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

Characterization of ModL using Prime Modulus

T.C. Vijayaraghavan [*]
{email: vijay@cmi.ac.in}

20th September 2009

Abstract

The complexity class ModL was defined by Arvind and Vijayaraghavan in [AV04] (more precisely in [Vij08, Definition 1.4.1][AV, Definition 3.1]). In this paper, under the assumption that 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 string x, we have

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

As a consequence under the assumption that NL = UL we show that

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

We prove the characterization of ModL stated above by showing the following property of #L. Assuming NL = UL, if $f \in$ #L and $g \in$ FL such that $g(x)$ is a positive integer k in unary that depends on the input x then the function $\binom{f(x)}{k} \in$ #L.

1 Introduction

The complexity class ModL was introduced in [AV04] to tightly classify the complexity of solving a system of linear equations modulo prime powers. The main assumption of this problem was that the prime power with respect to which we need to carry out the modulo operation is given as a part of the input in unary. One of the main results regarding ModL shown in [AV04][Vij08, Lemma 3.2.2][AV, Lemma 3.2] is that GapL and ModL are equivalent under logspace Turing reductions. In other words it is shown that $\mathrm{L}^{\mathrm{GapL}} = \mathrm{L}^{\mathrm{ModL}}$.

[*]This paper was written when I was a Post Doctoral Fellow in the Chennai Mathematical Institute, Siruseri 603103, India; and the results shown here are based on the results obtained in my Ph.D thesis [Vij08]. The previous version of this paper was submitted to the FSTTCS 2009 conference.

ISSN 1433-8092

In this paper we continue to study the complexity class ModL by obtaining a characterization that shows that just a #L function f and a FL function g that outputs a prime number in unary are sufficient to decide if a given input x is in a language $L \in$ ModL, however under the assumption that NL = UL. We prove this characterization in Theorem 3.6.

Very much the same kind of results have already been shown for modulo-based logspace counting classes. More precisely, Buntrock *et.al.* in [BDHM92, Theorem 7] show that if p^e is a prime power then $\mathrm{Mod}_{p^e}\mathrm{L} = \mathrm{Mod}_p\mathrm{L}$. In fact the proof of this result is exactly the same as in the polynomial time setting that $\mathrm{Mod}_{p^e}\mathrm{P} = \mathrm{Mod}_p\mathrm{P}$ shown in [BG92, Theorem 7.2]. Both these proofs are essentially based on choosing a set of $k > 0$ distinct accepting computation paths from the computation tree of a NL-machine or a NP-machine on a given input x. The very same proof works to prove our characterization in Theorem 3.6 but needs an extension of the above mentioned property of #L to the case when the integer $k > 0$ depends on the input and is output by a FL function in unary. More precisely we need to show that if $f \in$ #L and $g \in$ FL such that $g(x)$ is a positive integer k in unary that depends on the input x then the function $\binom{f(x)}{k} \in$ #L. We prove this property in Theorem 3.4 under the assumption that NL = UL. We now recall some related results which are precursors to the above stated property of #L and the new approach we have used in our proof.

Buntrock *et.al.* in [BDHM92, Lemma 2(ii)] show that if $f \in$ #L and $k > 0$ is a constant then the function $\binom{f(x)}{k} \in$ #L. They arrive at this result by showing that we can non-deterministically choose k distinct computation paths of the NL-machine corresponding to f on a given input and check if all of them end in the accepting configuration. At every stage we keep track of each of these paths chosen by storing the current configuration in each of them. Since any configuration is of size $O(\log n)$ and k is a constant the space bound is preserved and we get the result. The above observation has been extended in [AO96, Theorem 9] to the case when $f \in$ GapL and the constant k is computed by a FL function g.

Also Buntrock *et.al.* in [BDHM92, Lemma 12] show the same property for $f \in$ #L with the assumption that the FL function g outputs a positive integer k in the unary representation that varies with the input. However they impose a restriction on the NL-machine corresponding to f 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. This added assumption implies the number of accepting computation paths in such machines is at most a polynomial in the input size. 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.

Continuing along similar lines, in Theorem 3.4 under the assumption that NL = UL we show that if $f \in$ #L and $g \in$ FL such that $g(x)$ is a positive integer k in unary that depends on the input x then the function $\binom{f(x)}{k} \in$ #L. However as opposed to the results mentioned above, rather than considering the computation tree of a NL-machine on a given input we consider the NL-complete problem stated in Theorem 2.4 and the #L-complete function that follows from it to prove our result.

Once again as in the well known result that NSPACE$(f(n))$ is closed under complement due to Immerman and Szelepcsényi, where $f(n) \in \Omega(\log n)$, we also use the method of inductive counting to prove Theorem 3.4. It is well known and also follows from [BDHM92, Section 5] that by using inductive counting we can count the number of vertices that are reachable from a specified vertex s in a graph G by paths of length at most $i \geq 0$. However it is not obvious as to how to ensure that we do not count a vertex that is reachable from s more than once when using this method. This precisely happens to be the bottleneck in our case of choosing exactly k distinct directed paths from a vertex s to a vertex t when given an instance G of the directed st-connectivity problem stated in Theorem 2.4 as input. To keep track of k distinct paths, at different stages of our algorithm in Theorem 3.4 we need to repeatedly check if a vertex v chosen is in at least one directed path from s to t in G. However non-deterministically choosing a directed path from s to t and verifying if v lies in this path increases the number of accepting computation paths of the NL-machine and so it does not yield the desired result. However if we assume that NL = UL then we get a NL-machine for which there exists exactly one accepting computation path that verifies if v lies in a directed path from s to t in G. As a result under this assumption we show in Theorem 3.4 that the number of ways of choosing exactly k distinct directed paths from amongst all directed paths from s to t in G is in #L. It is widely believed that NL = UL and in [RA00, Corollary 2.3] it is shown that NL/poly = UL/poly. Also using well known derandomization techniques that depend on the existence of functions that require circuits of exponential size to be computed (for example [ARZ99, AV]) we can show that NL = UL.

