Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 82 (2009)

A combinatorial property of ♯L assuming NL = UL and its implications for ModL

T. C. Vijayaraghavan
Department of Computer Science and Applications,
Bharath Institute of Higher Education and Research,
Selaiyur, Chennai-600073,
Tamil Nadu, India.
Email: vijayaraghavan.cbcs.cs@bharathuniv.ac.in

***Abstract*—We first show that** SLDAGSTCON **which is the** st**-connectivity problem for simple layered directed acyclic graphs, where the vertex** s **is in the** 1^{st} **row and the vertex** t **is in the last row of the input graph, is complete for** NL **under logspace many-one reductions. Let** $G = (V, E)$ **be a directed graph given as input and let** $s, t \in V$**. Also let** p **be a positive integer whose unary representation can be computed by a deterministic** $O(\log |G|)$ **space bounded Turing machine. Then we can determine if the number of directed paths from** s **to** t **in** G **is at least** $(p + 1)$ **in** FNL**. Otherwise if the number of directed paths from** s **to** t **in** G **is lesser than** $(p+1)$ **then the** FNL **machine outputs the number of directed paths from** s **to** t **in** G**. This in turn shows that verifying if there are polynomially many accepting computation paths for a** NL **machine on a given input is in** FNL**. Assume that** NL = UL**. Let** Σ **be the input alphabet,** $f \in$ ♯L **and** $g \in$ FL **such that, for the given input string** $x \in \Sigma^*$**,** $g(x)$ **is a positive integer** k **in the unary representation. Then we show that the function** $\binom{f(x)}{k} \in$ ♯L**. In this paper we obtain consequences of this combinatorial property of** ♯L **for the complexity class** ModL **defined in [AV10, Definition 3.1]. We prove the following characterization of** ModL**: assuming** NL = UL**, we show that for every language** $L \in$ ModL**, there exists a function** $f \in$ ♯L **and a function** $g \in$ FL **such that on any input** x**, we have**

- $g(x) = 0^p$ **for some prime** p**, and,**
- **if** $x \in L$ **then** $f(x) \equiv 1$**(mod** p**),**
- **if** $x \notin L$ **then** $f(x) \equiv 0$**(mod** p**).**

As a result, assuming NL = UL**, we are able to show that**

1) ModL **is the logspace analogue of the complexity class** ModP **defined by Köbler and Toda in [KT96, Definition 3.1], and**
2) ModL **is closed under complement.**

***Index Terms*—computational complexity, space bounded computation, logarithmic space bounded counting classes**

I. Introduction

As the main result of this paper I show in Theorem 4.1 by assuming NL = UL that if $f \in$ ♯L and $g \in$ FL such that $g(x)$ is the unary representation of a positive integer k, where $x \in \Sigma^*$ is the input, then the number of ways of choosing exactly $k = |g(x)|$ distinct paths from amongst the $f(x)$ accepting computation paths of the NL machine corresponding to f is in ♯L. In Theorem 3.1, I have shown that the st-connectivity problem for simple layered directed acyclic graphs (denoted by SLDAGSTCON) is complete for NL under logspace many-one reductions and this result plays the key role for the results shown in this paper. The algorithm I describe in Theorem 4.1 is analogous to the algorithm described in the well known Immerman-Szelepscényi Theorem [Sip13, Theorem 8.27] and uses the method of double inductive counting. Regarding our assumption that NL = UL in Theorem 4.1, it has been shown in [RA00, Theorem 2.2] that NL $\subseteq$ UL/poly and in [RA00, Corollary 2.3] it is shown that NL/poly = UL/poly. Assuming the existence of a language $L \in$ DSPACE(n) such that, to compute the characteristic function of L we require Boolean circuits of exponential size, it is possible to construct pseudo-random generators (see [ARZ99, Theorem 5.5]) using which it is possible to derandomize the randomized algorithm in [RA00] which will imply that NL = UL. In [BD$^+$92, Lemma 2(ii)] it is shown that if $f \in$ ♯L and $k > 0$ is a constant then the function $\binom{f(x)}{k} \in$ ♯L. Also in [BD$^+$92, Lemma 12] the same property for ♯L is shown with the assumption that the FL function g outputs a positive integer that varies with the input. However they impose a restriction on the NL-machine that the number of distinct computation paths that end in any accepting configuration is at most one. Such machines are called weakly unambiguous Turing machines. Subsequently [ARZ99, Theorem 4.3] show that if $f, g \in$ ♯L and $f(x)$ is at most a polynomial in the size of x then $\binom{f(x)}{g(x)} \in$ FNL/poly.

