CLASSIFYING CERTAIN ALGEBRAIC PROBLEMS USING LOGSPACE COUNTING CLASSES

By

T.C. Vijayaraghavan

The Institute of Mathematical Sciences, Chennai.

A thesis submitted to the
Board of Studies in Mathematical Sciences

In partial fulfillment of the requirements

For the Degree of

DOCTOR OF PHILOSOPHY

of

HOMI BHABHA NATIONAL INSTITUTE

November 2008

Homi Bhabha National Institute

Recommendations of the Viva Voce Board

As members of the Viva Voce Board, we recommend that the dissertation prepared by **T.C. Vijayaraghavan** entitled "Classifying certain Algebraic Problems using Logspace Counting Classes" may be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

---------- **Date :**
Chairman and Convener: V. Arvind

---------- **Date :**
Member : Satya Lokam

---------- **Date :**
Member : Meena Mahajan

---------- **Date :**
Member : K.V. Subrahmanyam

---------- **Date :**
Member : C.R. Subramanian

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copies of the dissertation to HBNI.

I hereby certify that I have read this dissertation prepared under my direction and recommend that it may be accepted as fulfilling the dissertation requirement.

---------- **Date :**
Guide : V. Arvind

DECLARATION

I, hereby declare that the investigation presented in the thesis has been carried out by me. The work is original and the work has not been submitted earlier as a whole or in part for a degree/diploma at this or any other Institution or University.

T.C. Vijayaraghavan

ACKNOWLEDGEMENTS

I am grateful to Prof. V. Arvind for the sustained encouragement and push he had given me ever since I joined The Institute of Mathematical Sciences, Chennai for the Ph.D program in August 2001. The series of introductory lectures on Computational Complexity given by him during the second semester was the primary motivation to focus on Complexity Theory in my doctoral program, which eventually set me apart. Together we had several discussions on a wide variety of topics from Pseudorandomness, Derandomization, Applications of Algebraic Techniques in Complexity Theory and finally Space Bounded Computation, the topic on which results of this thesis is based.

I am also thankful to Eric Allender and Pierre McKenzie for email discussions and their valuable comments on my work, particularly on the results of Chapters 3 and 4. I am also grateful to Jacobo Torán and Eric Allender for kindly agreeing to present the results in [AV04, AV05] at the CCC 2004 and STACS 2005 conferences which I could not attend due to unforeseen circumstances.

I had the opportunity to work with Piyush Kurur, while he was doing his Ph.D here, and we obtained a number of interesting results connected to permutation groups and the Graph Isomorphism Problem. Working with Piyush taught me more about the academic and the social life of a student in a research institute, which was truly a learning experience.

I am also grateful to the faculty members Kamal Lodaya, Meena Mahajan, R. Ramanujam, Venkatesh Raman and C.R. Subramanian of the Theoretical Computer Science group in Matscience, and Madhavan Mukund, K. Narayan Kumar and K.V. Subrahmanyam of the Computer Science group in Chennai Mathematical Institute for the lectures they delivered during my initial days as a student at Matscience. These lectures were highly instrumental to me for getting a better understanding of the basic ideas in Theoretical Computer Science.

I am grateful to the Director, Matscience and other administrative staff for creating a conducive environment that helped me pursue research. I am also grateful to the Director, Chennai Mathematical Institute and other faculty members of the Computer Science group for offering me a post-doctoral position and creating a suitable environment that helped me complete writing this thesis.

I am thankful to S.P. Suresh and B. Meenakshi for sharing their experiences and views on several aspects I discussed with them during the early days of my stay in Matscience. I am also thankful to fellow students M.N. Jayalal Sarma, Nutan Limaye, Vinu Lukose, Rahul Muthu, N. Narayanan and C. Prakash for being with me during my stint at Matscience. Special thanks is due to Naru who almost on all occasions helped

me when I faced difficulties in using Linux and Latex.

Above all, I would like to thank my parents, without whose support and encouragement this work would have been impossible. They stood by me during turbulent times when I lacked confidence, and ensured that I eventually completed this work successfully.

Abstract

In this thesis we obtain results showing a finer classification of the complexity of several algebraic problems that have efficient polynomial time algorithms. The problems we consider are based on Group Theory and Linear Algebra.

One of the main problems we study in this thesis is LCON. Here we are given a matrix $A \in \mathbb{Z}^{m \times n}$, a column vector $\mathbf{b} \in \mathbb{Z}^m$ and a positive integer q in terms of its prime factorization $q = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$ where each $p_i^{e_i}$ is given in unary, $1 \leq i \leq k$, as input and the problem is to determine if $A\mathbf{x} = \mathbf{b}$ is a feasible system of linear equations over $\mathbb{Z}_q$. McKenzie and Cook defined this problem in [MC87] and showed that LCON is in NC^3. In this thesis we present a randomized parallel algorithm to solve LCON and place it in $\mathrm{BP{\cdot}NC}^2$. Along the way we also introduce a new logspace counting class called ModL and show that $\mathrm{L}^{\mathrm{ModL}} = \mathrm{L}^{\mathrm{GapL}}$. The $\mathrm{BP{\cdot}NC}^2$ upper bound for LCON also shows LCON is in $\mathrm{L}^{\mathrm{ModL}}/\mathrm{poly}$. Given such a feasible system $(A, \mathbf{b}, q)$ as input we also show that the problem of computing a solution to $A\mathbf{x} = \mathbf{b}$ over $\mathbb{Z}_q$, denoted by LCONX (defined in [MC87]), is in $\mathrm{BP{\cdot}NC}^2$ and in $\mathrm{L}^{\mathrm{ModL}}/\mathrm{poly}$. Some of the well known techniques of Polynomial Identity Testing and the Isolating Lemma are two main ingredients in the above results. Using LCON and LCONX we also show that the problem of computing a basis for the nullspace, denoted by LCONNULL (defined in [MC87]), of a mapping from $\mathbb{Z}_q^m$ to $\mathbb{Z}_q^m$ given in terms of a matrix over $\mathbb{Z}_q$ is also in $\mathrm{BP{\cdot}NC}^2$ and in $\mathrm{L}^{\mathrm{ModL}}/\mathrm{poly}$. The above three problems are also shown to be logspace many-one hard for ModL.

Continuing further we define and study a generalization of LCON: testing feasibility of a system of linear equations over a finite ring R having unit element. We assume that the ring R is given by its addition and multiplication tables (where the additive abelian group $(R, +)$ is given as a direct sum of cyclic subgroups of prime power order). As one of our main results we show that testing feasibility of linear equations over R is also in $\mathrm{L}^{\mathrm{ModL}}/\mathrm{poly}$.

McKenzie and Cook in [MC87] also consider a number of problems on Abelian permutation groups and show them to be NC^1-Turing equivalent to the above three problems on linear congruences. We re-examine these reductions and show that all these problems are in fact logspace Turing equivalent. As a consequence the upper bounds and hardness results obtained for LCON, LCONX and LCONNULL carry over to these permutation group theoretic problems as well.

Using known derandomization techniques we also show that all the problems discussed above are in fact in uniform $\mathrm{L}^{\mathrm{ModL}}$ assuming the existence of a language L in $\mathrm{DSPACE}(n)$ that requires circuits of size at least $2^{\epsilon n}$ for all but finitely many n, where $\epsilon > 0$ is a constant.

We then consider the Orbit problem studied by Kannan and Lipton in [KL86]. Given

$A \in \mathbb{Q}^{n \times n}$ and $\mathbf{x}, \mathbf{y} \in \mathbb{Q}^n$ the problem is to check if there exists a non negative integer i such that $A^i\mathbf{x} = \mathbf{y}$. We analyze the polynomial time algorithm given in [KL86] and place this problem in the GapL hierarchy. The problem is also shown to be logspace many-one hard for $\mathrm{C_{=}L}$.

We also consider the matroid intersection problem for linearly representable matroids. Given linear representations of matroids $M_1 = (S, \mathcal{I}_1)$ and $M_2 = (S, \mathcal{I}_2)$ as input the matroid intersection problem is to find an independent set of maximum cardinality in both M_1 and M_2. Its decision version is then to check if there is an independent set of size at least k in $\mathcal{I}$, where k is given as part of the input. We consider a promise version of the above problem denoted by LINMATINTpoly. Here we assume the number of independent sets in the intersection of M_1 and M_2 is bounded by a polynomial in the input size and show that it is in $\mathrm{L}^{\mathrm{GapL}}$. This problem is also shown to be logspace many-one hard for co-$\mathrm{C_{=}L}$. We also place the general linear matroid intersection problem in nonuniform $\mathrm{L}^{\mathrm{GapL}}$. We then consider the problem of checking if two linear representations M_1 and M_2 over $\mathbb{Q}$ represent the same matroid, denoted by ECLR. The question of whether there is a polynomial time algorithm for this problem is left open.

Finally we examine the complexity of problems on groups given by their Cayley table as input. We show that many of these problems such as testing whether the input group is simple, nilpotent, solvable and computing normal closure, centralizer and so on are all logspace computable.

Contents

List of Figures

List of Tables

1 Introduction

Computational Complexity forms an integral part of Theoretical Computer Science which deals with studying the intrinsic difficulty of solving a computational problem. Firstly, to achieve this goal one needs to introduce rigorous mathematical notions of computational models on which the problem is to be solved and parameters related to the model that should be taken into account to explain if the problem is efficiently solvable. The parent branch of Computational Complexity, namely Computability Theory, precisely provides such notions and thereby stands as a foundation upon which Complexity Theory has developed. Fundamental contributions to Computability Theory dates back to the work of Alonso Church, Kurt Gödel, Emil Post, Alan Turing and many others, who have provided the mathematical framework that is well suited to study the complexity of computational problems that are considered.

1.1 Computability Theory

The ideas brought forth from Computability Theory ensure that the difficulty of solving a computational problem or deciding the truth of a mathematical statement which is encoded in a suitable form over an alphabet Σ is independent of the model of computation that is considered. Relevant to the results to be presented in this thesis the most commonly used model of computation is the *Turing machine*.

The input instances of a computational problem are assumed to be encoded as strings in Σ^*, for a finite alphabet Σ. A *decision problem* is a computational problem for which the output is either a "yes" or a "no". We say that a decision problem is *decidable* if there is a Turing machine that halts on all inputs with the correct output. While there are numerous examples of decidable problems, a classic example of an undecidable decision problem is the *Halting Problem*: given a Turing machine code M and an input x, the problem is to decide if M will halt on input x. In a very broad sense we can say that in

Computability Theory we are mainly interested in studying such limitations of computing.

The goal of Complexity Theory is to consider decidable problems and determine how efficiently such problems can be solved in a reasonable model of computation. The subject is both concerned with the amount of resources that are necessary and the amount of resources that are sufficient to solve a problem.

Standard parameters of interest are the *time taken* and the *space used* by a Turing machine. We say that t is the time taken and s is the space used by a Turing machine M to decide if an input x is in L if M requires t steps and uses s tape cells to halt in the accepting or the rejecting state upon receiving input x.

1.1.1 Nondeterminism, Randomness and Oracles

In this subsection and in the next section of this chapter, we introduce fundamental concepts required to present our results. These notions are well known and discussed in detail in standard texts such as [BDG88, BDG91, Pap94, Sip01]. We refer to these when further clarifications are needed.

Turing machines described so far do computations in a deterministic manner. A deterministic Turing machine starting from a particular configuration switches to another configuration in fixed and predefined manner, depending on the contents of the cell scanned by the tape head and state of the Turing machine. Researchers have also studied another notion called *nondeterminism* which allows the Turing machine to move to one among several configurations from the current configuration based on the input symbol scanned and the state of the Turing machine. Such Turing machines are called *nondeterministic Turing machines*. From the point of view of computability, nondeterminism does not add to the power of Turing machines. In other words, every nondeterministic Turing machine can be simulated by a deterministic Turing machine, of course with a considerable overhead on the time taken and space used to solve the computational problem.

However when we consider resource-bounded Turing machines, nondeterminism seems to be more powerful than determinism. A standard example of this is to determine if a propositional formula ϕ is satisfiable. It is easy to define a nondeterministic Turing machine that accepts satisfiable formulas in polynomial time: the machine will nondeterministically choose boolean values for the variables in ϕ, substitute the values, evaluate ϕ, and accept if and only if it evaluates to true. It should be noted that there is so far no deterministic Turing machine running in time polynomial in the size of ϕ, that determines if ϕ is satisfiable. It is generally believed that no such deterministic procedure exists and proving such a non-existence is also known as the P vs NP problem (we elaborate on what P and NP mean in Section 1.2).

In the model, one can also consider *randomness* instead of nondeterminism. That is, a Turing machine instead of making nondeterministic moves, can make the next move based on the outcome of an unbiased coin toss. Turing machines that are thus equipped are called *randomized Turing machines*. In this case, apart from the time taken and the space used by the Turing machine, the number of random bits used and the probability of obtaining the correct output for any given input are also used as possible parameters to analyse the performance of an algorithm on a given input.

We also have the notion of *oracle Turing machines*. Here the machine has an extra tape called the *oracle tape* which is used to decide in one time step if some arbitrary string written in it is in some pre-specified set, called the *oracle*. Apart from the oracle tape, the oracle Turing machine also has three special states q_{QUERY}, q_{YES} and q_{NO}. When the Turing machine enters the q_{QUERY} state it writes a string x on the oracle tape. In the next time step the oracle Turing machine switches to q_{YES} or q_{NO} depending on whether x is in the oracle set. As the machine switches to q_{YES} or q_{NO}, the contents of the oracle tape get instantly erased. We say that a language A is accepted by a Turing machine relative to oracle B, if there is an oracle Turing machine with B as the oracle set accepting A. An oracle Turing machine can well be either deterministic or nondeterministic or randomized. Also it is easy to observe that a deterministic or nondeterministic or randomized Turing machine without oracle can be viewed as an oracle Turing machine wherein the oracle can be taken to be the empty set. The concepts of nondeterminism, randomness and oracle Turing machines are idealized notions that help us in understanding the nature of computation and difficulty of the problem being studied.

Since most of the standard operations such as keeping track of variables, updating them while a computation is performed, executing a set of instructions several times (that is looping), branching based on the truth value of a condition, deciding the next move nondeterministically or randomly or using oracles can all be described by giving a suitable definition of the Turing machine, we give only high-level descriptions of Turing machines by presenting them as algorithms or procedures.

A decision problem is usually identified with the language $L \subseteq \Sigma^*$ of its "yes" instances, where inputs to the are strings over alphabet Σ. For example, $SAT = \{\phi \mid \phi$ is a satisfiable propositional formula$\}$ is the language containing all satisfiable propositional formulas encoded over some alphabet Σ.

1.2 Complexity Classes and Reductions

A *complexity class* is a class of languages accepted by Turing machines (or some other model of computation) with suitable resource bound restrictions placed on them. We

measure resource bounds as a function of the input size. We now define some of the standard complexity classes that are required to present our results.

Definition 1.2.1. *1. Let Σ be a finite alphabet. We define* P *to be the complexity class containing all languages $A \subseteq \Sigma^*$ that are accepted by a deterministic algorithm running in time polynomial in the size of the given input.*

2. Let Σ be a finite alphabet. We define NP *to be the complexity class containing all languages $A \subseteq \Sigma^*$ that are accepted by a nondeterministic algorithm running in time polynomial in the size of the given input.*

We can also define a complexity class based on the amount of space used by an algorithm accepting a language A.

Definition 1.2.2. *1. Let Σ be a finite alphabet. We define* L *to be the complexity class containing all languages $A \subseteq \Sigma^*$ that are accepted by a deterministic algorithm using space at most $O(\log n)$, where n is the size of the given input.*

2. Let Σ be a finite alphabet. We define NL *to be the complexity class containing all languages $A \subseteq \Sigma^*$ that are accepted by a nondeterministic algorithm using space at most $O(\log n)$, where n is the size of the given input.*

It is easy to observe that $\mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{P} \subseteq \mathrm{NP}$.

Definition 1.2.3. *Let $\mathcal{C}$ be a complexity class. Then $co\text{-}\mathcal{C} = \{\overline{L} | L \in \mathcal{C}\}$ is the complexity class containing the complement of all languages $L \in \mathcal{C}$.*

A fundamental notion in Complexity Theory (inherited from Computability Theory) that enables us to compare the relative difficulty of two decision problems is that of a *reduction.*

Definition 1.2.4. *Let Σ be a finite alphabet. A* many-one reduction *from a language $A \subseteq \Sigma^*$ to another language $B \subseteq \Sigma^*$, is a total computable function $f : \Sigma^* \longrightarrow \Sigma^*$ such that, $x \in A$ if and only if $f(x) \in B$. We then say that A is many-one reducible to B, and denote it by $A \leq_m B$.*

So if $A \leq_m B$ and B is decidable, then to check if some input $x \in A$, we can compute $f(x)$ on input x and check if $f(x) \in B$ using a procedure that accepts B. Similar to the many-one reduction we define *Turing reductions.*

Definition 1.2.5. *Let Σ be a finite alphabet. A* Turing reduction *from a language $A \subseteq \Sigma^*$ to another language $B \subseteq \Sigma^*$ is an oracle Turing machine M that accepts A using B as an oracle and M halts on all inputs. We then say that A is Turing reducible to B.*

The above notions are from Computability Theory. In Complexity Theory we tend to place time and space bounds in computing the function f (for many-one reductions) or the Turing machine M (in case of Turing reductions).

Definition 1.2.6. *Let $L_1, L_2 \subseteq \Sigma^*$. A* Karp reduction *is a polynomial time many-one reduction from L_1 to L_2 and is denoted by $L_1 \leq_m^P L_2$.*

Definition 1.2.7. *Let $L_1, L_2 \subseteq \Sigma^*$. A* Cook reduction *is a polynomial-time Turing reduction from L_1 to L_2 and is denoted by $L_1 \leq_T^P L_2$.*

These standard notions are tailored to the P vs NP setting.

In this thesis, since we are concerned with classifying problems within the class P, we will be mainly interested in deterministic many-one and Turing reductions that are *logspace computable*.

In general, we observe that a reduction procedure reducing a language A to another language B is useful if and only if the amount of resources it uses, such as time or space, is strictly less than the amount of resources used by any procedure accepting A. Thus, different reducibility notions are suitable for different complexity classes.

Having defined complexity classes and reductions, we move onto defining when a problem is *hard* for a complexity class $\mathcal{C}$.

Definition 1.2.8. *Let Σ be a finite alphabet. We say that a language $L \subseteq \Sigma^*$ is* hard for $\mathcal{C}$ under many-one reductions *(or* many-one hard for $\mathcal{C}$*), if every language in $\mathcal{C}$ reduces to L by a many-one reduction. Moreover if $L \in \mathcal{C}$ then we say that L is* many-one complete for $\mathcal{C}$.

If the many-one reduction in the above definition were computable in time $p(n)$ for some polynomial $p(n)$, or using at most $O(\log n)$ space, where n is the size of the given input, then we say that L is *polynomial time many-one hard for* $\mathcal{C}$, or L is *logspace many-one hard for* $\mathcal{C}$ respectively.

Definition 1.2.9. *Let Σ be a finite alphabet. We say that a language $L \subseteq \Sigma^*$ is* hard for $\mathcal{C}$ under Turing reductions *(or* Turing hard for $\mathcal{C}$*), if every language in $\mathcal{C}$ reduces to L by a Turing reduction. Moreover if $L \in \mathcal{C}$ then we say that L is* Turing complete for $\mathcal{C}$.

If the Turing reduction in the above definition were computable in time $p(n)$ for some polynomial $p(n)$, or using at most $O(\log n)$ space, where n is the size of the given input, then we say that L is *polynomial time Turing hard for* $\mathcal{C}$, or L is *logspace Turing hard for* $\mathcal{C}$ respectively.

1.3 Complexity Classes contained in P

In Complexity Theory, it has been long argued that languages accepted by Turing machines running in time polynomial in the size of the input capture the notion of problems that can be efficiently solved. Such languages constitute the complexity class P and are often called *tractable*. If L is a language for which there are no efficient algorithm accepting it except the brute force methods which could consume unreasonable amount of resources, then L is referred to as *intractable*. A number of references exist that discuss tractability, intractability and when a language or function is efficiently computable. We refer to standard texts such as [Pap94, Sip01] for more along these lines.

Even though languages in P have efficient algorithms, interest in classes contained in P arose due to the need for a finer classification of the complexity of problems in P. The problems we consider in this thesis already have such efficient polynomial time algorithms. To achieve a finer classification of complexity, we firstly note that any Turing machine needs at least linear time to read the input provided and hence reducing the amount of time taken to be sub linear may not be possible. Natural questions arise, such as how the complexity of the problem studied changes if the parameter used to measure the efficiency is chosen to be different from the time taken by the Turing machine to solve the problem. One of the first and standard examples illustrating such a reduction in complexity when examined from a different setting is to check if there is a path between vertices s and t in a directed graph. It has been shown that the directed $s-t$ connectivity problem is complete for NL, the class of languages accepted by nondeterministic Turing machines using space at most $O(\log n)$. A more recent result is that the st-connectivity problem for undirected graphs is complete for L [Rei05]

Turing machine model discussed so far, its nondeterministic or randomized variants basically perform computations in a sequential manner. That is, the machine is restricted to performing no more than a pre-specified number of operations in each time step. Alternately, a different notion called *Parallel Computation* has been developed over the years in which the underlying computational model has several smaller units, called processors, each of which can perform computations concurrently. Several models of computation are known to implement parallel computation. One such commonly referred to model, which is relevant to the results to be presented is the *Boolean circuit* (we elaborate more on Boolean circuits as we proceed). It turns out that the notion of parallel computation can have potential advantages to solve some particular class of problems, from which we might observe a reduction in complexity for those problems.

Clearly when we shift our focus from using Turing machines to other computational models, parameters considered to measure the efficiency of solving the problem also

change. Another pleasing fact is that most of these computational models implementing parallel computation can be efficiently simulated by Turing machines itself.

1.3.1 Parallel Computation and Boolean Circuits

In *Parallel Computation*, the processors are the basic units that perform necessary computations. They are provided with a memory which can be used to compute values as needed, or to store the results of computation performed. Here, access to such memory registers for reading and writing contents is synchronised between the processors in such a way that no conflict occurs. As mentioned above, several models of computation such as *Parallel Random Access Machines, Boolean circuits* have been proposed that put the above idea into practice. We refer to [Pap94, Chapter 15] for a more detailed exposition on parallel computation.

With relevance to the results of this thesis we mainly take up Boolean circuits as our model for describing parallel computation. A *Boolean circuit* is a simple directed acyclic graph $C = (V, E)$ with a set of vertices V called *gates*, and a set of directed edges E. In any edge $(i, j) \in E$, we call i as the tail and j as the head of the edge. Any gate i can be the tail or the head of arbitrarily many number of edges. The number of edges for which i is the head is the *fan-in* of i, while the number of edges for which i is the tail is the *fan-out* of i. Those gates having fan-in zero are called the input gates, while nodes having fan-out zero are called the output gates of C. Input to the circuit is essentially a string from $\{0,1\}^*$ and it is fed through the input gates. Apart from the input gates, any other gate in C is defined to perform one of the operations: $\vee$ (Boolean OR), $\wedge$ (Boolean AND), and $\neg$ (Boolean negation). Any $\neg$ gate is assumed to have fan-in one.

When we say that C is acyclic, we mean C does not contain any directed cycle while the underlying undirected graph can have cycles. Since there is no directed cycle, we can suitably number the gates in C such that if (i, j) is an edge in C, we have $i < j$. We can stratify the gates in a circuit into different levels based on the longest distance of any gate from an input gate. We assume that input gates are at level 0. Any gate is at level 1 if the length of a directed path from any input gate to gate j is 1. Similarly we say that gate k is at level l if the length of the longest directed path from an input gate to gate k is l. The length of the longest directed path of any gate in C from any input gate is called the *depth of the circuit* C. The number of gates in the circuit C is known as the *size of* C. It is easy to note that the number of edges in any such circuit C is small (at most quadratic) in the number of gates in the circuit C.

Assume that the circuit C_n has n input gates. Upon receiving an input of length n from $\{0,1\}^n$, all the gates in level 1 of C_n compute their corresponding Boolean function ($\vee, \wedge$ or $\neg$) in parallel based on the values at level 0 in a single time step. The values

computed in level 1 and level 0 are then passed onto gates in level 2. All the gates in level 2 carry out necessary computation in parallel within a single time step and the values obtained from levels $0, 1, 2$ are passed onto gates in level 3. This procedure continues until the circuit computes the output after which the circuit stops.

From the above description of a Boolean circuit, we infer the following. Since the number of input gates (or in general number of gates) is fixed in a circuit C, unlike Turing machines, C can decide the membership, or compute values of functions for inputs of fixed length only. Thus if x and y were inputs whose lengths are different, then we need to use different circuits that accept inputs of length $|x|$ and $|y|$ to decide their membership in any language. Thus for any language L containing infinitely many strings, we need an infinite family of circuits $C = (C_1, C_2, \ldots)$ accepting L, where C_i denotes a circuit that decides if some string of length i belongs to L.

Note that we can define circuit families that accept undecidable languages also. To observe this, consider an undecidable language $L \in \{0,1\}^\star$ and define $L_1 = \{1^n | n = 1x \in L\}$. Clearly strings in L_1 are unary representations of strings in L and no two strings in L_1 have the same length. We can easily define a family of linear-size circuits accepting L_1 using $\wedge$ and $\neg$ gates. These difficulties prompt us to explore the feasibility of constructing a circuit that decides the membership of strings of a particular length. A notion that captures such a feasibility is called *uniformity*. For instance, we say that a circuit family accepting strings of length n of a language L is L-*uniform*, if there is an algorithm that uses at most $O(\log n)$ space which when given 1^n as input, outputs a circuit C_n that decides if any input string of length n is in L. Similarly we can define languages accepted by circuit families that are P-uniform (polynomial time uniform) and so on.

From the above description of Boolean circuits it is natural to have fan-in and fan-out of gates in the circuits, size and depth of circuits, or the extent of uniformity as possible parameters to judge the difficulty of solving a problem. Several complexity classes have been defined based on these parameters. We recall some of them that are required to present our results. The definitions given below are standard, and well known. We refer to [Vol99] when further details and clarifications are needed.

Definition 1.3.1. *1. Let $\Sigma = \{0,1\}$ be the finite alphabet. For $k \geq 0$, we define NC^k to be the complexity class of all functions $f : \Sigma^* \to \Sigma^*$ computed by a logspace uniform Boolean circuit family $\{C_n\}_{n\geq 1}$ wherein C_n takes inputs of length n, with its size polynomial in n, and its depth being $O(\log^k n)$. Here each gate in C_n is assumed to have fan-in 2. The complexity class NC is defined to be $\cup_{k\geq 0}\mathrm{NC}^k$.*

2. Let $\Sigma = \{0,1\}$ be the finite alphabet. For $k \geq 0$, we define AC^k to be the complexity class of all functions $f : \Sigma^* \to \Sigma^*$ computed by a logspace uniform Boolean circuit

family $\{C_n\}_{n\geq 1}$ *wherein* C_n *takes inputs of length* n*, with its size polynomial in* n*, and its depth being* $O(\log^k n)$*. For any* $n \geq 1$*, we assume that gates in* C_n *have unbounded fan-in. The complexity class* AC *is defined to be* $\cup_{k\geq 0}\mathrm{AC}^k$*.*

The following relationship between is well known between the complexity classes discussed so far [Vol99]: $\mathrm{NC}^0 \subseteq \mathrm{AC}^0 \subset \mathrm{NC}^1 \subseteq \mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{AC}^1 \subseteq \cdots \subseteq \mathrm{P}$. In general, for $i \geq 1$, we have $\mathrm{NC}^i \subseteq \mathrm{AC}^i \subseteq \mathrm{NC}^{i+1}$, and hence $\mathrm{NC} = \mathrm{AC} \subseteq \mathrm{P}$.

There are several examples of problems solvable in NC. One important result required here is that computing the determinant of an integer matrix is in NC^2 [Ber84]. Subsequently [Tod91a, Vin91] gave a more exact characterization, showing that computing the determinant of an integer matrix is complete for the complexity class GapL (defined in Section 1.3.2) with respect to logspace many-one reductions.

We can also define *randomized Boolean circuits* as a circuit analogue of randomized algorithms. These are Boolean circuits C which apart from a usual input $x \in \{0,1\}^n$, also take as input a random string $w \in \{0,1\}^m$ picked uniformly at random from $\{0,1\}^m$. The acceptance probability of the randomized circuits is defined as the probability $\Pr_w[C(x,w) = 1]$. Using randomized circuit families of polynomial size and polylogarithmic depth we now define the randomized complexity class $\mathrm{BP{\cdot}NC^k}$.

Definition 1.3.2. *We say that a language* L *is in the complexity class* $\mathrm{BP{\cdot}NC^k}$ *for an integer* $k \geq 0$*, if there is a logspace uniform Boolean circuit family* $\{C_n\}_{n\geq 1}$ *of polynomial size and* $\log^k n$ *depth and constant fan-in circuits such that for* $x \in \Sigma^n$

$$x \in L \text{ implies } \Pr_w[C_n(x,w) = 1] \geq 2/3,$$
$$x \notin L \text{ implies } \Pr_w[C_n(x,w) = 1] \leq 1/3.$$

The complexity class $\mathrm{BP{\cdot}NC}$ *is* $\cup_{k\geq 0}\mathrm{BP{\cdot}NC^k}$*. When the randomized circuit does not err for inputs not in* L *then* L *is said to be in the subclass* RNC*.*

For an integer $k \geq 0$, we say that a function f is computable by a $\mathrm{BP{\cdot}NC^k}$ circuit family, whether each bit of $f(x)$ is computable in $\mathrm{BP{\cdot}NC^k}$.

We also consider circuits that have *oracle gates*. An oracle gate is used to decide if a given input string belongs to some language or to compute the value of some arbitrary function in one time step. There is an additional subtlety in defining constant fan-in, depth bounded oracle circuits as the oracle gates are not of bounded fan-in. The depth contributed by an oracle gate with k inputs is counted as $\log_2 k$. We come across such oracle circuits when we consider NC^1 and AC^0 Turing reductions in Chapters 4 and 5.

1.3.2 Logspace Counting Classes and Algebraic Problems

A complexity class $\mathcal{C}$ is said to be a *counting class* if we can decide the membership of any language L in $\mathcal{C}$ based on the number of rejecting paths of a nondeterministic Turing machine accepting L. As an example, it is easy to see that we can recast NP as a counting class. For any nondeterministic Turing machine M, let $acc_M(x)$, denote the number of accepting paths of M on input x. Then, any language $L \in$ NP if and only if there is a polynomial time bounded nondeterministic Turing machine M, such that any input string $x \in L$ if and only if $acc_M(x) \geq 1$.

Valiant in [Val79] defined the counting class #P to be the set of all functions $f : \Sigma^* \rightarrow \mathbb{N}$, such that there is a polynomial time bounded nondeterministic Turing machine M with $f(x) = acc_M(x)$. Valiant in [Val79] showed that computing the permanent of an integer matrix is #P-complete. There has been an extensive study of several counting classes that are defined based on the number of accepting and rejecting paths of a polynomial time bounded nondeterministic Turing machine. Results about their closure properties under different operations and their relation to other complexity classes are well known, for instance refer [BG92] and [Tod91b].

It is surprising that logspace bounded counting classes have turned out to capture several natural computational problems inside P and added to the rich structure of complexity classes within NC^2. In fact, it is mentioned in [ABO99] that there is no *a priori* reason to expect that space bounded analogs of counting classes such as #P would be interesting to study. However, similar to the result obtained for permanent, [Tod91a, Vin91, Dam91, Val92] have shown that computing the determinant of an integer matrix is complete for the counting class GapL (defined below) under logspace many-one reductions.

Definition 1.3.3. *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.*

Computing the determinant of an integer matrix is a problem that has been well studied for a long time. One of the most well known approaches uses Gaussian elimination to convert the input matrix into an upper triangular matrix, the product of whose diagonal entries equals the determinant of the original matrix. Several other methods exist, and in fact even a combinatorial algorithm that does not involve any division is also known [MV97]. The significance of this problem is profound that logspace counting classes have captured the complexity of a number of linear algebraic problems.

The results that we prove in this thesis are precisely based on such known results and these space-bounded counting classes. Before summarizing our main results, we introduce few other counting classes that are essential to present our results.

Definition 1.3.4. *We define* #L *to be the class of functions* $f : \Sigma^* \to \mathbb{N}$*, 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* $acc_M(x)$ *denotes the number of accepting computation paths of* M *on input* x*.*

Definition 1.3.5. *A language* L *is in* $\mathrm{C_{=}L}$ *if there exists a function* $f \in \mathrm{GapL}$ *such that* $x \in L$ *if and only if* $f(x) = 0$*.*

As an immediate corollary of characterization of the complexity of determinant of integer matrices in terms of GapL, we see that the problem of checking if an integer matrix is singular is complete for $\mathrm{C_{=}L}$. The question of whether $\mathrm{C_{=}L}$ is closed under complement is open.

Definition 1.3.6. *Let* $k \geq 2$ *be an integer. A language* L *is in* $\mathrm{Mod_kL}$ *if there exists a function* $f \in \#\mathrm{L}$ *such that* $x \in L$ *if and only if* $f(x) \not\equiv 0 (mod\ k)$*.*

We also need to define hierarchies that are formed using logspace counting classes. In defining such hierarchies, we need to deal with space bounded nondeterministic oracle Turing machines. In this context, we follow the *Ruzzo-Simon-Tompa oracle access mechanism* [ABO99]. According to this, any nondeterministic oracle Turing machine is allowed to write its queries in the oracle tape in a deterministic manner only. As a consequence, any nondeterministic logspace bounded oracle Turing machine can submit only polynomially many queries to the oracle. Also these queries can be submitted to the oracle in a single step, even before the logspace machine starts performing any computation with the given input.

