procedure is A with $p \geq \frac{2}{3} + \frac{1}{poly(l)}$. Using Adleman's argument (Lemma 2), we get an L/poly algorithm B which when given as input a CNF formula C, outputs a list of $t = poly(n)$ CNF formulas, $\langle C'_1, C'_2, \ldots, C'_t \rangle$ such that:

- Every satisfying assignment of each C'_i satisfies C.
- If C is satisfiable, at least p'-fraction of C'_i are uniquely satisfiable, $p' = p - \frac{1}{cl^c}$, c chosen such that $\forall l \in \mathbb{N}, p'(l) > \frac{2}{3}$.

Let R be a relation such that, given $C_1 \in Yes$ with w_1 as its unique satisfying assignment, $C_2 \in (Yes \cup No)$, $R(C_1, C_2)$ holds if and only if either:

- w_1 satisfies C_2.
- w_1 satisfies less that p'-fraction of C'_i of $B(C)$ where $C = C_1 \vee C_2$.

We show that R satisfies the properties required for Lemma 3.

For (K1), we see suppose $R(C_1, C_2)$ then either w_1 satisfies C_2 or w_1 satisfies less than p'-fraction of $B(C)$. In the latter case, from the construction of B we know that there are more than p'-fraction successful isolations of $B(C)$, therefore there must be an isolation of $B(C)$ which does not admit w_1. That gives an isolation of C_2. Thus in both cases, $C_2 \in Yes$.

For (K2), we assume for contradiction that $C_1, C_2 \in Yes$, $|C_1| = |C_2|$ but neither $R(C_1, C_2)$ nor $R(C_2, C_1)$ hold. Let w_1, w_2 be unique certificates of C_1, C_2, respectively. This means that neither w_1 satisfies C_2 nor w_2 satisfies C_1 and $w_1 \neq w_2$. Further, this means that both satisfy at least p'-fraction of $B(C)$. It is easy to see there must be at least $2p' - 1$ C'_is which admit both w_1 and w_2. $2p' - 1 > \frac{1}{3} > 1 - p'$ contradicting that at least p'-fraction of $B(C)$ are successful isolations.

$R(C_1, C_2)$ can be computed by a logspace machine using $C_1, w_1, p't$ as advice and calling B as a sub-routine. As B is in L/poly and deciding satisfiability of a CNF formula for given assignments can also be done with the same upper bound, we get that prUSAT $\in$ L/poly thus leading to $NL \subseteq$ L/poly.

For *(iii)* $\implies$ *(iv)*, we modify the relation R as follows, given C_1, w_1, C_2 (as above), $R(C_1, C_2)$ holds if and only if either:

- w_1 satisfies C_2,

- w_1 does not satisfy at least one C'_i from $B(C)$.

For (K1), if $R(C_1, C_2)$ and $C_1 \in Yes$, then either w_1 satisfies C_2 or w_1 does not satisfy one of the C_is. In the latter case, given that the isolation is satisfiability preserving, the formula not satisfied by w_1 must be satisfied by a witness of C_2. Thus in both cases $C_2 \in Yes$.

For (K2), we assume for contradiction that $C_1, C_2 \in Yes$, $|C_1| = |C_2|$ but neither $R(C_1, C_2)$ nor $R(C_2, C_1)$ hold. This means w_1 does not satisfy C_2, w_2 does not satisfy C_1, where w_1, w_2 are unique certificates of C_1, C_2, respectively and $w_1 \neq w_2$. Also, w_1 and w_2 satisfy all of $B(C)$ which contradicts the fact that at least one of $B(C)$ is an isolation.

As $R(C_1, C_2)$ can be computed by a logspace machine using C_1, w_1 as advice and calling B as a sub-routine, and as B is in $\mathsf{L/poly}$, we get that $\mathsf{prUSAT} \in \mathsf{L/poly}$. □

6 Further Discussion

Consider the following four properties for a procedure. On input G, a DAG:

1. it outputs DAGs $\langle G_1, G_2, \ldots, G_t \rangle$, where $t = poly(n)$ (not necessarily on the same vertices as G),
2. if G is satisfiable then *all* $\langle G_1, G_2, \ldots, G_t \rangle$ are satisfiable else none are,
3. if G is satisfiable, at least one of the G_is is uniquely satisfiable,
4. there is an $L/poly$ procedure to decide whether a given path in G will induce a path in an output graph or not.

We would like to note that our procedure from Lemma 6 satisfies properties 1, 2, and 3. Whereas, the isolation procedure proposed in [7] satisfies properties 1, 3, and 4. It is also interesting to note that if we get an isolation procedure for which all of the above properties hold then it would imply $\mathsf{NL} \subseteq \mathsf{L/poly}$ using a small variation of the proof of Theorem 8.

Acknowledgements. We would like to thank Meena Mahajan for useful discussions. We would also like to thank the anonymous referees for their valuable comments which helped to improve the manuscript.

References

[1] L. Adleman. Two theorems on random polynomial time. In *Foundations of Computer Science, 1978., 19th Annual Symposium on*, pages 75–83. IEEE, 1978.