A. A logarithmic space bounded modulo counting class

The complexity class ModL was defined by Arvind and Vijayaraghavan in [AV10, Definition 3.1] to tightly classify the complexity of solving a system of linear equations modulo a composite number k, where k is given in terms of its prime factorization such that every distinct prime power divisor that occurs in the prime factorization of k is given in the unary representation. In this paper we study the complexity class ModL and obtain a characterization of ModL that shows that a ♯L function f and a FL function g that outputs a prime number in unary representation are sufficient to decide if a given input x is in a language $L \in$ ModL, however under the assumption that NL = UL. More precisely we are able to show a characterization of ModL in Theorem 5.2 that if we assume NL = UL and we have a language $L \subseteq \Sigma^*$ with $L \in$ ModL then we can decide whether an input $x \in \Sigma^*$ is in L using $f \in$ ♯L and $g \in$ FL where $g(x)$ is a prime number p that is output by g in the unary representation such that if $x \in L$ then $f(x) \equiv$

ISSN 1433-8092

$1(\text{mod } p)$ and if $x \notin L$ then $f(x) \equiv 0(\text{mod } p)$. The proof of Theorem 5.2 is similar to the result that $\text{Mod}_{p^e}\text{P} = \text{Mod}_p\text{P}$ shown in [BG92, Theorem 7.2] but assumes that NL = UL since we use Theorem 4.1 as its important ingredient. As an immediate consequence of Theorem 5.2, assuming NL = UL we are able to show that ModL is the logspace analogue of the complexity class ModP defined by Köbler and Toda in [KT96, Definition 3.1]. As another corollary we show that ModL is closed under complement assuming NL = UL in Corollary 5.4.

II. Definitions

Definition 2.1: We define SLDAG $= \{G|G = (V, E)$ is a simple layered directed acyclic graph which does not have any self-loops on vertices or directed cycles or parallel edges between vertices. Also vertices in G are arranged as a square matrix such that there are n rows and every row has n vertices. Any edge in this graph is from a vertex in the i^{th} row to a vertex in the $(i+1)^{st}$ row, where $1 \leq i \leq (n-1)$ and $n \geq 2\}$.

Definition 2.2: Let $G = (V, E) \in$ SLDAG and $s, t \in V$ such that s is a vertex in the first row and t is a vertex in the last row of G. We define SLDAGSTCON $= \{(G, s, t)|\exists$ a directed path from s to t in $G\}$.

Definition 2.3: Let Σ be the input alphabet. We define the complexity class NL $= \{L \subseteq \Sigma^* |$ there exists a nondeterministic $O(\log n)$ space bounded Turing machine M such that $L = L(M)\}$.

Definition 2.4: [AJ93] Let Σ be the input alphabet. The complexity class $\sharp$L is defined to be the class of functions $f : \Sigma^* \to \mathbb{Z}^+$ such that there exists a NL machine M for which we have $f(x) = acc_M(x)$ where $acc_M(x)$ denotes the number of accepting computation paths of M on any input $x \in \Sigma^*$.

Definition 2.5: [BJ$^+$91] Let Σ be the input alphabet. We say that $L \subseteq \Sigma^*$ is a language in the complexity class UL if there exists a function $f \in \sharp$L such that for any input $x \in \Sigma^*$ we have $f(x) = 1$ if $x \in L$ and $f(x) = 0$ if $x \notin L$.

Definition 2.6: Let Σ be the input alphabet and let Γ be the output alphabet. We define FNL to be the complexity class of all functions $f : \Sigma^* \to \Gamma^*$ such that for any given input string $x \in \Sigma^*$ there exists a NL machine M which outputs $f(x)$ at the end all of its accepting computation paths.

Definition 2.7: Let Σ be the input alphabet and let Γ be the output alphabet. We define FUL to be the complexity class of all functions $f : \Sigma^* \to \Gamma^*$ such that for any given input string $x \in \Sigma^*$ there exists a NL machine M which has atmost one accepting computation path and M outputs $f(x)$ at the end of its unique accepting computation path.

Definition 2.8: [AV10, Definition 2.2] Let Σ be the input alphabet. The complexity class GapL is defined to be the class of functions $f : \Sigma^* \to \mathbb{Z}$ such that there exists a NL machine M for which we have $f(x) = acc_M(x) - rej_M(x)$ where $acc_M(x)$ and $rej_M(x)$ denote the number of accepting and the number of rejecting computation paths of M on any input $x \in \Sigma^*$ respectively. We also denote $(acc_M(x) - rej_M(x))$ by $gap_M(x)$.

Definition 2.9: [Vij08, Definition 1.4.1] [AV10, Definition 3.1] Let Σ be the input alphabet. A language $L \subseteq \Sigma^*$ is in the complexity class ModL if there is a function $f \in$ GapL and a function $g \in$ FL such that on any input $x \in \Sigma^*$,

- $g(x) = 1^{p^e}$ for some prime p and a positive integer e, and
- $x \in L \Leftrightarrow f(x) \not\equiv 0(\text{mod } p^e)$.

III. Results on directed st-connectivity

Theorem 3.1: SLDAGSTCON is complete for NL under logspace many-one reductions.