Using Theorem 3.4 and closure properties of #L from [BDHM92, Lemma 2(i)] we show under the assumption that NL = UL, the proof of [BG92, Theorem 7.2] holds for ModL also and thereby enables us to prove our characterization in Theorem 3.6. As an immediate consequence 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 a corollary we also show that ModL is closed under complement assuming NL = UL.

2 Preliminaries

We start by defining some logspace counting classes.

Definition 2.1. [AO96] *We define* #L *to be the class of functions $f : \Sigma^* \rightarrow \mathbb{Z}^+$, for which there is a logspace bounded nondeterministic Turing machine M such that on any input $x \in \Sigma^*$, we have $f(x) = acc_M(x)$, where $acc_M(x)$ is the number of accepting computation paths of M on input x.*

Definition 2.2. [AO96] *We define* GapL *to be the class of functions $f : \Sigma^* \rightarrow \mathbb{Z}$, for which there is a logspace bounded nondeterministic Turing machine M such that on any input $x \in \Sigma^*$, we have $f(x) = acc_M(x) - rej_M(x)$, where $acc_M(x)$ and $rej_M(x)$ denote the number of accepting and rejecting computation paths of M on input x respectively.*

Definition 2.3. [BDHM92, Definition 4] *Let $k \geq 2$ be an integer. A language L is in* $\mathrm{Mod}_k\mathrm{L}$ *if there exists a function $f \in$ #L such that on any input x, we have $x \in L$ if and only if $f(x) \not\equiv 0 (\mathrm{mod}\ k)$.*

The completeness result stated below follows as a corollary of the reduction from iterated integer matrix multiplication to computing the determinant of an integer matrix shown in [Tod91].

Theorem 2.4. [ABO99, HT03] *The st-connectivity problem for graphs that are layered directed acyclic* (LDAG-*st*-CON) *is complete for* NL *under logspace many-one reductions. Here we assume the number of layers in the input graph is n. Also every layer has n vertices and vertex s is in layer 1 and vertex t is in layer n. Any edge in this graph is from a vertex in layer i to a vertex in layer $(i+1)$, where $1 \leq i \leq (n-1)$, and the number of edges between any two vertices is at most 1. As a result any path from s to t is of length $(n-1)$.*

Counting the number of paths from vertex s in layer 1 to vertex t in layer n in the graph described above is complete for #L *under logspace many-one reductions.*

Remark 1. *Let $L \in$ NL and M be a* NL *machine that accepts L. It is not difficult to see that in proving Theorem 2.4, we always have a reduction from L to* LDAG-*st*-CON *such that if we are given an input x for M then the graph G that we obtain is such that the number of accepting computation paths of M on input x equals the number of directed paths from vertex s to vertex t in G. In other words the reduction is parsimonious.*

Definition 2.5. [RA00, Section 1] *A language L is in* UL *if there exists a logspace bounded non-deterministic Turing machine M such that on any input x, we have if $x \in L$ then the number of accepting computation paths of M on input x is 1. Otherwise if $x \notin L$ then the number of accepting computation paths of M on input x is 0.*

The following relation is then immediate from the definitions of these logspace counting classes: $\mathrm{L} \subseteq \mathrm{UL} \subseteq \mathrm{NL}$. Also $\mathrm{UL} \subseteq \mathrm{Mod}_k\mathrm{L}$ for all integers $k \geq 2$.

2.1 ModL

Definition 2.6. [AV04] (more precisely in [Vij08, Definition 1.4.1][AV, Definition 3.1]). *A language L 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 string x,*

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

It follows from the above definition that ModL generalizes the class $\mathrm{Mod}_{p^e}\mathrm{L}$ where p^e is a prime power. As a result it follows that $\mathrm{Mod}_{p^e}\mathrm{L} \subseteq \mathrm{ModL}$ and so $\mathrm{UL} \subseteq \mathrm{ModL}$.

A canonical problem that is complete for ModL is $\mathrm{ModpeDet} = \{(A, 0^{p^e}) \mid \det(A) \not\equiv 0 (\bmod\ p^e)$, where $A \in \mathbb{Z}^{n\times n}$ and p^e is a prime power$\}$. It is easy to see that ModpeDet $\in$ logspace-uniform TC^1. This is essentially due to the fact that computing the determinant of a square integer matrix is in logspace-uniform TC^1. Moreover given two integers a and b each of size n, computing $\lfloor a/b \rfloor$ is in logspace-uniform TC^0 [HAB02, Corollary 6.3][CDL01]. As a result we can check for any input matrix $A \in \mathbb{Z}^{n\times n}$, if $\det(A) \not\equiv 0 (\bmod\ p^e)$. This shows ModpeDet $\in$ logspace-uniform TC^1 from which we get ModL $\subseteq$ logspace-uniform TC^1.