In some of the definitions we need to have functions as oracles. In such cases we assume that the value of oracle function upon submitting input x is retrieved in a bit by bit manner. In other words, we assume that, length l of the value of the function when given an input of size n is known before hand. By submitting l many queries to the oracle, we finally retrieve the function value in a bit-by-bit manner.

Definition 1.3.7. *Define* $\#\mathrm{LH}_1$ *to be* #L*. For* $i \geq 1$*, define* $\#\mathrm{LH}_{i+1}$ *to be the class of functions* f*, such that for some nondeterministic logspace oracle Turing machine* M *with a function* $g \in \#\mathrm{LH}_i$ *as oracle, we have* $f(x) = acc_M(x)$*. We denote the* #L *hierarchy by* $\#\mathrm{LH} = \cup_{i \geq 0} \#\mathrm{LH}_i$*.*

From the definitions of GapL and #L it is easy to see that GapL is the closure of #L under subtraction. Since the number of computation paths of a logspace bounded nondeterministic Turing machine can be determined for inputs of length n, we can replace the #L oracle in the above definition with GapL oracle instead. In other words, the #L hierarchy defined above coincides with a hierarchy defined similarly in terms of GapL.

Definition 1.3.8. *Define* $\mathrm{C_{=}LH}_1$ *to be* $\mathrm{C_{=}L}$. *For* $i \geq 1$, *define* $\mathrm{C_{=}LH}_{i+1}$ *to be the class of languages* L, *such that for some nondeterministic logspace oracle Turing machine* M *with a language* $L' \in \mathrm{C_{=}LH}_i$ *as oracle, we have* $x \in L$ *if and only if* $acc_M(x) = rej_M(x)$. *We denote the* $\mathrm{C_{=}L}$ *hierarchy by* $\mathrm{C_{=}LH} = \cup_{i \geq 0} \mathrm{C_{=}LH}_i$.

In [AO96] it has been shown that #LH and $\mathrm{C_{=}LH}$ can be defined in terms of AC^0 reduction to #L and $\mathrm{C_{=}L}$ respectively. We first describe circuit-based reductions. For further clarifications we refer to [ABO99].

An oracle circuit is a Boolean circuit which apart from $\vee$ (Boolean OR), $\wedge$ (Boolean AND), and $\neg$ (Boolean negation) is equipped with oracle gates. An oracle gate that computes a function takes in a number of input bits in some fixed order and outputs a number of bits that correspond to the value of the function on that input. Notice that the output of one oracle gate can be fed as the input of another oracle gate which is at a higher level, while we have the Ruzzo-Simon-Tompa oracle access mechanism for nondeterministic oracle Turing machines [ABO99]. We once again recall that according to this any nondeterministic oracle Turing machine is allowed to write its queries in the oracle tape in a deterministic manner only. As a consequence, for instance AC^0 circuits equipped with #L oracle gates accept languages in $\#\mathrm{LH}_2$ (we refer to [AO96] for more on this), even though $\mathrm{AC}^0 \subset \mathrm{L}$. Recall that in describing Boolean circuits, we had assumed the underlying directed graphs to be simple. However, when dealing with oracle circuits, due to the nesting of oracle gates as mentioned above, circuit could lose the property that the underlying graph is simple. That is, there can exist more than one directed edge from a gate to another. But this can result in having exponentially many wires between polynomially many number of gates. Thus for oracle circuits, we assume that the maximum among the number of gates and the number of wires in the circuit to be the size of the circuit. Evaluation of the oracle circuit proceeds similar to that of a Boolean circuit. Gates at level i perform computation in parallel and pass their output to the gates at higher levels.

Definition 1.3.9. *A function* $f : \Sigma^* \to \Sigma^*$ *is logspace uniform* AC^0*-reducible to a function* g *if there exists a logspace uniform* AC^0 *oracle circuit family* $\{C_n\}_{n \geq 1}$ *in which oracle gates compute* g *on a given input, such that for inputs* x *of length* n, C_n *outputs* $f(x)$. *Here we denote* $f \in \mathrm{AC}^0(g)$ *or that* $f \leq_T^{\mathrm{AC}^0} g$.

Definition 1.3.10. *A function f is logspace uniform NC^1-reducible to a function g if there exists a logspace uniform NC^1 oracle circuit family $\{C_n\}_{n\geq 1}$ in which oracle gates compute g on a given input, such that for inputs x of length n, the circuit C_n outputs $f(x)$. Note that any gate in C_n, except oracle gates for g, have fan-in two. For any oracle gate computing g, if there are m inputs, then we add* $\log\ m$ *to the depth of C_n. Here we denote $f \in \mathrm{NC}^1(g)$ or that $f \leq_T^{\mathrm{NC}^1} g$.*

Definition 1.3.11. *A function f is logspace uniform NC^1-Turing equivalent to a function g, if $f \leq_T^{\mathrm{NC}^1} g$, and $g \leq_T^{\mathrm{NC}^1} f$. We denote this by $f \equiv_T^{\mathrm{NC}^1} g$.*

The above definitions regarding NC^1, and AC^0 reducibility carry over to languages as well. In this case, we replace f and g by the characteristic functions of the languages considered. For instance, $\mathrm{AC}^0(\mathrm{C_=L})$ denotes the set of languages logspace uniform AC^0-reducible to the problem of checking if an integer matrix is singular. As mentioned above, in [AO96] it has been shown that $\#\mathrm{LH} = \mathrm{AC}^0(\#\mathrm{L})$ and $\mathrm{C_=LH} = \mathrm{AC}^0(\mathrm{C_=L})$. In [ABO99], it has also been shown that $\mathrm{C_=LH}$ collapses to $\mathrm{L}^{\mathrm{C_=L}}$ and that the collapse would go down to $\mathrm{C_=L}$ if and only if $\mathrm{C_=L}$ is closed under complement.

In Section 1.3.1, we had briefly discussed the notion of uniformity for circuit complexity classes. Along similar lines, we can also define non-uniform complexity classes based on Turing machines. We introduce necessary definitions and terminology regarding non-uniform complexity classes relevant to present our results.

Definition 1.3.12. *Let $A(n)$ be a function mapping positive integers to strings in Σ^*. Then $\mathrm{L}^{\mathrm{GapL}}/\mathrm{poly}$ is the class of languages $L \subseteq \Sigma^*$ accepted by a $\mathrm{L}^{\mathrm{GapL}}$ machine with advice $A(n)$ of length $p(n)$, which is a polynomial in n (the size of the input), such that $x \in L$ if and only if $M(x, A(|x|)) = 1$.*

1.4 Our Contribution

We now summarize the main results to be proved in this thesis. Necessary mathematical background along with definitions required to present our results are covered in detail in Chapter 2.

The first part of our results is concerned with classifying the complexity of a number of problems on abelian permutation groups. In the permutation group theoretic problems we assume that an input group is given by a set of generating permutations, where each generator permutation is in $\mathrm{Sym}(\Omega)$, for the set $\Omega = \{1, \cdots, n\}$, with n given in unary. The problems studied are the following.

AGM: (abelian group membership) Given an abelian permutation group in terms of its generating permutations $G = \langle g_1, \ldots, g_r \rangle$, and another permutation h, determine if

$h \in G$.

AISO: (abelian group isomorphism) Given abelian permutation groups $G = \langle g_1, \ldots, g_r \rangle$ and $H = \langle h_1, \ldots, h_s \rangle$, determine if G are H are isomorphic groups.

AORDER: (abelian group order) Given abelian permutation group $G = \langle g_1, \ldots, g_r \rangle$ compute the *prime factorization* of $o(G)$, the cardinality of G.

AGMX: (search version of AGM) This is the search version of AGM in which, given an abelian permutation group $G = \langle g_1, \ldots, g_r \rangle$ by its generating permutations g_i $(1 \leq i \leq r)$, and a permutation h, we need to determine if $h \in G$ and in such a case, the problem is to find integers t_i where $1 \leq i \leq r$, such that $h = g_1^{t_1} \cdots g_r^{t_r}$.

AINTER: (abelian group intersection) Given abelian permutation groups in terms of their generating permutations, $G = \langle g_1, \ldots, g_r \rangle$ and $H = \langle h_1, \ldots, h_s \rangle$, the problem is to compute a generating set for $G \cap H$.

AGP: (abelian group presentation) Given an abelian group G by generators $g_1, \ldots, g_r$ compute integer vectors $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathbb{Z}^r$ which generate the kernel of the onto homomorphism $\phi : \mathbb{Z}^r \longrightarrow G$ defined by $\phi : (t_1, \ldots, t_r) \mapsto g_1^{t_1} \cdots g_r^{t_r}$.

The above set of problems were previously studied by McKenzie and Cook in [MC87], where it was shown that these problems are in NC^3. [MC87] show this complexity upper bound by first showing NC^1-Turing equivalence between the above mentioned group theoretic problems and certain linear congruence problems to be defined below. As the next step, the linear congruence problems were shown to be in NC^3 from which the results followed. We now state the linear congruence problems.

1. Given a matrix $A \in \mathbb{Z}^{m \times n}$ and a column vector $\mathbf{b} \in \mathbb{Z}^m$, the problem LCON is to determine whether $A\mathbf{x} = \mathbf{b}$ is a feasible system of linear equations over the ring $\mathbb{Z}_q$. Here q is a positive integer given as part of the input in terms of its prime factorization $q = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$, such that each $p_i^{e_i}$ is *tiny* (i.e. given in unary).

2. The search version LCONX of LCON wherein we compute a solution to $A\mathbf{x} = \mathbf{b} (\mathrm{mod}\ q)$ if it exists.

3. Given a matrix $A \in \mathbb{Z}^{m \times n}$, and a positive integer q in terms of its prime factorization $q = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$, such that each $p_i^{e_i}$ is tiny (i.e. given in unary), the problem LCONNULL is to compute a spanning set for the null space of the mapping represented by the matrix A over $\mathbb{Z}_q$. In other words, we want to compute a spanning set for the module $\{\mathbf{x} \in \mathbb{Z}^n | A\mathbf{x} \equiv 0 (\mathrm{mod}\ q), A \in \mathbb{Z}^{m \times n}\}$.

We show that the above mentioned group theoretic and linear algebraic problems are in $\mathrm{BP{\cdot}NC}^2$. To classify them more precisely, we introduce a new logspace counting class called ModL.

Definition 1.4.1. *A set L belongs to the complexity class* ModL *if there is a function $f \in$ GapL and a function $g \in$ FL such that for all strings x,*

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

The complexity class ModL is the logspace analogue of the class ModP introduced by Köbler and Toda in [KT96]. The definition of ModL is such that, it seems more natural to express the above mentioned results on linear algebra and abelian permutation groups in terms of this logspace counting class, rather than in terms of BP·NC^2. Actually, our BP·NC^2 upper bound yields a L$^{\text{ModL}}$/poly algorithm for LCON, where the advice is a randomly picked string. We define L$^{\text{ModL}}$/poly by replacing the GapL oracle with a ModL oracle in Definition 1.3.12. We also obtain a conditional derandomization of this result: assuming the existence of a language in DSPACE(n) that requires Boolean circuits of exponential size, we show that it is possible to derandomize the algorithm and get rid of the random advice string to show that the above mentioned problems are in fact in L$^{\text{ModL}} \subseteq$ NC^2. Along with these results, we also show that these problems are hard for L$^{\text{ModL}}$ under logspace Turing reductions, and thus we have a fairly tight characterization of the complexity of problems mentioned above. The results mentioned here are from [AV04, AV05] (we note here that in [AV05] we show the upper bound of L$^{\text{ModL}}$/poly which corrects our earlier claim in [AV04] that it is in L$^{\text{ModL}}$).

Extending the result obtained for LCON, we consider the problem of testing feasibility of linear equations over a finite ring R. We show that when the input ring R is given explicitly in terms of its addition and multiplication tables (wherein the additive abelian group $(R, +)$ is given as a direct sum of cyclic subgroups of prime power order), the problem of testing if a system of linear equations over R is feasible or not, is also in L$^{\text{ModL}}$/poly.

We next study the complexity of the *orbit problem* defined below.

> Given $A \in \mathbb{Q}^{n \times n}$ and $\mathbf{x}, \mathbf{y} \in \mathbb{Q}^n$, does there exist a non negative integer i such that $A^i\mathbf{x} = \mathbf{y}$.

Kannan and Lipton in [KL86] gave a deterministic polynomial time algorithm for the orbit problem. We observe that some of the underlying operations involved in their algorithm are linear algebraic subroutines such as solving a system of linear equations over $\mathbb{Q}$, computing the rank of a matrix over $\mathbb{Q}$ [ABO99], computing the inverse of a non-singular matrix over $\mathbb{Q}$, and computing the characteristic polynomial and minimal polynomial of matrices over $\mathbb{Q}$ [HT03]. We analyze their algorithm more carefully to place

the orbit problem in the GapL hierarchy GapLH. In the process we show that factoring a univariate polynomial $f \in \mathbb{Q}[x]$ for the special case when the roots of f are all complex roots of unity is in GapLH. We also show that orbit problem is logspace many-one hard for $\mathrm{C_{=}L}$. These results appear in Chapter 5.

In Chapter 6, we study the complexity of matroid intersection of two linearly representable matroids.

A major open problem is whether the perfect matching problem is in deterministic NC, even for bipartite graphs. Under the promise that the input graph has at most polynomially many perfect matchings, Grigoriev and Karpinski [GK87] show deterministic NC algorithms for finding and enumerating all perfect matchings. Recently, Agrawal *et al.* [AHT07] improve the upper bound to $\mathrm{L}^{\mathrm{GapL}}$. We study a similar promise version of linearly representable matroid intersection problem.

> Let $M_1, M_2 \in \mathbb{Q}^{m \times n}$ be two $m \times n$ matrices that linearly represent matroids $M_1 = (S, \mathcal{I}_1)$ and $M_2 = (S, \mathcal{I}_2)$, where $S = [n]$. Additionally, suppose the matroids fulfil the promise that their intersection $\mathcal{I}$ contains at most $p(n)$ many sets of cardinality m, where $p(n)$ is a fixed polynomial. Then, the problem LINMATINTpoly is to determine if $\mathcal{I}$ has a set of size m and if so then compute such a set.

We show that the above problem is in the class $\mathrm{L}^{\mathrm{GapL}}$ and is logspace many-one hard for co-$\mathrm{C_{=}L}$. Additionally, we also observe that the RNC algorithm of [NSV94] for the general linearly representable matroid intersection actually places the problem in $\mathrm{L}^{\mathrm{GapL}}/\mathrm{poly}$ for a random advice string. Furthermore, under a hardness assumption we can obtain a derandomization to get a uniform $\mathrm{L}^{\mathrm{GapL}}$ upper bound for the general linearly representable matroid intersection.

We also consider the problem of checking if two input linear representations M_1 and M_2 over $\mathbb{Q}$ represent the same matroid or not (denoted by ECLR). Any set of elements that form a minimal dependent set (also known as a *circuit*) in one matroid but is independent in the other is a witness to the fact that given two linear representations represent different matroids. We show that the problem of searching for one such witness and deciding whether such a witness exists are in fact polynomial time equivalent. In addition, the problem of counting the number of such witnesses that show the input linear representations represent different matroids is also shown to be #P-complete. We leave the problem of classifying the complexity of ECLR as an open question.

In the final chapter of the thesis we study the complexity of a number of problems on groups input in the form of a Cayley table (that is the multiplication table of the group). The complexity of these problems is first investigated in [BKLM01]. However, in [BKLM01] the authors take a more descriptive complexity approach.

The central observation we use in this chapter is that, given a group G in terms of its Cayley table, elements of a subset $C \subseteq G$ and $h \in G$, the problem of checking if h is in the group generated by the elements in C is decidable in L. This is an easy consequence of Reingold's logspace algorithm for undirected graph connectivity [Rei05]. As a consequence of this result we can show that several problems for groups given as multiplication tables, such as testing nilpotence, solvability, checking if the input group is simple or not, computing the normal closure, centralizer, and so on get classified into L. Finally, we also show a randomized test with constant error probability, to check if an input group G given by a Cayley table is abelian. This test makes constant number of queries to the Cayley table of G. However, we are unable to provide any matching hardness result for these problems.

2

Preliminaries and Notations

In this chapter we provide the necessary mathematical background needed to present our results. We assume familiarity with basic notions such as sets, mappings, binary operations and matrices.

2.1 Group Theory

Let G be a nonempty set of elements, and let $*$ be a binary operation defined on the elements in G. We say that G is a group under the binary operation $*$ (denoted by $(G, *)$) if it satisfies the following conditions.

- G *is closed under* $*$, that is for any two elements $g_1, g_2 \in G$ the element $g_1 * g_2 \in G$,
- G *is associative under* $*$, that is for any $g_1, g_2, g_3 \in G$ we have $(g_1 * g_2) * g_3 = g_1 * (g_2 * g_3)$,
- G *contains the identity element*, that is there exists an element $e \in G$ such that for all $g \in G$, we have $g * e = e * g = g$, and,
- *every element in* G *has inverse*, that is for any $g \in G$ there exists a $h \in G$ such that $g * h = h * g = e$, where e is the identity element in G.

It is easy to observe that the identity element e in any group G is unique. Similarly for any given element $g \in G$ its inverse is also unique.

For notational convenience in a group $(G, *)$ we denote $g * g$ by g^2, the element $g * g * g$ by g^3 and so on. The inverse of an element $g \in G$ would be denoted by g^{-1}.

We say that a group $(G, *)$ is *abelian* if for any $g, h \in G$, we have $g * h = h * g$. A group $(G, *)$ is said to be *cyclic* if there is an element $g \in G$ such that, for every $h \in G$ there is a positive integer m with $g^m = h$. Such an element g is said to be a *generator* of the group cyclic group G. Occasionally we also denote a cyclic group with generator g

by $\langle g \rangle$. Let $(G, *)$ be a group and $H \subseteq G$. We say that H is a subgroup of G, denoted by $H \leq G$, if H is also a group with respect to $*$. Given a set of elements $g_1, g_2, \ldots, g_r \in G$ the *group generated* by $g_1, g_2, \ldots, g_r \in G$ is the smallest subgroup of G containing g_i, for all $1 \leq i \leq r$.

Let $(G, *)$ be a group. The *order of an element* $g \in G$, denoted by $o(g)$, is defined to be the least non-negative integer n such that $g^n = e$, where e is the identity element in G. The *order of* G, denoted by $o(G)$, is the number of elements G. We say that $g \in G$ is a *p-element* for a prime p if $o(g)$ is a power of p. We say that G *is a p-group* for a prime p if $o(G)$ is a power of p. A subgroup H of G is said to be a p-subgroup if H is a p-group under $*$. We say that a p-subgroup $H \leq G$ of order p^r is a *Sylow p-subgroup* of G if p^r divides $o(G)$ but p^{r+1} does not divide $o(G)$.

We now list some well known group theoretic results that we frequently use.

Theorem 2.1.1. [Hal59, Her64]

1. *Let $(G, *)$ be a group of order n and let p be a prime dividing n. Then there is an element $g \in G$ such that $o(g) = p$.*

2. *Let $(G, *)$ be a group and let H be a subgroup of G. Then $o(H)$ divides $o(G)$.*

3. *If $(G, *)$ is an abelian group and p is a prime dividing $o(G)$ then the Sylow p-subgroup of G is unique.*

4. *(Sylow's Theorem) If $(G, *)$ is an abelian group of order $n = p_1^{r_1} p_2^{r_2} \cdots, p_k^{r_k}$, where $p_1, p_2, \ldots p_k$ are distinct primes, then G is a direct product of its Sylow subgroups $S_{p_1}, S_{p_2}, \ldots, S_{p_k}$. Here each S_{p_i} is of order $p_i^{r_i}$ and is the direct product of cyclic groups of orders $p_i^{r_{i_1}}, p_i^{r_{i_2}}, \ldots, p_i^{r_{i_l}}$ where $r_{i_1} + r_{i_2} + \cdots + r_{i_l} = r_i$.*

Let $(G_1, *)$ and $(G_2, \circ)$ be two groups. Then a mapping $\phi : G_1 \to G_2$ is said to be a homomorphism if $\phi(g * h) = \phi(g) \circ \phi(h)$. We define the kernel of ϕ, denoted by K_ϕ, to be the following subgroup of G: $\{g \in G |\ \phi(g) = e$, where e is the identity element in $G\}$. Moreover if ϕ is one-one and onto then we say that G_1 and G_2 are isomorphic and denote it by $G \cong H$.

In a group $(G, *)$, when the binary operation $*$ used is clear from the context, we avoid using the symbol $*$ and denote $g * h$ by gh itself for $g, h \in G$.

2.1.1 Permutation Groups

A major part of our results in this thesis deals with permutation groups. We recall definitions and basic results about permutation groups that are used in the chapters to follow.

Let Ω be a set containing n points. A *permutation* g over Ω is a one-one mapping from Ω onto itself. The set of all permutations over Ω is denoted by $\mathrm{Sym}(\Omega)$. Given $\alpha \in \Omega$ and a permutation $g \in \mathrm{Sym}(\Omega)$ *the image of* α *in* g is denoted by α^g. For any two permutations g and h we can define the product of g and h to be the permutation obtained by composing g and h as mappings. Thus α^{gh} denotes the point $(\alpha^g)^h$ in Ω. It is easy to observe that $\mathrm{Sym}(\Omega)$ forms a group having the above defined product of permutations as the binary operation. For a permutation $g \in \mathrm{Sym}(\Omega)$ the set $\{\beta \mid \alpha^{g^l} = \beta$, for some integer $l \geq 0\}$ is defined to be the *orbit of* α *with respect to* g. We denote this orbit by $\alpha^{\langle g \rangle}$. The set $\alpha^G = \{\alpha^g \mid g \in G\}$ is said to be the *G-orbit of* α. We say that G *is transitive on* Ω if for any $\alpha \in \Omega$ we have $\alpha^G = \Omega$. A *transposition* is a permutation which is a cycle of length 2. The following results are well known.

Proposition 2.1.2. [Wie64]

1. *Let* $G \leq \mathrm{Sym}(\Omega)$ *be an abelian transitive group on* Ω. *Then* $|G| = |\Omega|$.

2. *Any permutation* $\pi \in \mathrm{Sym}(\Omega)$ *is a product of transpositions.*

If a permutation is a product of even number of transpositions, then it is said to be an *even permutation*; otherwise it is said to be an *odd permutation.*

2.2 Linear Algebra

Let $(R, +, *)$ be a nonempty set with two binary operations $+$ and $*$ defined on the elements in R. Then, $(R, +, *)$ is a ring if it satisfies the following conditions.

- R *is an abelian group with respect to the binary operation* $+$,
- R *is closed under* $*$, that is, for any two elements $a, b \in G$, the element $a * b \in G$,
- R *is associative under* $*$, that is, if for any $a, b, c \in G$, we have $(a*b)*c = a*(b*c)$, and,
- $a * (b + c) = (a * b) + (a * c)$ and $(a + b) * c = (a * c) + (b * c)$.

In the above definition, the identity element in R under the binary operation $+$ is denoted by 0. The last condition mentioned above is the *distributivity law*, stating that $*$ distributes over $+$ when applied either from the left or from the right.

If in the ring $(R, +, *)$ there is an element 1 such that $1 * a = a * 1 = a$ for all $a \in R$, then 1 is said to be the *unit element in* R. In this case, we say that R *is a ring with unit element.* Note that just as the identity element in a group is unique, if R is a ring with

unit element, then the unit element is also unique. If $a * b = b * a$ for all $a, b \in R$, then we say that R is a commutative ring.

Let $(R, +, *)$ be a commutative ring. Then any non-zero element $a \in R$ is a *zero divisor*, if there exists another non-zero element $b \in R$ such that $a * b = 0$. A commutative ring R is said to be an *integral domain* if it does not contain zero divisors. A commutative ring R is said to be a *field* if the non-zero elements in R form a group with respect to the binary operation $*$.

Let $(R, +, *)$ be a ring. A nonempty subset U of R is a *(two-sided) ideal of* R if U is a subgroup of R under $+$ and for every $u \in U$ and $r \in R$ we have $ru, ur \in U$.

A non-empty set M is a *R-module* over a ring $(R, +_R, *_R)$ (or a module over the ring R) if M is an abelian group with respect to a binary operation $+$ such that for every $\alpha \in R$ and $a \in M$ there is an element denoted by $\alpha a \in M$ such that the following conditions hold.

- $\alpha(m + n) = \alpha m + \alpha n$,
- $(\alpha +_R \beta)m = \alpha m + \beta m$, and,
- $\alpha(\beta m) = (\alpha *_R \beta)m$,

for all $m, n \in M$ and $\alpha, \beta \in R$.

A non-empty set V is a *vector space* over a field $(\mathbb{F}, +_\mathbb{F}, *_\mathbb{F})$ if V is an abelian group with respect to a binary operation $+$ and for every $\alpha \in \mathbb{F}$ and $v \in V$, there is an element denoted by $\alpha v \in V$ such that the following conditions hold.

- $\alpha(u + v) = \alpha u + \alpha v$,
- $(\alpha +_\mathbb{F} \beta)v = \alpha v + \beta v$,
- $\alpha(\beta v) = (\alpha *_\mathbb{F} \beta)v$, and,
- $1v = v$, where 1 is the unit element in the field $\mathbb{F}$,

for all $u, v \in V$ and $\alpha, \beta \in \mathbb{F}$. We refer to elements in V as vectors.

From the above two definitions it is clear that a module over a ring generalizes what a vector space is over a field.

Let V be a vector space over a field $\mathbb{F}$, and let $u_1, \ldots, u_n \in V$. For $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$, any element of the form $(\alpha_1 u_1 + \cdots + \alpha_n u_n) \in V$ is said to be a *linear combination* of $u_1, \ldots, u_n$. A set of vectors $\{v_1, \ldots, v_n\} \subseteq V$ is *linearly dependent* if there exists $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$ such that $\alpha_1 v_1 + \ldots + \alpha_n v_n = 0$, where not all α_i are 0 and $1 \leq i \leq n$. Any set of vectors that is not linearly dependent is said to be *linearly independent*. We say that a set of linearly independent vectors $S \subseteq V$ is a *basis* (or a *spanning set*) for the vector space V

over the field $\mathbb{F}$, if every $u \in V$ is a linear combination of vectors in S. We say that V is a finite-dimensional vector space if the number of elements in a basis S of V is finite. In particular, if the number of elements in S is d, then V is referred to as a d-dimensional vector space over $\mathbb{F}$.

Let $(U, +_U, *_U)$ and $(V, +_V, *_V)$ be vector spaces over a field $\mathbb{F}$. Then a mapping T from U into V is said to be a *linear transformation* if for any $u_1, u_2 \in U$, we have $T(u_1 +_U u_2) = T(u_1) +_V T(u_2)$ and $T(\alpha u_1) = \alpha T(u_1)$. Then, we define the *kernel of* T to be $\{u \in U |\ T(u) = 0$, where 0 is the identity element of V with respect to $+_V\}$. We can also associate a matrix representation to every such linear transformation. Entries of such a matrix are elements of the base field $\mathbb{F}$. To every element $u \in U$ we associate a vector formed by the coefficients occurring in the linear combination of the basis elements of U. Thus if $\mathbf{x}_u$ is the vector corresponding to u then $T(u) \in V$ is element formed by the linear combination of entries of the vector $A\mathbf{x}_u$ with the basis elements of V.

Let $A = (a_{ij})_{1 \leq i,j \leq n}$ be a $n \times n$ matrix with entries from $\mathbb{F}$. Then, the determinant of A, denoted by $\det(A)$, is $\sum_{\sigma \in \mathrm{Sym}(\Omega)} (-1)^{\mathrm{sign}(\sigma)} (\prod_{1 \leq i \leq n} a_{i,\sigma(i)})$, where $\Omega = \{1, 2, \ldots n\}$, and $\mathrm{sign}(\sigma) = 1$ if σ is an even permutation and it is -1 otherwise. We say that a matrix A is singular if $\det(A) = 0$. A matrix is said to be unimodular if $\det(A)$ is -1 or 1. The characteristic polynomial of a matrix A is $\det(A - I\mathbf{x})$, where I is the $n \times n$ identity matrix, and $\mathbf{x}$ is a n-dimensional column vector of indeterminates. The minimal polynomial $f(x)$ of A, is the least degree monic irreducible polynomial with coefficients from $\mathbb{F}$ such that $f(A) = 0$.

Note 1. In Chapters 3 and 4 we study various algorithmic problems based on linear algebra such as solving linear equations wherein entries to matrices are from a finite ring, such as $\mathbb{Z}_q$ for a composite integer q. When q is not a prime, $\mathbb{Z}_q^n$ is not a vector space since $\mathbb{Z}_q$ is not a field. Actually, $\mathbb{Z}_q^n$ is a *module* over the ring $\mathbb{Z}_q$. Still, we will refer to elements of $\mathbb{Z}_q^n$ as vectors (or column vectors). We hope this terminology is not confusing. Also, several other vector space related definitions and terminology are applicable to modules defined over rings. Specifically, *linear combination of elements, linear transformation,* and giving *a matrix representation to a linear transformation,* naturally generalize to the setting of modules.

An important difference arises due to the presence of zero divisors in rings. This will be made clear in Chapter 3 where we study the complexity solving linear equations over finite rings. Unlike solving linear equations over fields, over rings we do not have the usual connections between rank, linear independence, and feasibility of a system of linear equations.

Theorem 2.2.1. (Cauchy-Binet Theorem) *Let $(R, +, *)$ be a ring. Given two matrices $A, B \in R^{m \times n}$ with $n \geq m$, we have*

$$\det(AB^T) = \sum_{\alpha} \det(A_\alpha) \det(B_\alpha),$$

where $\alpha \subseteq \{1, \ldots, n\}$ with $|\alpha| = m$ representing all possible ways of choosing m indexes from a set of n indexes. Here A_α and B_α denote $m \times m$ sub matrices of A, and B respectively, formed by picking columns corresponding to indexes in α.

We state two other basic number theoretic results that are used in subsequent chapters.

Theorem 2.2.2. (Prime Number Theorem) [Apo86] *Let n be a positive integer and let $\pi(n)$ denote the number of primes less than or equal to n. Then $\pi(n) = \Theta(\frac{n}{\log n})$.*

Theorem 2.2.3. (Chinese Remainder Theorem) [Apo86] *Let $m_1, \ldots m_r$ be positive integers that are pairwise relatively prime. Also let $b_1, \ldots, b_r$ be arbitrary integers. Then, there is a unique integer $a \in \mathbb{Z}_M$, where $M = m_1 \cdots m_r$, such that $a \equiv b_i (mod\ m_i)$, for all $1 \leq i \leq r$.*

2.3 Probability Theory

We recall some definitions and results in probability theory that are required to explain our results in the thesis. For more clarification we refer to standard texts such as [MR95].

Definition 2.3.1. *Let X be a random variable defined over a sample space Ω with a probability measure* Pr*. Then, the expectation of X, denoted by $E[X] = \Sigma_{x \in \Omega} x \Pr[X = x]$.*

Theorem 2.3.2. (Linearity of Expectation) *Let $X_1, \ldots, X_k$ be k arbitrary random variables defined over a sample space Ω with corresponding probability measures defined for each of them. Then, $E[X_1 + \cdots + X_k] = \Sigma_{i=1}^{k} E[X_i]$.*

Theorem 2.3.3. (Chernoff bound) *Let $X_1, \ldots, X_k$ be independent boolean random variables such that $p = \Pr[X_i = 1]$, for all $1 \leq i \leq n$ with $0 \leq p \leq 1$. Also let $X = \Sigma_{i=1}^{n} X_i$, and let e denote the base of the natural logarithm. Then for any $\delta > 0$,*

$$\Pr[X > (1 + \delta)\mu] < [e^\delta / ((1 + \delta)^{(1+\delta)})]^\mu \leq e^{-\delta^2 \mu / 3},$$

where $\mu = np$ is the expectation of the random variable X.

3

Solving Linear Equations over a Finite Ring

3.1 Introduction

In this chapter, we tightly classify the complexity of a number of problems on solving a system of linear equations over a given finite ring R. More precisely, we consider problems LCON, LCONX, and LCONNULL, defined in Section 1.4 of Chapter 1, and show that these problems are in BP·NC^2. One of the main motivations behind studying these problems is, problems such as solving linear equations over $\mathbb{Q}$ and over finite fields such as $\mathbb{Z}_p$, where p is a prime, have been shown to be complete for $L^{C_=L}$, and Mod_pL respectively by [ABO99, BDHM92]. However, no such tight result is known for solving linear equations over $\mathbb{Z}_q$, for a composite integer q, or over a finite ring R. Yet another reason is the pivotal role that problems such as LCON play in classifying the complexity of a number of problems based on abelian permutation groups. McKenzie and Cook in [MC87] show that these permutation group theoretic, and linear algebraic problems are in fact NC^1 Turing equivalent. Using these reductions and the upper bound results that we obtain, we show that all these problems are in BP·NC^2. We define and discuss the complexity of abelian permutation group theoretic problems that are of interest to us in Chapter 4.

In defining LCON and related problems on linear algebra over which the system of linear equations is to be solved, we specify the ring $\mathbb{Z}_q$ as a part of the input. To be more explicit, the basic problem is to check if a system of linear congruences has a solution or not. In this context, as mentioned in Section 1.4 of Chapter 1, the modulo operation in Definition 1.4.1 of the logspace counting class ModL makes it more natural to express the complexity of problems such as LCON with respect to this class. Thus the BP·NC^2 upper bound for these problems on linear algebra in turn show that these problems are also in L^{ModL}/poly. Also these problems are shown to be hard for L^{ModL} under logspace Turing reductions. Thus we obtain a fairly tight classification of the complexity

of problems that are studied. Regarding the complexity class ModL, we also show that $\mathrm{L}^{\mathrm{GapL}} = \mathrm{L}^{\mathrm{ModL}}$, which makes it interesting to study.