Proof. It is a well known result of W. Savitch that the st-connectivity problem for directed graphs is complete for NL under logspace many-one reductions (refer [Sip13, Theorem 8.25]). Therefore the st-connectivity problem for directed graphs that do not have any self-loops on vertices or parallel edges between vertices is also complete for NL under logspace many-one reductions. In other words the st-connectivity problem for simple directed graphs is complete for NL under logspace many-one reductions. Let $G = (V, E)$ be a simple directed graph and $s, t \in V$. Also let $n = |V|$. Given G, we obtain the directed graph $G' = (V', E')$, where $V' = V$ and $E' = E \cup (t, t)$. We now reduce G' to $G'' = (V'', E'') \in$ SLDAG. In G'', the vertex set V'' has n^2 vertices arranged as a $n \times n$ matrix obtained by creating n copies of V'. Let the vertex s in the first row be denoted by $(1, s)$ and the vertex t in the last row be denoted by (n, t). A (i, j) vertex in G'' is the j^{th} vertex in the i^{th} row of G'', where $1 \leq i, j \leq n$. Here $E'' = \{((i, j_1), (i+1, j_2))|(j_1, j_2) \in E',$ where $1 \leq i \leq n-1$ and $1 \leq j_1, j_2 \leq n\}$. Now it is easy to note that there exists a directed path from s to t in G if and only if there exists a directed path from $(1, s)$ to (n, t) in G''. Since we can obtain G'' from G using a logspace many-one reduction the result follows. ■

Corollary 3.2: Let (G, s, t) be an input instance of SLDAGSTCON. Then counting the number of directed paths from s to t in G is complete for $\sharp$L under logspace many-one reductions.

Corollary 3.3: Let Σ be the input alphabet and $f \in \sharp$L using a NL machine M such that $f(x) = acc_M(x)$ on any input $x \in \Sigma^*$. Then for an input string $x \in \Sigma^*$, $g(x) = (G, s, t)$, where g is the canonical logspace many-one reduction used to show that SLDAGSTCON is NL-complete and (G, s, t) is an input instance of SLDAGSTCON, and $f(x)$ is equal to the number of directed paths from s to t in G.

Lemma 3.4: [Vij08, Lemma 6.3.1] Let $X = \{1, \ldots, n\}$ be a set and let $\mathcal{F} \subseteq 2^X$ such that $|\mathcal{F}| \leq p(n)$ for a polynomial $p(n)$. Let $r > (n+1)^2 p^2(n)$ be a prime number and for each $1 \leq i \leq r$ and $j \in X$ define the weight function $w_i : [n] \longrightarrow \mathbb{Z}_r$ as $w_i(j) = (i^j \text{mod } r)$. Further for each subset $Y \subseteq X$ define

$$w_i(Y) = \Sigma_{j \in Y} w_i(j)(\text{mod } r).$$

Then there exists a weight function w_m such that $w_m(Y) \neq w_m(Y')(\text{mod } r)$ for any two distinct $Y, Y' \in \mathcal{F}$.

Proof. For any $1 \leq m \leq r$ and $Y \in \mathcal{F}$, we can interpret $w_m(Y)$ as the value of the polynomial $q_Y(z) = \Sigma_{j \in Y} z^j$ at the point $z = m$ over the field $\mathbb{Z}_r$. For $Y \neq Y'$, notice that the polynomials $q_Y(z)$ and $q_{Y'}(z)$ are distinct and their degrees are at most n. Hence, $q_Y(z)$ and $q_{Y'}(z)$ can be equal for at most n values of z in the field $\mathbb{Z}_r$. Equivalently, if $Y \neq Y'$ then $w_i(Y) = w_i(Y')$ for at most n weight functions w_i. Since there are $\binom{|\mathcal{F}|}{2}$ pairs of distinct sets in $\mathcal{F}$, it follows that there are at most $\binom{|\mathcal{F}|}{2} \cdot n < n \cdot p^2(n)$ weight functions w_i for which $w_i(Y) = w_i(Y')$ for some pair of sets $Y, Y' \in \mathcal{F}$. Since $r > n \cdot p^2(n)$, there is a weight function as claimed by the lemma. ∎

Theorem 3.5: Let (G, s, t) be an input instance of SLDAGSTCON and let the number of directed paths from s to t in G be at most $p(n)$ for some polynomial $p(n)$, where $G = (V, E) \in$ SLDAG and $n = |E|$. Also let $E = \{1, \ldots, n\}$ and let $r > (n+1)^2 p^2(n)$ be a prime number and for each $1 \leq i \leq r$ and $j \in E$ define the weight function $w_i : E \longrightarrow \mathbb{Z}_r$ as $w_i(j) = (i^j \text{mod } r)$. Further for each subset $Y \subseteq E$ define

$$w_i(Y) = \Sigma_{j \in Y} w_i(j) (\text{mod } r).$$

Now let $\mathcal{F} \subseteq 2^X$ such that if $X \in \mathcal{F}$ then edges in X form a directed path from s to t in G. It is then possible to determine a weight function w_m such that $w_m(X) \neq w_m(X') (\text{mod } r)$ for any two distinct $X, X' \in \mathcal{F}$ and the number of directed paths from s to t in G in FNL.