We also know the following result.

Lemma 2.7. [AV04][Vij08, Lemma 3.2.2][AV, Lemma 3.2] $\mathrm{L}^{\mathrm{ModL}} = \mathrm{L}^{\mathrm{GapL}}$.

3 A characterization of ModL

In this section we obtain a characterization of ModL in Theorem 3.6 which is based on the unproven assumption that NL = UL.

3.1 Choosing polynomially many distinct paths in #L

Under the assumption that NL = UL, we first show in Theorem 3.4 that if $f \in \#\mathrm{L}$ and $g \in \mathrm{FL}$ such that the function g outputs an integer $k > 0$ which varies with the input then the function $\binom{f(x)}{k} \in \#\mathrm{L}$. We need the following observations.

Lemma 3.1. *Given an instance G of* LDAG-*st*-CON *as described in Theorem 2.4, if there exists a path from vertex s to vertex t in G, then in* FNL *we can obtain the subgraph H of G such that any vertex or edge of G is in H if and only if the corresponding vertex or edge is in some path from s to t in G.*

As a consequence given a vertex or an edge of G it is possible to decide if the vertex or edge is in the subgraph H in NL*.*

Proof. Given an instance G of LDAG-*st*-CON, a logspace machine can use the NL-complete problem of directed *st*-connectivity as an oracle to check for each vertex v in G if there exists a directed path from s to v and a directed path from v to t in G. If both these are true then we output the vertex v. As a result we can obtain the subset of vertices that lie in every directed path from s to t in FNL.

It is also easy to see that a logspace machine with the above procedure as an oracle can output the subgraph H of G that is induced by the vertices we have obtained in the previous step. This also shows that the subgraph H can be output by a FNL machine. Having shown the FNL upper bound for obtaining H it is clear that verifying if some vertex or edge in G is in H is in NL. ∎

Given this subgraph H as input we now show how we can the check if the number of paths from s to t in G is at least $(p + 1)$ in UL, where p is bounded by a polynomial in $|G|$. The proof of this result uses the deterministic weight assignment scheme from [AHT07, Section 3, Lemma 3.2] to isolate all the polynomially many paths from s to t in H. This deterministic weight assignment scheme is used in [AHT07] to solve the polynomially bounded perfect matching problem in undirected graphs. This deterministic weight assignment scheme has also been used in [Vij08, Chapter 6] to solve the polynomially bounded linearly representable matroid intersection problem. Let FUL be the set of functions computable by a UL machine.

Lemma 3.2. *Let G be an instance of* LDAG-*st*-CON *and let H be the subgraph of G that we obtain from the* FNL *procedure described in Lemma 3.1. Then given the graph H as input we can check if the number of directed paths from vertex s to vertex t in G is at least $(p+1)$*

in UL*, where p is bounded by a polynomial in $|G|$. Otherwise if the number of paths from s to t in G is less than $(p+1)$ then the* FUL *procedure outputs the number of paths from s to t in G.*

Proof. It follows from the definition of the subgraph H that there exists a directed path from vertex s to vertex t in G if and only if the same directed path exists from vertex s to vertex t in H also. As a result number of paths from vertex s to vertex t in G is equal to the number of paths from vertex s to vertex t in H. Moreover any two paths from s to t in H are distinct if and only if there is an edge in one of the paths that is not in the other path. Since H always contains at least one path from s to t it follows that $(p+1)$ is bounded by a polynomial in $|H|$.

To verify if the number of paths from vertex s to vertex t is at least $(p+1)$, we iteratively consider subgraphs H_λ of H formed by the vertices in the first λ layers of H where $\lambda \geq 2$. Any path from s to a vertex in layer λ is of length $(\lambda - 1)$. Also if ϕ is the maximum number of paths from s to a vertex in layer $(\lambda - 1)$ and if m is the maximum outdegree of any vertex in layer $(\lambda - 1)$ then the number of paths from s to vertices in layer λ is at most $\phi m n_\lambda$ where n_λ is the number of vertices in layer λ in H_λ. Whenever we consider H_λ we always ensure that $\phi m n_\lambda$ is at most $(p+1)$ or bounded by a polynomial in $|H|$. This is therefore computable in $O(\log |H|)$ space.

We now use the deterministic weight assignment scheme from [AHT07, Section 3, Lemma 3.2] to assign weights to the edges of H_λ using at most $O(\log |H|)$ space. The weight of any path in H_λ is the sum of the weights of the edges in the path and is therefore bounded by a polynomial in $|H|$. Number of paths from s to vertices in layer λ is polynomially bounded in $|H|$. Also it follows from [AHT07, Section 3, Lemma 3.2] that every such path of length $(\lambda - 1)$ from vertex s in H_λ gets a unique weight. Now given H_λ with a vertex t_λ in layer λ as input verifying if there exists a path from s to t_λ having weight w is in UL (for example refer [RA00, Theorem 2.2]). We can now check if such a path exists for all the possible polynomially many weights and obtain the number of paths from s to vertices in layer $(\lambda - 1)$. We repeat these steps for H_λ from $\lambda \geq 2$ until we get the number of paths from vertex s to be at least $(p+1)$. If at some stage we get the number of paths to be greater than p then we output p. Otherwise we would have computed the number of paths from s to t in H which we output. ∎

We now recall the complexity class UL from Definition 2.5. It has been shown in [RA00, Theorem 2.2] that NL $\subseteq$ UL/poly. The proof of this result is based on using the isolating lemma to identify a unique path from vertex s to vertex t on a given instance G of the directed st-connectivity problem. It is widely believed that NL = UL and under possible derandomization assumptions we can show that NL = UL.