We also try to generalize LCON by considering how feasible it is to solve linear equations over any finite ring R. It turns out that when the input ring R is given to us in an explicit manner in terms of addition and multiplication tables of elements in R (wherein the additive abelian group $(R, +)$ is given as a direct sum of cyclic subgroups of prime power order), the complexity of solving system of linear equations over R is also $\mathrm{L}^{\mathrm{ModL}}/\mathrm{poly}$. We obtain this result by giving a matrix representation for each element in R and finally reducing the problem to several instances of solving linear equations over $\mathbb{Z}_{p^e}$, for different prime powers p^e. By repeatedly invoking the algorithm for LCON on these instances, we finally determine if the given system of linear equations over R has a solution and hence we place this problem in $\mathrm{L}^{\mathrm{ModL}}/\mathrm{poly}$. This result is described in detail in Section 3.5.

3.2 Feasibility of Linear Congruences Modulo Composites

We recall the definition of LCON from Chapter 1.[1] Given a matrix $A \in \mathbb{Z}^{m \times n}$ and a column vector $\mathbf{b} \in \mathbb{Z}^m$, the problem LCON is to determine whether $A\mathbf{x} = \mathbf{b}$ is a feasible system of linear equations over the ring $\mathbb{Z}_q$. Here q is a positive integer given as part of the input in terms of its prime factorization $q = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$, such that each $p_i^{e_i}$ is tiny (i.e. given in unary).

McKenzie and Cook in [MC87], showed that the problem LCON of checking if the congruence $A\mathbf{x} = \mathbf{b} (\mathrm{mod}\ q)$ has a solution, for a composite integer $q = p_1^{e_1} \cdots p_k^{e_k} \geq 2$, is in the complexity class NC^3. Here we assume that q is given in terms of its prime factorization with each prime power $p_i^{e_i}$, for $1 \leq i \leq k$, specified in unary. The basic idea used there, is to solve $A\mathbf{x} = \mathbf{b} \pmod{p_i^j}$, by first solving $A\mathbf{x} = \mathbf{b} \pmod{p_i}$, for $1 \leq i \leq k$, and then "lifting" the solution (essentially Hensel lifting [NZM01, Lemma 2.23]) repeatedly to solutions modulo p_i^j for increasing values of j, until a solution to $A\mathbf{x} = \mathbf{b} \pmod{p_i^{e_i}}$ is obtained. The solutions for different prime powers $p_i^{e_i}$, where $1 \leq i \leq k$, are then combined using the Chinese remainder theorem to obtain a solution for the original congruence.

To arrive at our results, we start by presenting a $\mathrm{BP{\cdot}NC}^2$ algorithm that avoids the lifting process mentioned above. By standard probability amplification techniques we show that there exists a string of polynomial length which can be supplied as ad-

[1] We also recall Note 1 from Chapter 2.

vice to the algorithm instead of the random bits it requires. As a consequence, we get LCON $\in \mathrm{L}^{\mathrm{ModL}}/\mathrm{poly}$. Under a possible hardness assumption that there is a language in DSPACE(n) which requires circuits of sub-exponential size we show that the upper bound for LCON holds in the uniform setting also; that is LCON $\in \mathrm{L}^{\mathrm{ModL}}$. Along with these results we show that LCON is hard for ModL under logspace many-one reductions. Thus from these observations we obtain a fairly tight classification of the complexity of LCON in terms of logspace counting classes.

Proposition 3.2.1. *Let $L \in$ ModL be witnessed by a* GapL *function $gap_M(x)$, where M is a nondeterministic logspace Turing machine accepting L. Also let $g \in$ FL be a function that outputs a prime power in unary. Then given any $f(x) \in$ GapL and $x \in \Sigma^*$, we have a nondeterministic logspace Turing machine M' such that $x \in L \Leftrightarrow gap_{M'}(x) \not\equiv f(x)(mod\ |g(x)|)$, where $gap_{M'}(x) = acc_{M'}(x) - rej_{M'}(x)$.*

Proof. Since GapL [AO96] is closed under addition and subtraction, we have a nondeterministic logspace Turing machine M' such that $gap_{M'}(x) = gap_M(x) + f(x)$. Thus $x \in L \Leftrightarrow gap_M(x) \not\equiv 0(\mathrm{mod}|g(x)|) \Leftrightarrow gap_{M'} \not\equiv f(x)(\mathrm{mod}\ |g(x)|)$. ∎

Following is a nice result that relates ModL introduced in Definition 1.4.1 of Section 1.4 from Chapter 1, and the logspace counting class GapL.

Lemma 3.2.2. $\mathrm{FL}^{\mathrm{ModL}} = \mathrm{FL}^{\mathrm{GapL}}$.

Proof. To see this we first observe that ModL $\subseteq \mathrm{L}^{\mathrm{GapL}}$. Suppose $L \in$ ModL is witnessed by an $f \in$ GapL and a function $g \in$ FL that computes tiny prime powers in unary. On query x to the oracle f, an $\mathrm{L}^{\mathrm{GapL}}$ computation can retrieve all the bits of $f(x)$ from the least significant to the most significant. If the i^{th} bit from the right of $f(x)$ is 1, we compute $2^i \pmod{|g(x)|}$ in logspace and add it to the current sum modulo $|g(x)|$. When all the bits of $f(x)$ are scanned we would have computed $f(x)(\mathrm{mod}\ |g(x)|)$.

For the reverse inclusion let $L \in \mathrm{L}^{\mathrm{GapL}}$ computed by a logspace oracle machine with access to a GapL complete function f as oracle. (It is easy to note that since $\mathrm{L}^{\mathrm{GapL}} = \mathrm{L}^{\#\mathrm{L}}$ we can assume the function f to be always non-negative for all inputs x). For $x \in \Sigma^n$, we have size$(f(x)) \leq p(n)$, for some polynomial $p(n)$. By the Prime Number Theorem, the number of primes between 2 and $p^2(n)$ is $p^2(n)/O(\log n)$ which exceeds $p(n)$ for sufficiently large n. Thus the first $p(n)$ primes are each of size $O(\log n)$ bits. Furthermore, the product of the first $p(n)$ primes exceeds $f(x)$. Now, it is easy to see that checking if an $O(\log n)$ bit integer is a prime can be done in logspace. Furthermore, in logspace we can compute the i^{th} prime for $1 \leq i \leq p(n)$. Let p_i denote the i^{th} prime. We define the function $g \in$ FL as follows $g(x, 0^{p(|x|)}, i) = p_i$ if $i \leq p(|x|)$ and it is defined as 0^2 otherwise.

We define the following language in ModL

$$L' = \{\langle x, 0^{p(|x|)}, i, k\rangle \mid i \leq p^2(|x|), k \leq p^2(|x|) \text{ and } f(x) \equiv k \pmod{g(x, 0^{p(|x|)}, i)}\}.$$

In order to show that $L \in \mathrm{L}^{\mathrm{ModL}}$ we need to simulate the $\mathrm{L}^{\mathrm{GapL}}$ machine for L with a $\mathrm{L}^{\mathrm{ModL}}$ computation. Clearly, it suffices to show that each GapL query $f(x)$ made by the base logspace machine can be simulated in $\mathrm{L}^{\mathrm{ModL}}$. For each $1 \leq i \leq p(n)$ we can query L' for $\langle x, 0^p(|x|), i, k\rangle$ for different values of $k \leq p^2(|x|)$ to find $f(x) (\mathrm{mod}\ p_i)$.

Now, by Chinese Remaindering $f(x)$ is uniquely determined by $f(x) (\mathrm{mod}\ p_i)$, for $1 \leq i \leq p(n)$. Moreover, given these residues $f(x) (\mathrm{mod}\ p_i)$ for $1 \leq i \leq p(n)$ it is possible to compute $f(x)$ in logspace by the results of [CDL01, HAB02]. Hence a logspace oracle machine with access to the ModL oracle L' can recover $f(x)$ for each query x. It follows easily that L is in $\mathrm{L}^{\mathrm{ModL}}$. ∎

Remark 1. In the proof of the above result, to show that $\mathrm{L}^{\mathrm{GapL}} \subseteq \mathrm{L}^{\mathrm{ModL}}$, rather than assuming $f \in \#\mathrm{L}$ and is hence non-negative, we can also construct a suitable non-negative GapL function from f and the primes p_i while defining L'. In such cases, as the base logspace machine retrieves the values of the new GapL function in its Chinese Remainder representation it subtracts the required positive integer to obtain the value of the actual GapL function f.

Lemma 3.2.3. *Let $A \in \mathbb{Z}^{m \times n}$, $\mathbf{b} \in \mathbb{Z}^m$, and $q \geq 2$ be a positive integer given in terms of its prime factorization $p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$, such that each $p_i^{e_i}$ is tiny (i.e. given in unary) with $1 \leq i \leq k$. Then $A\mathbf{x} = \mathbf{b}$ (mod q) is feasible if and only if $A\mathbf{x} = \mathbf{b}$ (mod $p_i^{e_i}$) is feasible for every $1 \leq i \leq k$. Moreover if the solution to $A\mathbf{x} = \mathbf{b}$(mod $p_i^{e_i}$) is given for all $1 \leq i \leq k$, then a solution to $A\mathbf{x} = \mathbf{b}$(mod q) is computable in logspace.*

Proof. If $A\mathbf{x} = \mathbf{b} (\mathrm{mod}\ q)$ has a solution $\mathbf{x}$ then it is obvious that $\mathbf{x}$ also satisfies $A\mathbf{x} = \mathbf{b} (\mathrm{mod}\ p_i^{e_i})$, for all $1 \leq i \leq k$. The converse is an application of the Chinese Remainder Theorem. Assume that $A\mathbf{x} = \mathbf{b} (\mathrm{mod}\ p_i^{e_i})$ has a solution $\mathbf{x}_i$, for each $1 \leq i \leq k$. Then, $\mathbf{x}_i$ satisfies $A\mathbf{x}_i = \mathbf{b} + p_i^{e_i}\mathbf{z}_i$ for some vector $\mathbf{z}_i \in \mathbb{Z}^m$. We now describe a method to lift these solutions over $\mathbb{Z}_{p_i^{e_i}}$ for $1 \leq i \leq k$ to obtain a solution for $A\mathbf{x} = \mathbf{b} (\mathrm{mod}\ q)$.

For $1 \leq i \leq k$, let $\gamma_i = \frac{q}{p_i^{e_i}}$. Notice that, γ_i is invertible in $\mathbb{Z}_{p_i^{e_i}}$. We can compute its multiplicative inverse β_i in $\mathbb{Z}_{p_i^{e_i}}$ in logspace by exhaustive search. Thus, the term γ_i so obtained satisfies $\gamma_i\beta_i \equiv 1 (\mathrm{mod}\ p_i^{e_i})$.

Now, define $\mathbf{y} = \sum_{i=1}^{n} \gamma_i \beta_i \mathbf{x}_i$. Then we have,

$$\begin{aligned} A\mathbf{y} &= \sum_{i=1}^{k} \gamma_i \beta_i A\mathbf{x}_i \\ &= \sum_{i=1}^{k} \gamma_i \beta_i (\mathbf{b} + p_i^{e_i} \mathbf{z}_i) \\ &= \mathbf{b}(\sum_{i=1}^{k} \gamma_i \beta_i) + q \sum_{i=1}^{k} \beta_i \mathbf{z}_i \\ &= \mathbf{b}(\sum_{i=1}^{k} \gamma_i \beta_i) (\text{mod } q). \end{aligned} \tag{3.1}$$

However, we note that for every $1 \leq i \leq k$, we have $\sum_{j=1}^{k} \gamma_j \beta_j \equiv 1 (\text{mod } p_i^{e_i})$. That is, $p_i^{e_i} | (\sum_{j=1}^{n} \gamma_j \beta_j - 1)$, for all $1 \leq i \leq k$. Since $q = p_1^{e_1} \cdots p_k^{e_k}$, where $p_1^{e_1}, \ldots, p_k^{e_k}$ are relatively prime, it follows that $q | (\sum_{u=1}^{n} \gamma_u \beta_u - 1)$. Or, in other words

$$\sum_{u=1}^{n} \gamma_u \beta_u \equiv 1 (\text{mod } q). \tag{3.2}$$

Thus from (3.1) and (3.2) we get

$$A\mathbf{y} \equiv \mathbf{b} (\text{mod } q).$$

It is clear that if the solutions for each congruence $A\mathbf{x} = \mathbf{b} (\text{mod } p_i^{e_i})$ is given, then the rest of the computation involving multiplying and adding integers of size at most polynomial in the size of the given input can be done in logspace. Thus, a logspace machine can compute the required solution for $A\mathbf{x} = \mathbf{b} (\text{mod } q)$ and hence the result follows. ■

Thus we now focus on the problem of testing if the system $A\mathbf{x} = \mathbf{b} \pmod{p^e}$ is feasible, where p is a prime and p^e is tiny. If this system is feasible, then we also compute a solution for the same. In other words, we are testing if $A\mathbf{x} = \mathbf{b}$ has a solution in the finite ring $\mathbb{Z}_{p^e}$. For this, we first transform the problem to solving a system of linear Diophantine equations in the following proposition.

Proposition 3.2.4. *Let A be an $m \times n$ integer matrix, $\mathbf{b}$ be an m integer column vector, and p be a prime and e a positive integer. The system of linear equations $A\mathbf{x} = \mathbf{b} \pmod{p^e}$ is feasible (in the finite ring $\mathbb{Z}_{p^e}$) if and only if $A\mathbf{x} + p^e\mathbf{y} = \mathbf{b}$ has a solution in $\mathbb{Z}$.*

Proof. Clearly, if $A\mathbf{x} + p^e\mathbf{y} = \mathbf{b}$ has a solution $\mathbf{x}', \mathbf{y}'$ in $\mathbb{Z}$, then $A\mathbf{x}' = \mathbf{b} \pmod{p^e}$. Conversely, if $\mathbf{x}'$ is a solution to $A\mathbf{x} = \mathbf{b} \pmod{p^e}$ then $A\mathbf{x}'$ must be of the form $\mathbf{b} + p^e\mathbf{y}'$

for some integral vector $\mathbf{y}'$. Consequently, $(\mathbf{x}', -\mathbf{y}')$ is an integral solution to $A\mathbf{x}+p^e\mathbf{y} = \mathbf{b}$.

■

Remark 2. Polynomial time algorithms for solving linear Diophantine equations are well known (see e.g. [Sch98]). However the problem is not known to be in NC. It is observed in [ABO99] that testing existence of integral solutions to $A\mathbf{x} = \mathbf{b}$ is RNC reducible to checking if $\gcd(a_1, a_2, \ldots, a_n) = \gcd(b_1, \ldots, b_m)$, for integers a_i and b_j. It is a long standing open problem if the latter problem is in NC (even randomized NC).

However, the system $A\mathbf{x}+p^e\mathbf{y} = \mathbf{b}$ of linear Diophantine equations has a form whose structure we will be able to exploit and avoid computation of the GCD of integers.

Let us consider the following set of rationals, $\mathrm{Z}_{(\mathrm{p})}$ (contained in $\mathbb{Q}$):

$$\mathrm{Z}_{(\mathrm{p})} = \{\frac{\mathrm{a}}{\mathrm{b}} \mid \mathrm{a}, \mathrm{b}, \in \mathbb{Z} \ : \ \gcd(\mathrm{a}, \mathrm{b}) = 1 \text{ and } \gcd(\mathrm{p}, \mathrm{b}) = 1\}.$$

$\mathrm{Z}_{(\mathrm{p})}$ is the set of all rationals a/b, wherein the denominator b is relatively prime to the numerator a and the prime p. It is easy to see that $\mathrm{Z}_{(\mathrm{p})}$ is an integral domain with unit element 1 under the usual addition and multiplication of rationals.

Lemma 3.2.5. *Let A be an $m \times n$ integer matrix, $\mathbf{b}$ be an $m \times 1$ integer column vector, p be a prime and e a positive integer. The system $A\mathbf{x} + p^e\mathbf{y} = \mathbf{b}$ has a solution in $\mathbb{Z}$ if and only if $A\mathbf{x} + p^e\mathbf{y} = \mathbf{b}$ has a solution in the ring $\mathrm{Z}_{(\mathrm{p})}$.*

Proof. If $A\mathbf{x} + p^e\mathbf{y} = \mathbf{b}$ has a solution in $\mathbb{Z}$ then obviously that solution lies in $\mathrm{Z}_{(\mathrm{p})}$ as well.

Conversely, suppose $A\mathbf{x} + p^e\mathbf{y} = \mathbf{b}$ has a solution $\mathbf{x}', \mathbf{y}'$ in $\mathrm{Z}_{(\mathrm{p})}$. Each entry of $\mathbf{x}'$ and $\mathbf{y}'$ is a rational number. Let $\alpha \in \mathbb{Z}$ be the least common multiple of the denominators of the entries in $\mathbf{x}', \mathbf{y}'$. Let $\mathbf{x}'' = \alpha\mathbf{x}'$ and $\mathbf{y}'' = \alpha\mathbf{y}'$. Both $\mathbf{x}''$ and $\mathbf{y}''$ are integral vectors and it follows that

$$A\mathbf{x}'' + p^e\mathbf{y}'' = \alpha\mathbf{b}.$$

Since $\mathbf{x}', \mathbf{y}'$ is a solution in $\mathrm{Z}_{(\mathrm{p})}$, it follows that $(\alpha, p) = 1$. Thus there are integers $s, t \in \mathbb{Z}$ such that $sp^e + t\alpha = 1$. Consequently, we have $tA\mathbf{x}'' + tp^e\mathbf{y}'' = (1 - sp^e)\mathbf{b}$. Rearranging terms, we obtain $tA\mathbf{x}'' + p^e(s\mathbf{b} + t\mathbf{y}'') = \mathbf{b}$ yielding a solution in $\mathbb{Z}$. ■

We observe one further property of the linear system $A\mathbf{x} + p^e\mathbf{y} = \mathbf{b}$. We can rewrite it as $B\mathbf{z} = \mathbf{b}$. Notice that the matrix $B = (A; p^eI)$ is an $m \times (m + n)$ matrix of rank m and $\mathbf{z} = (\mathbf{x}, \mathbf{y})$.

Proposition 3.2.6. *$A\mathbf{x}+p^e\mathbf{y} = \mathbf{b}$ is a system of linear equations with coefficient matrix $[A; p^eI]$ of full row rank.*

Let B be an $m \times n$ integer matrix of full row rank, and $\mathbf{b}$ be an integral column vector. The theory of linear Diophantine equations precisely characterizes when the system of linear equations $B\mathbf{z} = \mathbf{b}$ has an *integral* solution. We state the following useful characterization from [Sch98, pp. 51] and [Dic92, pp. 82].

Theorem 3.2.7. [Sch98, pp. 51] *Let B be an $m \times n$ integer matrix of full row rank and $\mathbf{b}$ be an integral column vector. The system of linear equations $B\mathbf{z} = \mathbf{b}$ has an* integral *solution $\mathbf{z}$ if and only if[2] the GCD of all the nonzero $m \times m$ subdeterminants of B equals the GCD of all the nonzero $m \times m$ subdeterminants of the augmented matrix $[B; \mathbf{b}]$.*

Intuitively, this follows from the fact that the GCD of the $m \times m$ subdeterminants of B is the volume of fundamental parallelepiped in the integral lattice generated by the columns of B and the GCD of the $m \times m$ subdeterminants of $[B; \mathbf{b}]$ is the volume of fundamental parallelepiped in the integral lattice generated by the columns of $[B; \mathbf{b}]$. $B\mathbf{z} = \mathbf{b}$ is feasible if and only if $\mathbf{b}$ lies in the lattice of B and the vector $\mathbf{b}$ will lie in this lattice if and only if the volume of the fundamental parallelepiped in the lattice generated by columns of $[B; \mathbf{b}]$ equals the volume of the fundamental parallelepiped in the lattice generated by the columns of B.

Based on the above theorem, we now give a similar characterization for the feasibility of the linear equations $B\mathbf{z} = \mathbf{b}$ over $\mathrm{Z}_{(\mathrm{p})}$. This will be useful for proving our new upper bound result. For a positive integer d, let $ord_p(d)$ be the largest nonnegative integer e such that p^e divides d.

Theorem 3.2.8. *Let B be an $m \times n$ integer matrix of full row rank and $\mathbf{b}$ be a m-dimensional integer column vector. Let r denote the GCD of all the nonzero $m \times m$ subdeterminants of B and s denote the GCD of all the nonzero $m \times m$ subdeterminants of the augmented matrix $[B; \mathbf{b}]$. The system of linear equations $B\mathbf{z} = \mathbf{b}$ has a solution in $\mathrm{Z}_{(\mathrm{p})}$ if and only if $ord_p(r) = ord_p(s)$.*

Proof. Firstly, notice that s is a factor of r for any integer matrix B of full row rank and any column vector $\mathbf{b}$ (simply because B is a submatrix of $[B; \mathbf{b}]$), where s and r are defined in the statement above. Thus we can write $r = ds$, for some integer d.

Now, suppose $B\mathbf{z} = \mathbf{b}$ is feasible over $\mathrm{Z}_{(\mathrm{p})}$. Then, by clearing denominators of the solution, it follows that there is a positive integer $\alpha \in \mathbb{Z}$ such that $\gcd(\alpha, p) = 1$ and $B\mathbf{z} = \alpha\mathbf{b}$ is feasible over $\mathbb{Z}$. Let t denote the GCD of all nonzero $m \times m$ sub determinants of $[B; \alpha\mathbf{b}]$. Applying Theorem 3.2.7 to the system $B\mathbf{z} = \alpha\mathbf{b}$, it follows that $r = t$. Thus $r = t = ds$. If u denotes the GCD of all nonzero $m \times m$ sub determinants of

[2] Our statement is slightly different but equivalent to that in [Sch98]. For, the GCD of the $m \times m$ subdeterminants of the augmented matrix $[B; \mathbf{b}]$ will in any case divide the GCD of all the nonzero $m \times m$ subdeterminants of B.

$[B; \mathbf{b}]$ containing the column vector $\mathbf{b}$ and v denotes the GCD of all nonzero $m \times m$ sub determinants of $[B; \alpha\mathbf{b}]$ containing the column vector $\alpha\mathbf{b}$, it is easy to observe that $s = \gcd(r, u)$ and $t = \gcd(r, v)$. But $t = ds = d\ \gcd(r, u)$. This implies that d divides α. Since we also have $\gcd(\alpha, p) = 1$ we get $ord_p(r) = ord_p(s)$.

Conversely, suppose $ord_p(r) = ord_p(s)$. Since B has full row rank m, the linear system $B\mathbf{z} = \mathbf{b}$ has a rational solution $\mathbf{z}'$. Let $p^e\alpha$ be the LCM of the denominators of entries in $\mathbf{z}'$. The α mentioned here is the divisor of the LCM of the denominators of entries in $\mathbf{z}'$, such that $\gcd(\alpha, p) = 1$. Multiplying by $p^e\alpha$ on both sides of the equation $B\mathbf{z}' = \mathbf{b}$ we get $B\mathbf{z}'' = p^e\alpha\mathbf{b}$, where $\mathbf{z}''$ has integer entries. Let t denote the GCD of all $m \times m$ sub determinants of $[B; p^e\alpha\mathbf{b}]$. By applying Theorem 3.2.7 to the system $B\mathbf{z} = p^e\alpha\mathbf{b}$, it follows that $r = t$. Thus $r = t = ds$. But $p \nmid d$ as $ord_p(s) = ord_p(r)$. It follows that the GCD of all $m \times m$ sub determinants of the matrix $[B; \alpha\mathbf{b}]$ is also r. Again applying Theorem 3.2.7 to $B\mathbf{z} = \alpha\mathbf{b}$, it follows that $B\mathbf{z} = \alpha\mathbf{b}$ has an *integral* solution (call it $\mathbf{z}_0$). From the definition of α, we have $\gcd(\alpha, p) = 1$. Thus, it follows that $\frac{1}{\alpha}\mathbf{z}_0$ is a $\mathrm{Z}_{(\mathrm{p})}$ solution to $B\mathbf{z} = \mathbf{b}$. This completes the proof. ■

3.2.1 The Upper Bound Result

A square integer matrix M is *unimodular* if $det(M)$ is ± 1. Let $A \in \mathbb{Z}^{m\times n}$ with $m \leq n$. Then there exists a unique integer matrix $S = (D; 0)$ and unimodular matrices $P \in \mathbb{Z}^{m\times m}$ and $Q \in \mathbb{Z}^{n\times n}$ such that $S = PAQ$, where D is a $m \times m$ integer diagonal matrix. The matrix S is called the *Smith Normal Form* of A. If r is the rank of A then the diagonal matrix D has diagonal $diag(s_1, \ldots, s_r, 0, \ldots, 0)$, where $s_i \neq 0$ for $1 \leq i \leq r$ such that $s_i | s_{i+1}$ for each i. Furthermore, if d_k denotes the GCD of all $k \times k$ minors of A for $1 \leq k \leq r$, then $s_1 = d_1$ and $s_k = d_k/d_{k-1}$ for $2 \leq k \leq r$. The number d_k is the k^{th} *determinantal divisor* of A, $1 \leq k \leq r$, and s_k are the *invariant factors* of A.

The algorithm that we are going to describe for LCON is based on the ideas and results of Giesbrecht [Gie95] in which the author describes a randomized polynomial time algorithm to compute the Smith Normal Form of an integer matrix.

We can now give a straightforward reformulation of the characterization of Theorem 3.2.8 for the feasibility of $B\mathbf{z} = \mathbf{b}$ over $\mathrm{Z}_{(\mathrm{p})}$ in terms of determinantal divisors.

Theorem 3.2.9. *Let B be an $m \times n$ integer matrix of full row rank and $\mathbf{b}$ be an integral column vector of length m. Let d_m be the m^{th} determinantal divisor of B and d'_m be the m^{th} determinantal divisor of the augmented matrix $[B; \mathbf{b}]$. The system of linear equations $B\mathbf{z} = \mathbf{b}$ has a solution in the ring $\mathrm{Z}_{(\mathrm{p})}$ if and only if $ord_p(d_m) = ord_p(d'_m)$.*

Proof. As given above, the k^{th} determinantal divisor of a matrix $A \in \mathbb{Z}^{m\times n}$ be the GCD of all $k \times k$ minors of A, for $1 \leq k \leq m \leq n$. We obtain our result by choosing $k = m$

and adapting the definition of the m^{th} determinantal divisor in the statement of Theorem 3.2.8. ■

Thus the problem of testing feasibility of $B\mathbf{z} = \mathbf{b}$ over the ring $Z_{(p)}$ is equivalent to checking if $ord_p(d_m) = ord_p(d'_m)$, where d_m is the m^{th} determinantal divisor of B and d'_m is the m^{th} determinantal divisor of the matrix $[B; \mathbf{b}]$.

The difficulty with computing d_m and d'_m lies in the number of $m \times m$ submatrices of B, and $[B; \mathbf{b}]$ that we need to consider. This number can be exponential in the size of the input. Also the problem of computing the gcd of a set of integers is not known to be in NC. These reasons prompt us to explore new ways of computing d_m and d'_m. We will use the following result of Giesbrecht [Gie95] and design a randomized algorithm to test if $ord_p(d_m) = ord_p(d'_m)$, without actually computing the numbers d_m and d'_m.

Recall that the *content*, denoted by $cont(f)$, of a multivariate polynomial f (over any Euclidean Domain, in particular integers) is the GCD of all the coefficients of f.

Theorem 3.2.10. [Gie95, Theorem 2.1] *Let B be an $m \times n$ integer matrix of rank r. Let $X = (X_{ij})$ be an $r \times m$ matrix and $Y = (Y_{lk})$ be an $n \times r$ matrix of distinct indeterminates X_{ij} and Y_{lk}, $1 \leq i, k \leq r$, $1 \leq j \leq m$, and $1 \leq l \leq n$. Then the content of the determinant of the t^{th} leading minor of the $r \times r$ matrix XBY equals the t^{th} determinantal divisor d_t, $1 \leq t \leq r$.*

As a direct consequence of Theorem 3.2.9 and Theorem 3.2.10, we obtain the following.

Lemma 3.2.11. *Let B be an $m \times n$ integer matrix of full row rank and let $\mathbf{b}$ be an integral column vector of dimension m. The system of linear equations $B\mathbf{z} = \mathbf{b}$ has a solution in $Z_{(p)}$ if and only if $ord_p(cont(\det(XBY))) = ord_p(cont(\det(X[B; \mathbf{b}]Z))$, where X, Y, and Z are matrices of indeterminates of dimension $m \times m$, $n \times m$ and $(n+1) \times m$ respectively.*

We now focus on the problem of computing $ord_p(cont(\det(XBY)))$, where B is an $m \times n$ integer matrix of rank m. Notice that computing $\det(XBY)$ is inefficient as there are exponentially many terms that contribute to this multivariate polynomial. Instead, following Giesbrecht [Gie95] and in analogy with the Schwartz-Zippel test, the idea is to compute the determinant $\det(XBY)$, where values for the indeterminates in X and Y are randomly picked from a suitable domain (over which computing the determinant will be easy). We will use the following variant of the Schwartz-Zippel test (as stated in Giesbrecht [Gie95]). The proof given below is a analogous to the proof of the Schwartz-Zippel theorem given in [MR95]. We give a proof of this result for completeness.

Lemma 3.2.12. [Gie95, Lemma 2.2] *Let $g \in D[z_1, z_2, \ldots, z_s]$ be a nonzero polynomial, where D is an integral domain. Let W be a finite subset of D. Suppose elements $a_1, \ldots, a_s$*

are picked independently at random from D with the probability of choosing an element being at most ϵ. Then $\text{Prob}[g(a_1, \ldots, a_s) = 0; a_i \in W] \leq \epsilon deg(g)$*, where $deg(g)$ is the total degree of g.*

Proof. We use induction on the number of variables in $g(z_1, \ldots, z_s)$ to prove the result. Let $s = 1$ and $d = \deg(g)$. Then, g is a univariate polynomial of degree d, and so has no more than d distinct roots in D. If a is picked independently at random from D such that it is equal to any $a_i \in D$ with probability at most ϵ, then $\Pr_{a \in_r D}[a \text{ is a root of } g(z)] \leq \epsilon d$. In other words, $\Pr_{a \in_r D}[g(a) = 0] \leq \epsilon d$. This completes the base case.

Assume the result to be true for all polynomials having at most $(s-1)$ variables. Let $g(z_1, \ldots, z_s)$ be a polynomial containing s variables. Then,

$$g(z_1, \ldots, z_s) = \sum_{i=0}^{k} z_1^i g_i(z_2, \ldots, z_s),$$

where $k \leq d$ is the largest power of z_1 in g. Thus $\deg(g_k(z_2, \ldots, z_s)) \leq (d-k)$. Therefore if $(s-1)$ values are picked at random from D, such that the probability of each of these values being equal to any $a_i \in D$ is at most ϵ, then by the induction hypothesis we have,

$$\Pr_{a_2, \ldots, a_s \in_r D}[g_k(a_2, \ldots, a_s) = 0] \leq \epsilon(d-k).$$

Having assigned such values for $z_2, \ldots, z_s$ from D, now consider the univariate polynomial $h(x_1) = g(x_1, a_2, \ldots, a_s)$. If $g_k(a_2, \ldots, a_s) \neq 0$ then $h(x_1)$ is a nonzero univariate polynomial in x_1 of degree k. Choosing $a_1 \in_r D$ such that it is equal to any $a_i \in D$ with probability at most ϵ, it follows that $\Pr[h(a_1) = 0 | g_k(a_2, \ldots, a_s) \neq 0]$ is

$$\Pr_{a_1, \ldots, a_s \in_r D}[g(a_1, \ldots, a_s) = 0 | g_k(a_2, \ldots, a_s) \neq 0] \leq \epsilon k.$$

Now let A denote the random event $g(a_1, a_2, \ldots, a_s) = 0$ and B denote the event $g_k(a_2, \ldots, a_s) = 0$, where the a_i are picked independently at random from W as in the statement of the lemma.

Then we have

$$\begin{aligned}
\Pr[A] &= \Pr[A \cap B] + \Pr[A \cap \overline{B}] \\
&= \Pr[B] \cdot \Pr[A|B] + \Pr[\overline{B}] \cdot \Pr[A|\overline{B}] \\
&\leq \Pr[B] + \Pr[A|\overline{B}] \\
&\leq \epsilon(d-k) + \epsilon k = \epsilon d
\end{aligned}$$

as claimed in the statement. ∎

For ease of notation in the sequel we denote the multivariate polynomial $\det(XBY)$ by $f(z_1, \ldots, z_s) \in \mathbb{Z}[z_1, \ldots, z_s]$, where indeterminates in X and Y have been renamed as the z_i's. Our goal is to compute $ord_p(cont(f))$. By factoring out the content of f, we can write $f(z_1, \ldots, z_s) = c \cdot g(z_1, z_2, \ldots, z_s)$, where $cont(g) = 1$. We are interested in computing $ord_p(c)$.