Proof. We iteratively start with the first weight function and first replace each edge in G with weight w by a directed path of length w. Let the resulting directed graph obtained be G'. It is easy to note that the number of directed paths from s to t in G is equal to the number of directed paths from s to t in G'. Since it is shown in Theorem 3.1 that SLDAGSTCON is NL-complete under logspace many-one reductions it is possible to obtain an input instance (G'', s'', t'') of SLDAGSTCON when we are given (G', s, t) as input. Upon following the proof of Theorem 3.1 we infer that the number of directed paths from s to t in H' is equal to the number of directed paths from s'' to t'' in G''. Let us initialize a counter c to 0. We consider the simple layered directed acyclic graph structure of G'' and vertices which are copies of t in all the polynomially many layers of G''. We query the NL oracle if there exists a directed path in G'' from s'' to the copy of the vertex t in each of the layers of G''. If the oracle returns "yes" then we increment the counter c by 1. For a given weight function we can compute c in $O(\log |G|)$ space. Using Lemma 3.4 it follows that there exists a weight function w_m such that $w_m(X) \neq w_m(X') (\text{mod } r)$ for any two distinct $X, X' \in \mathcal{F}$, where $1 \leq m \leq r$. Clearly any such weight function will result in the maximum value for the counter c. As a result by storing c for successive weight functions and updating it by comparison we can find the weight function w_m also. Note that the maximum value of c is the number of directed paths from s to t in G itself. Since $|G''|$ is a polynomial in $|G|$ it follows that we can find the weight function w_m in FL^{NL} which is FNL. ∎

Theorem 3.6: Let $G = (V, E)$ be a directed graph given as input and let $s, t \in V$. Also let p be a positive integer whose unary representation can be computed by a deterministic $O(\log |G|)$ space bounded Turing machine. Then we can determine if the number of directed paths from s to t in G is at least $(p+1)$ in FNL. Otherwise if the number of directed paths from s to t in G is lesser than $(p+1)$ then the FNL machine outputs the number of directed paths from s to t in G.

Proof. Due to Theorem 3.1 we know that SLDAGSTCON is NL-complete under logspace many-one reductions. We therefore follow Theorem 3.1 and obtain $H \in$ SLDAG and vertices $s', t' \in V(H)$ from G. Upon following the proof of Theorem 3.1 we infer that the number of directed paths from s to t in G is equal to the number of directed paths from s' to t' in H. Let us consider the subgraph of H induced by vertices that are in at least one directed path from s' to t' in H. Let this subgraph be H'. We use induction on the number of layers $\lambda \geq 2$ in H' and consider subgraphs of H' formed by vertices in the first λ layers of H' such that the number of directed paths from s' to vertices in layer λ of H' is at most $(p+1)$ or a polynomial in $|G|$. Note that it is easy to compute this upperbound on the number of directed paths using $O(\log |G|)$ space. We now once again use Theorem 3.1 and Theorem 3.5 to compute the number of directed paths from s' to all the vertices in layer λ in H'. If the number of directed paths is greater than or equal to $(p+1)$ then we move to the accepting configuration, output $(p+1)$ and stop. Otherwise finally we would have computed the number of directed paths from s' to t' in H which is lesser than $(p+1)$. We then move to the accepting configuration, output this value and stop. In all the stages of this proof throughout, we use a deterministic $O(\log |G|)$ space bounded Turing machine with access to the FNL oracle. The deterministic $O(\log |G|)$ space bounded Turing machine submits queries deterministically to the FNL oracle. In the intermediary stages, the reductions are done by submitting queries deterministically to the FNL oracle and after reading the reply given by the oracle on the oracle tape. As a result given the input G it is possible to determine if the number of directed paths from s to t is at least $(p+1)$ is in FL^{FNL} which is FNL. ∎

Corollary 3.7: Verifying if there are polynomially many accepting computation paths for a NL machine on a given input is in FNL.

Proof. Let M be a NL machine and let g be the canonical logspace many-one reduction from $L(M)$ to SLDAGSTCON obtained by using the seminal result of W. Savitch that the st-connectivity problem for directed graphs is complete for NL under logspace many-one reductions [Sip13, Theorem 8.25] and Theorem 3.1. We note that if x is the input string then $g(x) = (G, s, t)$ and the number of accepting computation paths of M on x is equal to the number of directed paths

from s to t in G. We now use Theorem 3.6 to complete the proof. ■

Corollary 3.8: Assume that NL = UL. Let $G = (V, E)$ be a directed graph given as input and let $s, t \in V$. Also let p be a positive integer whose unary representation can be computed by a deterministic $O(\log |G|)$ space bounded Turing machine. Then we can determine if the number of directed paths from s to t in G is at least $(p+1)$ in FUL. Otherwise if the number of directed paths from s to t in G is lesser than $(p+1)$ then the FUL machine outputs the number of directed paths from s to t in G.

Note 1: The number of accepting computation paths of a NL machine is not altered if it simulates a NL machine of a language $L \in$ UL during intermediate stages to verify if some input string is in L.

IV. A CONDITIONAL COMBINATORIAL PROPERTY OF ♯L

Theorem 4.1: Assume that NL = UL. Let Σ be the input alphabet, $f \in \sharp\text{L}$ and $g \in$ FL such that, for the given input string $x \in \Sigma^*$, $g(x)$ is a positive integer k in the unary representation. Then the function $\binom{f(x)}{k} \in \sharp\text{L}$.