[2] R. Arora, A. Gupta, R. Gurjar, and R. Tewari. Derandomizing isolation lemma for K3,3-free and K5-free bipartite graphs. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, pages 10:1–10:15, 2016.

[3] S. Arora and B. Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[4] V. Arvind and P. Mukhopadhyay. Derandomizing the isolation lemma and lower bounds for circuit size. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques, 11th International Workshop, APPROX 2008, and 12th International Workshop, RANDOM 2008, Boston, MA, USA, August 25-27, 2008. Proceedings*, pages 276–289, 2008.

[5] H. Dell, V. Kabanets, D. van Melkebeek, and O. Watanabe. Is valiant-vazirani's isolation probability improvable? In *Computational Complexity (CCC), 2012 IEEE 27th Annual Conference on*, pages 10–20. IEEE, 2012.

[6] S. A. Fenner, R. Gurjar, and T. Thierauf. Bipartite perfect matching is in quasi-NC. *CoRR*, abs/1601.06319, 2016.

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

[8] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

[9] K.-I. Ko. On self-reducibility and weak p-selectivity. *Journal of Computer and System Sciences*, 26(2):209–221, 1983.

[10] K. Mulmuley, U. V. Vazirani, and V. V. Vazirani. Matching is as easy as matrix inversion. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 345–354. ACM, 1987.

[11] R. Niedermeier and P. Rossmanith. Unambiguous auxiliary pushdown automata and semi-unbounded fan-in circuits. *Information and computation*, 118(2):227–245, 1995.

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

[13] I. H. Sudborough. On the tape complexity of deterministic context-free languages. *Journal of the ACM*, 25, 1978.

[14] L. G. Valiant and V. V. Vazirani. NP is as easy as detecting unique solutions. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 458–463. ACM, 1985.

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

A Appendix

A.1 Proof of Lemma 5

Proof of Lemma 5. To prove the claim we need to give an NL algorithm with properties mentioned in the claim. We do this by making a slight modification in the algorithm for Reach as presented in [12]. They keep track of two variable, c_k is the number of vertices reachable from s within k distance while Σ_k is the sum of distances of vertices reachable from s within k distance, distance being the length of shortest path from s. Note that they also check if the input graph is not min-unique. We only remove this check. The main procedure used sub-procedure 2 for computing c_k, Σ_k from c_{k-1}, Σ_{k-1} and sub-procedure 2 used sub-procedure one for checking $d(x) \leq k, x \in V$.

Algorithm 1 Reach algorithm: Main procedure

Input: G
Output: whether G is satisfiable

$c_0 := 1; \Sigma_0 := 0; k = 0$
repeat
  $k = k + 1$
  compute c_k, Σ_k from (c_{k-1}, Σ_{k-1})
until $c_{k-1} = c_k$
return whether $d(t) \leq k$.

Algorithm 2 Reach Algorithm: sub-procedure 1

Input: (G, k, v, c_k, Σ_k)
Output: halts the algorithm and rejects on some paths, and on others returns whether $d(v) \leq k$

$count := 0; sum := 0; foundPath := false$
for all $x \in V$ **do**
  Guess if $d(x) \leq k$
  if Guess is $d(x) \leq k$ **then**
    Guess a path of length $l \leq k$ from s to x
    if Found such a path **then**
      $count = count + 1; sum = sum + l$
      if $x == v$ **then**
        $foundPath = true$
      end if
    else
      halt and reject
    end if
  end if
end for
if $count == c_k$ **and** $sum == \Sigma_k$ **then**
  return $foundPath$
else
  halt and reject
end if

Algorithm 3 Reach algorithm: sub-procedure 2

Input: $(G, k, c_{k-1}, \Sigma_{k-1})$
Output: (c_k, Σ_k)

```
1: c_k := c_{k-1}; Σ_k = Σ_{k-1};
2: for all v ∈ V do
3:   if ¬(d(v) ≤ k − 1) then
4:     for all x: (x, v) ∈ E do
5:       if d(x) ≤ k − 1 then
6:         c_k = c_k + 1; Σ_k = Σ_k + k
7:         break this loop
8:       end if
9:     end for
10:  end if
11: end for
```

Now we prove the correctness of these procedures. (They follow from the correctness arguments presented in [12] but we describe them here for the sake of completeness.) For the correctness of sub-procedure 1, observe that there are two guesses it makes. If it incorrectly guesses $d(x) \leq k$, it will not be able to find a path of that length, if it incorrectly guesses $d(x) > k$ then the final value of c_k can not be right thus in both cases it halts and rejects. If it stops in the search of a path prematurely, it will not get the final value of c_k right and if it finds a path longer than the actual value of $d(x)$ and still gets the correct value of c_k, it cannot get the correct value of Σ_k thus in both cases it halts and rejects. Thus the only case in which it does not halt the algorithm is when it correctly finds $d(x)$ for all x thus it answers $d(v) \leq k$ correctly.