Now, suppose we substitute for z_i a univariate polynomial $a_i(x) \in \mathbb{Z}[x]$, $1 \leq i \leq s$. We claim that $ord_p(c) = ord_p(cont(f(a_1(x), \ldots, a_s(x))))$ if and only if $g(a_1(x), \ldots, a_s(x)) \neq 0 (\text{mod } p)$. It follows because $ord_p(c) \leq ord_p(cont(f(a_1(x), \ldots, a_s(x))))$ and we have $ord_p(c) < ord_p(cont(f(a_1(x), \ldots, a_s(x))))$ if and only if $cont(g(a_1(x), \ldots, a_s(x)))$ is divisible by p.

Now, we define the following finite subset V of $\mathbb{Z}[x]$ from which we will randomly pick the polynomials a_i, and argue that with high probability we have $g(a_1(x), \ldots, a_s(x)) \neq 0 \pmod{p}$. Choose $\beta = 2p + 1$, and let $L = \{1, \ldots, \beta\}$. Let $t = \deg(g)$. Define $V = \{a(x) \mid \deg(a) \leq t - 1 \text{ and coefficient of } a \text{ are in } L\}$.

We now prove a lemma that is a modified version of [Gie95, Lemma 2.6].

Lemma 3.2.13. [Gie95] *Let g be a polynomial in $\mathbb{Z}[z_1, \ldots, z_s]$ of degree t and $cont(g) = 1$. If $(a_1, \ldots, a_s)$ are s elements chosen independently and uniformly at random from V, then*

$$\text{Prob}[g(a_1, \ldots, a_s) = 0 (mod\ p)] \leq t(4/5)^t.$$

Proof. Let Γ be an irreducible polynomial of degree t modulo p. Consider the domain D of Lemma 3.2.12 to be the finite field $\mathbb{Z}[x]/(p, \Gamma)$ of size p^t. Notice that we can consider g to be a nonzero polynomial in $D[z_1, \ldots, z_s]$ (g is surely nonzero modulo p as its content is 1).

Recall the set V defined above. We wish to consider the set V as a subset W of D: an element a of V is already a polynomial of degree at most $(t-1)$ and the coefficients of a have to be reduced modulo p to get the corresponding element in W. Now if we pick an element $a \in V$ independently and uniformly at random, we wish to bound the probability that it is equal to a specific element $a' \in W$. Each coefficient of a when reduced modulo p takes any specific value in $\mathbb{Z}_p$ with probability at most $\lceil \frac{\beta}{p} \rceil \cdot 1/\beta \leq (1/p + 1/\beta) \leq 4/5$. Here a' is a polynomial of degree $(t-1)$ and hence contains t coefficients from $\mathbb{Z}_p$. Therefore for a to be equal to a' after the modulo operation all the t coefficients need to be equal to that of a'. Thus it follows that for any $a' \in W$, the $\text{Prob}_{a \in_r V}[a = a' \pmod{p}] \leq (4/5)^t$.

Now, applying Lemma 3.2.12 to the polynomial $g \in D[z_1, \ldots z_s]$ we immediately get the desired probability bound. ∎

We have the following corollary.

Corollary 3.2.14. *Let B be an $m \times n$ integer matrix of rank m. Also let X and Y be matrices of indeterminates of dimension $m \times m$ and $n \times m$ respectively. Let each indeterminate in X and Y be assigned a value independently and uniformly at random from the set W, defined in the proof of Lemma 3.2.13. If X' and Y' are the resulting matrices, then we have*

$$\text{Prob}[ord_p(cont(\det(XBY))) = ord_p(cont(\det(X'BY')))] \geq 1 - 2m(4/5)^{2m}.$$

Proof. Notice that the degree of $\det(XBY)$ is $2m$. Thus, setting $g = \det(XBY)$, $t = 2m$, and $\epsilon = (4/5)^{2m}$, in Lemma 3.2.12 we obtain the probability bound immediately. ■

Now we return to the problem LCON. As a consequence of the Chinese Remainder Theorem, stated in Theorem 2.2.3, the system $A\mathbf{x} = \mathbf{b} (\text{mod } q)$ is also feasible if and only if $A\mathbf{x} = \mathbf{b} (\text{mod } p_i^{e_i})$ is feasible, for every $1 \leq i \leq k$. Moreover, in Lemma 3.2.3, we have already described a logspace procedure to construct a solution to $A\mathbf{x} = \mathbf{b} (\text{mod } q)$ from the solutions obtained for $A\mathbf{x} = \mathbf{b} (\text{mod } p_i^{e_i})$, for $1 \leq i \leq k$. Thus we focus on checking if the system $A\mathbf{x} = \mathbf{b} (\text{mod } q)$ is feasible, where $q = p^e$ with p being a prime and e is a positive integer.

By applying Proposition 3.2.4, Lemma 3.2.5, and Theorem 3.2.9, we can easily (in logspace) transform the input into a system of linear equations $B\mathbf{z} = \mathbf{b}$, where B and $\mathbf{b}$ are integral and B is full row rank. Now by Lemma 3.2.11 we can further transform this into the problem of checking if $ord_p(cont(\det(XBY))) = ord_p(cont(\det(X[B; \mathbf{b}]Z)))$. Continuing further, we apply Corollary 3.2.14. More precisely, we consider the domain of univariate polynomials W mentioned in Corollary 3.2.14. Pick polynomials independently and uniformly at random from W and substitute them for the indeterminates in X and Y to obtain matrices X' and Y' respectively. It is easy to see that $\det(X'BY')$ is a polynomial in x of degree $2m(2m-1)$. Let $\sum_{i=0}^{2m(2m-1)} \mu_i x^i$ be this polynomial. Here the size of each μ_i is at most $m' = O(me \log p)$, which is polynomially bounded in the size of the given input.

To retrieve each μ_i, where $0 \leq i \leq 2m(2m-1)$, we substitute $2^{2m'}$ for the indeterminate x in the matrix B. After this substitution step, it is easy to note that $\det(X'BY')$ is an integer whose length is polynomially bounded in the size of the given input. Moreover, for $0 \leq i \leq 2m(2m-1)$, every bit of μ_i occurs in this integer and also has a unique index. Thus a logspace machine can compute the index of all the bits that form each μ_i, where $0 \leq i \leq 2m(2m-1)$, and hence retrieve these coefficients with access to a GapL oracle. We also have p^e to be specified in unary. Therefore in logspace the algorithm can also keep track of the highest power of p that divides $\mu_0, \ldots, \mu_i$ as they are computed. Repeating this step for all coefficients we can compute $ord_p(cont(\det(X'BY')))$, which is

correct with high probability, as proved in Corollary 3.2.14. We can similarly compute $ord_p(cont(\det(X[B;\mathbf{b}]Z))$. Final step involves comparing the two values and to output the system is feasible over $\mathbb{Z}_{p^e}$ if they are equal. Otherwise if the values are unequal, we output that the system $A\mathbf{x} = \mathbf{b}$ is not feasible over $\mathbb{Z}_{p^e}$.

We now analyze the probability of error in the above randomized algorithm. Assume that we are given $(A, \mathbf{b}, p^e)$ as input such that $A\mathbf{x} = \mathbf{b}(\text{mod } p^e)$ is feasible. The algorithm described above returns the system is feasible if the randomized algorithm of Corollary 3.2.14 computes correct values of both $ord_p(cont(\det(XBY)))$ and $ord_p(cont(\det(X[B;\mathbf{b}]Z)))$. Thus by Corollary 3.2.14 and the union bound, the error probability is bounded by $4m(4/5)^{2m}$. Therefore, the algorithm outputs $(A, \mathbf{b}, p^e) \in$ LCON with probability at least $1 - 4m(4/5)^{2m}$ in this case.

Conversely, suppose $A\mathbf{x} = \mathbf{b}(\text{mod } p^e)$ is not feasible. Then $ord_p(cont(\det(XBY)))$ and $ord_p(cont(\det(X[B;\mathbf{b}]Z)))$ are different. Again by Corollary 3.2.14 and the union bound the error probability is bounded by $4m(4/5)^{2m}$ implying that the algorithm outputs $(A, \mathbf{b}, p^e) \notin$ LCON with probability at least $1 - 4m(4/5)^{2m}$.

We can amplify the success probability by repeating the algorithm polynomially many times and taking the majority of the outcomes (for example, refer [MR95, Chapter 4]). The error probability can be reduced to a chosen inverse exponential fraction using Chernoff bound (see Theorem 2.3.3 in Chapter 2). In particular, we can amplify the success probability so that most random strings will work as the correct advice string for the underlying $\mathrm{L}^{\mathrm{GapL}}$ computation (notice that for each fixed random string the algorithm performs have an $\mathrm{L}^{\mathrm{GapL}}$ computation). This shows that LCON is in BP·NC^2. More specifically, LCON is in $\mathrm{L}^{\mathrm{GapL}}$/poly for random advice strings.

Theorem 3.2.15. *The problem* LCON *is in* BP·NC^2*, and also in* $\mathrm{L}^{\mathrm{ModL}}$/poly.

Note 2. As inputs for the problem LCON, we are given each prime power $p_i^{e_i}$ dividing q in unary, where $1 \leq i \leq k$. In the algorithm given above we keep track of the highest power of p that divides each μ_j, where $0 \leq j \leq 2m(2m-1)$. Since the size of each μ_j is polynomially bounded in the size of the input, the exponent that we need to keep track of is of size atmost $O(\log e_i)$, where $0 \leq j \leq 2m(2m-1)$, and $1 \leq i \leq k$. Therefore we can actually relax the definition of LCON: for the result in Theorem 3.2.15 it is sufficient to assume that p_i and e_i are in unary, rather than requiring $p_i^{e_i}$ is in unary, where $1 \leq i \leq k$.

We also show that LCON is hard for $\mathrm{L}^{\mathrm{ModL}}$ under logspace Turing reductions.

Theorem 3.2.16. LCON *is logspace many-one hard for* ModL.

Proof. A language $L \in$ ModL can be defined by a GapL function f and an FL function g that outputs a prime power p^e in unary, so that $x \in L$ if and only if $f(x) \not\equiv 0(\text{mod } |g(x)|)$.

Since the integer determinant is hard for GapL under logspace many-one reductions, there is a many-one reduction that maps $x \mapsto A \in \mathbb{Z}^{m \times m}$ such that $f(x) = \det(A)$ for all strings x. Thus, the condition for membership in L is $\det(A) \not\equiv 0 (\mathrm{mod}\ |g(x)|)$. But this is equivalent to the non-singularity of A over the ring $\mathbb{Z}_{|g(x)|}$. In other words $x \in L$ if and only if there is a matrix $X \in \mathbb{Z}_{|g(x)|}^{m \times m}$ such that $AX \equiv I\ (\mathrm{mod}\ |g(x)|)$. However in logspace, $AX \equiv I\ (\mathrm{mod}\ |g(x)|)$ can be expressed as a system of linear congruences $(A', \mathbf{b}, g(x))$ with X being a matrix of indeterminates. We can then check if $(A', \mathbf{b}, g(x))$ is in LCON and hence determine if $x \in L$. Thus L is logspace many-one reducible to LCON. ■

3.2.2 A Conditional Uniform Upper Bound for LCON

By derandomizing the randomized algorithm for LCON shown in Theorem 3.2.15 under a suitable hardness assumption, we show that we can obtain membership of LCON in $\mathrm{L}^{\mathrm{ModL}}$.

Recent work on derandomization [ARZ99, KvM02], based on [NW94], present techniques to derandomize space-bounded randomized algorithms under suitable hardness assumptions. The BP·NC^2 upper bound obtained for LCON in Theorem 3.2.15 is a typical example of such a problem that can be derandomized under a hardness assumption. In Theorem 3.2.15, we showed that LCON is also in $\mathrm{L}^{\mathrm{ModL}}$/poly. Furthermore, as observed, a random advice string is good with high probability for the $\mathrm{L}^{\mathrm{ModL}}$ (respectively the BP·NC^2 circuit). Assuming that there is a language in DSPACE(n) that does not have circuits of size $2^{\epsilon n}$ for all but finitely many n, we can derandomize the algorithm of Theorem 3.2.15 to obtain an $\mathrm{L}^{\mathrm{ModL}}$ for LCON.

Our method for obtaining these results is completely analogous to the results of Allender *et al.* in [ARZ99], which in turn is based on results of [KvM02]. We first recall some definitions and terminology.

Definition 3.2.17. *We define a pseudorandom generator to be a family of functions $G_n : \{0,1\}^{s(n)} \to \{0,1\}^n$, for $n \in \mathbb{N}$, $s : \mathbb{N} \to \mathbb{N}$ with $s(n) < n$, such that for any n-input circuit C of size n we have,*

$$|\mathrm{Prob}[C(y) = 1] - \mathrm{Prob}[C(G_n(x)) = 1]| < (1/n),$$

where x, and y are independently and uniformly distributed over $\{0,1\}^{s(n)}$, and $\{0,1\}^n$ respectively. Here the function $s(n)$ is the seed length of the pseudorandom generator G_n.

Assuming the existence of a language $L \in \mathrm{DSPACE}(n)$ with average-case hardness $2^{\epsilon n}$ for some $\epsilon > 0$, based on [KvM02] the authors in [ARZ99] construct a pseudorandom generator computable in deterministic logspace.

The following result from [ARZ99] summarizes what we require. Further details can be found in [ARZ99, Section 5.2].

Theorem 3.2.18. [ARZ99, Theorem 5.5] *Let $L \in \mathrm{DSPACE}(n)$ such that for some constant $\epsilon > 0$ and all but finitely many n, no n-input circuit C of size at most $2^{\epsilon n}$ accepts exactly strings of length n in L. Then there exists a function (pseudorandom generator) $G_n : \{0,1\}^{k \log n} \to \{0,1\}^n$ computable in logspace such that if C is a circuit of size at most n we have*

$$|\mathrm{Prob}[C(y) = 1] - \mathrm{Prob}[C(G_n(x)) = 1]| < (1/n),$$

where $k \geq 1$ is a constant and x and y are independently and uniformly distributed over $\{0,1\}^{k \log n}$ and $\{0,1\}^n$ respectively.

Using the above theorem it is shown in [ARZ99] that the perfect matching problem is in SPL (which is a logspace counting class contained in NC^2), under the hardness assumption of the theorem.

We now apply Theorem 3.2.18 to prove the conditional derandomization result.

Theorem 3.2.19. *Suppose $L \in \mathrm{DSPACE}(n)$ such that for some constant $\epsilon > 0$ and all but finitely many n, no n-input circuit C of size at most $2^{\epsilon n}$ accepts exactly strings of length n in L. Then* LCON *is in* $\mathrm{L}^{\mathrm{ModL}}$.

Proof. Consider LCON inputs of size n. Suppose the $\mathrm{L}^{\mathrm{ModL}}$/poly algorithm of Theorem 3.2.15 takes advice strings of length $n^{c'}$ for some constant c'. Also, suppose the $\mathrm{L}^{\mathrm{ModL}}$ computation on inputs of size n can be simulated in time $n^{c''}$ for some constant $c'' > 0$. Suppose $1 - \delta$ fraction of the advice strings are correct advice strings for a suitably small δ. Let $c = \max\{c', c''\}$. Thus, on a length n LCON input x, the $\mathrm{L}^{\mathrm{ModL}}$/poly algorithm can be simulated by a circuit of size n^c that takes as input, apart from x, a random advice string of length n^c. Clearly, under the hardness assumption, the output of the pseudorandom generator G_{n^c} can be used as the advice string. The error probability can change to at most $\delta + n^{-c}$. It follows that for a suitably chosen δ the majority vote on all the pseudorandom strings as advice strings will give the correct answer.

To put it together, for inputs of length n the $\mathrm{L}^{\mathrm{ModL}}$ algorithm for LCON will cycle over all seeds of length $kc \log n$ and use the output of G_{n^c} as the advice string in the $\mathrm{L}^{\mathrm{ModL}}$/poly algorithm. It will keep counters for the yes and no answers to take the majority vote. Since G_{n^c} is computable in space $O(\log n)$, the overall computation is in $\mathrm{L}^{\mathrm{ModL}}$. ∎

3.3 Constructing Solutions for Feasible Instances

We recall the definition of LCONX. Given a matrix $A \in \mathbb{Z}^{m \times n}$, a column vector $\mathbf{b} \in \mathbb{Z}^m$, and a positive integer $q = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$ given by its prime factorization, such that each $p_i^{e_i}$ is tiny, the problem is to find a solution to $A\mathbf{x} = \mathbf{b} (\text{mod } q)$ if the system is feasible. We show in this section that LCONX is in $\text{FL}^{\text{ModL}}/\text{poly}$. Our approach is as follows. By the Chinese Remainder Theorem and Lemma 3.2.3, we can compute a solution to $A\mathbf{x} = \mathbf{b} (\text{mod } q)$ from solutions to $A\mathbf{x} = \mathbf{b} (\text{mod } p_i^{e_i})$, $1 \leq i \leq k$, in logspace. Thus it suffices to consider the case when q is a prime power p^e given in unary.

We will find a solution to a feasible system $A\mathbf{x} = \mathbf{b} (\text{mod } p^e)$ using the Isolating Lemma. Every n-dimensional vector over $\mathbb{Z}_{p^e}$ is assigned a weight according to a randomly picked weight function w. If $\mathcal{F}$ is the set of all solutions to $A\mathbf{x} = \mathbf{b} (\text{mod } p^e)$, then by the Isolating Lemma, with high probability there exists a unique minimum weight vector in $\mathcal{F}$.

The process of finding the minimum weight solution to $A\mathbf{x} = \mathbf{b} (\text{mod } p^e)$ crucially depends on the way in which we define the weight functions. We first state and prove the Isolating Lemma in a form suited for LCONX and then show how we use it to find a solution to $A\mathbf{x} = \mathbf{b} (\text{mod } p^e)$.

This version of the Isolating Lemma is based on an article by Klivans and Spielman [KS01, Lemma 4]. Let $S = \{1, \ldots, n\}$ denote the indices of any solution vector to the system $A\mathbf{x} = \mathbf{b} (\text{mod } p^e)$. Also let $w : S \to \{1, \ldots, 2np^{2e}\}$ be a *weight function* that assigns a weight to each index. Then the weight $w(\mathbf{x})$ of a n-dimensional vector $\mathbf{x} = (x_1, \ldots, x_n)$ is defined as $w(\mathbf{x}) = \sum_{i=1}^{n} w(i) x_i$ (for notational convenience we also denote $w(i)$ by w_i), where the entries x_i of $\mathbf{x}$ are treated as integers in the range $\{0, \ldots, p^e - 1\}$. When the weights are not yet assigned to the indices in S, the expression $w(\mathbf{x})$ is a linear form in the n variables $w_i, 1 \leq i \leq n$. We are interested in those linear forms whose coefficients form a solution to $A\mathbf{x} = \mathbf{b} (\text{mod } p^e)$.

Lemma 3.3.1. (Isolating Lemma)[KS01] *Let C be the collection of linear forms in n variables $w_1, \ldots, w_n$ with coefficients in the range $\{0, \ldots, p^e - 1\}$. If $w_1, \ldots, w_n$ are chosen independently and uniformly at random from $\{1, \ldots, 2np^{2e}\}$, then the linear form having minimum weight is unique with probability at least $1/2$.*

Proof. We say that an index i ambiguous, if there exists two forms in C with different coefficients for w_i having the same minimum weight. Since all the forms in C are distinct, if more than one form achieve the minimal weight, then some index will be ambiguous. We show that for any given i, the property of i being ambiguous is at most $1/(2n)$. Thus the probability that there exists such an ambiguous element is at most $1/2$.

Assume that we have assigned values from the range $\{1, \ldots, 2np^{2e}\}$ to all variables, except w_i for some $1 \leq i \leq n$. Then each linear form in C becomes a linear polynomial in w_i with the constant term depending on the values set for w_j, where $1 \leq j \leq n$ and $i \neq j$. We group these polynomials by coefficients of w_i into at most p^e classes. It is clear that the polynomial with the smallest constant term can achieve the least weight when a weight is assigned to w_i. Let this polynomial be the representative for this class.

We would have w_i to be ambiguous if and only if representatives of two different classes have the same weight when value is assigned to w_i and this weight is the minimum among the weights of the set of representatives. Number of possible values that when assigned to w_i could make the above event to happen is at most $p^e(p^e-1)/2$. But however, the weights to w_i are assigned from the interval $\{1, \ldots, 2np^{2e}\}$, which is sufficiently large. Consequently, we have Prob[the index i is ambiguous with respect to the random weight function w] $\leq 1/(2n)$. Therefore, the probability that there exists some element $1 \leq i \leq n$ that is ambiguous with respect to the weight function w is $\leq n(1/(2n)) = 1/2$. This completes the proof. ∎

Suppose the instance $A\mathbf{x} = \mathbf{b}(\text{mod } p^e)$ is feasible. We use the Theorem 3.3.1 to construct a solution to this system as follows. The following proposition is easy to prove.

Proposition 3.3.2. *Any n-dimensional vector $\mathbf{x} \in \mathbb{Z}_{p^e}^n$ is a solution to $A\mathbf{x} = \mathbf{b}(mod\ p^e)$ if and only if it is a solution to the system $(p^lA)\mathbf{x} = (p^l\mathbf{b})(mod\ p^{e+l})$, where l is a positive integer.*

Theorem 3.3.3. LCONX *is in* BP·NC^2. *More precisely, there is an* FL$^{\text{ModL}}$/poly *algorithm for* LCONX *for which a randomly picked advice string is correct with high probability.*

Proof. The proof will apply the Isolating Lemma proved in Theorem 3.3.1. Let C denote the set of linear forms in n variables such that the coefficient vectors formed from each of these linear forms are a solution to the system $A\mathbf{x} = \mathbf{b}(\text{mod } p^e)$. Clearly the variables w_i in these forms describe the weight functions to be assigned values independently and uniformly at random from $\{1, \ldots, 2np^{2e}\}$. Now it follows from Proposition 3.3.2 that $\mathbf{x}$ is a solution to $A\mathbf{x} = \mathbf{b}(\text{mod } p^e)$ if and only if $\mathbf{x}$ is a solution to $(p^lA)\mathbf{x} = (p^l\mathbf{b})(\text{mod } p^{e+l})$. Let $A' = p^lA$ and $\mathbf{b}' = p^l\mathbf{b}$. Here we choose l as the smallest integer such that $p^{e+l} > 2n^2p^{3e}$. It is easy to compute such a l in logspace.

Let w be a randomly picked weight function. Let A'' be the $(m+1) \times n$ matrix obtained from A' by including as $(m+1)^{st}$ row the vector $(w(1), \ldots, w(n))$ formed from the weight function w. Clearly the weight $w(\mathbf{x})$ of any solution vector $\mathbf{x}$ lies between 1 and $2n^2p^{3e}$. Let s be an integer between 1 and $2n^2p^{3e}$ be a candidate value for the minimum

weight of a solution. Correspondingly, let $\mathbf{b}''$ be the $(m+1)$-dimensional column vector formed from $\mathbf{b}'$ by including s as the $(m+1)$ row of $\mathbf{b}'$.

By the choice of l, $\mathbf{x}$ is a solution to $A''\mathbf{x} = \mathbf{b}''(\text{mod } p^{e+l})$ if and only if $w(\mathbf{x}) = s$ and $\mathbf{x}$ is a solution to $A\mathbf{x} = \mathbf{b}(\text{mod } p^e)$. In particular, if $A\mathbf{x} = \mathbf{b}(\text{mod } p^e)$ has a unique solution of weight s then $A''\mathbf{x} = \mathbf{b}''(\text{mod } p^{e+l})$ has a *unique* solution which must be of weight s. Now we can retrieve the entries of such a solution $\mathbf{x}$ to $A''\mathbf{x} = \mathbf{b}''(\text{mod } p^{e+l})$ by querying the oracle for LCON. To compute x_i, we try setting $x_i = j$ for $0 \leq j \leq p^e - 1$ and check if the resulting system $A''\mathbf{x} = \mathbf{b}''(\text{mod } p^{e+l})$ with $x_i = j$ is feasible using LCON as oracle. If so, then we can output x_i. Clearly, we can find the entire vector $\mathbf{x}$.

As a consequence of the Isolating Lemma of Theorem 3.3.1, there exists a unique solution of minimum weight with probability at least 1/2. Thus we can apply the above method to try and solve $A''\mathbf{x} = \mathbf{b}''(\text{mod } p^{e+l})$ for each candidate value of s in the range $1 \leq s \leq 2n^2p^{3e}$. In logspace we can check if the output vector $\mathbf{x}$ is indeed a solution. The Isolating Lemma guarantees that with probability 1/2 the algorithm will succeed for some value of s. Finally, notice that the algorithm is a logspace base machine with access to LCON as oracle, with the weight function w chosen at random. By standard probability amplification techniques, we obtain the FL^{ModL}/poly upper bound for LCONX with a suitably chosen weight function as the advice string. ■

In Section 3.2.2, we obtained a uniform upper bound L^{ModL} for LCON based on the hardness assumption for a language in DSPACE(n). It is easy to observe that essentially the same assumption will yield an FL^{ModL} upper bound for LCONX.

Theorem 3.3.4. *Suppose $L \in$ DSPACE(n) such that for some constant $\epsilon > 0$ and all but finitely many n, no n-input circuit C of size at most $2^{\epsilon n}$ accepts exactly strings of length n in L. Then* LCONX *is in* FL^{ModL}.

3.4 Computing a Spanning Set for the Nullspace

We recall the definition of LCONNULL from Chapter 1. Given a matrix $A \in \mathbb{Z}^{m \times n}$, and a positive integer q with its prime factorization $q = p_1^{e_1}p_2^{e_2}\cdots p_k^{e_k}$, such that each $p_i^{e_i}$ is tiny (i.e. given in unary), the problem LCONNULL is to compute a spanning set for the nullspace of the mapping represented by the matrix A. In other words, we want to compute a spanning set for the $\mathbb{Z}$-module $\{\mathbf{x} \in \mathbb{Z}^n | A\mathbf{x} \equiv 0(\text{mod } q), A \in \mathbb{Z}^{m \times n}\}$.

As in the case of LCON and LCONX, we show that FL^{ModL}/poly upper bound holds for LCONNULL also. To solve LCONNULL, we apply Chinese Remainder Theorem stated in Theorem 2.2.3 of Chapter 2 as follows. We obtain a basis for the null space of the mapping represented by A over $\mathbb{Z}_{p_i^{e_i}}$, for each $1 \leq i \leq k$. Then using these sets we obtain

a basis for the null space of the mapping represented by A over $\mathbb{Z}_q$ using the construction as presented in Lemma 3.2.3. We present details of this later in the section.

Lemma 3.4.1. *Given a matrix $A \in \mathbb{Z}^{m \times n}$ and a prime power p^e in unary, the problem of computing a basis for the nullspace of the mapping represented by the matrix A over $\mathbb{Z}_{p^e}$ is in* $\mathrm{FL}^{\mathrm{ModL}}/\mathrm{poly}$.

Proof. Let the given matrix be $A = (A_1, \ldots, A_n) \in \mathbb{Z}^{m \times n}$. For each $1 \leq i \leq n$, we find the smallest nonzero $\alpha_i \in \mathbb{Z}_{p^e}$ such that the system $A\mathbf{x}_i = 0 (\mathrm{mod}\ p^e)$ has a solution using LCON as a subroutine. If one were to exist, use LCONX to compute such a solution $\mathbf{x}_i$ and output $\mathbf{x}_i = (0, \ldots, 0, \alpha_i, x_{i+1}^{(i)}, \ldots, x_n^{(i)})$. If no such α_i exists, then we output $\mathbf{x}_i = 0$.

Consider any $\mathbf{x}_i \in \mathbb{Z}_{p^e}^n$ obtained from the above procedure, where $1 \leq i \leq n$. Clearly $\mathbf{x}_i$ lies in the null space of the module $\{\mathbf{x} \in \mathbb{Z}^n | A\mathbf{x} \equiv 0 (\mathrm{mod}\ p^e),\ A \in \mathbb{Z}^{m \times n}\}$, where $1 \leq i \leq n$, and the first $(i-1)$ entries of $\mathbf{x}_i$ are zero. The following claims are also easy to observe.

Claim 3.4.2. *For any $1 \leq i \leq n$, the vector $\mathbf{x}_i$ obtained above is nonzero if and only if $\alpha_i \neq 0$ which holds if and only if there exists indexes $(i+1) \leq j_1 \leq \cdots \leq j_l \leq n$, and nonzero scalars $\beta_{j_1}, \ldots, \beta_{j_l} \in \mathbb{Z}_{p^e}$ such that $(\alpha_i A_i + \beta_{j_1} A_{j_1} + \cdots + \beta_{j_l} A_{j_l}) = 0 (mod\ p^e)$.*

Proof of Claim. It is clear from the procedure outlined above that $\mathbf{x}_i$ is nonzero if and only if $\alpha_i \neq 0$. For the other equivalence if nonzero scalars $\beta_{j_1}, \ldots, \beta_{j_l} \in \mathbb{Z}_{p^e}$ exist such that $(\alpha_i A_i + \beta_{j_1} A_{j_1} + \cdots + \beta_{j_l} A_{j_l}) = 0 (\mathrm{mod}\ p^e)$, for $(i+1) \leq j_1 \leq \cdots \leq j_l \leq n$, then a nonzero $\mathbf{x}_i$ always exists satisfying $A\mathbf{x}_i = 0 (\mathrm{mod}\ p^e)$. The converse is similar since the existence of any nonzero $\mathbf{x}_i$ guarantees the existence of such scalars and a linear combination of columns whose indexes is greater than or equal to i, that evaluates to 0 in $\mathbb{Z}_{p^e}$. This proves the claim.

Claim 3.4.3. *For any $1 \leq i \leq n$, let $\mathbf{x}_i$ be a nonzero vector obtained from the procedure outlined above and $\mathbf{y} = (0, \ldots, 0, y_i, \ldots, y_n) \in \mathbb{Z}^n$ satisfying $A\mathbf{y} = 0 (mod\ p^e)$. Then there exists $\beta \in \mathbb{Z}_{p^e}$ such that $\beta\alpha_i = y_i (mod\ p^e)$.*

Proof of Claim. Assume that the claim is not true. That is, for none of the $\beta \in \mathbb{Z}_{p^e}$ we have $\beta\alpha_i = y_i (\mathrm{mod}\ p^e)$. In this case, we can always find $\gamma \in \mathbb{Z}$ such that $(\gamma\alpha_i - y_i)$ is nonzero but strictly less than α_i in the ring $\mathbb{Z}_{p^e}$. Since $\mathbf{x}_i$ and $\mathbf{y}$ lie in the null space of the mapping represented by A, any linear combination of $\mathbf{x}_i$ and $\mathbf{y}$ is also in the same null space. Thus $(\gamma\mathbf{x}_i - \mathbf{y})$ is nonzero and lies in the null space of the mapping represented by A. However the i^{th} term of $(\gamma\mathbf{x}_i - \mathbf{y})$ is nonzero but lesser than α_i, which leads to a contradiction. This completes the proof of the claim.

Consider any $\mathbf{y} = (y_1, \ldots, y_n) \in \mathbb{Z}_{p^e}^n$ in the null space of the mapping represented by A. Without loss of generality assume that $y_1 \neq 0$. Then, from the procedure given above,

we have $\mathbf{x}_1 = (x_1^{(1)}, \ldots, x_n^{(1)})$ such that $x_1^{(1)} \neq 0$. It follows from Claim 3.4.3 that y_1 is a multiple of $x_1^{(1)}$ in the ring $\mathbb{Z}_{p^e}$. In other words, there exists some $\beta_1 \in \mathbb{Z}_{p^e}$ such that $y_1 = \beta_1 x_1^{(1)}$. Thus, $(\mathbf{y} - \beta_1 \mathbf{x}_1)$ also lies in the null space of the mapping represented by A and its first component is zero.

Note that in computing $(\mathbf{y} - \beta_1 \mathbf{x}_1)$, it might turn out that the j^{th} component of $(\mathbf{y} - \beta_1 \mathbf{x}_1)$ becomes 0 in $\mathbb{Z}_{p^e}$, for some $2 \leq j \leq n$. In such cases it can be observed from Claim 3.4.2 that the corresponding $\mathbf{x}_j$ output by the above procedure is also the zero vector. This happens when all linear combinations involving the j^{th} column of A that evaluate to 0 over $\mathbb{Z}_{p^e}$ have at least one column whose index is strictly less than j. Since we have driven the components corresponding to columns whose index is strictly less than j to zero in previous steps, the j^{th} component vanishes as well.

Let $2 \leq i' \leq n$ be the least index such that the corresponding component is nonzero in $(\mathbf{y} - \beta_1 \mathbf{x}_1)$. Then we need to repeat the argument as done above with $\mathbf{x}_{i'}$. That is, there exists a nonzero $\beta_{i'}$ such that $(\mathbf{y} - \beta_1 \mathbf{x}_1 - \beta_{i'} \mathbf{x}_{i'})$ has its first i' components to be zero, and so on. Continuing this argument further, it follows that any $\mathbf{y}$ in the null space of the mapping repesented by A over $\mathbb{Z}_{p^e}$ can always be expressed as a linear combination of the nonzero n-dimensional vectors output by the above procedure. The vectors so obtained are in a lower triangular form with a nice column echelon form like structure.

The main tasks involved in the above procedure are to check feasibility of linear equations over $\mathbb{Z}_{p^e}$ and to obtain solutions for such systems. This step also involves iteratively finding the smallest element $\alpha_i \in \mathbb{Z}_{p^e}$ that occurs as the i^{th} component of any such solution. Since p^e is given in unary, we can keep track of these elements in logspace, and hence compute solutions for such systems in $\mathrm{FL}^{\mathrm{ModL}}/\mathrm{poly}$. ∎

Theorem 3.4.4. LCONNULL $\in \mathrm{FL}^{\mathrm{ModL}}/\mathrm{poly}$.