Proof. Let $x \in \Sigma^*$ be the input string. Given x, we obtain the number $k = |g(x)|$ in FL. It is easy to note that k is upper bounded by a polynomial in $|x|$. It follows from our assumption that NL = UL and Definition 2.5 that there exists a NL machine M' to which if we give a directed graph G' along with two vertices s' and t' in G' as input then M' outputs "yes" at the end of the unique accepting computation path and "no" at the end of all the other computation paths if there exists a directed path from s' to t' in G'. Otherwise if there does not exist any directed path from s' to t' in G' then M' outputs "no" at the end of all of its computation paths. Now given the input x, we obtain an instance of SLDAGSTCON, say (G, s, t), using a logspace many-one reduction from Theorem 3.1. It is easy to note that the graph G that we obtain is of size polynomial in $|x|$. Once again we assume that s is in row 1 and t is in row n. Also any two paths are distinct if there exists a vertex in one of the paths that is not in the other path. It follows from Corollary 3.3 that $f(x)$ is equal to the number the directed paths from s to t in G. We associate the label (i, j) to a vertex of G if it is the j^{th} vertex in the i^{th} row of G, where $1 \leq i, j \leq n$. In the following algorithm the row number is denoted by λ and a vertex in row λ is denoted by (λ, ω), where $1 \leq \lambda, \omega \leq n$. The number of distinct paths we have chosen till row λ is denoted by φ and η denotes the number of vertices in row λ that are in the φ distinct paths. We use nondeterminism to compute φ and η. Let us consider the SHARPLCFL algorithm described below which is based on the algorithm described in the well known Immerman-Szelepscényi Theorem [Sip13, Theorem 8.27] and uses the method of double inductive counting. Input to the SHARPLCFL algorithm is $(0^k, (G, s, t))$ where (G, s, t) is an input instance of SLDAGSTCON. If using SHARPLCFL we are able to nondeterministically choose exactly $k = |g(x)|$ distinct directed paths from s to t in G then the computation path ends in the accepting configuration. Otherwise the computation path ends in the rejecting configuration.

Algorithm 1 SHARPLCFL$(0^k, (G, s, t))$

```
1: λ ← 1, φ ← 1, η ← 1
2: while λ ≤ (n − 1) do
3:     φ′ ← 0, η′ ← 0, φ″ ← 0, η″ ← 0, ω ← 1
4:     while (ω ≤ n) do
5:         Nondeterministically either choose (λ, ω) or skip (λ, ω)
6:         if (λ, ω) is chosen nondeterministically then
7:             if (M′(G, s, (λ, ω)) returns"yes") and (M′(G, (λ, ω), t) returns "yes") then
8:                 η′ ← η′ + 1
9:                 if η′ > η then
10:                    reject the input
11:                else
12:                    σ ← 0, ψ ← 0
13:                    while ∃ a neighbour (λ+1, ρ) of (λ, ω) that is yet to be visited do
14:                        Nondeterministically either choose (λ + 1, ρ) or skip (λ + 1, ρ)
15:                        if (λ + 1, ρ) is chosen nondeterministically then
16:                            σ ← σ + 1
17:                            if M′(G, (λ + 1, ρ), t) returns "no" then
18:                                reject the input
19:                            else if (φ = k or φ″ = k) and σ ≥ 2 then
20:                                reject the input
21:                    if (σ = 0 or σ > k) then
22:                        reject the input
23:                    if #(directed paths from s to (λ, ω)) > φ − φ′ then
24:                        ψ ← max(1, φ − φ′)
25:                    else
26:                        ψ ← #(directed paths from s to (λ, ω))
27:                    Nondeterministically choose a number α from 1 to ψ
28:                    φ′ ← φ′ + α
29:                    Nondeterministically choose a number β from 0 to σ
30:                    φ″ ← αβ + φ″
31:                    η″ ← η″ + β
32:                    if (η″ = 0) or ((φ = k or φ″ > k) and η″ > η) then
33:                        reject the input
34:                    if φ″ > k then
35:                        φ″ ← k
36:            else
37:                reject the input
38:        ω ← ω + 1
39:    if λ + 1 < n and (η′ ≠ η or φ′ < φ) then
40:        reject the input
41:    else if λ + 1 = n and (η′ ≠ η or φ″ ≠ k) then
42:        reject the input
43:    λ ← λ + 1, φ ← φ″, η ← η″
44: accept the input
```

We note the following points about the SHARPLCFL algorithm.

Always $1 \leq \lambda \leq n$ and $1 \leq \omega \leq n$. In the SHARPLCFL algorithm, if we choose a vertex nondeterministically we verify if it is in a directed path from s to t in lines 7 and 17.

The variable η' is the number of vertices we are choosing nondeterministically in row λ and it must be equal to η after we have made nondeterministic choices on all the vertices in row λ failing which we reject the input in lines 39-42. φ' is used to verify if the number of distinct directed paths from s to t that pass through η' vertices chosen nondeterministically in row λ is atleast φ failing which we reject the input in lines 39-40.

The variable σ computed in line 16 inside the while loop from line 13 to 20 is the number of vertices chosen nondeterministically as the neighbours of (λ, ω) in row $\lambda + 1$ such

that these vertices are in at least one directed path from s to t in G. Here $1 \leq \sigma \leq n$.