Corollary 3.3. *Assume that* NL = UL*. Given an instance G of* LDAG-st-CON *as described in Theorem 2.4, if there exists a path from vertex s to vertex t in G, then in* FUL *we can obtain the subgraph H of G such that any vertex or edge of G is in H if and only if the corresponding vertex or edge is in some path from s to t in G.*

As a consequence given a vertex or an edge of G it is possible to decide if the vertex or edge is in the subgraph H in UL.

Also given this subgraph H of G as input, we can check if the number of directed paths from vertex s to vertex t in G is at least $(p+1)$ in UL, *where p is bounded by a polynomial in $|G|$. Otherwise if the number of paths from s to t in G is less than $(p+1)$ then the* FUL *procedure outputs the number of paths from s to t in G.*

Proof. The proof follows directly from our assumption that NL = UL, Lemma 3.1 and Lemma 3.2. ∎

Note 1. *Let H be the subgraph of G that we obtain from Lemma 3.1. It is easy to note that given an instance G of* LDAG-*st*-CON *and vertices s and t the subgraph H is unique. Also the* FNL *machine may have more than one accepting computation path in which it outputs the subgraph H and stops. However in Corollary 3.3 we have shown in that this subgraph H is output by a* FUL *machine under the assumption that* NL = UL. *It therefore follows that any such* FUL *machine will output the subgraph H on a unique accepting computation path and stop. Any other computation path of this* FUL *machine ends in a rejecting state in which it may output some vertices or edges which is not the subgraph H and stop. Also the* UL *machine that verifies if some vertex or an edge is in H has a unique accepting path, if any. It is important to note that the number of accepting computation paths of a* NL *machine does not change even if it simulates a* UL *machine during intermediate stages to verify if some string is in a language $L \in$* UL. *We need this observation along with the parsimonious reduction for* LDAG-*st*-CON *mentioned in Remark 1.*

Theorem 3.4. *Assume that* NL = UL. *Then for any function $f \in$ #L and a function $g \in$ FL such that if $g(x)$ is a positive integer k in unary that depends on the input string x then the function $\binom{f(x)}{k} \in$ #L.*

Proof. Given an input string x we obtain a graph G as an instance of LDAG-*st*-CON using a logspace many-one reduction mentioned in Theorem 2.4. Once again we assume vertex s in layer 1 and vertex t is in layer n. It follows from Remark 1 that $f(x)$ is equal to the number the directed paths from s to t in G. We start by first checking if there exists a path from vertex s to vertex t in G using the FUL procedure described in Corollary 3.3. If this procedure ends in a rejecting state then the NL machine we describe also ends in a rejecting state and stops.

We now proceed by assuming this UL procedure ends in an accepting state. Firstly we note that the graph G that we obtain is of size polynomial in $|x|$. We associate the label (i, j) to a vertex of G if it is the j^{th} vertex in the i^{th} layer of G. Here j is called its vertex number. It follows from Theorem 2.4 that $1 \leq i, j \leq n$. Also any two paths are distinct if there exists a vertex in one of the paths that is not in the other path.

We now show how to non-deterministically choose a set of $k = |g(x)|$ distinct paths using at most $O(\log |x|)$ space from the set of all paths from s to t in G. As a result we obtain a NL machine whose number of accepting computation paths is $\binom{f(x)}{k}$ on input x. When

$f(x) < k$ the NL machine so constructed has no accepting computation paths. In such cases we define $\binom{f(x)}{k} = 0$.

In order to non-deterministically choose k distinct paths from s to t we keep track of the following variables:

- the layer number we are currently in, denoted by λ,
- number of distinct paths we have chosen so far till layer λ, denoted by φ, and
- number of distinct vertices in layer λ that lie in the φ distinct paths that we have chosen so far. We denote this variable by η. Note that there could be a vertex in layer λ that is in more than one of the φ distinct paths that we have chosen so far. However η does not count such vertices more than once.

We start by assuming the vertex s has the label $(1, j)$. The variables defined above are initialized to $\lambda = 1$, $\varphi = 1$ and $\eta = 1$. Also we assume $\lambda < n$. In every step of our algorithm we do the following operations.

1. we have variables η' and φ' that are initialized to 0 before we choose any vertex from layer λ.

2. starting from $\omega = 1$ until $\omega \leq n$, non-deterministically choose distinct vertices (λ, ω) in the increasing order from layer λ.

3. if the vertex (λ, ω) is chosen then we use the UL procedure described in Corollary 3.3 to verify if there exists a directed path from s to (λ, ω) and if there exists a directed path from (λ, ω) to t in G.

4. if this UL procedure ends in a rejecting state then we also reject the input and stop.

5. otherwise if this UL procedure ends in an accepting state then there is a directed path from s to (λ, ω) and there is a directed path from (λ, ω) to t in G. We now increment η' by 1. If we have $\eta' > \eta$ then we reject the input and stop.