Proof. As inputs we are given $A \in \mathbb{Z}^{m \times n}$, $\mathbf{b} \in \mathbb{Z}^n$, and a positive integer q in terms of its prime factorization $q = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$, where each $p_i^{e_i}$ is tiny (i.e. given in unary).

We solve this problem using two steps. Firstly, obtain a spanning set for the null space of the mapping represented by the matrix A over $\mathbb{Z}_{p_i^{e_i}}$ using Lemma 3.4.1, for $1 \leq i \leq k$. Let S_i be the spanning set over $\mathbb{Z}_{p_i^{e_i}}$ obtained from the above step. Using constructions similar to the one in Lemma 3.2.3, which is based on the Chinese Remainder Theorem stated in Theorem 2.2.3 of Chapter 2, we then construct a spanning set S for the null space of the mapping represented by A over $\mathbb{Z}_q$ from S_i, where $1 \leq i \leq k$.

For $1 \leq i \leq k$, let $r_i = q/(p_i^{e_i})$. Then $\gcd(r_i, p_i^{e_i}) = 1$, and so there exists integers α_i, β_i, such that $\alpha_i r_i + \beta_i p_i^{e_i} = 1$. Then, for every i and any $\mathbf{y} \in \mathbb{Z}^n$, we have $A\mathbf{y} = 0 (\mathrm{mod}\ p_i^{e_i})$ if and only if $Ar_i\mathbf{y} = 0 (\mathrm{mod}\ q)$. Thus $S_i' = \{r_i \mathbf{y} | \mathbf{y} \in S_i\}$ is contained in the null space of the mapping represented by A over $\mathbb{Z}_q$.

We claim that the set $S = \cup_{i=1}^{k} S_i'$ spans the nullspace of the mapping represented by A over $\mathbb{Z}_q$. To see this, suppose $\mathbf{z} \neq 0$ is in the nullspace so that $A\mathbf{z} = 0 (\text{mod } q)$. Let $\mathbf{z}_i = \mathbf{z} (\text{mod } p_i^{e_i})$. Then, $A\mathbf{z} = A\mathbf{z}_i = 0 (\text{mod } p_i^{e_i})$ and so $\mathbf{z}_i$ is in the $\mathbb{Z}_{p_i^{e_i}}$ span of the elements in S_i. By the Chinese Remainder Theorem, we have $\mathbf{z} = \sum_{i=1}^{k} r_i \alpha_i \mathbf{z}_i (\text{mod } q)$ since $\mathbf{z} = \sum_{i=1}^{k} r_i \alpha_i \mathbf{z}_i (\text{mod } p_i^{e_i})$, for each $1 \leq i \leq k$.

Since $r_i \mathbf{z}_i \in S_i'$, it follows that $\mathbf{z}$ is in the $\mathbb{Z}_q$ span of $\cup_{i=1}^{k} S_i'$. Thus the set $\cup_{i=1}^{k} S_i'$ spans the null space of the mapping represented by the matrix A over $\mathbb{Z}_q$. Since S is computable in logspace from the sets $S_i, 1 \leq i \leq k$ it easily follows that S is computable in $\text{FL}^{\text{ModL}}/\text{poly}$ such that a random advice string is correct with high probability. ■

As in the case of LCON and LCONX it can be observed that LCONNULL is hard for ModL under logspace Turing reductions. To prove this recall the proof of the hardness of LCON given in Theorem 3.2.16. To show that any $L \in \text{ModL}$ reduces to LCONNULL we had to check if $\det(A) \not\equiv 0 (\text{mod } |g(x)|)$ which is true if and only if the null space of A contains only the all 0 vector. In other words we need to check if $(A, 0, q) \in \text{LCONNULL}$. Again from Theorem 3.2.16 it follows that the above reduction is logspace computable and hence we have the following.

Theorem 3.4.5. LCONNULL *is logspace many-one hard for* ModL.

Similar to the derandomization results obtained for LCON and LCONX it is easy to observe the following.

Theorem 3.4.6. *Suppose* $L \in \text{DSPACE}(n)$ *such that for some constant* $\epsilon > 0$ *and all but finitely many* n*, no* n*-input circuit* C *of size at most* $2^{\epsilon n}$ *accepts exactly strings of length* n *in* L*. Then* LCONNULL *is in* FL^{ModL}.

3.5 Solving Linear Equations over a Finite Ring

As a natural generalization of LCON, we consider the problem of solving a system of linear equations over a finite ring R. We assume that the input ring R is given explicitly by its addition (denoted by $+$) and multiplication (denoted by concatenation) tables. It follows from the fundamental theorem of finite abelian groups [Her64], that any additive abelian group can be decomposed as a direct sum of cyclic subgroups, each of prime power order. Since the ring R under $+$ is an abelian group, the above result holds true for $(R, +)$ also. Thus we have $(R, +) = C_1 \oplus \cdots \oplus C_r$, where each C_i is a cyclic group of prime power order and $1 \leq i \leq r$. We are also given the elements of R in an explicit manner as a part of the input. Thus, if the number of elements in R is n, we can obtain the prime factorization of $n = p_1^{e_1} \cdots p_r^{e_r}$ in logspace, where each $p_i^{e_i}$ is a distinct prime

power with $1 \leq i \leq r$. Once the prime powers have been computed, in logspace we can make a brute force search in $(R, +)$ and identify elements of order $p_i^{e_i}$ which are in fact generators of the cyclic group C_i mentioned above, with $1 \leq i \leq r$. It then folllows that, any element of R is a linear combination of the generators of C_i obtained from the above step, where the coefficients in this linear combination are arbitrary integers.

When the input is presented in this form, we show that this problem in fact reduces to solving several instances of LCON all of which have to be true for the given system of linear equations to have a solution over R. This reduction is computable by a $\mathrm{L}^{\mathrm{ModL}}$ machine and since LCON is in $\mathrm{L}^{\mathrm{ModL}}/\mathrm{poly}$, the problem of solving a system of linear equations over a finite ring is also in $\mathrm{L}^{\mathrm{ModL}}/\mathrm{poly}$.

We now study the complexity of the following general problem: Given as input a finite ring R with unity and a system of linear equations $A\mathbf{x} = \mathbf{b}$, where A is an $m \times n$ matrix and $\mathbf{b}$ is an m-dimensional column vector over R, test if there is a solution for $\mathbf{x}$ over R. Here we assume that R is given by its addition (denoted by $+$) and multiplication (denoted by concatenation) tables. From the arguments given above, it is clear that the direct sum decomposition of the additive abelian group $(R, +)$, also denoted by R^+, into $C_1 \oplus \cdots \oplus C_r$ is computable in logspace, where each C_i is a cyclic group of prime power order.

Notice that the ring R is small as its size can be encoded in unary in the size of the input. The above problem generalizes the problem of solving $A\mathbf{x} = \mathbf{b}$ modulo p^e, where p^e is tiny, as we can set $R = \mathbb{Z}_{p^e}$. In this section we show that the above problem is logspace reducible to the problem of solving $A\mathbf{x} = \mathbf{b}$ modulo composites q (with tiny prime-power factors). Thus we show that the above problem is also in the class $\mathrm{L}^{\mathrm{ModL}}/\mathrm{poly}$.

Remark 3. Notice that the ring R is not assumed to be commutative. The following example indicates how our claimed reduction is going to work and also motivates our approach: Let $R = M_k(\mathcal{F}_q)$, the ring of $k \times k$ matrices over the finite field $\mathcal{F}_q$. Now, consider linear equations $A\mathbf{x} = \mathbf{b}$ over $M_k(\mathcal{F}_q)$, where A is an $m \times n$ matrix and $\mathbf{b}$ an m-vector over $M_k(\mathcal{F}_q)$. By expanding each entry of $\mathbf{x}$ into a $k \times k$ block of variables (that will take values in $\mathcal{F}_q$), and likewise treating A as an $mk \times nk$ matrix and $\mathbf{b}$ as an $m \times k$ matrix, both over $\mathcal{F}_q$, we can consider the equations $A\mathbf{x} = \mathbf{b}$ as a system of linear equations over $\mathcal{F}_q$. Now, applying ideas from [ABO99], we can easily see that testing feasibility of this system is in $\mathrm{L}^{\mathrm{GapL}}$.

We proceed to show that the idea in the above remark can be extended to handle any finite ring R with unity, and reduce it to LCON.

Let $|R| = n$ and $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$ be the prime factorization of n. As R is an abelian group under addition, by the fundamental theorem of finite abelian groups, $(R, +)$ can

be written as a direct sum of its Sylow subgroups. Let R_i denote the p_i-Sylow subgroup of R, $1 \leq i \leq k$. Decomposing the additive group $(R, +)$ into its Sylow subgroups R_i we can write

$$R = R_1 \oplus R_2 \oplus \cdots \oplus R_k.$$

Now, let $x \in R$ and $a \in R_i$. Notice that the (additive) order[3] of xa must divide $p_i^{e_i}$ as $p_i^{e_i} xa$ can be written as $x(p_i^{e_i} a)$, and $p_i^{e_i} a = 0$ since $a \in R_i$. Since $(R, +)$ is an abelian group, R_i is the set of all elements of R whose order is a power of p_i. Thus, $xa \in R_i$. Similarly, $ax \in R_i$. Therefore, each R_i is a two-sided ideal of R. Since R has unity, it follows that $RR_i = R_i R = R_i$ for each i. Furthermore, it is easy to see that for $i \neq j$, $R_i R_j = 0$. This follows because $R_i R_j$ is contained in $R_i \cap R_j$ which contains only the additive identity 0. Putting it together, we can see that the R_i's actually yield a ring decomposition $R = R_1 \oplus R_2 \oplus \cdots \oplus R_k$. Thus, we can express each $x \in R$ uniquely as $x = x_1 + \cdots + x_k$, where $x_i \in R_i$.

There is another crucial property of R_i. Since R has unity 1, the above ring decomposition gives a unique expression for 1 as $1 = a_1 + a_2 + \cdots + a_k$, $a_i \in R_i$.

We claim that $a_i \neq 0$. Furthermore, we also claim that a_i is *not* a zero divisor in the subring R_i. To see this, consider any $y \in R_i$. We can write $y = y \cdot 1 = y(a_1 + \cdots + a_k) = ya_1 + \cdots + ya_k$. Now, since $y \in R_i$, for any $j \neq i$ it holds that $ya_j = 0$. Thus, $a_i = 0$ forces $y = 0$ for all $y \in R_i$ which is a contradiction as R_i is a p_i-Sylow subgroup of R. By the same argument, a_i cannot be a zero divisor of R_i. For, if $ya_i = 0$ for $y \in R_i$ then the above equation forces $y = 0$. We summarize our observations below.

Lemma 3.5.1. *Let R be a finite ring with unity. Then R has the ring decomposition $R = R_1 \oplus R_2 \oplus \cdots \oplus R_k$, where each R_i is a Sylow subgroup of R. Furthermore, each R_i has at least one nonzero element which is not a zero-divisor of R_i.*

Since $R = R_1 \oplus R_2 \oplus \cdots \oplus R_k$ is a direct sum decomposition, it is clear that we can decompose A and $\mathbf{b}$ in the linear system into A_i and $\mathbf{b}_i$ (which are the components of the entries of A and $\mathbf{b}$ in R_i) for each i. Thus, it follows easily that $A\mathbf{x} = \mathbf{b}$ is feasible over R if and only if $A_i\mathbf{x} = \mathbf{b}_i$ is feasible over R_i for each i. Since R is given by its addition table, we can find the ring decomposition of R even in logspace. Thus, the above reduction can be carried out it logspace.

We can henceforth assume that R is of size p^e and we have to test feasibility of $A\mathbf{x} = \mathbf{b}$ over R. Notice that R need not have unity. However, by Lemma 3.5.1 we can assume that R has at least one element which is not a zero-divisor(namely, the element a_i in R_i where $1 = \sum_{i=1}^{k} a_i$).

[3]When we talk of order of an element $a \in R$, we shall mean the order of a as an element of the additive group $(R, +)$. In other words, it is the least positive integer t such that $ta = 0$.

We now give a suitable matrix representation to a finite ring R which has an element that is not a zero divisor where $|R|$ is a prime power p^e. This will be an important step in the reduction of feasibility testing of linear equations over R to linear equations over $\mathbb{Z}_{p^e}$.

In the sequel, we denote the additive abelian group $(R, +)$ by R^+. By the fundamental theorem of finite abelian groups, the abelian p-group R^+ can be expressed as a direct sum of cyclic groups: $R^+ = C_1 \oplus \cdots \oplus C_r$, where each $|C_i| = p^{e_i}$, such that $e_1 \geq e_2 \geq \cdots \geq e_r$, and $e = \sum_{i=1}^{r} e_i$. The tuple $(e_1, \ldots, e_r)$ characterizes the abelian p-group up to isomorphism.

We are interested in describing the endomorphisms of the group R^+ (an endomorphism of R^+ is a group homomorphism from R^+ to R^+). The following theorem [Sho28] shows that each endomorphism of R^+ can be given a matrix representation. To see this we first note that R^+ can be expressed as the direct sum $C_1 \oplus \cdots \oplus C_r$, we can choose an independent generating set for R^+ by picking a generator g_i for each cyclic group C_i in the above direct sum. Thus, the elements of R^+ are of the form $\sum_{i=1}^{r} x_i g_i$, where x_i is an integer modulo p^{e_i} for each i. Hence, R^+ can be identified with the set of integer column vectors $(x_1, x_2, \ldots, x_r)^T$, where x_i is an integer modulo p^{e_i}, and addition of these vectors is done coordinate-wise, where addition in the ith coordinate is modulo p^{e_i}.

Therefore, an endomorphism ψ of R^+ can be described by writing down $\psi(g_i)$ for each i as a linear combination $\sum_{j=1}^{r} h_{ij} g_j$. The $r \times r$ matrix with integral entries h_{ij} will describe an endomorphism. The following theorem [Sho28] characterizes the integral matrices that define endomorphisms of R^+ (The original paper writes $\psi(g_i)$ as a row vector, whereas we write it as a column vector).

Theorem 3.5.2. [Sho28, Satz1] *Let A be an abelian p-group of order p^e of type $(e_1, \ldots, e_r)$. I.e. $A = C_1 \oplus \cdots \oplus C_r$ with $|C_i| = p^{e_i}$ for each i. For $1 \leq i, j \leq r$, define integers μ_{ij} as follows: $\mu_{ij} = 1$ for $i \geq j$ and $\mu_{ij} = p^{e_i - e_j}$ for $i < j$.*

Then an $r \times r$ integral matrix $M = (m_{ij})$ describes an endomorphism of A if and only if $m_{ij} = \mu_{ij} h_{ij}$, for some integer h_{ij}, where m_{ij} is an integer computed modulo p^{e_i} for $1 \leq i, j \leq r$.

As explained in [Sho28], the set of integral matrices defined by Theorem [Sho28] forms a ring $\mathrm{End}(A)$ (the endomorphism ring). The addition and multiplication of two matrices in $\mathrm{End}(A)$ is defined as the usual matrix operation where the entries are computed with the modulo operation prescribed by Theorem 3.5.2: the ijth entry is computed modulo p^{e_i}. It is easy to verify that $\mathrm{End}(A)$ is a ring under these operations.

Now we show that the ring R can be embedded inside $\mathrm{End}(R^+)$. Thus, R is essentially a subring of $\mathrm{End}(R^+)$, which means that we can view the elements of R as $r \times r$ integral matrices.

To every element $a \in R$, we associate the endomorphism $T_a \in \mathrm{End}(R^+)$ defined as $T_a(x) = ax$ for $x \in R^+$. We claim that T_a defines the zero element of $\mathrm{End}(R^+)$ if and only if $a = 0$. To see this, recall that: R has an element a_0 which is not a zero divisor. Thus, if T_a defines the zero endomorphism, $T_a(a_0) = aa_0 = 0$. Since a_0 is not a zero divisor, we have $a = 0$. As an immediate consequence, we have the following lemma (that R can be seen as a subring of $\mathrm{End}(R^+)$).

Lemma 3.5.3. *The homomorphism $\psi : R \longrightarrow \mathrm{End}(R^+)$ defined by $\psi(a) = T_a$, for $a \in R$ is an embedding (i.e. ψ has trivial kernel and is thus 1-1).*

Given R as input by its addition and multiplication tables, we can construct a logspace machine that converts every $a \in R$ into the matrix $T_a \in \mathrm{End}(R^+)$: it follows essentially from the assumption that the decomposition $R^+ = C_1 \oplus \cdots \oplus C_r$ is given as part of the input. Let g_i be a generator for C_i for each i. Thus, we can identify any element $y \in R$ with the corresponding integer vector $\overline{y} = (x_1, \ldots, x_r)$, where $y = \sum x_i g_i$ and x_i is computed modulo p^{e_i}. Now, given $a \in R$, it is easy to see that the jth column of the matrix T_a is the vector $\overline{ag_j}$. Now, a logspace machine can compute $\overline{y}$ for any given $y \in R$. Thus, a logspace machine can compute T_a, given a.

Therefore, without loss of generality, we can assume that the ring R is already given by $r \times r$ matrices denoting elements of $\mathrm{End}(R^+)$, where the additive abelian group R^+ is given by decomposition $R^+ = C_1 \oplus \cdots \oplus C_r$.

Now, consider the system of linear equations $A\mathbf{x} = \mathbf{b}$ over R, where each entry of A and $\mathbf{b}$ is an $r \times r$ integer matrix, and each entry of the column vector $\mathbf{x}$ is an indeterminate that will take values in R. As we did earlier with matrices in $M_n(\mathcal{F}_q)$, we can convert $A\mathbf{x} = \mathbf{b}$ into a system of linear equations modulo prime powers (the main difference is that different equations may be computed modulo different powers of p):

We replace each variable x_i of $\mathbf{x}$ by the linear combination $\sum_{a \in R} y_{ai} T_a$, where $y_{ai} \in \mathbb{Z}_{p^e}$. This ensures that x_i will take values only in R. Thus, A is now an $mr \times nr$ matrix with integer entries. Now, notice that $\mathbf{b}$ is an $mr \times r$ matrix, where the (i,j)th entry in each $r \times r$ block is evaluated modulo p^{e_i}. Thus, corresponding to each entry of the $mr \times r$ matrix $\mathbf{b}$, if it is the (i,j)th entry of an $r \times r$ block, we get a linear equation modulo p^{e_i}. It will assume the form $\sum_{k=1}^{nr} \alpha_j z_j = \beta \pmod{p^{e_i}}$, where the indeterminates z_j are actually appropriate y_{aj}'s and α_j are from the appropriate entries of A. As $p^{e_i} \leq p^e$, the above linear equation is equivalent to $\sum_{k=1}^{nr} p^{e-e_i} \alpha_j z_j = p^{e-e_i} \beta \pmod{p^e}$.

Thus, we have reduced the feasibility of $A\mathbf{x} = \mathbf{b}$ over R to an instance of LCON (modulo a tiny prime power p^e). We can now derive the following.

Theorem 3.5.4. *The problem of testing feasibility of linear equations $A\mathbf{x} = \mathbf{b}$ over a finite R with unity is in $\mathrm{L}^{\mathrm{ModL}}/\mathrm{poly}$, where R is given as input by its addition (denoted*

by +) and multiplication (denoted by concatenation) tables, and the additive abelian group $(R,+)$, denoted R^+ is given as a direct sum $C_1 \oplus \cdots \oplus C_r$, where each C_i is a cyclic group of prime power order.

3.6 Discussion

We had initially believed that LCON is in the uniform class $\mathrm{L}^{\mathrm{ModL}}$, as we claimed in [AV04]. This was based on an observation in [ABO99] about computing ranks of matrices over general commutative rings. Subsequently, it was pointed out to us by Eric Allender and Pierre McKenzie that the notion of rank over rings (such as $\mathbb{Z}_q$, for composite q) is not well defined. Unlike the case of linear equations over fields, there does not seem to be a notion of rank of matrices over rings that can be used to test feasibility of linear equations over rings. In this chapter we find a different approach to the problem, but succeed in proving only the weaker upper bound of $\mathrm{L}^{\mathrm{ModL}}/\mathrm{poly}$. As we show in the next chapter, for several abelian permutation group problems we obtain the same $\mathrm{L}^{\mathrm{ModL}}/\mathrm{poly}$ upper bound.

It is remarked in [ABO99], based on the results of [Gie95], that solving linear Diophantine equations is randomized NC reducible to computing the GCD of a list of integers. With this as a starting point, we have explored the problem of feasibility of linear equations modulo composites. We also consider the feasibility of linear equations over arbitrary rings with unity. Surprisingly, it turns out that, by giving a suitable matrix representation to elements of the arbitrary ring, we can reduce this problem to solving linear equations modulo prime powers.

Specifically, we have shown in this chapter that the problem LCON of testing the feasibility of linear equations modulo composites q (with tiny prime power factors) is in the class $\mathrm{L}^{\mathrm{ModL}}/\mathrm{poly}$. Indeed, under a hardness assumption, it is in $\mathrm{L}^{\mathrm{ModL}}$. As explained in this chapter, we can easily show that finding a solution to an instance of LCON is in the function class $\mathrm{FL}^{\mathrm{ModL}}/\mathrm{poly}$ (which can also be derandomized under the same hardness assumption as used in Theorem 3.2.19). As we show in Section 3.5, it turns out that over arbitrary (even noncommutative) rings with unity the same upper bound holds for the feasibility problem.

We leave open the question if the upper bounds can be improved to $\mathrm{L}^{\mathrm{ModL}}$ without the hardness assumption.

4

Abelian Permutation Group Problems

4.1 Introduction

Research on the algorithmic complexity of permutation group problems has been done extensively for more than three decades. There is in fact a huge library of *efficient algorithms* (algorithms that run in polynomial time) for various problems on permutation groups; see for example the survey by [Luk93, Ser03]. Over the years, one of the main motivations for studying these problems is the connection that they enjoy with a variety of computational problems, most notably the *Graph Isomorphism Problem*. Given two input graphs, the Graph Isomorphism Problem is to determine if there is a relabelling (permutation) of the vertices of one of the input graphs that produces the other. Since this problem is in NP $\cap$ coAM [BDG91, pp. 239 and Theorem 11.5] it is unlikely to be NP-complete as that would imply a collapse of the polynomial-time hierarchy PH to its second level. On the other hand a polynomial-time algorithm has eluded researchers. Our motivation for studying permutation groups problems is complexity theoretic: we seek to precisely characterize these problems using complexity classes. Ideally, we would like to show matching upper bounds and hardness results.

Problems like testing if a permutation is in a given permutation group have efficient NC algorithms and are thus unlikely to be P-complete. Still, to the best of our knowledge, the complexity of permutation groups problems, notably membership testing, has not been shown to be complete for any complexity class. Here, we initiate a study in this direction and provide fairly tight upper and lower bounds for abelian permutation group theoretic problems using logspace counting classes. The problems we consider are from the work of McKenzie and Cook in [MC87]. Moreover [MC87] have also shown these problems to be equivalent to the linear algebraic problems defined in the previous chapter which makes it interesting to study. We first define the problems of interest to us and summarize the results obtained in [MC87] for these problems.

We start by recalling the terminology and notation from Section 2.1 of Chapter 2 to present our results on problems regarding abelian permutation groups. In each problem instance we assume that the input permutations are from $\mathrm{Sym}(\Omega)$, where Ω is a finite set of elements.

AGM: (abelian group membership) Given an abelian permutation group $G = \langle g_1, g_2, \ldots, g_r \rangle$ by a generating set of permutations and a permutation h, we need to determine if $h \in G$.

AISO: (abelian group isomorphism) Given abelian permutation groups $G = \langle g_1, \ldots, g_r \rangle$ and $H = \langle h_1, \ldots, h_s \rangle$ determine if G are H are isomorphic groups.

AORDER: (abelian group order) Given abelian permutation group $G = \langle g_1, \ldots, g_r \rangle$ compute the *prime factorization* of $o(G)$, the cardinality of G.

AGMX: (search version of AGM) This is the search version of AGM in which, given an abelian permutation group $G = \langle g_1, g_2, \ldots, g_r \rangle$ by its generating permutations g_i $(1 \leq i \leq r)$ and a permutation h, we need to determine if $h \in G$ and in such a case, the problem is to find integers t_i where $1 \leq i \leq r$, such that $h = g_1^{t_1} g_2^{t_2} \cdots g_r^{t_r}$.

AINTER: (abelian group intersection) Given abelian permutation groups $G = \langle g_1, \ldots, g_r \rangle$ and $H = \langle h_1, \ldots, h_s \rangle$ the problem is to compute a generating set for $G \cap H$.

AGP: (abelian group presentation) Given an abelian group G by generators $g_1, g_2, \ldots, g_r$ compute integer vectors $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathbb{Z}^r$ which generate the kernel of the onto homomorphism $\phi : \mathbb{Z}^r \longrightarrow G$ defined by $\phi : (t_1, \ldots, t_r) \mapsto g_1^{t_1} \cdots g_r^{t_r}$.

McKenzie and Cook in [MC87] show that the above problems can be classified into four NC^1 Turing-equivalent classes. We summarize their results below. Recall problems LCON, LCONX, and LCONNULL defined in Chapters 1 and 3.

Theorem 4.1.1. [MC87, Theorem 6.8, Proposition 6.13, Theorem 7.10]

1. AGM, AISO *and* AORDER, *and* LCON *are* NC^1 *Turing-equivalent,*
2. AGMX *is* NC^1 *Turing-equivalent to* LCONX,
3. AINTER *is* NC^1 *Turing-reducible to* AGP *and,*
4. AGP *is* NC^1 *Turing-equivalent to* LCONNULL.

McKenzie and Cook in [MC87] showed that LCON, LCONX, and LCONNULL are in NC^3, and hence place the abelian permutation group theoretic problems defined above in NC^3.

We carefully examine the reductions stated above and make minor changes to show that AGM, AISO and AORDER are in fact logspace Turing equivalent. Also, AGM, AISO and AORDER reduce to AINTER by logspace Turing reductions.[1] Then we show that

[1] We note that logspace Turing reducibility is stronger than NC^1-Turing reducibility, mainly because of the Ruzzo-Simon-Tompa oracle access explained in the first chapter.

AGM and AGMX reduce to LCON and LCONX respectively under logspace Turing reductions. From these reductions, it follows that AGP reduces to LCONNULL by a logspace Turing reduction. Now, using Theorems 3.2.15, 3.3.3 and 3.4.4 from Chapter 3, we place the above defined abelian permutation group theoretic problems in $\mathrm{L}^{\mathrm{ModL}}/\mathrm{poly}$. Due to the above mentioned logspace Turing equivalence between these problems, we note that hardness and derandomization results obtained for LCON, LCONX, and LCONNULL in Theorems 3.2.16, 3.2.19, 3.3.4, 3.4.5 and 3.4.6 carry over to these abelian permutation group theoretic problems as well.

4.2 Reductions and Equivalences

We start by examining the reductions between problems on abelian permutation groups shown in [MC87], and observe that such reductions are in fact logspace computable. Recall Definitions 1.2.9, 1.3.9 and 1.3.10 from Chapter 1. It follows from [AO96, ABO99] that the NC^1-Turing reducibility (or even AC^0-Turing reducibility) is potentially more powerful than the logspace-Turing reducibility. This is essentially due to the fact that oracle queries can be nested in the NC^1 oracle circuit implementing the reduction. In other words, the output of a query submitted to an oracle gate in such circuits can be fed as the input of another oracle gate in a higher level. However in the case of logspace Turing machines, we use the Ruzzo-Simon-Tompa oracle access mechanism (recall the discussion following Definition 1.3.6 in Section 1.3.2 of Chapter 1). As a consequence number of queries that the logspace machine can generate is polynomially bounded in the length of the input, and in fact all these queries can be submitted to the oracle in a single step to obtain the corresponding replies from it.

Lemma 4.2.1. *As inputs to the problem* AGM*, let $g_1, \ldots, g_r$ be a set of permutations over n elements from Ω that pairwise commute. Let $G = \langle g_1, \ldots, g_r \rangle$ be the group generated by these permutations. We are also given another permutation h over Ω that commutes with each g_i, where $1 \leq i \leq r$. Then* AGM *logspace Turing reduces to* AISO, AORDER*, and* AINTER*.*

Proof. The permutation h is in the group G if and only if h can be written as a product of permutations in $\{g_1, \ldots, g_r\}$. This holds if and only if the group generated by $\{g_1, \ldots, g_r\}$ and the group generated by $\{g_1, \ldots, g_r, h\}$ is the same, and hence isomorphic which is denoted by $G \cong \langle g_1, \ldots, g_r, h \rangle$. Clearly a logspace machine can output the above two generating sets upon receiving $\{g_1, \ldots, g_r\}$ and h as input. Equivalently, the cardinality of G does not increase even if h is added to the generating set of G, denoted by $o(G) = o(\langle g_1, \ldots, g_r, h \rangle)$. Once again a logspace machine can output $\{g_1, \ldots, g_r\}$ and

$\{g_1, \ldots, g_r, h\}$ which completes the reduction. Also this is true if and only if h lies in G, that is $h \in G \cap \langle h \rangle$. It is once again easy to note that this is also logspace computable. The above equivalences show logspace many-one reductions from AGM to AISO, AORDER, and AINTER. ■

Moreover given any arbitrary abelian permutation group G, it follows from Theorem 2.1.1 (parts 3 and 4) that AISO and AORDER reduce to AISO for Sylow p-subgroups and AORDER for Sylow p-subgroups respectively. The following lemma shows a method to construct generators for the Sylow p-subgroup of an abelian permutation group G given by a set of generators.

Let p be a prime and k be a nonnegative integer. Then let $ord_p(k)$ denote the largest integer l such that p^l divides k. Let $rest_p(k)$ denote $k/(p^{ord_p(k)})$. Clearly, $\gcd(ord_p(k), rest_p(k)) = 1$, for any positive integer k.

Lemma 4.2.2. [MC87, Lemma 3.8] *Let G be an abelian group given by a set of generators $\{g_1, \ldots, g_r\}$. If p is a prime dividing $o(G)$, then the Sylow p-subgroup of G is generated by $\{g_1^{e_1}, \ldots, g_r^{e_r}\}$, where $e_i = rest_p(o(g_i))$ for $1 \leq i \leq r$.*

Proof. Let $S_p = \langle g_1^{e_1}, \ldots, g_r^{e_r} \rangle$. Clearly every generator of S_p is a p-element. Also let $H = \{g \in G | \ \gcd(p, o(g)) = 1\}$. It follows from the definitions of S_p and H that $S_p \cap H = \{1\}$ and p does not divide $o(H)$. Also H is a subgroup of G. Furthermore, since G is abelian S_pH is a subgroup of G. We claim that $G = S_pH$. To prove this it suffices to show that any generator g_i of G is in S_pH, where $1 \leq i \leq r$. Fix g_i and let $d_i = ord_p(o(g_i))$. As d_i and e_i are relatively prime there exists integers s and t such that $se_i + td_i = 1$. Also p does not divide $o(g_i^{td_i})$, since it follows from $(g_i^{td_i})^{e_i} = 1$, that $o(g_i^{td_i})$ divides e_i. Thus $g_i^{td_i} \in H$. However $g_i = g_i^1 = g_i^{se_i + td_i} = g_i^{se_i} g_i^{td_i}$. But $g_i^{se_i} \in S_p$, and $g_i^{td_i} \in H$ from which we have $g_i \in S_pH$. Hence $G = S_pH$. As p does not divide $o(H)$ it follows that all S_p contains all elements of G whose order is a power of the prime p, which means S_p is the unique p-Sylow subgroup of G by Theorem 2.1.1. ■

Thus, given an abelian permutation groups by generators, we can compute in logspace a generating set for each Sylow subgroup of G.

Lemma 4.2.3. *Let G be an abelian group given by a generating set of permutations $\{g_1, \ldots, g_r\}$. Then computing a generating set for a non-trivial Sylow p-subgroup of G for any prime p is in* L.

Proof. It follows from Lemma 4.2.2 only primes dividing the order of any of the generators of G will yield non-trivial Sylow subgroups. Firstly note that the order of any element $g \in G$ in logspace computable. To observe this, we first compute the size of the orbit

of every element in Ω with respect to the permutation g. Since the size of the orbit computed for each element in Ω is logarithmic in the size of g, we can compute the least common multiple (LCM) of the sizes of the orbits, once again in L. This LCM is precisely $o(g)$. Simultaneously we can also compute the prime factors of $o(g)$. It is then clear that the size of any such prime p is $O(\log n)$ where n is the size of the input. Thus if g_i is a generator of G and if $p|o(G)$, then $ord_p(o(g_i))$ and hence $e_i = rest_p(o(g_i))$ are also computable in L.

Using Lemma 4.2.2, it then follows that the Sylow p-subgroup of G is generated by $\{g_1^{e_1}, g_2^{e_2}, \ldots, g_r^{e_r}\}$ from which the result follows. ■

Next, we recall from [MC87, Proposition 6.4] and show that given two abelian p-groups G and H, the problem of checking if G and H are isomorphic, denoted by $G \cong H$, is logspace Turing reducible to AORDER. As a consequence, it follows that AISO is also logspace Turing reducible to AORDER.