We compute ψ in lines 24 and 26 in FUL by Corollary 3.8 without altering the number of accepting computation paths. Note that $\psi \geq 1$ since (λ, ω) is in at least one directed path from s to t in G. After lines 24 and 26, ψ is either $\max(1, \varphi - \varphi')$ or #(directed paths from s to (λ, ω)). Therefore $1 \leq \psi \leq \varphi$ always. From lines 1, 32-33 and 43 it follows that $1 \leq \varphi \leq k$ always and so in line 30 we always have $\varphi'' \leq kn + \varphi'' \leq 2kn^2$. $1 \leq \alpha \leq \psi$ and so $0 \leq \varphi' \leq 2kn$ always. Also $0 \leq \varphi'' \leq k$ in line 4 at the begining of the while loop. Therefore $0 \leq \varphi'' \leq 2kn^2$ always.

Always $0 \leq \beta \leq \sigma$, $0 \leq \eta' \leq n$, $0 \leq \eta'' \leq n^2$ and $1 \leq \eta \leq n^2$. If $\eta > n$ then we reject the input due to the condition in lines 39 and 41. Variables φ', η' and η'' are updated in the while loop from line 4 to 38 and these values do not decrease inside this loop.

Using nondeterminism to increment η'' in lines 29 and 31 by β is to nondeterministically avoid the possibility of counting vertices in row $\lambda + 1$ that are common neighbours of two distinct vertices in row λ more than once.

In lines 27 and 29, assume that we are always nondeterministically choosing the correct value of α and β respectively in the algorithm. Then our algorithm proceeds correctly and ends in an accepting configuration if and only if we have nondeterministically chosen exactly k distinct directed paths from s to t in G. On the contrary if φ'' is updated with an incorrect value of α or if η'' is incremented by an incorrect value of β, then in those iterations of the while loop from line 4 to 38, the algorithm proceeds by assuming that an alternate set of nondeterministic choices have been made on vertices in row $\lambda + 1$ which agree with η'' and φ''.

Any two directed paths formed by nondeterministically choosing two different vertices in row λ in lines 5-6 are distinct irrespective of their neighbours nondeterministically chosen in the row $\lambda + 1$ in lines 14-15. As a result if the number of directed paths from s to t in G is lesser than k then the value of φ'' computed in line 30 is always lesser than k and these inputs are rejected in lines 41-42.

At the end of the while loop in line 38, φ'' is the number of directed paths from s to t computed nondeterministically, that we need for subsequent stages of our algorithm. Also η'' is the number of distinct vertices that are in row $\lambda + 1$ in φ'' distinct directed paths from s to t in G which is also computed nondeterministically.

Now assume that the number of directed paths from s to t in G is at least k. The cases where we reject the input since the nondeterministic choices made on the neighbours of (λ, ω) in row $\lambda+1$ results in increasing the number of distinct directed paths nondeterministically chosen to be greater than k is in lines 19, 20, 32 and 33. Lines 19 and 20 are pertaining to the case when we are in vertex (λ, ω) in row λ and we are visiting the neighbours of (λ, ω) in row $\lambda + 1$. In this case we have already nondeterministically chosen k distinct directed paths from s to t and we have also chosen vertices in row $\lambda + 1$ in excess that results in increasing the number of distinct directed paths chosen nondeterministically from s to t to be greater than k. Similarly in lines 32 and 33 we have the case when we have chosen a (λ, ω) in row λ and we have visited all the neighbours of (λ, ω) in row $\lambda + 1$. If ($\varphi = k$ or $\varphi'' > k$) and $\eta'' > \eta$ then it implies that we have chosen more than k distinct directed paths from s to t in G that most recently includes directed paths that pass through (λ, ω) and η'' distinct neighbours of (λ, ω) in row $\lambda + 1$ and so in lines 32 and 33 we reject the input.

The case when we reject the input since we have not chosen η vertices in row $\lambda < n-1$ or at least φ distinct directed paths till row $\lambda < n - 1$ is in lines 39 and 40. The case when we reject the input since we have not chosen exactly k distinct directed paths from s to t but we have moved till row $n - 1$ is in lines 41 and 42.

In the SHARPLCFL algorithm, since we keep track of only a constant number of variables all of which take non-negative integer values and the values of these variables are upper bounded by a polynomial in the size of the graph G, which is once again polynomial in the input size $|x|$ we get a NL machine that executes the SHARPLCFL algorithm. Since we have assumed that NL = UL and we use M' to verify if there exists a directed path from a vertex s' to a vertex t' in lines 13 and 23, it follows that the number of accepting computation paths of the NL machine described by our SHARPLCFL algorithm does not get altered upon simulating M' (also see Note 1). As a result it follows that the NL machine described by the SHARPLCFL algorithm stops in an accepting state if and only if we start from vertex s in row 1 and reach vertex t in row n via exactly k distinct directed paths. Now as mentioned in Corollary 3.3, since $f(x)$ is equal to the number of directed paths from s to t in G and since we have assumed NL = UL, it follows that the number of accepting computation paths of this NL machine is $\binom{f(x)}{k}$. When $f(x) < k$ this NL machine has no accepting computation paths and so we get $\binom{f(x)}{k} = 0$. This shows that $\binom{f(x)}{k} \in \sharp\text{L}$. ■

V. Characterization of ModL

Lemma 5.1: Let $L \in$ ModL. Then there exists a function $f \in \sharp$L and a function $g \in$ FL such that on any input string x,

- $g(x) = 0^{p^e}$ for some prime p and a positive integer e, and
- $x \in L$ if and only if $f(x) \not\equiv 0 (\text{mod } p^e)$.