6. we now check if the number of directed paths from s to (λ, ω) in G is at least φ. For this we first construct a graph G'_ω that contains the first λ layers of G without the vertices in layer λ whose vertex numbers are greater than ω. We also add a new directed path of length $(\lambda - 1)$ from s to (λ, ω) in G'_ω and a self-loop to the vertex (λ, ω). It is easy to see that number of paths from s to (λ, ω) in G is one less than the number of paths from s to (λ, ω) in G'_ω. We then use the logspace many-one reduction mentioned in Theorem 2.4 to obtain the instance G_ω with vertices s_ω and t_ω of LDAG-*st*-CON from G'_ω. Since there exists a directed path from s to (λ, ω) in G'_ω we know that there exists a directed path from s_ω to t_ω in G_ω. Also it follows from Remark 1 that this reduction from G'_ω to G_ω is parsimonious.

   Now giving G_ω as input to the FUL procedure in Corollary 3.3 we can verify if the number of paths from vertex s_ω to t_ω in G'_ω is at least $(\varphi + 1)$. Or equivalently whether the number of paths from s to (λ, ω) in G is at least φ. If this procedure ends in a rejecting state then we also reject the input and stop.

Otherwise from using the FUL procedure we can determine if the number of paths from s to (λ, ω) in G is at least φ and increment φ' by φ. Otherwise if the number of paths from s to (λ, ω) in G, say ϕ, is lesser than φ, then we increment φ' by ϕ.

7. if we have $\lambda \leq (n-1)$ then we choose a non-empty set of distinct neighbours of (λ, ω) from layer $(\lambda + 1)$ in the increasing order. We have variables η'' that is initialized to 0 and φ'' that is initialized to φ before we choose any vertex from layer $(\lambda + 1)$. Once we have made non-deterministic choices on all vertices in layer λ we would have also chosen η'' distinct vertices from layer $(\lambda + 1)$ to obtain $\varphi'' \geq \varphi$ distinct paths from vertex s to t.

8. if we have $(\lambda + 1) = n$ and we non-deterministically choose a neighbour of (λ, ω) that is not the vertex t from layer n then we reject the input and stop.

   Assume that $\lambda < (n-1)$ and that we have non-deterministically chosen the neighbour $(\lambda + 1, \rho)$ of (λ, ω) in G. We then we use the UL procedure in Corollary 3.3 to check if there exists a directed path from $(\lambda + 1, \rho)$ to t in G. If this UL procedure rejects the input then we also reject the input and stop. Otherwise there exists at least one directed path from s to t that passes through $(\lambda + 1, \rho)$.

9. We say that a vertex (λ, α) is a predecessor of $(\lambda + 1, \rho)$ if there is an edge from (λ, α) to $(\lambda + 1, \rho)$ in G. Now {if we have $(\eta' = 1)$ or (if we have $\eta'' = 0$) or (if there are no predecessors (λ, α) of $(\lambda + 1, \rho)$ such that $\alpha < \omega$)} then we increment η'' by 1 and go to step 18.

10. otherwise if all these three conditions are false then we use the UL procedure described in Corollary 3.3 as follows to find out the number of predecessors (λ, α) of $(\lambda + 1, \rho)$ that are in at least one path from the vertex s to the vertex t in G such that $\alpha < \omega$.

11. we do this as follows. We have a counter pred_ρ that is initialized to 0 to count the number of predecessors of $(\lambda + 1, \rho)$ in layer λ whose vertex numbers are lesser than ω and which are in at least one directed path from s to t in G. First note that there is at least one directed path from s to t that passes through $(\lambda + 1, \rho)$. Now let (λ, α) be a predecessor of $(\lambda + 1, \rho)$. As a result there exists a path from (λ, α) to t. Therefore to check if (λ, α) is reachable from s we follow the construction in step 6 to obtain graphs G'_α and the instance G_α of LDAG-*st*-CON with vertices s_α and t_α. Number of paths from s to (λ, α) in G is one less than the number of paths from s to (λ, α) in G'_α. Since there exists at least one path from s to (λ, α) in G'_α we know that there exists at least one path from s_α to t_α in G_α. Also it follows from Remark 1 that this reduction from G'_α to G_α is parsimonious.

    Now giving G_α as input to the FUL procedure in Corollary 3.3 we can verify if the number of paths from vertex s to (λ, α) in G'_α is at least 2. If this FUL procedure ends in a rejecting state then we also reject the input and stop.

12. otherwise we obtain the number of directed paths from s to (λ, α) in G'_α. We check if this is at least 2. If this is not true then (λ, α) is not reachable from vertex s in G and so we move to the next predecessor in layer λ, if any, of $(\lambda+1, \rho)$ in G in the increasing order of vertex numbers till we reach (λ, ω). If for all predecessors of $(\lambda+1, \rho)$ there does not exist any path from vertex s in G then we increment η'' by 1 and go to step 18. However if there exists a directed path from s to (λ, α) in G then we increment the counter pred_ρ by 1.

13. now assume that pred_ρ is at least 1. We have then found at least one predecessor (λ, α) of $(\lambda+1, \rho)$ whose vertex number is lesser than ω that is reachable from vertex s in G. We once again use the UL procedure from Corollary 3.3 and the method used in the steps 11 and 12 to count the number of distinct neighbours of these pred_ρ predecessors of $(\lambda+1, \rho)$ in layer $(\lambda+1)$ that are in at least one path from s to t in G. Let this counter be succ_ρ. We decrement succ_ρ by 1 to leave out the vertex $(\lambda+1, \rho)$ from this set.