Proposition 4.2.4. [Hal59] *Any abelian p-group G is isomorphic to $C_p \times \cdots \times C_p \times C_{p^2} \times \cdots \times C_{p^2} \times \cdots \times C_{p^k} \times \cdots \times C_{p^k}$ for some integer k, where C_{p^i} is the unique cyclic group of order p^i, $(1 \leq i \leq k)$ and C_{p^i} occurs with multiplicity d_i in the above product. Here $o(G) = p^{\sum_{i=1}^{k} i d_i}$. Also, $(d_1, \ldots, d_k)$ is defined as the signature of G and is unique for the group G upto isomorphism.*

Lemma 4.2.5. [MC87, Proposition 6.4] *Let $G = \langle g_1, \ldots, g_r \rangle$ and $H = \langle h_1, \ldots, h_s \rangle$ be abelian p-groups, and let $p^k = max\{o(g_1), \ldots, o(g_r), o(h_1), \ldots, o(h_s)\}$. Then $G \cong H$ if and only if $o(\langle g_1^{p^i}, \ldots, g_r^{p^i} \rangle) = o(\langle h_1^{p^i}, \ldots, h_s^{p^i} \rangle)$, for all $0 \leq i < k$.*

Proof. Let $G_i = \{g^{p^i} | g \in G\}$ and $H_i = \{h^{p^i} | h \in H\}$ for $0 \leq i < k$. From the definitions of G_i and H_i we note that $G_i = \langle g_1^{p^i}, \ldots, g_r^{p^i} \rangle$ and $H_i = \langle h_1^{p^i}, \ldots, h_s^{p^i} \rangle$. If $G \cong H$ then $o(G_i) = o(H_i)$ for each $0 \leq i < k$, since both are isomorphism invariants and there is a one-one and onto mapping from G_i to H_i. We now prove the converse part.

Assume $o(G_i) = o(H_i)$, for all $0 \leq i < k$. We need to show $G \cong H$. From Proposition 4.2.4, it is sufficient to show that the signatures (defined in Proposition 4.2.4) of G and H are the same.

Now, assume $o(G_i) = o(H_i)$ for each $0 \leq i < k$. From the definition of k and Proposition 4.2.4, it follows that the signature of G and that of H each can contain at most k non-zero entries since p^k is the largest order of any element in either G or H.

Let $(d_1, \ldots, d_k)$ and $(e_1, \ldots, e_k)$ denote the signatures of G and H, respectively. Thus, the direct product decomposition of G contains d_2 copies of the cyclic group C_{p^2}. Now, $G_1 = \{g^p \mid g \in G\}$. Notice that if $\langle a \rangle$ is a cyclic group of order p^j in G then $\langle a^p \rangle$ is a cyclic group of order p^{j-1} in G_1 for every $1 \leq j < k$. Thus all d_j occurrences of

C_{p^j} in the signature of G will become d_j occurrences of $C_{p^{j-1}}$ in the signature of G_1 for $1 \leq j < k$. Similarly, all e_j occurrences of C_{p^j} in the signature of H will become e_j occurrences of $C_{p^{j-1}}$ in the signature of H_1. It follows that $(d_2, \ldots, d_k)$ and $(e_2, \ldots, e_k)$ are the signatures of G_1 and H_1, respectively. Likewise, the signature of G_i is $(d_{i+1}, \ldots, d_k)$ and the signature of H_i as $(e_{i+1}, \ldots, e_k)$ for all i.

We now show $G \cong H$ by an induction on k. For $k = 1$ it is trivially true. Assume as induction hypothesis that it is true for abelian p-groups for $k = \ell - 1$. Suppose $k = \ell$ for two abelian p-groups G and H such that $o(G_i) = o(H_i)$, for all $0 \leq i < k$. By the induction hypothesis applied to the groups G_1 and H_1, it follows immediately that $G_1 \cong H_1$. Hence their signatures $(d_2, \ldots, d_k)$ and $(e_2, \ldots, e_k)$ are the same. It remains to show that $d_1 = e_1$. But that follows immediately because $o(G) = o(H)$. This completes the proof. ∎

In the next result we recall another logspace Turing reduction from the AORDER problem for p-groups to AGM given in [MC87]. With this reduction we finally relate AGM, AORDER, AISO and AINTER.

Lemma 4.2.6. [MC87, Proposition 6.6] *Let $G = \langle g_1, \ldots, g_r \rangle$ be a finite abelian group. Then $o(G) = t_1 \cdots t_r$, where t_j is the least positive integer such that $g_j^{t_j} \in \langle g_{j+1}, \ldots, g_r \rangle$ with $1 \leq j \leq r$.*

Proof. We show by induction on r, that there is a unique way to write any element $g \in G$ as $g_1^{s_1} \cdots g_r^{s_r}$, with $0 \leq s_j < t_j$ where $1 \leq j \leq r$. The base case $r = 1$ is clear since G is cyclic. Let us assume the statement to be true for $H = \langle g_2, \ldots, g_r \rangle$. To see that an arbitrary $g \in G$ is expressible in the desired form, consider any expression for g in terms of the generators where the exponent α of g_1 is non-negative. Writing $\alpha = ut_1 + v$ for $0 \leq v < t_1$, and using the expression for $g_1^{t_1}$ in terms of $g_2, \ldots, g_r$, we find that $g = g_1^v h$, for $h \in H$. Hence g is expressible as in the statement.

To see that any $g \in G$ is uniquely expressible, assume that $g = g_1^{\alpha} h_1 = g_2^{\beta} h_2$, with $h_1, h_2 \in H$, and $0 \leq \alpha, \beta < t_1$. If we can show $\alpha = \beta$, it completes the proof since $h_1 = h_2$ and by the induction hypothesis it is representable in a unique way as product of powers of $g_2, \ldots, g_r$. But $g_1^{|\alpha - \beta|} \in H$, and so $|\alpha - \beta|$ cannot be positive by the choice of t_1, which implies $\alpha = \beta$.

From the observations made above any element $g \in G$ is and only if it can be uniquely expressed as a product of powers of the generators of G where the exponents are in the form stated above. This clearly means that the number of elements in G is $t_1 \cdots t_r$. ∎

Using the results proved above, we obtain the following.

Theorem 4.2.7. *The problems* AGM, AORDER *and* AISO *are logspace-Turing equivalent, and logspace-Turing reducible to* AINTER.

Proof. From Lemma 4.2.1, we have AGM reduces to AISO by a logspace Turing reduction. Nextly, it follows from Lemma 4.2.3 that given an abelian group, the set of generators for any of its Sylow p-subgroups can be obtained in logspace. Since every abelian group is a direct product of its Sylow p-subgroups, given an abelian group, we can reduce AISO to checking if its Sylow p-subgroups are isomorphic (due to the same reason given an abelian group G, the problem of computing the order of G reduces to computing the order of Sylow p-subgroups of G). Now using Lemma 4.2.5, we can reduce the the problem of checking if two Sylow p-subgroups are isomorphic to computing the order of an abelian group. The logspace Turing reduction from AORDER to AGM shown in Lemma 4.2.6 now completes the proof that AGM, AISO, and AORDER are logspace Turing equivalent. Moreover we have already shown in Lemma 4.2.1 that AGM is logspace many-one reducible to AINTER which completes the proof. ∎

We now prove upper bounds on the complexity of the problems defined above by showing that AGM and AGMX are logspace Turing reducible to LCON and LCONX respectively. The proof of this reduction also shows that AGP is logspace Turing reducible to LCONNULL. The $\mathrm{L}^{\mathrm{ModL}}$/poly upper bound for LCON, LCONX and LCONNULL proved in Chapter 3 then completes the proof.

Let Ω denote a set containing n elements over which all our permutations are defined. As an input instance for LCON and LCONX, we are given an abelian permutation group G by its generators $\{g_1, \ldots, g_r\}$ and a test permutation h. Following [MC87], we define the homomorphism

$$\psi : \mathbb{Z}^r \to G, \text{ where } \mathbf{y} = (y_1, \ldots, y_r) \mapsto \prod_{1 \leq j \leq r} g_j^{y_j}.$$

Now for AGM, we need to check if there exists $\mathbf{y} \in \mathbb{Z}^r$ such that $\psi(\mathbf{y}) = h$. If such a solution were to exist, we also need to compute one such solution for the problem AGMX. Equivalently, we need to compute $\mathbf{y} \in \mathbb{Z}^r$ such that

$$\alpha^h = \alpha^{\prod_{1 \leq j \leq r} g_j^{y_j}}, \ \forall \alpha \in \Omega.$$

Fix some $\alpha \in \Omega$. Then checking if there is a $\mathbf{y} \in \mathbb{Z}^r$ such that $\alpha^h = \alpha^{\prod_{1 \leq j \leq r} g_j^{y_j}}$ is an instance of the undirected st-connectivity problem in the *operator graph* [Ros93] defined on the points in Ω by the generators of G. In this graph, the vertex set is Ω and (α, β) is an undirected edge if $\alpha^{g_i} = \beta$ or $\alpha = \beta^{g_i}$ for some generator g_i, where $1 \leq i \leq r$. This graph can be generated from G by a logspace machine, and checking if there is a

path from α to α^h can also be done in L [Rei05]. Corresponding to every such α, let us associate the following set of integer vectors:

$$V_\alpha = \{(y_1, \ldots, y_r) \in \mathbb{Z}^r \mid \alpha^h = \alpha^{\Pi_j g_j^{y_j}}\}.$$

By repeatedly solving the above reachability problem we obtain such a $\mathbf{y} = (y_1, \ldots, y_r) \in V_\alpha$ as follows. Let $\Sigma \subseteq \Omega$ be the orbit of α with respect to G. When the generators of G are restricted to Σ, the group generated by these permutations, say H, forms a transitive abelian permutation group over Σ. Therefore, the size of H is $o(\Sigma)$, the cardinality of Σ. As a consequence, the order of each generator g_j when restricted to Σ is small, more precisely bounded by $o(\Sigma)$.

Let $i = 1$. We obtain $\mathbf{y}$ by starting with $g_i^{y_i}$, where $y_i = o(\Sigma) - 1$ initially. Using the logspace algorithm of [Rei05], check if there is an undirected path between $\alpha^{g_i^{y_i}}$ and α^h in the graph defined above. If no such path exists, then we decrement y_i by 1 until $y_i = 0$. If for all $0 \leq y_i \leq o(\Sigma) - 1$ no such path exists, then we output the given permutation h is not in G. Otherwise, if for some $0 \leq y_i \leq o(\Sigma) - 1$ we get a path from $\alpha^{g_i^{y_i}}$ to α^h, then we output y_i and retain $\gamma = \alpha^{g_i^{y_i}}$. Now increment i by 1. In the next step, the graph that we generate on the points in Ω is based on generators $\{g_i, \ldots, g_r\}$ restricted to Σ. In other words, the logspace machine does not include generators $\{g_1, \ldots, g_{i-1}\}$ to define edges in the graph that is generated in the next step. When the above algorithm is repeated for every $1 \leq i \leq n$, we finally end up with a vector $\mathbf{y} = (y_1, \ldots, y_n)$ that lies in V_α. Since the main step involved in each iteration is to generate the undirected graph and to check if there exists a path between two vertices in it, both of which are logspace computable, we observe the entire procedure is computable in L.

Let us call the solution obtained from the algorithm given above, as $\mathbf{b}_\alpha$. Let

$$W_\alpha = \{(y_1, \ldots, y_r) \in \mathbb{Z}^r \mid \alpha = \alpha^{\Pi_j g_j^{y_j}}\}.$$

Firstly, W_α is a group under component wise addition of r-dimensional vectors. This follows since W_α is closed under addition: given $\mathbf{y}_1, \mathbf{y}_2 \in W_\alpha$, we have $\alpha^{\psi(\mathbf{y}_1 + \mathbf{y}_2)} = (\alpha^{\psi(\mathbf{y}_1)})^{\psi(\mathbf{y}_2)} = \alpha$. Additive inverse exists for every element in W_α, that is, for every $\mathbf{y} \in W_\alpha$, we have $(-\mathbf{y}) \in W_\alpha$ and the zero vector is in V_α. Moreover, component wise addition of r-dimensional vectors is also associative, from which it follows that W_α forms a group. Also, V_α is the coset $\mathbf{b}_\alpha + W_\alpha$. This follows since given $\mathbf{z}_1, \mathbf{z}_2 \in V_\alpha$, we have $\mathbf{z}_1 - \mathbf{z}_2 \in W_\alpha$. The proof is similar to the one showing W_α is closed under addition and additive inverses.

We can also find a spanning set of integer vectors for W_α in logspace by repeatedly solving the reachability problem for the undirected graphs defined above in a way similar

to the one used to find $\mathbf{b}_\alpha$. The fact that cardinality of the group G when restricted to Σ, the orbit containing α, is small (in fact equal to $o(\Sigma) \leq o(\Omega)$) is once again used. We summarize the steps involved as a procedure below.

CONSTRUCT VECTOR $(g_1, \ldots, g_r)$
Let Σ be the orbit of α.
for $(i \leftarrow 1$ to $r)$
    Construct the operator graph G on Σ with respect to generators $\{g_i, \ldots, g_r\}$.
    $j \leftarrow o(\Sigma) - 1$.
    while $(j \geq 0)$ **do**
        if $((j = 0)$ or $(\exists$ a path between $\alpha^{g_i^j}$ and α in $G))$ **then**
            $\alpha \leftarrow \alpha^{g_i^j}$.
            Output j.
        endif
        $j \leftarrow j - 1$.
    endwhile
endfor

Note that the above procedure always returns an output since the r-dimensional zero vector trivially satisfies $\alpha = \alpha^{\psi(\mathbf{Z})}$. It can be easily seen that the non-zero vectors we obtain from this procedure form a lower triangular matrix, similar to the column echelon form. Let this matrix be denoted by A_α.

The column vectors of A_α span W_α. The proof of this is similar to the one used in Claim 3.4.3 of Lemma 3.4.1 in Section 3.4 of Chapter 3. We need to use the fact that the topmost nonzero entry in any column of A_α is the smallest integer between 1 and $o(\Sigma)$. Thus if we have any vector in W_α, we can always write it as a linear combination of columns in A_α, for otherwise the minimality of the topmost nonzero entry in some column of A_α will be contradicted.

From the procedures given above, it is clear that entries of $\mathbf{b}_\alpha$ and A_α are computable in logspace. We now recall (minor variants of) propositions from [MC87].

Proposition 4.2.8. [MC87, Proposition 7.5] *Let G be an abelian permutation group given by a set of generators $\{g_1, \ldots, g_r\}$ and let h be a permutation. For any $\mathbf{y} \in \mathbb{Z}^r$, we have $h = \psi(\mathbf{y})$ if and only if there exist vectors $\mathbf{x}_\alpha \in \mathbb{Z}^r$, for each $\alpha \in \Omega$, such that $\mathbf{y} = \mathbf{b}_\alpha + A_\alpha \mathbf{x}_\alpha$.*

Proof. Let $\mathbf{y} = (y_1, \ldots, y_r)$. Then,

$$\begin{aligned} h = \psi(\mathbf{y}) &\iff \alpha^h = \alpha^{\Pi_{1\le j\le r} g_j^{y_j}},\ \forall \alpha \in \Omega \\ &\iff \alpha^{\psi(\mathbf{y})} = \alpha^{\psi(\mathbf{b}_\alpha)},\ \forall \alpha \in \Omega \\ &\iff \alpha = \alpha^{\psi(\mathbf{y})-\psi(\mathbf{b}_\alpha)},\ \forall \alpha \in \Omega \\ &\iff \alpha = \alpha^{\psi(\mathbf{y}-\mathbf{b}_\alpha)},\ \forall \alpha \in \Omega. \end{aligned}$$

Thus for any $\mathbf{y} \in \mathbb{Z}^r$, the equality $h = \psi(\mathbf{y})$ holds if and only if for every $\alpha \in \Omega$, we have $\mathbf{y} - \mathbf{b}_\alpha \in W_\alpha$.

Let us now consider W_α. For $1 \le j \le r$, we denote by W_α^j, the subgroup of W_α consisting of all vectors whose first $(j-1)$ components are 0. From W_α^j, we choose vectors $\mathbf{y}_\alpha^{(j)}$ such that its first $(j-1)$ entries are 0 and the j^{th} entry is positive and minimal among all vectors in W_α^j. Also let $A_\alpha = (\mathbf{y}_\alpha^{(1)}, \mathbf{y}_\alpha^{(2)}, \ldots, \mathbf{y}_\alpha^{(r)})$ denote the $r \times r$ matrix formed by such vectors. Now we prove that any vector in W_α is just a linear combination of columns in A_α. We show by induction on l that the last l columns of A_i generate W_α^{r-l+1}. When $l = 1$, the claim automatically follows since $\mathbf{y}_\alpha^{(r)}$ generates W_α^r. Now for $l > 1$, let $\mathbf{y}$ be any vector in W_α^j, where $j = r - l + 1$. Clearly, $a_{jj}^{(i)}$ divides the j^{th} component of $\mathbf{y}$ exactly, since if not it contradicts the minimality of $a_{jj}^{(i)}$. Hence $\mathbf{y} - u\mathbf{y}_\alpha^{(j)} \in W_\alpha^{j+1}$ for some $u \in \mathbb{Z}$. By induction hypothesis $\mathbf{y} - u\mathbf{y}_\alpha^{(j)}$ is a linear combination of the last l columns of A_α which completes the proof of the claim. As observed before note that V_α is $\mathbf{b}_\alpha + W_\alpha$ which completes the proof of this result. ■

Proposition 4.2.9. [MC87, Proposition 7.6] *Let $q = lcm(o(g_1), \ldots, o(g_r))$. First, if $h \in G$ then the equations, $\mathbf{y} = \mathbf{b}_\alpha + A_\alpha \mathbf{x}_\alpha$ with variables $\mathbf{y}$ and the $\mathbf{x}_\alpha$ mentioned in the previous proposition are solvable modulo q. Second, if $\mathbf{y}, \mathbf{x}_\alpha \in \mathbb{Z}^r$ satisfy $\mathbf{y} = \mathbf{b}_\alpha + A_\alpha \mathbf{x}_\alpha (mod\ q)$ for all $\alpha \in \Omega$, then $\psi(\mathbf{y}) = h$.*

Proof. When $h \in G$, then obviously there exits $\mathbf{x}_\alpha$ such that $\mathbf{y} = \mathbf{b}_\alpha + A_\alpha \mathbf{x}_\alpha$ and the same equation holds modulo q as well. The second part follows from the definition of $\mathbf{b}_\alpha, A_\alpha, \mathbf{x}_\alpha, \mathbf{y}$, and the fact that W_α contains vectors of the form $q\mathbf{z}$, where $\mathbf{z} \in \mathbb{Z}^r$. ■

By rearranging equations $\mathbf{y} = \mathbf{b}_\alpha + A_\alpha \mathbf{x}_\alpha$, we can combine them into a single system of congruences of the form $AX = B (\mathrm{mod}\ q)$ in logspace, where $A \in \mathbb{Z}^{rn \times (rn+r)}$, $B \in \mathbb{Z}^{rn}$ and $X \in \mathbb{Z}^{rn+r}$. Note that q is also computable in logspace since when restricted to the orbit of α, order of any of the generator is small, that is $O(\log n)$ in the size of any permutation. Thus, we can compute the LCM of the orders of these elements in logspace itself. Now, $h \in G$ if and only if there exists a solution to the above congruence. If a solution were to exist, then using the terms occurring in the solution vector we can also construct an

expression for h in terms of the generators of G. This completes the description of a many-one reduction from AGM to LCON, and also from AGMX to LCONX. As already explained, the reduction is logspace computable since the *st*- connectivity problem for undirected graphs is shown to be in L [Rei05]. Summing up the observations made above, and using the upperbounds for LCON and LCONX shown in Theorem 3.2.15 and 3.3.3 of Chapter 3, we obtain the following result.

Theorem 4.2.10. AGM, AISO, AORDER *and* AGMX *are in* $\mathrm{L}^{\mathrm{ModL}}/\mathrm{poly}$.

Given a set of generators $\{g_1, \ldots, g_r\}$ of the group G, we have an onto homomorphism $\psi : \mathbb{Z}^r \to G$ defined as $\psi(\mathbf{x}) = g_1^{x_1} \ldots g_r^{x_r}$, where $\mathbf{x} = (x_1, \ldots, x_r)$. A *relator* of G is any vector $\mathbf{x} \in \mathrm{Ker}\ \psi$. In other words, a relator is a vector $\mathbf{x}$ such that $\psi(\mathbf{x}) = e$, where e is the identity element in G. The problem Abelian Group Presentation (AGP) is to compute a set of relators that span Ker ψ. AGP has been shown to be NC^1-Turing equivalent to LCONNULL by [MC87]. Recall the procedure used to show that AGMX is logspace Turing reducible to LCONX from Theorem 4.2.10. By making some minor modifications to this reduction we can also show a logspace Turing reduction from AGP to LCONNULL. We just need to note that the permutation h is replaced by the identity permutation e, from which it follows that the set V_α becomes W_α. Once a spanning set A_α for W_α has been obtained, for each $\alpha \in \Omega$, we proceed as in Theorem 4.2.10 to reduce the problem to computing solutions for a system of linear equations of the form $AX = 0 (\mathrm{mod}\ q)$, where $A \in \mathbb{Z}^{rn \times (rn+r)}$, $B \in \mathbb{Z}^{rn}$ and $X \in \mathbb{Z}^{rn+r}$. Now using LCONNULL as an oracle, we can obtain a basis for the solutions of above system in logspace. This completes the logspace Turing reduction from AGP to LCONNULL.

Also [MC87] have proved that the problem of computing the intersection of two abelian permutation groups (AINTER) is NC^1-Turing reducible to AGP. We recall this proof and observe that the reduction is in fact logspace computable.

Lemma 4.2.11. AINTER *is logspace Turing reducible to* AGP.

Proof. Let $G = \langle g_1, \ldots, g_r \rangle$, and $H = \langle h_1, \ldots, h_s \rangle$. Also let $M = \{\mathbf{x} = (x_1, \ldots, x_{r+s}) \in \mathbb{Z}^{r+s} | g_1^{x_1} \cdots g_r^{x_r} h_1^{x_{r+1}} \cdots h_s^{x_{r+s}} = e$, where e is the identity element in $G\}$. Then $g_1^{x_1} \cdots g_r^{x_r} \in G \cap H$ if and only if there exists $\mathbf{x} \in M$ with $x_1, \ldots, x_r$ as its first r entries. Thus the mapping, $\phi : M \to G \cap H$ defined as $\phi(\mathbf{x}) = g_1^{x_1} \cdots g_r^{x_r}$, is an onto homomorphism. Let $\psi : M \to \langle g_1, \ldots, g_r, h_1, \ldots, h_s \rangle$ be a mapping defined as $\psi(\mathbf{x}) = g_1^{x_1} \cdots g_r^{x_r} h_1^{x_{r+1}} \cdots h_s^{x_{r+s}}$, where $\mathbf{x} = (x_1, \ldots, x_{r+s}) \in \mathbb{Z}^{r+s}$. It is then easy to note that ψ is a onto homomorphism. If $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ generate the kernel of ψ then $G \cap H = \langle \phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_m) \rangle$. Here, $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ can be obtained in logspace using a AGP oracle gate. As these vectors are obtained, we can compute the product $g_1^{x_1} \cdots g_r^{x_r}$, once again in logspace and output it, and hence the result follows. ∎

Using the observations made above regarding AGP and AINTER we obtain the following.

Theorem 4.2.12. AGP *and* AINTER *are in* $\mathrm{L}^{\mathrm{ModL}}/\mathrm{poly}$.

4.3 Hardness Results

Having obtained upper bounds we prove hardness results for all the problems on abelian permutation groups defined in Section 4.1. We obtain this by showing that LCON, LCONX and LCONNULL are logspace many-one reducible to AGM, AGMX and AGP respectively. Recall that in Theorem 3.2.16 and Theorem 3.4.5 of Chapter 3, we had shown LCON, LCONX and LCONNULL to be hard for ModL under logspace many-one reductions. Using this result we then conclude that the problems on abelian permutation groups studied in this chapter are hard for ModL under logspace many-one reductions. The underlying method to obtain our results is based on ideas from [MC87].

Theorem 4.3.1. *1.* LCON *is logspace many-one reducible to* AGM.

2. LCONX *is logspace many-one reducible to* AGMX.

3. LCONNULL *is logspace many-one reducible to* AGP.

Proof. In LCON, LCONX and LCONNULL we are given as input, a $m \times n$ matrix $A = (a_{ij}) \in \mathbb{Z}^{m \times n}$ and a positive integer q in terms of its factorization into prime powers $p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$ where each $p_i^{e_i}$ is given in unary. We now try to define a suitable group G that effects a logspace many-one reduction from LCON, LCONX and LCONNULL to AGM, AGMX and AGP respectively.

Consider a permutation π with disjoint cycle representation $\psi_1, \psi_2, \ldots, \psi_k$, where ψ_i is a cycle of length $p_i^{e_i}$ for $1 \leq i \leq k$. Clearly the order of π is q and π is definable in L. Let $\pi_1, \pi_2, \ldots, \pi_m$ be m copies of π with each π_i, for $1 \leq i \leq m$ acting on a separate set of points. The group G effecting the reduction would be a subgroup of the abelian group $\langle \pi_1, \pi_2, \ldots, \pi_m \rangle$. Let us define $G = \langle g_1, \ldots, g_n \rangle$, where $g_j = \pi_1^{a_{1j}} \pi_2^{a_{2j}} \cdots \pi_m^{a_{mj}}$ for $1 \leq j \leq n$.

For problems LCON and LCONX, apart from A and q we are also given a vector $\mathbf{b} = (b_i)_{1 \leq i \leq m} \in \mathbb{Z}^m$. Now, let us define a permutation $h = \pi_1^{b_1} \pi_2^{b_2} \cdots \pi_m^{b_m}$. Given any vector $\mathbf{x} = (x_i)_{1 \leq i \leq n} \in \mathbb{Z}^n$, we have

$$g_1^{x_1} g_2^{x_2} \cdots g_n^{x_n} = \pi_1^{\sum_{j=1}^n a_{1j} x_j} \cdots \pi_m^{\sum_{j=1}^n a_{mj} x_j}.$$

Notice that the exponents of π_i, for $1 \leq i \leq m$, in the expression given above are in fact the terms occurring in the vector $A\mathbf{x}$. Therefore it now follows that the system of linear

equations $A\mathbf{x} = \mathbf{b}$ have a solution $\mathbf{x} = (x_i)_{1 \leq i \leq n} \in \mathbb{Z}^n$ if and only if $g_1^{x_1} \cdots g_n^{x_n} = h$. In other words there is a solution for the system of linear equations if and only if h is in G. That is, LCON and LCONX reduce to AGM and AGMX respectively.

To reduce LCONNULL to AGP we use the same group G constructed above and observe that $g_1^{x_1} g_2^{x_2} \cdots g_n^{x_n} = 1$ if and only if $A\mathbf{x} = 0 (\mathrm{mod} q)$. Note that we use the fact that each π_i, for $1 \leq i \leq m$, is of order q. ∎

The following result is then immediate.

Theorem 4.3.2. AGM, AISO, AORDER, AINTER, AGMX *and* AGP *are hard for* ModL *under logspace many-one reductions.*

Due to the equivalence of linear algebraic problems LCON, LCONX and LCONNULL, and the abelian permutation group problems AGM, AISO, AORDER, AINTER, AGMX and AGP under logspace Turing reductions, it follows that as done in Chapter 3 the non-uniform upper bounds on these problems can be relaxed to obtain an upper bound of $\mathrm{L}^{\mathrm{ModL}}$ under hardness assumption that there is a language in DSPACE(n) that is not accepted by circuits of subexponential size.

Theorem 4.3.3. *Suppose* $L \in \mathrm{DSPACE}(n)$ *such that for some constant* $\epsilon > 0$ *and all but finitely many* n*, no* n*-input circuit* C *of size at most* $2^{\epsilon n}$ *accepts exactly strings of length* n *in* L*. Then* AGM, AISO, AORDER, AINTER, AGMX *and* AGP *are in* $\mathrm{L}^{\mathrm{ModL}}$.

4.4 Discussion

In this chapter we provide reasonably tight upper and lower bounds for problems defined on abelian permutation groups. The observations shown are a natural fall out of results obtained in Chapter 3 and the NC^1-Turing reductions shown by [MC87]. Our main tool has been to show that the various reductions proved by [MC87] are in fact logspace computable. Then we use the upper bound and hardness results of Chapter 3 to finally get the results proved above. An interesting area for further work is to study these problems for larger classes of permutation groups. The membership problem for general permutation groups is known to be in NC [BLAS87]. We would like to obtain a tight complexity-theoretic classification, at least for the easier cases of solvable or nilpotent permutation groups.

5

Orbit Problem

5.1 Introduction

The *Orbit problem* is defined as follows.

> Given $A \in \mathbb{Q}^{n \times n}$ and $\mathbf{x}, \mathbf{y} \in \mathbb{Q}^n$, does there exist a non-negative integer i such that $A^i\mathbf{x} = \mathbf{y}$.

The goal of this chapter is to give a new upper bound for the complexity of the orbit problem using logspace counting classes. We show that the orbit problem is in $\mathrm{AC}^0(\mathrm{GapL})$.

Kannan and Lipton in [KL86] gave a polynomial time algorithm for the orbit problem. Their approach was to reduce it to the *Matrix power problem*. In the matrix power problem, we are given two matrices $B, D \in \mathbb{Q}^{n \times n}$ as input and we need to check if there exists a non-negative integer i such that $B^i = D$. Kannan and Lipton further show that (B, D) is a yes instance of the matrix power problem if and only if $B^i = q(B)$ for some nonnegative integer i, where $q(x) \in \mathbb{Q}[x]$ is a polynomial that depends on B and D and its coefficients can be computed in polynomial time. Here the degree of $q(x)$ is one less than the degree of the minimal polynomial of B. The rest of the algorithm in [KL86] focuses on checking if there is an $i \in \mathbb{Z}^+$ satisfying $B^i = q(B)$. Assume that we have computed the polynomial $q(x)$, and let α be a root of $q(x)$. Now, if there exists $i \in \mathbb{Z}^+$ such that $B^i = q(B)$ then $\alpha^i = q(\alpha)$. The algorithm in [KL86] uses this fact repeatedly while considering different cases: wherein $q(x)$ has a root that is not a root of unity, or when all its roots are roots of unity with multiplicity at most 1, or the case when all the roots of $q(x)$ are roots of unity, but there exists at least one root with multiplicity greater than 1. Kannan and Lipton design their algorithm based on this case analysis.

In this chapter, we broadly follow the Kannan-Lipton algorithm [KL86], but we need to differently analyze the complexity of the main steps involved in it. This forces us to modify several subroutines in the algorithm. Since these steps basically require linear

algebraic computation over $\mathbb{Q}$, we obtain an upper bound in the GapL hierarchy. Some of the steps involve checking if a set of vectors are linearly independent over $\mathbb{Q}$, computing the determinant of a matrix over $\mathbb{Q}$, computing the inverse of a matrix, computing powers and the minimal polynomial of a rational matrix etc. We also need to compute the greatest common divisor of two polynomials in $\mathbb{Q}[x]$. Using the GapL upper bound of [Dam91, Tod91a, Val92, Vin91] for computing the determinant of integer matrices, we show that computing the gcd of two given polynomials with rational coefficients is in $\mathrm{L}^{\mathrm{GapL}}$. Moreover, [ABO99, HT03] have classified the complexity of the remaining subroutines using logspace counting classes. Finally, we show that the orbit problem is hard for $\mathrm{C_{=}L}$ under logspace many-one reductions.

We leave open a tight classification of the orbit problem using logspace counting classes.

5.2 Basic Results

In this section we introduce the basic definitions, notation, terminology and results required to solve the orbit problem. Much of the material on algebra and number theory in this section are standard. For more details we refer to standard texts such as [BL65, Mar77].

Definition 5.2.1. *1. We say that a complex number θ is an n^{th} root of unity if $\theta^n - 1 = 0$.*

2. We say that θ is a primitive n^{th} root of unity if θ is a n^{th} root of unity and $\theta^m - 1 \neq 0$ for all integers $0 < m < n$.

Let e denote the base of the natural logarithm. Then, from the above definition it follows that any n^{th} root of unity is of the form $e^{(2\pi\sqrt{-1})j/n}$ for $0 \leq j \leq (n-1)$. Also note that $e^{(2\pi\sqrt{-1})j/n}$ is a primitive n^{th} root of unity if and only if $\gcd(j, n) = 1$. Following standard notation, we denote $\sqrt{-1}$ by ι.

Let $\varphi(j)$ denote the Euler totient function: the number of positive integers less than and relatively prime to j.

Definition 5.2.2. *Let $\theta_1, \ldots, \theta_{\varphi(j)}$ be primitive j^{th} roots of unity. Then, the j^{th} cyclotomic polynomial, denoted by $C_j(x)$, is defined as $C_j(x) = \prod_{i=1}^{\varphi(j)}(x - \theta_i)$.*

It is well known that $C_j(x)$ is irreducible over $\mathbb{Q}$. It follows that $C_j(x)$ must divide any polynomial $h(x) \in \mathbb{Q}[x]$ that has as root one of the primitive n^{th} roots of unity. We state this as a fact.

Fact 5.2.3. *Let $h(x) \in \mathbb{Q}[x]$. If $h(\theta) = 0$, where θ is a primitive n^{th} root of unity, then $h(\theta') = 0$ for any other primitive n^{th} root of unity θ'.*

We assume that each rational entry of an input matrix $A \in \mathbb{Q}^{n \times m}$ is given in terms of its numerator and denominator. Also, we will assume that an algorithm computing $\det(A)$ for a rational matrix $A \in \mathbb{Q}^{n \times n}$ will output two integers p and q such that $\det(A) = p/q$. Furthermore, we will *not* require that p and q be relatively prime, that is $\gcd(p, q)$ need not be 1. This assumption is necessary because computing the GCD of two integers is not known to be in NC. This representation of rationals does not affect our algorithm so long as the size in binary of the two integers p and q is bounded by a polynomial in the size of the input. We will make a similar assumption for other computations involving rational inputs.

We now recall the following results concerning rational matrices. These are usually stated for integer matrices.