Proof. Let $L \in$ ModL be witnessed by functions $f' \in$ GapL and $g \in$ FL as in Definition 2.9. Now given $f' \in$ GapL there exists $f_1, f_2 \in \sharp$L such that on any input string x we have $f'(x) = f_1(x) - f_2(x)$. Consider $f(x) = f_1(x) + (p^e - 1) f_2(x)$. Since $\sharp$L is closed under multiplication by a FL function that outputs a positive integer and also under addition [BD$^+$92, Lemma 2(i)], we have $f(x) \in \sharp$L. Moreover on a given input x, we have $f'(x) \not\equiv 0 (\text{mod } p^e)$ if and only if $f'(x) = f_1(x) - f_2(x) \not\equiv 0 (\text{mod } p^e)$ if and only if $f(x) = f_1(x) + (p^e - 1) f_2(x) \not\equiv 0 (\text{mod } p^e)$. As a result we can replace the GapL

function f' by the $\sharp$L function f to decide if any given input string x is in L. ■

Theorem 5.2: Assume that NL = UL and let $L \in$ ModL. Then there exists a function $f \in \sharp$L and a function $g \in$ FL such that on any input string x,

- $g(x) = 0^p$ for some prime $p > 0$, and,
- if $x \in L$ then $f(x) \equiv 1 (\text{mod } p)$,
- if $x \notin L$ then $f(x) \equiv 0 (\text{mod } p)$.

Proof. Let $L \in$ ModL. It follows from Lemma 5.1 that there exists $f' \in \sharp$L and $g' \in$ FL such that on any input string x we have $g'(x) = 0^{p^e}$ for some prime p and a positive integer e, and $x \in L$ if and only if $f'(x) \not\equiv 0 (\text{mod } p^e)$. Let g be a FL function that outputs the prime p in unary when given the input x. Now assume that there exists a $f'' \in \sharp$L such that $f'(x) \not\equiv 0 (\text{mod } p^e)$ if and only if $f''(x) \not\equiv 0 (\text{mod } p)$. Then $x \in L$ if and only if $f''(x) \not\equiv 0 (\text{mod } p)$. Define $f(x) = (f''(x))^{(p-1)}$. Using Fermat's Little Theorem [BG92, Theorem 5.1] we have, if $x \in L$ then $f(x) \equiv 1 (\text{mod } p)$. Otherwise if $x \notin L$ then $f(x) \equiv 0 (\text{mod } p)$. We therefore prove the theorem statement if we define the function f'' such that $f'(x) \not\equiv 0 (\text{mod } p^e)$ if and only if $f''(x) \not\equiv 0 (\text{mod } p)$.

It is easy to see that we can compute the largest power of p that divides $p^e = |g'(x)|$ in FL. If $e = 1$ then we define $f'' = f'$ where f' is the $\sharp$L function in Lemma 5.1. It is clear that on an input string x we have $g(x) = g'(x) = 0^p$ for some prime p and $x \in L$ if and only if $f'(x) \not\equiv 0 (\text{mod } p^e)$ which is true if and only if $f''(x) \not\equiv 0 (\text{mod } p)$. Otherwise $e \geq 2$. Here we have $g'(x) = 0^{p^e}$ and $g(x) = 0^p$. We follow the proof of [BG92, Theorem 7.2] and use induction on $(e-1)$ to define f''. Inductively assume that for $1 \leq i \leq (e-1)$ we have functions $f_i \in \sharp$L such that $f'(x) \not\equiv 0 (\text{mod } p^i)$ if and only if $f_i(x) \not\equiv 0 (\text{mod } p)$. For the case when $(e-1) = 1$ we can have $f_{e-1} = f'$. Then it is clear that given an input string x, we have $f'(x)$ is divisible by p^e if and only if

1) $f'(x)$ is divisible by p^{e-1}, and
2) the coefficient of p^{e-1} in the base-p expansion of $f'(x)$ is zero.