    In the next few steps we do a case analysis on pred_ρ and succ_ρ to determine if the vertex $(\lambda+1, \rho)$ has already been counted in choosing at most k distinct paths from s to t.

14. $\{(\text{if } ((\eta'-1) \geq \text{pred}_\rho)) \text{ and } (\text{if } (\eta'' > \text{succ}_\rho))\}$ then we do not increment η''. If these two conditions are satisfied then since $\text{pred}_\rho \geq 1$ we assume that we have chosen at least one of the pred_ρ predecessors of $(\lambda+1, \rho)$ from layer λ whose vertex numbers are $< \omega$ that are reachable from s and that we have also chosen $(\lambda+1, \rho)$ from layer $(\lambda+1)$ using the edge from one such predecessor. As a result we do not increment η'' so that we do not count $(\lambda+1, \rho)$ more than once.

15. otherwise $\{(\text{if } ((\eta'-1) \geq \text{pred}_\rho)) \text{ and } (\text{if } (\eta'' \leq \text{succ}_\rho))\}$ then increment η'' by 1. If these two conditions are satisfied then since $\text{pred}_\rho \geq 1$ we assume that we have chosen at least one of the pred_ρ predecessors of $(\lambda+1, \rho)$ from layer λ whose vertex numbers are $< \omega$ that are reachable from s but that we have not chosen $(\lambda+1, \rho)$ so far from layer $(\lambda+1)$. However since we have chosen $(\lambda+1, \rho)$ using the edge from (λ, ω) we increment η'' by 1.

16. otherwise $\{(\text{if } ((\eta'-1) < \text{pred}_\rho)) \text{ and } (\text{if } (\eta'' > \text{succ}_\rho))\}$ then we do not increment η''. If these two conditions are satisfied then it follows that we have not chosen all of the predecessors of $(\lambda+1, \rho)$ that are reachable from s. However since $\text{pred}_\rho \geq 1$ we assume that we have chosen at least one predecessor of $(\lambda+1, \rho)$ from layer λ whose vertex numbers are $< \omega$ that is reachable from s and we have also chosen $(\lambda+1, \rho)$ from layer $(\lambda+1)$ using the edge from one such predecessor. As a result we do not count $(\lambda+1, \rho)$ more than once and so we do not increment η''.

17. otherwise $\{(\text{if } ((\eta'-1) < \text{pred}_\rho)) \text{ and } (\text{if } (\eta'' \leq \text{succ}_\rho))\}$ then increment η'' by 1. If these two conditions are satisfied then since $\text{pred}_\rho \geq 1$ we assume that we have chosen at least one but not all of the predecessors of $(\lambda+1, \rho)$ whose vertex numbers are $< \omega$

that is reachable from s. Also we assume that we have not chosen all of the neighbours of these pred_ρ predecessors in layer $(\lambda+1)$. Therefore we can assume that we have not chosen $(\lambda+1, \rho)$ so far and since we have chosen $(\lambda+1, \rho)$ using the edge from (λ, ω) we increment η'' by 1.

18. we have the variable φ'' that is initialized to φ before we choose any vertex from layer $(\lambda+1)$. Having non-deterministically chosen (λ, ω) from layer λ, it follows from step 6 that there are at least ϕ' (which is $\leq \varphi$) paths from s to (λ, ω) in G. Assume that we have nondeterministically chosen m neighbours of (λ, ω) from layer $(\lambda+1)$. If $m = 0$ then we reject the input and stop. Otherwise if $\phi' \leq (m-1)$ then we nondeterministically choose an integer ψ from $(m - \phi')$ to $\phi' m$ and increment φ'' by ψ. However if $\phi' \geq m$ then we nondeterministically choose an integer ψ from 0 to $\phi' m$ and increment φ'' by ψ.

19. if we have $\varphi'' > k$ then we reject the input and stop.

20. {(if we have $(\lambda+1) < n$) and (if we have not made non-deterministic choices on all the n vertices in layer λ)} then go to step 2.

21. otherwise {(if we have $(\lambda+1) < n$) and (if we have made non-deterministic choices on all the n vertices in layer λ) and {(if we have φ' is not greater than or equal to φ) or (if we have η' is not equal to η)}} then we reject the input and stop.

22. otherwise {(if we have $(\lambda+1) < n$) and (if we have made non-deterministic choices on all the n vertices in layer λ) and (if we have $\varphi' \geq \varphi$) and (if we have $\eta' = \eta$)} then we increment λ by 1, re-initialize $\varphi \leftarrow \varphi''$, $\eta \leftarrow \eta''$ and continue the algorithm from step 2.

23. otherwise if we have $(\lambda+1) = n$ then {{(if not all the vertices chosen in layer n is t, that is η'' is not equal to 1) or (if we have φ'' is not equal to k)} then we reject the input and stop. Otherwise if both these conditions are true then we accept the input and stop.}

We have mentioned in Note 1 that the number of accepting computation paths of a NL machine does not change even if it simulates a UL machine in its intermediary stages to verify if some input is in a language $L \in$ UL. It is also easy to observe that the same is true for the FUL function that verifies if the number of paths from s to t is at least φ.