Lemma 5.2.4. *Let $A \in \mathbb{Q}^{n \times m}$ be the given input rational matrix. Then,*

1. [AO96, Dam91, Tod91a, Val92, Vin91] *When $n = m$, computing the determinant of A denoted by $\det(A)$, computing the $(i, j)^{th}$ entry of A^{-1}, and computing the $(i, j)^{th}$ entry of A^l for a given positive integer l are complete for* GapL *under logspace many-one reductions.*

2. [ABO99] *Checking if the set of column vectors of A are linearly dependent is complete for* $\mathrm{C_{=}L}$ *under logspace many-one reductions.*

3. [ABO99] *Let $\mathbf{b} \in \mathbb{Q}^n$ be an n-dimensional rational vector. Then, determining if the system of linear equations $A\mathbf{x} = \mathbf{b}$ has a rational vector $\mathbf{x}$ as a solution is complete for* $\mathrm{L}^{\mathrm{C_{=}L}}$ *under logspace truth-table reductions.*

4. *Computing a maximal set of linearly independent columns from A is in* $\mathrm{FL}^{\mathrm{C_{=}L}}$.

5. [HT03] *Given $B \in \mathbb{Q}^{n \times n}$, we can compute the coefficients of the minimal polynomial of B in* $\mathrm{AC^0(GapL)}$.

Proof. Let $A \in \mathbb{Q}^{n \times m}$ be the given input rational matrix. Let $A_{ij} = p_{ij}/q_{ij}$, where $1 \leq i \leq n$ and $1 \leq j \leq m$. Also, we can assume the size of each p_{ij} and q_{ij} is at most $\max(m, n)$. Let q be the product of all the denominators of the entries in A. It is well known that, for any positive integer n, we can compute the i^{th} bit of the product of n integers, each of size n, using an $\mathrm{NC^1}$ circuit and therefore we can compute q which is a product of nm integers in $\mathrm{NC^1}$ as well. Let us consider the matrix (qA), obtained by multiplying each entry of A by q. Clearly (qA) is an integer matrix and $A = (qA)/q$.

In problems involving an additional vector $\mathbf{b}$, we multiply q with the denominators of the entries occurring in $\mathbf{b}$ to reduce the problem to the case when the inputs are integer matrices. In all these cases, the size of q as well as entries of (qA) and $(q\mathbf{b})$ are bounded by a polynomial in the size of the input, where $1 \leq i \leq n$ and $1 \leq j \leq m$. Thus we can compute the i^{th} bit of any entry of these matrices in logspace. The results stated above then follow by applying known complexity bounds (proven in the references appearing in the Theorem statement) on linear algebraic problems involving integer matrices to (qA), and $(q\mathbf{b})$. ■

Lemma 5.2.5. *Let $A \in \mathbb{Q}^{m\times n}$ and $\mathbf{b} \in \mathbb{Q}^n$. If the system of linear equation $A\mathbf{x} = \mathbf{b}$ is feasible, then a solution to it can be computed in* $\mathrm{AC}^0(\mathrm{GapL})$.

Proof. First, we can compute a maximal linearly independent set of columns of A with an $\mathrm{L}^{\mathrm{GapL}}$ computation as follows: for each index i such that $2 \leq i \leq n$ we check if the i^{th} column A_i of A is linearly independent of the first $i-1$ columns $\{A_1, A_2, \cdots, A_{i-1}\}$, and if it is independent we output the index i. Let $S \subseteq [n]$ denote the output set of indices, and let A' denote the matrix of these linearly independent columns. Notice that $A\mathbf{x} = \mathbf{b}$ is feasible if and only if $A'\mathbf{z} = \mathbf{b}$ is feasible, where $\mathbf{z}$ is an $n - |S|$ dimensional vector. Furthermore, given a solution $\mathbf{z}$ for $A'\mathbf{z} = \mathbf{b}$ we can extend it to a solution $\mathbf{x}$ of $A\mathbf{x} = \mathbf{b}$ by setting $x_i = 0$ for $i \notin S$. Since the columns of A' are linearly independent, the solution $\mathbf{z}$, if it exists, is unique. In order to find $\mathbf{z}$, we perform another round of $\mathrm{L}^{\mathrm{GapL}}$ computation in which we output a maximal linearly independent set of *rows* of A' (using the same method as above). Let $T \subseteq [m]$ denote the set of $|S|$ row indices output, and let B denote the corresponding $|S| \times |S|$ matrix. Furthermore, let $\mathbf{b}'$ denote the corresponding $|S|$-dimensional subvector of $\mathbf{b}$ picked out by index set T. Clearly, $A'\mathbf{z} = \mathbf{b}$ if and only if $B\mathbf{z} = \mathbf{b}'$ for any vector $\mathbf{z}$. Finally, since B is invertible we can compute B^{-1} using an $\mathrm{L}^{\mathrm{GapL}}$ computation to obtain the solution vector $\mathbf{z} = B^{-1}\mathbf{b}'$. Composing these three $\mathrm{L}^{\mathrm{GapL}}$ computations gives us the required $\mathrm{AC}^0(\mathrm{GapL})$ upper bound. ■

We first show that computing the GCD of two polynomials over $\mathbb{Q}$ is in the class $\mathrm{L}^{\mathrm{GapL}}$.

Let $f(x) = \sum_{i=0}^{m} f_i x^i$ and $g(x) = \sum_{j=0}^{n} g_j x^j$ be polynomials over $\mathbb{Q}$. Also let $h(x) = x^l + h_{l-1}x^{l-1} + \cdots + h_0$ be the monic polynomial over $\mathbb{Q}$ denoting the $\gcd(f(x), g(x))$. Then there exists $s(x) = \sum_{i=0}^{n-1} s_i x^i$, $t(x) = \sum_{j=0}^{m-1} t_j x^j \in \mathbb{Q}[x]$ such that $s(x)f(x) + t(x)g(x) = h(x)$. This can be seen as a system of linear equations over $\mathbb{Q}$, with the coefficients of $s(x), t(x)$ and $h(x)$ as the unknowns. That is, we have a linear system of the form $A\mathbf{y} = \mathbf{z}$, where A is the *Sylvester matrix* defined by the coefficients occurring in the polynomials

$f(x)$ and $g(x)$:

$$A(i,j) = \begin{cases} f_{m-i+j} & \text{for } 1 \leq j \leq i \leq m+j \leq m+n \\ g_{j-i} & \text{for } 1 \leq j-n \leq i \leq j \leq m+n \\ 0 & \text{otherwise} \end{cases}$$

$\mathbf{y} = (s_{n-1}, \ldots, s_0, t_{m-1}, \ldots, t_0)^T$ and $\mathbf{z} = (0, \ldots, 0, 1, h_{l-1}, \ldots, h_0)$. The result given below shows that it is possible to obtain $\mathbf{y}$ and hence $\mathbf{z}$ and $h(x)$ in $\mathrm{L}^{\mathrm{GapL}}$.

Lemma 5.2.6. *Given polynomials $f(x)$, $g(x) \in \mathbb{Q}[x]$, we can compute $h(x) = gcd(f(x), g(x))$ in $\mathrm{L}^{\mathrm{GapL}}$.*

Proof. Recall the definitions of A, $\mathbf{y}$ and $\mathbf{z}$ given above. The procedure that follows finds the smallest $l \geq 1$, where l is the degree of $h(x)$, for which the system $A\mathbf{y} = \mathbf{z}$ has a solution. The least l will clearly identify $\gcd(f(x), g(x)) = h(x)$.

Let $m = \deg(f(x))$ and $n = \deg(g(x))$ and assume $m = \min(m, n)$. Also let $h(x) = \gcd(f(x), g(x))$ such that $l = \deg(h(x))$. For $0 \leq d \leq m$, let $A^{(d)}$ be the matrix obtained by deleting the last d rows of A. Thus, $A^{(0)} = A$. Corresponding to $A^{(d)}$, let $\mathbf{z}^{(d)} = (0, \ldots, 0, 1) \in \mathbb{Q}^{m+n-d}$ be the vector with 0's in the first $m+n-d-1$ entries and a 1 in the last entry. If l is the degree of $\gcd(f, g)$, then notice that the least value of d such that the system of linear equations $A^{(d)}\mathbf{y} = \mathbf{z}^{(d)}$ has a solution is $d = l$. Since $0 \leq l \leq m$, it is sufficient to try each d in the range $0 \leq d \leq m$ and find the least such d. Now we focus on actually computing the coefficients of $\gcd(f, g)$. Let $\mathbf{z}'$ denote the vector whose first $m+n-l-1$ entries are 0, the $(m+n-l)^{th}$ entry is 1, and the last l entries are the indeterminates $h_{l-1}, h_{l-2}, \cdots, h_0$. Then, by the uniqueness of the GCD and the property of the Sylvester matrix A we observe that every solution to $A\mathbf{y} = \mathbf{z}'$ has to take a *unique* set of values for the indeterminates $h_{l-1}, h_{l-2}, \cdots, h_0$ occurring in $\mathbf{z}'$, namely, the coefficients of the monic $\gcd(f, g)$. Therefore, we will be able to find the coefficients of $\gcd(f, g)$ in parallel. In order to find h_j we consider a new matrix $B_j \in \mathbb{Q}^{(m+n-l+1)\times m+n}$ whose first first $m+n-l$ rows are the first $m+n-l$ rows of A and the last row of B_j is the $(m+n-j)^{th}$ row of A. We consider the system $B_j\mathbf{y} = \mathbf{z}'_j$, where z'_j is obtained from $\mathbf{z}'$ by taking the identical set of rows as we took for B_j. Notice that the only indeterminate in $\mathbf{z}'_j$ is h_j. Furthermore, by the uniqueness of GCD, there every solution to $B_j\mathbf{y} = \mathbf{z}'_j$ assigns the same value to h_j which we need to compute. We rewrite this system as

$$B_j\mathbf{y} + (0, 0, \cdots, 0, -1)^T \cdot h_j = (0, 0 \cdots, 1, 0)^T.$$

Since we can test linear independence of a set of vectors over $\mathbb{Q}$ with queries to a GapL oracle (more precisely, $\mathrm{C}_=\mathrm{L}$ would suffice [ABO99]), we can pick a maximal set of

columns $B_j^{(i_1)}, B_j^{(i_2)}, \cdots, B_j^{(i_k)}$ of B_j that, along with the column $(0, 0, \cdots, 0, -1)^T$ form a linearly independent set: the precise GapL query would be whether the i^{th} column of B_j is independent of $(0, 0, \cdots, 0, -1)^T$ and the last $m+n-i+1$ columns, and we output this column if and only if it is independent. This entire computation can be carried out in $\mathrm{L}^{\mathrm{GapL}}$. Thus, the system of equations now assumes the form

$$(0, 0, \cdots, 0, -1)^T \cdot h_j + C\mathbf{y}' = (0, 0 \cdots, 1, 0)^T,$$

with indeterminates h_j and $\mathbf{y}'$. With a similar $\mathrm{L}^{\mathrm{GapL}}$ computation we can now find a maximal linearly independent subset of rows from the coefficient matrix to obtain a system of equations of the form $C\mathbf{y}'' = \mathbf{b}$, where $\mathbf{y}''$ include the indeterminate h_j and the vector $\mathbf{b}$ is the corresponding subvector of $(0, 0 \cdots, 1, 0)^T$. Since now C is invertible and C^{-1} is computable in $\mathrm{L}^{\mathrm{GapL}}$, we can solve for $\mathbf{y}''$ in $\mathrm{L}^{\mathrm{GapL}}$ and hence recover h_j and output it. Putting it together, an $\mathrm{L}^{\mathrm{GapL}}$ can thus compute all the coefficients of $\gcd(f, g)$. This completes the proof. ∎

5.3 Kannan-Lipton Algorithm

We next recall the definition of the GapL hierarchy from [AO96].

Definition 5.3.1. *Define* GapLH_1 *to be* GapL. *For* $i \geq 1$, *define* GapLH_{i+1} *to be the class of functions* f, *such that for some logspace-bounded nondeterministic oracle Turing machine* M *with a function* $g \in \mathrm{GapLH}_i$ *as oracle, we have* $f(x) = acc_M(x)$. *We denote the* GapL *hierarchy by* GapLH.

As mentioned in Section 1.3.2 of Chapter 1, #LH is in fact equal to GapLH. Also, it is shown in [AO96] that $\mathrm{GapLH} = \mathrm{AC}^0(\mathrm{GapL})$. We now proceed to show that the orbit problem is in GapLH, and hence in $\mathrm{AC}^0(\mathrm{GapL})$.

We first describe the main steps in Kannan-Lipton algorithm [KL86] for the orbit problem. To obtain a polynomial time algorithm for the orbit problem, Kannan and Lipton in [KL86] reduce the orbit problem to the *Matrix Power problem* which is defined below.

> Given $B, D \in \mathbb{Q}^{n \times n}$ does there exists a non-negative integer i such that $B^i = D$.

We now describe the reduction. Let $(A, \mathbf{x}, \mathbf{y})$ be an instance of the orbit problem. Let $V \subseteq \mathbb{Q}^n$ denote the subspace spanned by $\{\mathbf{x}, A\mathbf{x}, A^2\mathbf{x} \cdots, A^{n-1}\mathbf{x}\}$. Clearly V is k-dimensional for the largest k such that $\{\mathbf{x}, A\mathbf{x}, A^2\mathbf{x} \cdots, A^{k-1}\mathbf{x}\}$ are linearly independent,

and a basis for V is this set $\{\mathbf{x}, A\mathbf{x}, A^2\mathbf{x}\cdots, A^{k-1}\mathbf{x}\}$. We can compute this basis in $\mathrm{AC}^0(\mathrm{GapL})$: with an $\mathrm{L}^{\mathrm{GapL}}$ computation we can first compute $A^j\mathbf{x}$ for $1 \leq j \leq n-1$. This machines output is taken as input by another $\mathrm{L}^{\mathrm{GapL}}$ computation that will find the largest k such that $\{\mathbf{x}, A\mathbf{x}, A^2\mathbf{x}\cdots, A^{j-1}\mathbf{x}\}$ is linearly independent.

An important property of the subspace V is that it is invariant under the linear transformation A. Thus, it follows that $A^i\mathbf{x} \in V$ for each $i \geq 0$. Consequently, $(A, \mathbf{x}, \mathbf{y})$ is a 'yes' instance for the orbit problem only if $\mathbf{y} \in V$. We can check if $\mathbf{y} \in V$ in $\mathrm{L}^{\mathrm{GapL}}$. If $\mathbf{y} \notin V$ then the reduction outputs the pair (O_n, I_n) of the matrix power problem, where O_n is the $n \times n$ zero matrix and I_n is the identity matrix. Therefore, in the sequel we can assume that $\dim(V) = k$ and $\mathbf{y} \in V$. Let

$$A^k\mathbf{x} = \sum_{j=0}^{k-1} \alpha_j A^j\mathbf{x},$$
$$\mathbf{x} = \sum_{j=0}^{k-1} \beta_j A^j\mathbf{x},$$
$$\mathbf{y} = \sum_{j=0}^{k-1} \gamma_j A^j\mathbf{x}.$$

We can compute the scalars $\alpha_j, \beta_j, \gamma_j$ in $\mathrm{L}^{\mathrm{GapL}}$ by solving each of the above three systems of linear equations using Cramér's rule.

The $k \times k$ matrix for the linear transformation A from V to V has $e_{j+1}, 1 \leq j \leq k-1$ as its first $k-1$ columns and $(\alpha_0, \cdots, \alpha_{k-1})^T$ as the last column.[1] Call this matrix A'. Likewise, let $\mathbf{x}' = (\beta_0, \cdots, \beta_{k-1})^T$ and $\mathbf{y}' = (\gamma_0, \cdots, \gamma_{k-1})^T$. Clearly, $(A', \mathbf{x}', \mathbf{y}')$ is a yes instance of the orbit problem if and only if $(A, \mathbf{x}, \mathbf{y})$ is a yes instance. This is because $A', \mathbf{x}', \mathbf{y}'$ are essentially A, $\mathbf{x}$, and $\mathbf{y}$ expressed using the basis $\mathbf{x}, A\mathbf{x}, \cdots, A^{k-1}\mathbf{x}$ of V. Now, let C denote the $k \times k$ invertible matrix $[\mathbf{x}'|A'\mathbf{x}'|\cdots|A'^{k-1}\mathbf{x}']$. Similarly, let C' denote the $k \times k$ matrix $[\mathbf{y}'|A'\mathbf{y}'|\cdots|A'^{k-1}\mathbf{y}']$. Then, there exists an $i \geq 0$ such that $A'^i\mathbf{x}' = \mathbf{y}'$ if and only if $A'^iC = C'$, which we can rewrite as $A'^i = C'C^{-1}$ as C is invertible. Thus, $(A', C'C^{-1})$ is the instance of the matrix power problem to which we have reduced $(A, \mathbf{x}, \mathbf{y})$. We formally state this as a lemma.

Lemma 5.3.2. *The orbit problem can be reduced to the matrix power problem in* $\mathrm{AC}^0(\mathrm{GapL})$.

Proof. The correctness of the reduction follows from the above argument. To see that it is computable in $\mathrm{AC}^0(\mathrm{GapL})$, we note that a set of $\mathrm{L}^{\mathrm{GapL}}$ computations need to be carried out that involves a nesting of at most two levels of GapL queries. ∎

[1] Here the vectors e_{j+1} denote the standard basis vectors of $\mathbb{R}^k$.

We now turn to the matrix power problem. Let $B, D \in \mathbb{Q}^{n\times n}$ be an input instance. Following [KL86] we further reduce it to a more tractable problem.

Lemma 5.3.3. *Given $B, D \in \mathbb{Q}^{n\times n}$, we can compute in* $\mathrm{AC}^0(\mathrm{GapL})$ *a polynomial $q(x) \in \mathbb{Q}[x]$ of degree at most $n-1$ such that there exists a non-negative integer i satisfying $B^i = D$ if and only if $B^i = q(B)$.*

Proof. Let $p(x)$ be the minimal polynomial of B which is computable in $\mathrm{AC}^0(\mathrm{GapL})$ [HT03]. We have $p(B) = 0$ and $\deg(p(x)) = r \leq n$. Thus, if there is an $i \geq 0$ such that $B^i = D$, then we claim that there is a polynomial $q(x)$ of degree at most $n-1$ such that $D = q(B)$. We divide x^i by $q(x)$ and take the remainder as the polynomial $q(x)$. Thus, $q(x) \equiv x^i (\mathrm{mod}\ p(x))$, and $\deg(q(x)) \leq (\deg(p(x)) - 1) \leq (n-1)$. Therefore, (B, D) is a yes instance of the matrix power problem only if such a polynomial $q(x)$ exists. We can test this and compute the coefficients of $q(x)$ by solving the following system of n^2 linear equations over n variables: $\sum_{j=0}^{(r-1)} q_j B^j = D$ where the unknowns are the coefficients q_j of the polynomial $q(x)$. Given B and D as input, an $\mathrm{L}^{\mathrm{GapL}}$ computation will first compute B^j for $1 \leq j \leq n-1$ and pass it as input to another $\mathrm{L}^{\mathrm{GapL}}$ computation to check the feasibility of the above system and find a solution $q(x)$ using Lemma 5.2.5. Thus, the polynomial $q(x)$ can be computed in $\mathrm{AC}^0(\mathrm{GapL})$. Clearly, $B^i = q(B)$ if and only if $B^i = D$. ∎

As mentioned previously, the overall reduction from the orbit problem involves composing computations, each of which is in some constant level of the GapL hierarchy. Since we will do only a constant number of such compositions the overall computation is still in a constant level of the GapL hierarchy.

Continuing with the proof, as a consequence of Lemma 5.3.2 and Lemma 5.3.3, we obtain the following.

Corollary 5.3.4. *Given an instance $A \in \mathbb{Q}^{n\times n}$ and $\mathbf{x}, \mathbf{y} \in \mathbb{Q}^n$ of the orbit problem, for some $m \leq n$ we can compute a matrix $B \in \mathbb{Q}^{m\times m}$ and a polynomial $q(x) \in \mathbb{Q}[x]$ of degree at most $(m-1)$ in* $\mathrm{AC}^0(\mathrm{GapL})$*, such that $A^i\mathbf{x} = \mathbf{y}$ for some $i \geq 0$ if and only if $B^i = q(B)$.*

The following lemma is a useful property for the next step.

Lemma 5.3.5. *Suppose $p(x) \in \mathbb{Q}[x]$ is the minimal polynomial of matrix $B \in \mathbb{Q}^{n\times n}$. For any two polynomials $r(x), q(x) \in \mathbb{Q}[x]$ we have $r(B) = q(B)$ if and only if $r(x) = q(x) (mod\ p(x))$.*

In particular, it follows that $B^i = q(B)$ for some $i \geq 0$ if and only if $x^i = q(x) (\mathrm{mod}\ p(x))$. As a consequence of Corollary 5.3.4 and Lemma 5.3.5, it suffices to solve in $\mathrm{AC}^0(\mathrm{GapL})$

the problem of checking if $x^i = q(x) (\text{mod } p(x))$ for some $i \geq 0$, where $p(x)$ is the minimal polynomial of the matrix B. We solve this problem in the next section.

5.3.1 Orbit Problem is in $\text{AC}^0(\text{GapL})$

Given polynomials $p, q \in \mathbb{Q}[x]$, where p is a monic, the goal is to test in $\text{AC}^0(\text{GapL})$ if $x^i = q(x) (\text{mod } p(x))$ for some $i \geq 0$. Following the Kannan-Lipton analysis [KL86], we need to handle different cases depending on the roots of the polynomial $p(x)$. A crucial property they use is a bound from algebraic number theory [KL86, Theorem 3] which we recall below.

For a polynomial $f \in \mathbb{Q}[x]$ let $|f|$ denote the ℓ_2 norm of the vector of its coefficients.

Theorem 5.3.6. [KL86, Theorem 3] *There exists a polynomial P such that for any algebraic number $\alpha \in \mathbb{C}$ that is not a root of unity and any polynomial $q(x) \in \mathbb{Q}[x]$, if $\alpha^i = q(\alpha)$ for some positive integer i then $i \leq P(deg(f_\alpha), \log(|f|), \log(|q|))$, where $f_\alpha \in \mathbb{Q}[x]$ is the minimal polynomial of α.*

Thus, if the given polynomial $p(x)$ has a root α that is not a root of unity then, by Theorem 5.3.6, we can test if there is an i such that $x^i = q(x) (\text{mod } p(x))$ by trying the polynomially many values of i in the range $i \leq P(\deg(f_\alpha), \log(|f_\alpha|), \log(|q|))$. Since f_α is an irreducible factor of $p(x)$, we know that $|f_\alpha|$ is polynomially bounded by $|p|$. Thus, the range of values for i is indeed polynomially bounded by the input size. Indeed, since this test involves only division of polynomials it can be carried out in logspace.

Thus, the harder case is when all the roots of $p(x)$ are complex roots of unity. We focus on this case. We shall use some key properties of the cyclotomic polynomials $C_j(x)$. First we show that $C_j(x)$ can be computed in $\text{AC}^0(\text{GapL})$ by an algorithm that takes j in unary as input.

Lemma 5.3.7. *Given 1^j as input the j^{th} cyclotomic polynomial $C_j(x)$ can be computed in $\text{AC}^0(\text{GapL})$.*

Proof. The j^{th} cyclotomic polynomial $C_j(x) = \prod_{r=1}^{\varphi(j)} (x - \omega_r)$ where the ω_r are the $\varphi(j)$ different primitive j^{th} roots of unity and $C_j(x)$ is an irreducible factor of $x^j - 1$.

We first define the polynomial

$$t_j(x) = \prod_{i=1}^{j-1} (x^i - 1).$$

The polynomial t_j is of degree $j(j-1)/2$. It is easy to see that each coefficient of $t_j(x)$ is GapL computable. Furthermore, it is clear that $b_j(x) = \gcd(t_j(x), x^j - 1)$ contains as

roots precisely all non-primitive j^{th} roots of unity. Therefore, it follows that $C_j(x)$ is the quotient obtained on dividing $x^j - 1$ by $b_j(x)$. Given the coefficients of $t_j(x)$ we can apply Lemma 5.2.6 to compute $\gcd(t_j(x), x^j - 1)$ in $\mathrm{L}^{\mathrm{GapL}}$. Therefore, the overall computation is clearly in $\mathrm{AC}^0(\mathrm{GapL})$. ■

We can easily show that testing if all roots of $p(x)$ are complex roots of unity is in $\mathrm{AC}^0(\mathrm{GapL})$.

Lemma 5.3.8. *Given $p(x) \in \mathbb{Q}[x]$ as input we can test in $\mathrm{AC}^0(\mathrm{GapL})$ if all roots of $p(x)$ are complex roots of unity, and if so we can factorize $p(x)$ into its irreducible factors in $\mathrm{AC}^0(\mathrm{GapL})$.*

Proof. Let $\deg(p(x)) = d$. We first compute $C_j(x), 1 \leq j \leq d$ using Lemma 5.3.7. Next, since division can be carried out in logspace, we can find the highest power of $C_j(x)$ that divides $p(x)$ in logspace. Putting it together will give us all the irreducible factors of $p(x)$, with multiplicity, from the set $C_j(x), 1 \leq j \leq d$. ■

After applying Lemma 5.3.8 we will know whether $p(x)$ has a root that is not a root of unity (in which case we can use the easy logspace algorithm based on Theorem 5.3.6). Thus, we now consider only the case when $p(x) = \prod_{j=1}^{d} C_j(x)^{k_j}$, where $k_j \geq 0$.

An easy and useful lemma is the following.

Lemma 5.3.9. *Let $q(x)$ be an arbitrary polynomial and let $C_j(x)$ be the j^{th} cyclotomic polynomial. The congruence $x^\ell \equiv q(x) \pmod{C_j(x)}$ holds for some nonnegative integer ℓ if and only if it holds for some* unique *ℓ in the range $0 \leq \ell \leq (j-1)$.*

Proof. Since $C_j(x)$ divides $x^j - 1$, it immediately follows that $x^\ell \equiv q(x) \pmod{C_j(x)}$ implies $x^{\ell \pmod j} \equiv q(x) \pmod{C_j(x)}$. ■

Using the above result we first dispense off the case when $k_j \in \{0, 1\}$ in $p(x) = \prod_{j=1}^{d} C_j(x)^{k_j}$.

Lemma 5.3.10. *If $p(x) = \prod_{j=1}^{d} C_j(x)^{k_j}$ for $k_j \in \{0, 1\}$, then the problem of testing for a given polynomial $q(x) \in \mathbb{Q}[x]$ if $x^i \equiv q(x) \pmod{p(x)}$ for some positive integer i, is in $\mathrm{AC}^0(\mathrm{GapL})$.*

Proof. By the chinese remainder theorem, it suffices to check if there is a positive integer i such that

$$x^i \equiv q(x) \pmod{C_j(x)}$$

for every C_j such that $k_j = 1$. By Lemma 5.3.9 there is an $i \geq 0$ such that $x^i \equiv q(x) \pmod{C_j(x)}$ if and only if there is an $i_j \in \{0, 1, \cdots, j-1\}$ such that $x^{i_j} \equiv$

$q(x) \pmod{C_j(x)}$. Notice that such an i_j, if it exists, has to be *unique*. If for some C_j such that $k_j = 1$ no such i_j exists we reject the input. Otherwise, we would have computed i_j for each C_j with $k_j = 1$. We only need to check if there exists a positive integer i such that

$$i \equiv i_j (\text{mod } j) \tag{5.1}$$

for all j such that $k_j = 1$. We cannot directly apply the chinese remainder theorem to check this congruence as the different j's need not be relatively prime. However, since each such j is bounded by d, it follows that j is of logarithmic size. Hence we can compute the prime factorization for each j such that $k_j = 1$ in deterministic logspace. Let $p_1, p_2, \cdots, p_k$ denote the set of all prime factors of any $j \leq d$. Clearly, each p_i is logarithmic in size and k is also logarithmic in the input size. Then we can rewrite the congruences in Equation 5.1 above as

$$i \equiv i_j (\text{mod } p_\ell^{r_{j,\ell}}), \tag{5.2}$$

where $1 \leq \ell \leq k$ and j such that $k_j = 1$ and $j = \prod p_\ell^{r_{j,\ell}}$.

Now, for each prime p_ℓ above we club together all congruences of the type $i \equiv i_j \pmod{p_\ell^{r_{j,\ell}}}$ for all the j's. Let j' be a value of j for which $r_{j',\ell}$ is maximum. Then, a necessary condition that Equation 5.2 has a solution for i is that $i_j = i_{j'} \pmod{p_\ell^{r_{j,\ell}}}$ for all j which we can check in logspace. Having checked this condition we can replace all the congruences in Equation 5.2 by the single congruence $i \equiv i_{j'} (\text{mod } p_\ell^{r_{j',\ell}})$. Thus, for each p_ℓ we will have a single congruence and we can *now* invoke the chinese remainder theorem to check in logspace if there is a solution for Equation 5.1. This completes the proof. ∎

It now remains to handle the case when for some j, the exponent k_j of $C_j(x)$ is at least 2 in the factorization of $p(x)$.

Lemma 5.3.11. *Given $q(x) \in \mathbb{Q}[x]$ and a cyclotomic polynomial $C_j(x)$, we can compute in deterministic logspace a set $S_{q(x),j}$ of positive integers such that $|S_{q(x),j}|$ is polynomially bounded in $\log |q|$ and j, with the property that $x^i \equiv q(x) (mod\ C_j(x)^2)$ can have solutions only for $i \in S_{q(x),j}$.*

Proof. Suppose $x^i \equiv q(x) (\text{mod } C_j(x)^2)$. Then we have $x^i - q(x) = r(x) C_j(x)^2$. Taking the formal derivative on both sides we obtain $ix^{i-1} - q'(x) = 2C_j(x)r(x) + r'(x)C_j(x)^2$, implying that $ix^{i-1} - q'(x) \equiv 0 \pmod{C_j(x)}$, where $q'(x)$ and $r'(x)$ are the derivatives

of $q(x)$ and $r(x)$ respectively. Let P_ℓ denote the polynomial $x^\ell \pmod{C_j(x)}$ for $0 \leq \ell \leq j-1$. Notice that each P_ℓ is of degree at most $\varphi(j)-1$. Furthermore, let $q_1'(x) = q'(x) \pmod{C_j(x)}$.

Thus, i is a candidate solution only if for some ℓ we have $iP_\ell = q_1'(x)$. We define the set

$$S_{q(x),j} = \{s \mid s = \frac{q_1'(x)}{P_\ell} \text{ for some } \ell\}.$$

Clearly, $|S_{q(x),j}| \leq j$ and can be computed in deterministic logspace. ∎

We obtain the following corollary which limits the search space for the index i to such a set $S_{q(x),j}$.

Corollary 5.3.12. *Suppose $p(x) = \prod_{j=1}^{d} C_j(x)^{k_j}$ such that $k_{j'} \geq 2$ for some j'. Then $x^i \equiv q(x) \pmod{p(x)}$ for some i if and only if $x^i \equiv q(x) \pmod{p(x)}$ for some $i \in S_{q(x),j'}$.*

The rest of the algorithm is as follows: we need to check if there is an $i \in S_{q(x),j'}$ such that for each $k_j > 0$ we have $x^i \equiv q(x) \pmod{C_j(x)^{k_j}}$. Such an i is a solution. Notice that we cannot directly check this by division because $i \in S_{q(x),j'}$ may be an integer that is polynomially many bits long. Thus we need to devise a different test for checking if $x^i \equiv q(x) \pmod{C_j(x)^{k_j}}$ for a given i. This is described in our final lemma that will also complete the upper bound description.

Lemma 5.3.13. *Given as input a polynomial $q(x) \in \mathbb{Q}[x]$, and integer i (encoded in binary), a cyclotomic polynomial $C_j(x)$ and an integer k, where k and j are encoded in unary, we can test in deterministic logspace if $x^i \equiv q(x) \pmod{C_j(x)^k}$.*

Proof. Let ω denote a primitive j^{th} root of unity. Since $C_j(x)$ is irreducible it follows that $C_j(x)^k$ divides $x^i - q(x)$ if and only if $(x-\omega)^k$ divides $x^i - q(x)$. That means ω is a root of multiplicity k for $f(x) = x^i - q(x)$. Equivalently, we need to check if ω is a root of the ℓ^{th} formal derivative $f^{(\ell)}(x)$ of the polynomial $f(x)$ for each $0 \leq \ell \leq k-1$. Notice that $f^{(\ell)}(x)$ assumes the form $i(i-1)\cdots(i-\ell)x^{i-\ell} - q^{(\ell)}(x)$. Computing the coefficient $i(i-1)\cdots(i-\ell)$ is iterated integer multiplication that can be done in deterministic logspace. Furthermore, the ℓ^{th} derivative of the polynomial can be done term by term, which will also involve a similar iterated integer multiplication for each term and it can be done in deterministic logspace. Now, checking if ω is a root of $f^{(\ell)}(x)$ is equivalent to checking if $C_j(x)$ divides $f^{(\ell)}(x)$, again by the irreducibility of $C_j(x)$. But $f^{(\ell)}(x)$ has the nice form $i(i-1)\cdots(i-\ell)x^{i-\ell} - q^{(\ell)}(x)$ which is easy to divide by $C_j(x)$ as we can replace the exponent $i-\ell$ in the first term by $(i-\ell) \pmod{j}$. This completes the proof. ∎

We now show that the orbit problem is hard for C$_=$L under logspace many-one reductions.