Here Condition 2 stated above is equivalent to $\binom{f'(x)}{p^{e-1}} \equiv 0 (\text{mod } p)$ by [BG92, Corollary 5.5]. Therefore $f'(x) \not\equiv 0 (\text{mod } p^e)$ if and only if $\{f'(x) \not\equiv 0 (\text{mod } p^{e-1})$ or $\binom{f'(x)}{p^{e-1}} \not\equiv 0 (\text{mod } p)\}$ which is true if and only if $\{f_{e-1}(x) \not\equiv 0 (\text{mod } p)$ or $\binom{f'(x)}{p^{e-1}} \not\equiv 0 (\text{mod } p)\}$. Let $f'_e(x) = (f_{e-1}(x))^{p-1}\binom{f'(x)}{p^{e-1}}^{p-1} + ((f_{e-1}(x))^{p-1} + p - 1)\binom{f'(x)}{p^{e-1}}^{p-1} + (f_{e-1}(x))^{p-1}(\binom{f'(x)}{p^{e-1}}^{p-1} + (p-1))$. Now define $f_e(x) = (f'_e(x))^{p-1}$. Using Theorem 4.1 and [BD$^+$92, Lemma 2(i)] it follows that $f_e \in \sharp$L. Moreover on an input x, if $f'(x) \equiv 0 (\text{mod } p^e)$ then $f_e(x) \equiv 0 (\text{mod } p)$. On the other hand if $f'(x) \not\equiv 0 (\text{mod } p^e)$ then we consider the following cases. Case 1: $f'(x) \equiv 0 (\text{mod } p^{e-1})$: Then $f_{e-1}(x) \equiv 0 (\text{mod } p)$ and we also have $\binom{f'(x)}{p^{e-1}} \not\equiv 0 (\text{mod } p)$. As a result from the definition of $f_e(x)$ we get $f_e(x) \equiv 1 (\text{mod } p)$.
Case 2: $f'(x) \not\equiv 0 (\text{mod } p^{e-1})$: Then $f_{e-1}(x) \not\equiv 0 (\text{mod } p)$ and we can have either $\binom{f'(x)}{p^{e-1}} \not\equiv 0 (\text{mod } p)$ or $\binom{f'(x)}{p^{e-1}} \equiv 0 (\text{mod } p)$. But in both of these cases we get $f_e(x) \equiv 1 (\text{mod } p)$. As a result defining $f'' = f_e$ we complete the proof. ■

Corollary 5.3: Assume that NL = UL. Then ModL is the logspace analogue of ModP.

Corollary 5.4: Assume that NL = UL. Then ModL is closed under complement.

Proof. Let $L \in$ ModL. Then by Theorem 5.2 there exists $f \in \sharp$L and $g \in$ FL such that on any input x, we have $g(x) = 0^p$ for some prime p and if $x \in L$ then $f(x) \equiv 1 (\text{mod } p)$. Otherwise if $x \notin L$ then $f(x) \equiv 0 (\text{mod } p)$.

Let $h(x) = (f(x) + (p-1))^{(p-1)}$. It follows from [BD$^+$92, Lemma 2(i)] that $h(x) \in \sharp$L. Using Fermat's Little Theorem [BG92, Theorem 5.1] we have, if $x \in L$ then $h(x) \equiv 0 (\text{mod } p)$ and if $x \notin L$ then $h(x) \equiv 1 (\text{mod } p)$. Clearly this shows $\overline{L} \in$ ModL or that ModL is closed under complement. ■

ACKNOWLEDGEMENTS

I am extremely grateful to V. Arvind for many motivating and useful discussions on the results shown in this paper, on ModL and on counting classes. I thank Johannes Köbler for some useful discussions on the results shown in this paper.

REFERENCES

[AJ93] Carme Àlvarez and Birgit Jenner. A very hard log-space counting class. *Theoretical Computer Science*, 107(1):3-30, Elsevier, 1993.

[ARZ99] Eric Allender, Klaus Reinhardt and Shiyu Zhou. Isolation, Matching and Counting: Uniform and Nonuniform Upper Bounds. *Journal of Computer and System Sciences*, 59(2): 164-181, Elsevier, 1999.

[AV10] V. Arvind and T. C. Vijayaraghavan. Classifying Problems on Linear Congruences and Abelian Permutation Groups using Logspace Counting Classes, *Computational Complexity*, 19(1):57-98, Birkhauser, 2010.

[BD$^+$92] Gerhard Buntrock, Carsten Damm, Ulrich Hertrampf and Christoph Meinel. Structure and Importance of Logspace-MOD Classes. *Mathematical Systems Theory*, 25(3):223-237, Springer-Verlag, 1992.

[BG92] Richard Beigel and John Gill. Counting classes: Thresholds, Parity, Mods and Fewness, *Theoretical Computer Science*, 103(1):3-23, Elsevier, 1992.

[BJ$^+$91] Gerhard Buntrock, Birgit Jenner, Klaus-Jörn Lange and Peter Rossmanith. Unambiguity and fewness for logarithmic space. In *FCT '91: Proceedings of the 8^{th} International Conference on Fundamentals of Computation Theory*, LNCS 529, pp. 168-179, Springer, 1991.

[KT96] Johannes Köbler and Seinosuke Toda. On the power of generalized MOD-classes, *Mathematical Systems Theory*, 29(1):33-46, Springer-Verlag, 1996.

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

[Sip13] Michael Sipser. *Introduction to the Theory of Computation, Third edition*, Cengage Learning, 2013, First Indian reprint, 2018.

[Vij08] T. C. Vijayaraghavan. *Classifying certain algebraic problems using logspace counting classes*, Ph.D Thesis, The Institute of Mathematical Sciences, Homi Bhabha National Institute, December 2008.

This paper is published in the International Virtual Conference on Advances in Data Sciences and Theory of Computing (ICADSTOC-2022), pages 51-56, Bharath Institute of Higher Education and Research, 2022. ISBN: 978-93-5607-750-8.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il