Also it follows from Theorem 2.4 that at any stage of the algorithm we can construct graphs G'_ω and G_ω using $O(\log |x|)$ space. We keep track of only a constant number of variables all of which take non-negative integer values. Also 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 $|x|$. As a result we get a NL machine that executes the above algorithm and ends in an accepting state on an input x if and only if it chooses k distinct directed paths from s to t in G. As mentioned in Remark 1 since $f(x)$ is equal to the number of directed paths from s to t in G it follows that $\binom{f(x)}{|g(x)|} \in \#\text{L}$. ∎

3.2 #L functions are sufficient for ModL

Functions in GapL can take negative integer values on a given input and many interesting results related to the sign of a GapL function are known. The most obvious is that the class of functions in #L is properly contained in GapL. Some other results are also shown in [AO96, Proposition 19] and [HT05, Theorem 4.5].

Buntrock *et.al.* in [BDHM92, Proposition 9] show that deciding if a square integer matrix has its determinant not congruent to zero modulo k, where $k \geq 2$ is a positive integer, is logspace many-one complete for $\mathrm{Mod}_k\mathrm{L}$. As a result we can replace the #L function by the determinant in defining $\mathrm{Mod}_k\mathrm{L}$. For ModL we obtain a similar result that shows that we can replace the GapL function by a #L function for deciding membership of any input in a language $L \in \mathrm{ModL}$.

Lemma 3.5. *Let $L \in \mathrm{ModL}$. Then there exists a function $f \in \#\mathrm{L}$ and a function $g \in \mathrm{FL}$ 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 $f(x) \not\equiv 0 (\mathrm{mod}\, p^e)$.*

Proof. Let $L \in \mathrm{ModL}$ be witnessed by functions $f' \in \mathrm{GapL}$ and $g \in \mathrm{FL}$ as in Definition 2.6. We know that GapL is the closure of #L under subtraction [AO96, Proposition 2]. In other words, given $f' \in \mathrm{GapL}$ there exists $f_1, f_2 \in \#\mathrm{L}$ 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 #L is closed under multiplication by a FL function that outputs a positive integer and under addition [BDHM92, Lemma 2(i)], we have $f(x) \in \#\mathrm{L}$. Moreover on a given input x, we have $f'(x) \not\equiv 0(\mathrm{mod}\, p^e)$ if and only if $f'(x) = f_1(x) - f_2(x) \not\equiv 0(\mathrm{mod}\, p^e)$ if and only if $f(x) = f_1(x) + (p^e - 1)f_2(x) \not\equiv 0(\mathrm{mod}\, p^e)$. As a result we can replace the GapL function f' by the #L function f to decide if any given input string x is in L. ∎

3.3 Prime modulus is sufficient for ModL

We now prove our characterization of ModL. The proof of this result is the same as that used to prove $\mathrm{Mod}_{p^e}\mathrm{P} = \mathrm{Mod}_p\mathrm{P}$ in [BG92, Theorem 7.2]. It should be noted that if $f \in \#\mathrm{P}$ and $g \in \mathrm{FP}$ such that $g(x)$ is a positive integer in unary that depends on the input string x then the function $\binom{f(x)}{|g(x)|} \in \#\mathrm{P}$ [FFK94]. Also [BG92, Property 2.7] show the same property to be true when $g(x)$ is a positive integer that is polynomially bounded in $|x|$ and $g \in \mathrm{UPF}$. In Theorem 3.4 we also showed the same property to be true in the logspace setting but it required the unproven assumption that NL = UL. As a consequence we also need to assume that NL = UL to prove our characterization of ModL.

Theorem 3.6. *Assume that* $\mathrm{NL} = \mathrm{UL}$ *and let $L \in \mathrm{ModL}$. Then there exists a function $f \in \#\mathrm{L}$ and a function $g \in \mathrm{FL}$ such that on any input string x,*

- *$g(x) = 0^p$ for some prime p, 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 $L \in \text{ModL}$. It follows from Lemma 3.5 that there exists $f' \in \#\text{L}$ and $g' \in \text{FL}$ 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 $f'' \in \#\text{L}$ 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] 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 #L function in Lemma 3.5. 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 $f_i \in \#\text{L}$ 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 20]. 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 $\binom{f'(x)}{p^{e-1}} \not\equiv 0 (\text{mod}\, p)\}$.

Now define $f_e(x) = f_{e-1}(x) + \binom{f'(x)}{p^{e-1}}$. Using Theorem 3.4 and [BDHM92, Lemma 2(i)] it follows that $f_e \in \#\text{L}$. Moreover on an input string x we have $f'(x) \not\equiv 0 (\text{mod}\, p^e)$ if and only if $f_e(x) \not\equiv 0 (\text{mod}\, p)$. As a result defining $f'' = f_e$ we complete the proof. ■

Corollary 3.7. *Assume that* $\text{NL} = \text{UL}$. *Then* ModL *is closed under complement.*

Proof. Let $L \in \text{ModL}$. Then by Theorem 3.6 there exists $f \in \#\text{L}$ and $g \in \text{FL}$ 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 [BDHM92, Lemma 2(i)] that $h(x) \in \#\text{L}$. Using Fermat's Little Theorem [BG92] 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 $\overline{L} \in \text{ModL}$ or that ModL is closed under complement. ■

4 Discussion

In an earlier submission to the STACS 2009 conference related to the complexity class ModL I had given a proof of Theorem 3.6, but without the assumption that NL = UL. Moreover the result had $f \in$ GapL and in the proof, the FL function g was defined to output a prime in the unary representation that is greater than $p^e = |g'(x)|$ instead of the prime p itself as shown in this paper. However referees of the STACS 2009 conference pointed out many errors in the proof that I had given. In this paper I have managed to partially correct this statement under the assumption that NL = UL and since #L $\subset$ GapL. However as in the proof of $\text{Mod}_{p^e}\text{P} = \text{Mod}_p\text{P}$ [BG92] and $\text{Mod}_{p^e}\text{L} = \text{Mod}_p\text{L}$ [BDHM92], the FL function g that we define in Theorem 3.6 only outputs the prime p that divides $p^e = |g'(x)|$.