Now, the proof of correctness of sub-procedure 2 is straightforward. It is simply checking for all vertices at distance more than $k-1$ if there is a vertex adjacent to it with distance less than or equal to $k-1$. For the vertices it finds that the condition is true, meaning that those vertices are actually at a distance of k from s, it adds them to c_k, Σ_k thus correctly calculating their values.

The proof of correctness of main procedure is trivial. The space bound is straightforward as it only stores a constant number of vertex labels at any time.

Further, if the graph is min-unique, there is only set of choices that sub-procedure 1 can make that can make it accept and return. As it is the only non-deterministic part in the whole algorithm, even though it may be called multiple times, there is only one computation path through which the algorithm can accept. □

A.2 Proof of Lemma 10

Proof of Lemma 10. We need to give an $AuxPDA(log(n), n^{\mathcal{O}(1)})$ with the properties as per the claim. We do this by modifying the inductive counting algorithm presented in [12] for SAC^1-EVAL. They count two things, c_k is the number of gates in a *weighted* semi-unbounded circuit which have a certificate of weight less than k, Σ_k is the sum of weights of minimum-weight certificates of gates which have a certificate of weight less than k. They give two sub-procedures and the main procedure used them to evaluate the circuit on the input. They also check whether circuits are *min-unique* or not, which we remove.

Let $d = \mathcal{O}(log(n))$ represent the depth of C and the size be n^l for n, the number of inputs.

Algorithm 4 SAC^1-$EVAL$ algorithm: Main procedure

Input: C, x

Output: whether C accepts x

$c_0 := n + 1; \Sigma_0 := 0; k = 0$
for $k = 1$ **to** $2^d 4n^{3l}$ **do**
  compute c_k, Σ_k from (c_{k-1}, Σ_{k-1})
end for
if $W(g) < \infty$ **then**
  output $true$
else
  output $false$
end if
return whether $d(t) \leq k$.

Algorithm 5 SAC^1-$EVAL$ Algorithm: sub-procedure 1

Input: $(C, x, k, g, c_k, \Sigma_k)$

Output: halts the algorithm and rejects on some paths, and on others returns whether $W(v) \leq k$

$count := 0; sum := 0; a := \infty$
for all gate h **do**
  Guess if $W(h) \leq k$
  if Guess is $W(h) \leq k$ **then**
    Guess a certificate of weight $l \leq k$ for h
    if success **then**
      $count = count + 1; sum = sum + l$
      if $h == g$ **then**
        $a = l$
      end if
    else
      halt and reject
    end if
  end if
end for
if $count == c_k$ **and** $sum == \Sigma_k$ **then**
  return a
else
  halt and reject
end if

Algorithm 6 SAC^1-$EVAL$ algorithm: sub-procedure 2

Input: $(C, x, k, c_{k-1}, \Sigma_{k-1})$
Output: (c_k, Σ_k)

$c_k := c_{k-1}; \Sigma_k := \Sigma_{k-1}$;
for all gate g **do**
  if $W(g) \geq k$) **then**
    if g is an AND gate, with inputs h_1, h_2 with edge weights w_1, w_2 **and** $W(h_1) + W(h_2) + w_1 + w_2 = k$ **then**
      $c_k = c_k + 1; \Sigma_k = \Sigma_k + k$
    else if g is an OR gate **then**
      for all h connected to g by edge of weight w **do**
        if $W(h) + w == k$ **then**
          $c_k = c_k + 1; \Sigma_k = \Sigma_k + k$
          break this loop
        end if
      end for
    end if
  end if
end for

Now we prove the correctness of these procedures. (They follow from the correctness arguments presented in [12] but we describe them here for the sake of completeness.) For the correctness of sub-procedure 1, observe it makes two guesses. If it wrongly guesses $W(h) \leq k$ is true, it will not not be able to find a certificate for h of weight less than or equal to k, thus it will halt and reject. If it wrongly guesses $W(h) \leq k$ is false, it will not reach the correct value of c_k thus halt and reject. Given it guessed correctly $W(h) \leq k$, it can only guess a certificate of weight equal or more than the actual minimum certificate for h. If it finds one with more weight, it will not get the correct value of Σ_k if it gets the correct value for c_k, thus will halt and reject. This means that it finds the actual minimum weight certificate for h if it is less than or equal to k.

Correctness of sub-procedure 2 is straightforward. It is simply checking that for gates with weight k is there a way of actually achieving it from the gates that are input to it.

Correctness of main-procedure is trivial.

For the resource bound, the only non-trivial part is when a certificate for a gate is to be guessed in sub-procedure 1. This can be done using a depth-first search using the pushdown store starting from the gate and using non-determinism at the OR gates with $\mathcal{O}(log(n))$ workspace.

When the circuit is min-unique on the given input, there is only one accepting path for sub-procedure 1 for each gate. This means only one accepting path for the whole sub-procedure 1. As it is the only non-deterministic part in the whole algorithm, this means a unique accepting computation path as a whole. □

ECCC ISSN 1433-8092
http://eccc.hpi-web.de