4.1 Open problems

The most obvious open problem that we obtain from the results shown above is to prove NL = UL so that we obtain the characterization of ModL shown in Theorem 3.6 unconditionally. Using this characterization we have also shown that ModL is closed under complement. However note that even under the assumption that NL = UL we do not know any other closure properties of ModL or its relation to logspace counting classes such as $\text{C}_=\text{L}$ and Mod_kL where $k > 2$ is a composite number having more than one prime divisor.

The most interesting result regarding ModL that does not depend on any unproven assumption is $\text{L}^{\text{ModL}} = \text{L}^{\text{GapL}}$ [AV04][Vij08, Lemma 3.2.2][AV, Lemma 3.2]. We do not know if ModL is closed under logspace Turing reductions. However note that irrespective of whether we assume NL = UL, proving that ModL is closed under logspace Turing reductions would show that if $f \in$ GapL then the language $L_f = \{(x, i, b) | f(x) = y_1 y_2 \cdots y_{p(|x|)}$ and $y_i = b$, where $p(|x|)$ is a polynomial, $1 \leq i \leq p(|x|)$ and $b \in \{0, 1\}\}$ is in ModL. In other words the closure of ModL under logspace Turing reductions will show that ModL captures formulation of GapL as a language. We find these questions interesting and leave them open.

Acknowledgments

I am grateful to Johannes Köbler for valuable discussions. I am also grateful to the anonymous referees of the STACS 2009 conference for pointing out many errors in the proofs of the results on the complexity class ModL in an earlier submission.

References

[AO96] Eric Allender and Mitsunori Ogihara. Relationships among PL, #L and the Determinant. *RAIRO - Theoretical Informatics and Applications*, 30: 1-21, 1996.

[ABO99] Eric Allender, Robert Beals and Mitsunori Ogihara. The complexity of matrix rank and feasible systems of linear equations. *Computational Complexity*, 8(2): 99-126, 1999.

[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, 1999.

[AV04] V. Arvind and T.C. Vijayaraghavan. Abelian Permutation Group Problems and Logspace Counting Classes. *CCC '04: Proceedings of the 19th IEEE Annual Conference on Computational Complexity*, pages 204-214, 2004.

[AHT07] Manindra Agrawal, Thanh Minh Hoang and Thomas Thierauf. The Polynomially Bounded Perfect Matching Problem Is in NC^2. In *STACS '07: Proceedings of the 24th Annual Symposium on Theoretical Aspects of Computer Science*, pages 489-499, 2007. Also available as ECCC Report TR 06-129.

[AV] V. Arvind and T.C. Vijayaraghavan. Classifying Problems on Linear Congruences and Abelian Permutation Groups using Logspace Counting Classes. Accepted in the journal *Computational Complexity* (to appear).

[BDHM92] Gerhard Buntrock, Carsten Damm, Ulrich Hertrampf and Christoph Meinel. Structure and Importance of Logspace-MOD Classes. *Mathematical Systems Theory*, 25(3): 223-237, 1992.

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

[CDL01] Andrew Chiu, George Davida and Bruce Litow. Division in logspace-uniform NC^1. *RAIRO - Theoretical Informatics and Applications*, 35(3): 259-276, 2001.

[FFK94] Stephen Fenner, Lance Fortnow and Stuart Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences*, 48(1): 116-148, 1994.

[HAB02] William Hesse, Eric Allender and David A. Mix Barrington. Uniform Constant-Depth Threshold Circuits for Division and Iterated Multiplication. *Journal of Computer and System Sciences*, 65(4): 695-716, 2002.

[HT03] Thanh Minh Hoang and Thomas Thierauf. The Complexity of the Characteristic and the Minimal Polynomial. *Theoretical Computer Science*, 295(1-3): 205-222, 2003.

[HT05] Thanh Minh Hoang and Thomas Thierauf. The Complexity of the Inertia and some Closure Properties of GapL. In *CCC '05: Proceedings of the 20th Annual IEEE Conference on Computational Complexity*, pages 28-37, 2005.

[KT96] Johannes Köbler and Seinosuke Toda. On the Power of Generalized MOD-Classes. *Mathematical Systems Theory*, 29(1): 33-46, 1996.

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

[Tod91] Seinosuke Toda. *Counting problems computationally equivalent to computing the determinant.* Technical Report CSIM 91-07, Department of Computer Science, University of Electro-Communications, Tokyo, Japan, May 1991.

[Vij08] T.C. Vijayaraghavan. *Classifying certain Algebraic Problems using Logspace Counting Classes.* Ph.D Thesis, The Institute of Mathematical Sciences, Homi Bhabha National Institute, India, December 2008.