Theorem 5.3.14. *The orbit problem is hard for* $\mathrm{C_{=}L}$ *under logspace many-one reductions.*

Proof. It is well known that given a directed graph $G = (V, E)$, and vertices $u, v \in V$, the problem of checking is there is a directed path from u to v is NL-complete. In fact, this problem remains NL-complete for input graphs that are layered, directed, and acyclic with u as its unique source node and v its unique sink node, where u is the unique node in the first layer and v is the unique node in the last layer. By a layered digraph we mean for each edge $(s, t) \in E$ in the graph if s is in layer i then t is in layer $(i + 1)$. The counting version of this problem: namely, counting the number of directed u-v paths is #L complete under logspace many-one reductions. Furthermore, verifying if the number of directed u-v paths is a given nonnegative integer m is $\mathrm{C_{=}L}$-complete under logspace many-one reductions. Therefore, it suffices to show a logspace many-one reduction from this problem to the orbit problem.

Let A be the adjacency matrix of an input digraph G as described above. Let 1 be its unique source node and let its sink node be n, where the vertex set is $V = \{1, 2, \cdots, n\}$. We want to check if the number of paths from 1 to n is m.

Since G is a layered digraph, it is easy to observe that all directed paths from 1 to n are of the same length, assuming there is a directed path from 1 to n in G. Furthermore, this number is the difference between the layer numbers of n and 1, say ℓ. Thus, G has exactly $\ell + 1$ layers, and there is exactly one vertex in G, namely vertex n, that is at distance ℓ from vertex 1.

Let A denote the adjacency matrix of the graph G. Notice that A is an $n \times n$ matrix with 0-1 entries and its rows and columns are indexed by the vertex set of G. It is easy to observe that for any positive integer k, the $(i, j)^{th}$ entry of A^k is the number of walks from vertex i to vertex j in G. Since the digraph G is acyclic, all walks are directed paths. Now, we define the vector $\mathbf{x} = (0, \ldots, 0, 1)^T \in \mathbb{Q}^{n\times 1}$, and the vector $\mathbf{y} = (m, 0, \ldots, 0)^T \in \mathbb{Q}^{n\times 1}$. Since G is a layered graph with 1 and n on the first and $(\ell + 1)^{st}$ layers respectively, it follows from the observations made above that number of directed paths in G from 1 to n is m if and only if $A^\ell \mathbf{x} = \mathbf{y}$. In other words, there is a nonnegative integer i such that $A^i \mathbf{x} = \mathbf{y}$ if and only if there are exactly m directed paths in G from 1 to n. ∎

5.4 Discussion

The interesting open problem here is to tightly classify the orbit problem in the GapL hierarchy. We would like to close the gap between the upper bound and hardness bound results reported in this chapter.

There are a number of other interesting questions that arise from our results. We have shown in Lemma 5.3.8 that factoring univariate polynomials whose roots are all complex roots of unity can be done in $\mathrm{AC}^0(\mathrm{GapL})$. By the well-known LLL algorithm (e.g. see [Sch98],) factoring univariate polynomials over $\mathbb{Q}$ is in polynomial time. To the best of our knowledge, there is no P-hardness result for the problem. It would be interesting to either obtain a better complexity upper bound or show P-hardness.

6

Intersection of Linearly Representable Matroids

6.1 Introduction

In this chapter, we study the complexity of the matroid intersection problem for linearly representable matroids. We start by recalling some definitions. For a more detailed exposition and further clarifications we refer to standard texts such as [Wes03].

Definition 6.1.1. *A* matroid M *is a pair* $(S, \mathcal{I})$*, where* S *is a finite set and* $\mathcal{I}$ *is a collection of subsets of* S *such that:*

1. *The empty set* $\emptyset$*, is in* $\mathcal{I}$*.*
2. *If* $X \in \mathcal{I}$ *and* $Y \subseteq X$*, then* $Y \in \mathcal{I}$*.*
3. *If* $X, Y \in \mathcal{I}$ *with* $|X| = |Y| + 1$*, then there exists* $x \in X - Y$ *such that* $Y \cup \{x\} \in \mathcal{I}$*. We refer to this condition as the* independence augmentation axiom.

We say that a subset X *of* S *is* independent *if* $X \in \mathcal{I}$*. Any subset of* S *not in* $\mathcal{I}$ *is said to be a* dependent set.

We next define linearly representable matroids.

Definition 6.1.2. *Let* $M = (S, \mathcal{I})$ *be a matroid and* $\mathbb{F}$ *be a field, where the underlying set* $S = \{1, 2, \cdots, |S|\}$ *without loss of generality. We say that* M *is linearly representable over* $\mathbb{F}$*, if for some positive integer* r*, there exists a matrix* $A \in \mathbb{F}^{r \times |S|}$ *such that a set of columns in* A *is linearly independent over* $\mathbb{F}$ *if and only if the corresponding set of column indices in* S *is in* $\mathcal{I}$*.*

Note 3. From the above definition it is easy to observe that if M is linearly representable over a field $\mathbb{F}$, then the representation need not be unique. For the results in this chapter we consider only linear representable matroids over $\mathbb{Q}$.

Let $M_1 = (S, \mathcal{I}_1)$ and $M_2 = (S, \mathcal{I}_2)$ be two matroids over the same set S. The *intersection* of the these matroids is a set system $(S, \mathcal{I})$ where

$$\mathcal{I} = \{A \subset S \mid A \in \mathcal{I}_1 \cap \mathcal{I}_2\}.$$

Given matroids $M_1 = (S, \mathcal{I}_1)$ and $M_2 = (S, \mathcal{I}_2)$ as input (accessed by their independence oracles) the *matroid intersection problem* is to find a set of maximum cardinality in $\mathcal{I}$. A decision version of this problem would be to check if there is a set of size at least k in $\mathcal{I}$, where k is given as part of the input.

Matroids are combinatorial objects that generalize the notions of linear independence and dependence of vectors in a vector space. The study of matroids, especially providing efficient algorithms for several problems related to matroids and in particular the matroid intersection problem is an important branch of combinatorial optimization [Wes03]. In fact the first polynomial time algorithm for the matroid intersection problem (not necessarily linear representable matroids) dates back to the work of Edmonds in [Edm65].

The focus of this chapter is the matroid intersection problem for linearly representable matroids. It is well known that the linearly representable matroid intersection problem generalizes the maximum matching problem for bipartite graphs. This is easy to observe. Let $G = (X, Y, E)$ be the given bipartite graph, where X and Y are the two disjoint subsets of the vertex set of G and E is the set of edges in G. We can now define two matroids keeping the underlying set S as E with respect to the partitions X and Y as follows. Let $M_X = (E, \mathcal{I}_X)$, where any subset $A \subseteq E$ is in $\mathcal{I}_X$ if and only if no two edges in A are incident with the same vertex from X. Similarly, we can define another matroid $M_Y = (E, \mathcal{I}_Y)$ with respect to the second partition Y of the vertex set of G. It is also easy to note that both these matroids are also linearly representable over $\mathbb{Q}$. The incidence matrix of the graph G with respect to the partition X is a linear representation of M_X over $\mathbb{Q}$, while the incidence matrix of G with respect to Y is a linear representation of the matroid M_Y over $\mathbb{Q}$. It then follows that the size of the maximum matching in G equals the maximum size of any set in $\mathcal{I}$, which is the collection of independent subsets obtained by intersecting matroids M_X and M_Y.

Just as the maximum matching problem was shown to be in RNC in [MVV87], it is shown in [NSV94] by Narayanan et.al that the matroid intersection problem for linearly representable matroids is in RNC. Their RNC algorithm closely follows the approach of [MVV87]. It is basically an application of the isolating lemma of [MVV87] combined with a clever use of the Cauchy-Binet theorem that enables them to pick out a maximum size set in the matroid intersection in RNC.

A major open problem in the area of parallel algorithms is whether the maximum

matching problem, or even the perfect matching problem is in deterministic NC. Indeed, this question is open even for bipartite graphs. Grigoriev and Karpinski [GK87] made some progress on this question. Under the promise that the input graph has at most polynomially many perfect matchings they show deterministic NC algorithms for finding and enumerating all perfect matchings. In a recent elegant paper by Agrawal et al [AHT07] the upper bound for the problem was improved $\mathrm{L}^{\mathrm{GapL}}$.

In this chapter we study a similar promise version of linearly representable matroid intersection LINMATINTpoly defined below.

> Let $M_1, M_2 \in \mathbb{Q}^{m \times n}$ be $m \times n$ matrices that linearly represent matroids $M_1 = (S, \mathcal{I}_1)$ and $M_2 = (S, \mathcal{I}_2)$, where $S = [n]$. Additionally, suppose the matroids fulfil the promise that their intersection $\mathcal{I}$ contains at most $p(n)$ many sets of cardinality m, where $p(n)$ is a fixed polynomial. Then the problem LINMATINTpoly is to determine if $\mathcal{I}$ has a set of size m and if so then compute such a set.

6.1.1 Our Results

We show that LINMATINTpoly is in the class $\mathrm{L}^{\mathrm{GapL}}$ and is hard for co-$\mathrm{C}_{=}\mathrm{L}$.

Remark 4. Notice that the problem LINMATINTpoly is actually parameterized by the polynomial $p(n)$ bounding the number of maximum cardinality independent sets in the intersection. However to avoid cumbersome notation we do not write the parameter with the problem.

Additionally, we also observe that the RNC algorithm of [NSV94] for the general linearly representable matroid intersection problem actually places the problem in $\mathrm{L}^{\mathrm{GapL}}/\mathrm{poly}$. Furthermore, under the conditional hardness stated in Chapter 3 (Theorem 3.2.18) we can obtain a derandomization to get an $\mathrm{L}^{\mathrm{GapL}}$ upper bound.

When an arbitrary unweighted bipartite graph is given as input, the authors in [AHT07, Lemma 3.2] describe a deterministic weight assignment scheme to the edges of the given input graph to isolate perfect matchings. For LINMATINTpoly we use their idea to give a similar deterministic weight assignment scheme to the columns of the given linear representation. This gives us the $\mathrm{L}^{\mathrm{GapL}}$ upper bound for LINMATINTpoly.

Essentially the same algorithm yields $\mathrm{L}^{\mathrm{GapL}}$ algorithms for counting and listing all sets of maximum cardinality for inputs to LINMATINTpoly.

Finally, we provide an algorithm to check if the intersection $(S, \mathcal{I})$ so obtained is itself a matroid or not in $\mathrm{L}^{\mathrm{GapL}}$. We then conclude with a discussion and an open problem concerning linear representation of matroids over $\mathbb{Q}$.

6.2 Basic Results

In this section we recall some basic properties of matroids from [Wes03].

Definition 6.2.1. *Let $M = (S, \mathcal{I})$ be a matroid, and let $X \in \mathcal{I}$. We say that X is a base if $X \not\subseteq Y$, where $Y \in \mathcal{I}$ with $X \neq Y$. In other words, a* base *is a maximal independent set of the given matroid M.*

Definition 6.2.2. *Let $M = (S, \mathcal{I})$ be a matroid, and let $X \in \mathcal{I}$. We say that X is a circuit if $X \notin \mathcal{I}$, but every proper subset Y of X is in $\mathcal{I}$. In other words, a* circuit *is a minimal dependent set of the given matroid M.*

We recall some properties of bases in a matroid. Using these results we show a characterization of matroids in terms of the bases in $\mathcal{I}$. The results in this subsection are well known and we refer to [Wes03] for further clarifications.

Proposition 6.2.3. *Let $M = (S, \mathcal{I})$ be a matroid, and $\mathcal{B}$ be the collection of all bases in $\mathcal{I}$. If $B_1, B_2 \in \mathcal{B}$ then $|B_1| = |B_2|$.*

Proof. Let us start by assuming the contrary, that is $|B_1| < |B_2|$. Since M is a matroid, by the independence augmentation axiom given in Definition 6.1.1, there is an element $x \in B_2 - B_1$ such that $B_1 \cup \{x\} \in \mathcal{I}$. But this contradicts the maximality of B_1 in $\mathcal{I}$. Thus $|B_1| \geq |B_2|$. Essentially the same argument holds to show that $|B_2| \geq |B_1|$ from which the claim follows. ∎

Lemma 6.2.4. *Let $M = (S, \mathcal{I})$ be a matroid, and $\mathcal{B}$ be the collection of all bases in $\mathcal{I}$. If $B_1, B_2 \in \mathcal{B}$ and $x \in B_1 - B_2$, then there exists $y \in B_2 - B_1$ such that $(B_1 - \{x\}) \cup \{y\} \in \mathcal{B}$.*

Proof. From Proposition 6.2.3 we have $|B_1| = |B_2|$. Let $x \in B_1 - B_2$ as given above and let $B_1' = B_1 - \{x\}$. Clearly $B_1' \subseteq B_1$ and so $B_1' \in \mathcal{I}$. According to the independence augmentation axiom given in Definition 6.1.1, we have $y \in B_2 - B_1'$ such that $B_1' \cup \{y\} \in \mathcal{I}$. Note that $y \neq x$, since $x \in B_1 - B_2$, which implies $y \in B_2 - B_1$. Moreover, $B_1' \cup \{y\} \in \mathcal{I}$ and so is contained in some maximal independent set B. Once again by Proposition 6.2.3, $|B| = |B_1| = |B_1' \cup \{y\}|$ which implies $B \subseteq B_1'$ and hence the claim follows. ∎

The condition stated in Lemma 6.2.4 satisfied by bases of a matroid is also known as the *base exchange axiom*. In fact the converse of the above result is also true.

Lemma 6.2.5. *Let S be a set and $\mathcal{B}$ be a collection of subsets of S such that $\mathcal{B}$ is non-empty. Also assume that for any $B_1, B_2 \in \mathcal{B}$ and $x \in B_1 - B_2$, there exists $y \in B_2 - B_1$ such that $(B_1 - \{x\}) \cup \{y\} \in \mathcal{B}$. Let $\mathcal{I}$ denote the collection of subsets of sets in $\mathcal{B}$. Then $M = (S, \mathcal{I})$ is a matroid with $\mathcal{B}$ as its collection of bases.*

As a result of Proposition 6.2.3, Lemma 6.2.4 and Lemma 6.2.5, we obtain the following characterization of a matroid in terms of bases.

Theorem 6.2.6. *Let S be a set of elements and $\mathcal{I}$ be a collection of subsets of elements in S. Then, $M = (S, \mathcal{I})$ is a matroid if and only if $\mathcal{B}$, the collection of maximal sets in $\mathcal{I}$, is non-empty and sets in $\mathcal{B}$ satisfy the base exchange axiom stated in Lemma 6.2.4.*

6.3 Polynomially Bounded Linear Matroid Intersection

We recall the definition of LINMATINTpoly. For notational convenience, we denote both the input matroids and their linear representations by M_1 and M_2, and it will be clear from the context.

We start with a deterministic Isolating Lemma based on the ideas of [AHT07], applied to any set system $(X, \mathcal{F})$.

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 $w_i(j) = (i^j \, mod \, r)$. Further for each subset $Y \subseteq X$ define*

$$w_i(Y) = \sum_{j \in Y} w_i(j) (mod \; r).$$

Then there exists a weight function w_m such that $w_m(Y) \neq w_m(Y') (mod \; r)$ 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 $q_Y(z) = \sum_{j \in Y} z^j$ 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. ∎

Remark 5. Recall our matroid intersection problem context: let $M_1, M_2 \in \mathbb{Q}^{m \times n}$ be the input to LINMATINTpoly. Then, in the above lemma, we would have X to be the elements of the underlying set $S = \{1, \cdots, n\}$, and $\mathcal{F}$ is the collection of size m sets in $\mathcal{I}$, where $\mathcal{I}$ is the collection of independent sets in the intersection of the matroids M_1 and M_2. The input promise for LINMATINTpoly guarantees that $|\mathcal{F}| \leq p(n)$ for the polynomial $p(n)$.

In [AHT07] permutations that constitute perfect matchings in bipartite and general graphs are identified similarly. The underlying set X corresponds to entries of the adjacency matrix of the graph (that is the edges of the bipartite graph) and the collection $\mathcal{F}$ corresponds to permutations that define perfect matchings.

Let $M_1, M_2 \in \mathbb{Q}^{m\times n}$ be an instance of LINMATINTpoly. We will apply the Cauchy-Binet theorem (see Theorem 2.2.1 of Chapter 2) to expand $\det(M_1 M_2^T)$. Recall that we will obtain

$$\det(M_1 M_2^T) = \sum_{\alpha} \det(M_{1,\alpha}) \det(M_{2,\alpha}),$$

where $\alpha \subseteq \{1, \ldots, n\}$ with $|\alpha| = m$ representing all possible ways of choosing m indexes from a set of n indexes. Here $M_{1,\alpha}$, and $M_{2,\alpha}$ denote $m \times m$ submatrices of M_1 and M_2 respectively, formed by picking columns corresponding to indexes in α.

Notice that a term indexed by α makes a nonzero contribution to this summation precisely when the subset α is a size m independent set in both matroids M_1 and M_2. In other words, the term indexed by α makes a nonzero contribution to the summation if and only if $\alpha \in \mathcal{F}$, where $\mathcal{F} \subset \mathcal{I}$ is the collection of the at most $p(n)$ many sets of size m in $\mathcal{I}$. Thus there are at most $p(n)$ many nonzero terms in the above summation.

In order to identify the terms in the summation, we assign weights given by Lemma 6.3.1 to the entries of the first matrix M_1 to get a new matrix M_1', before applying the Cauchy-Binet theorem to analyze $\det(M_1' M_2^T)$. We note that by assigning a weight w to a column of M_1 we mean multiplying the entries of that column by x^w, where x is an indeterminate.

Notice that $\det(M_1' M_2^T)$ is a univariate polynomial in $\mathbb{Q}[x]$ as $M_1' M_2^T$ is a matrix whose entries are univariate polynomials in $\mathbb{Q}[x]$. For any i, the coefficient of x^i in the above determinant is a GapL computable function [AO96, Tod91a, Vin91]. The choice of weights will allow us to retrieve the columns that contribute to size m subsets in the matroid intersection $\mathcal{I}$.

6.3.1 An $\mathrm{L}^{\mathrm{GapL}}$ Algorithm for LINMATINTpoly

We now formally describe the algorithm. The algorithm and its proof of correctness are based on Lemma 6.3.1. Let $n = |S|$ and $p(n)$ be the polynomial upper bounding the number of sets of maximum cardinality in $\mathcal{I}$.

CAUCHY-BINET(M_1, M_2)
Choose a prime $r > (n+1)^2 p^2(n)$.
for ($i \leftarrow 1$ to r)
    for ($j \leftarrow 1$ to n)

Let $w_i(j) \leftarrow i^j (\text{mod } r)$.
Multiply the j^{th} column of M_1 by $x^{w_i(j)}$.
(* Here x is an indeterminate *).
endfor
Let M_1' denote the resulting matrix.
Let $N^{(i)} \leftarrow M_1' M_2^T$.
Output $N^{(i)}$.
endfor

For each weight function given by Lemma 6.3.1, the procedure CAUCHY-BINET(M_1, M_2) produces a matrix $N^{(i)}$. We observe that $\det(N^{(i)})$ is a polynomial $P_i(x)$, of degree bounded by mr. Let $P_i(x) = \sum_{k=1}^{mr} P_{ik} x^k$. Then each P_{ik} is a GapL computable function.

Let $\{S_1, S_2, \cdots, S_t\} = \mathcal{F}$. That is, the S_i are the size m sets in the intersection $\mathcal{I}$ of the two input matroids, where $t \leq p(n)$. By Lemma 6.3.1 there is a weight function say w_j, which takes distinct values on all sets in $\mathcal{F}$. Then $w_j(S_k) \neq w_j(S_\ell)$, for $1 \leq k < \ell \leq t$. We now focus on w_j for the rest of the discussion.

As already observed, for a weight function w_i, in general $\det(N^{(i)})$ has exactly t nonzero terms in the Cauchy-Binet expansion, one for each index $\alpha = S_\ell$ $1 \leq \ell \leq t$. However notice that the polynomial $P_i(x) = \sum_{k=1}^{mr} P_{ik} x^k$ may have fewer than t terms if there are two different subsets $S_{\ell'}$ and S_ℓ that have the same weight k. In this case the terms corresponding to $S_{\ell'}$ and S_ℓ in the Cauchy-Binet expansion of $\det(N^{(i)})$ will both contribute to P_{ik}. However, for the weight function w_j that isolates the family $\mathcal{F}$, Lemma 6.3.1 guarantees that the terms corresponding to distinct subsets $S_{\ell'}$ and S_ℓ will necessarily have different weights and hence contribute to distinct P_{ik}. In other words, the polynomial $\det(N^{(i)}) = \sum_{k=1}^{mr} P_{jk} x^k$ has exactly t distinct nonzero terms, one corresponding to each subset $S_\ell \in \mathcal{F}$.

This will straightaway give an L^{GapL} algorithm for computing t. It is the maximum number of terms that any of the polynomials P_i can have. Conversely, it is also clear that a weight function w_i for which the number of terms in P_i attains the maximum is an isolating weight function for the family $\mathcal{F}$.

Theorem 6.3.2. *For inputs $M_1, M_2 \in \mathbb{Q}^{m \times n}$ to* LINMATINTpoly *there is a* L^{GapL} *algorithm for computing the number of size m independent sets in the matroid intersection.*

We now describe an L^{GapL} algorithm for listing all the sets in $\mathcal{F}$. Let w_j be an isolating weight function for $\mathcal{F}$, and let $\det(N^{(j)}) = P_{jk_1} x^{k_1} + P_{jk_2} x^{k_2} + \cdots + P_{jk_t} x^{k_t}$. For $1 \leq \ell \leq t$ let $S_\ell \in \mathcal{F}$ be the size m subset corresponding to the coefficient P_{jk_ℓ}. In order to find out if $s \in [n]$ belongs to S_ℓ we transform $N^{(j)} = M_1' M_2^T$ into a new matrix $M^{(j)} = M_1'' M_2^T$, where M_1'' is obtained from M_1' by multiplying each entry of the s^{th}

column with a new indeterminate y. It is easy to see that $\det(M^{(j)})$ assumes the form

$$\det(M^{(j)}) = \sum_{\ell=1}^{t} P_{jk_\ell} x^{k_\ell} y^{b_\ell},$$

where $b_\ell = 1$ if $s \in S_\ell$ and $b_\ell = 0$ if $s \notin S_\ell$.

It follows easily that testing if $s \in S_\ell$ for $1 \leq \ell \leq t$ can be done by an $\mathrm{L}^{\mathrm{GapL}}$ computation. Repeating this test for each $s \in [n]$ will identify all the sets $S_\ell \in \mathcal{F}$. We summarize the result below.

Theorem 6.3.3. *Given an input $M_1, M_2 \in \mathbb{Q}^{m \times n}$ to* LINMATINTpoly *there is an* $\mathrm{L}^{\mathrm{GapL}}$ *algorithm for listing all the size m independent sets in the intersection of the two matroids.*

We now show that the decision version of LINMATINTpoly is hard for co-$\mathrm{C}_=\mathrm{L}$ under logspace many-one reductions. The decision version of LINMATINTpoly has input instances $M_1, M_2 \in \mathbb{Q}^{m \times n}$. Here the matroid pairs (M_1, M_2) fulfil the promise of LINMATINTpoly and (M_1, M_2) is a yes instance if and only if there is a size m independent set in the matroid intersection.

The problem of checking if a matrix $M \in \mathbb{Q}^{n \times n}$ is non-singular or not is logspace many-one complete for the class co-$\mathrm{C}_=\mathrm{L}$ by [AO96]. Consider the matroid (also denoted M) that is linearly represented by such a matrix M. Since M has rank at most n, the corresponding matroid M has either one or no independent set of size n. The matrix $M \in \mathbb{Q}^{n \times n}$ is non-singular if and only if the matroid M and the matroid represented by the identity matrix I_n are identical. Therefore, given the input as M the reduction maps it to the instance (M, I_n, n). Notice that this is an instance of LINMATINTpoly because the number of size n independent sets in the intersection is at most 1. Furthermore, M is non-singular if and only if the size of the maximal independent set in intersection of the two matroids is 1.

Theorem 6.3.4. *The decision version of the* LINMATINTpoly *problem is logspace many-one hard for* co-$\mathrm{C}_=\mathrm{L}$.

6.4 Unrestricted Linear Matroid Intersection

We show in this section that there is a nonuniform $\mathrm{L}^{\mathrm{GapL}}$ for solving linear matroid intersection in general. The algorithm is exactly the Narayanan et al RNC algorithm [NSV94]. We only observe the nonuniform $\mathrm{L}^{\mathrm{GapL}}$ upper bound for it. For the sake of completeness we give a quick sketch of the proof.

Let $M_1, M_2 \in \mathbb{Q}^{m\times n}$ be the input instance of the problem, where our goal is to find a maximum cardinality set in the intersection. We first explain an easily computable transformation of (M_1, M_2) to another pair of matrices (N_1, N_2), where $N_1, N_2 \in \mathbb{Q}^{m\times(n+m^2)}$ such that M_1 is the first n columns of N_1 and M_2 is the first n columns of N_2. Thus, every subset $S \subseteq [n]$ that is in the intersection of matroids M_1 and M_2 is also in the intersection of matroids N_1 and N_2. Furthermore, the transformation will ensure that this set S can be extended to a size m independent set in the intersection of N_1 and N_2. This construction is from [NSV94] applied to general linear representable matroid intersection. We now explain the construction.

To obtain N_1 we simply augment m copies of the identity matrix I_m to M_1, so $N_1 = [M_1\ I_m \cdots I_m]$. To get N_2 we augment M_2 differently. Let $I_m^{(i)}$ denote the matrix obtained by an i-place cyclic shift of the columns of I_m, for $1 \le i \le m$. We augment M_2 by $I_m^{(i)}$, $1 \le i \le m$ to obtain N_2.

This construction guarantees the claimed extension property: for any set $S \subseteq [n]$ of, say, k columns that are independent in both M_1 and M_2, we can find a set T of $m-k$ indices in the range $n+1 \cdots n+m^2$ such that the columns indexed by $S \cup T$ are independent in both N_1 and N_2. In particular, this property holds for sets S of maximum cardinality in the intersection of matroids M_1 and M_2.

We again apply the Cauchy-Binet theorem to expand $\det(N_1 N_2^T)$. We obtain

$$\det(N_1 N_2^T) = \sum_{\alpha} \det(N_{1,\alpha}) \det(N_{2,\alpha}),$$

where $\alpha \subseteq \{1, \ldots, n+m^2\}$ with $|\alpha| = m$ representing all possible ways of choosing m indexes from a set of $n+m^2$ indexes. Here $N_{1,\alpha}$, and $N_{2,\alpha}$ denote $m \times m$ sub submatrices of N_1, and N_2 respectively, formed by picking columns corresponding to indexes in α.

Notice that a term indexed by α makes a nonzero contribution to this summation precisely when the subset α is a maximum cardinality independent set in both matroids N_1 and N_2. However, we are actually interested in the maximum cardinality independent sets in both M_1 and M_2. In any nonzero term $\det(N_{1,\alpha}) \det(N_{2,\alpha})$ the set of columns corresponding to indexes in $[n] \cap \alpha$ are linearly independent in both M_1 and M_2. In order to identify the contribution of the columns of M_1 and M_2 in this expansion, we will assign randomly chosen weights to the entries of the two matrices N_1 and N_2 before applying the Cauchy-Binet theorem. More precisely, we will assign weights to columns corresponding to N_1 and N_2 using the isolating lemma of [MVV87] as follows. We randomly pick $w_i \in [2(n+m^2)]$ for $1 \le i \le n+m^2$ and multiply the ith column of N_1 by x^{w_i} for $1 \le i \le n$ and by $x^{w_i+2m(n+m^2)}$ for $n+1 \le i \le n+m^2$. Let this new matrix be N_1'. Now we can use the Cauchy-Binet theorem to analyze $\det(N_1' N_2^T)$, which is a polynomial

in $\mathbb{Q}[x]$. As shown in [NSV94, Theorem 4.2], with probability at least 1/2 there is a unique minimum weight set α of maximum cardinality in the matroid intersection of N_1 and N_2. Let w_α denote its weight. Then the coefficient of the minimum power of x in $\det(N_1' N_2^T)$ (which is x^{w_α}) is nonzero with probability at least 1/2. Moreover, the extra weight of $2m(n+m^2)$ on each of the last m^2 columns of N_1' ensures that α must contain a maximum cardinality independent set from intersection of M_1 and M_2. We can extract this particular maximum cardinality independent set by using the same technique as in Section 6.3. For $1 \leq s \leq n$ we will multiply the sth column of N_1' by a new indeterminate y to obtain matrix N_1''. If we now compute $\det(N_1'' N_2^T)$ we will see that the coefficient of x^{w_α} will have y occurring in it if and only if s is in the isolated maximum size independent set of the intersection of M_1 and M_2. By standard probability amplification we can convert the random bits into a polynomial size advice string. The rest of the computation is clearly $\mathrm{L}^{\mathrm{GapL}}$.

Theorem 6.4.1. *Linear matroid intersection is in* $\mathrm{L}^{\mathrm{GapL}}/\mathrm{poly}$.

Applying Theorem 3.2.18 (of Section 3) we can obtain the following conditional upper bound.

Corollary 6.4.2. *Suppose* $L \in \mathrm{DSPACE}(n)$ *such that for some constant* $\epsilon > 0$ *and all but finitely many* n*, no* n*-input circuit* C *of size at most* $2^{\epsilon n}$ *accepts exactly strings of length* n *in* L*. Then the linear matroid intersection problem is in* $\mathrm{L}^{\mathrm{GapL}}$.

6.5 Discussion

Let us recall Definition 6.1.2. We assume the underlying field $\mathbb{F}$ in our case to be $\mathbb{Q}$, the set of all rational numbers. As mentioned in Note 3, it is easy to observe that for a matroid M, its linear representation need not be unique. For instance, the matroid represented by the $n \times n$ identity matrix I_n is the same as the matroid represented by any $n \times n$ non-singular matrix over $\mathbb{Q}$. Thus the following problem stems naturally from the definition of linear representation of matroids.

> Equality Checking for Linear Representations (ECLR): Given two linear representations over $\mathbb{Q}$, is there a polynomial time algorithm that determines if they both represent the same matroid.

From now on, we denote by ECLR the set of all pairs (M_1, M_2), where $M_1, M_2 \in \mathbb{Q}^{m \times n}$, such that the matroid represented by M_1 and by M_2 over $\mathbb{Q}$ is the same. Similarly, $\overline{\mathrm{ECLR}}$ denotes the set of all pairs (M_1, M_2), where $M_1, M_2 \in \mathbb{Q}^{m \times n}$, but the matroid represented by M_1 is not the matroid represented by M_2 over $\mathbb{Q}$.

In the following, we observe some basic results about ECLR. Given two linear representations M_1 and M_2 over $\mathbb{Q}$, any set of indexes such that columns corresponding to these indexes are linearly independent in M_i but not in M_j, where $1 \leq i, j \leq 2$ with $i \neq j$, is a witness showing that M_1 and M_2 represent different matroids. Since a nondeterministic machine could check if such a witness exists in polynomial time, it follows that ECLR is in co-NP. Checking if a rational matrix is non-singular or not is complete for co-$\mathrm{C_{=}L}$. This problem trivially reduces to ECLR. Given $M \in \mathbb{Q}^{n \times n}$ as input, we output M and the identity matrix I_n. Clearly, M is non-singular if and only if the matroid represented by M and I_n over $\mathbb{Q}$ are the same. Thus ECLR is hard for co-$\mathrm{C_{=}L}$.

6.5.1 Reduction from Search to Decision for $\overline{\text{ECLR}}$

In this section we show that the decision version and the search version of ECLR are polynomial time equivalent. Assume that there is a polynomial time algorithm that decides ECLR. Then, given linear representations $M_1, M_2 \in \mathbb{Q}^{m \times n}$, let $\text{ECLR}(M_1, M_2)$ be the subroutine that outputs 1 if the matroid represented by M_1, and M_2 is the same, and outputs 0 otherwise. We also denote the matroid represented by M_1 and M_2, by M_1 and M_2 respectively. Assume that the input M_1 and M_2 represent different matroids. The polynomial time procedure described below outputs a set of indexes such that columns corresponding to these indexes form a circuit (refer Definition 6.2.2) in M_i, but corresponding columns are linearly independent in M_j using the algorithm for deciding ECLR as an oracle, where $1 \leq i, j \leq 2$ with $i \neq j$.

Given any $X \subseteq S = \{1, \ldots, n\}$, and $j \in \{1, 2\}$, let $M_j^{(X)}$ denote the matrix obtained from M_j by retaining columns whose indexes correspond to integers in X. We denote the matroid so obtained from M_j by $(S, \mathcal{I}_j^{(X)})$, where $\mathcal{I}_j^{(X)} = \{X \cap I | \text{for } I \in \mathcal{I}_j\}$. We start by assuming that M_1 and M_2 represent different matroids. Let $i = 1$, $X = \{2, \ldots, n\}$, and $Y = \emptyset$. We now query the ECLR oracle if $M_1^{(X)}$ and $M_2^{(X)}$ represent the same matroid. If the oracle outputs 1, then it is clear that the i^{th} element of S, represented by the i^{th} column in M_1 and M_2, is in every subset of S that forms a circuit in $M_k^{(X)}$ but is linearly independent in $M_l^{(X)}$, where $1 \leq k, l \leq 2$ with $k \neq l$. In this case, we include i in the set Y, increment i, and re-initialize $X = Y \cup \{(i+1), \ldots, n\}$. However, if the ECLR oracle outputs 0 upon receiving input $M_1^{(X)}$ and $M_2^{(X)}$, it is clear that there exists some subset of X that forms a circuit in one of the input linear representations but is linearly independent in the other. In this case we do not include i in Y, but just increment i, and re-initialize $X = Y \cup \{(i+1), \ldots, n\}$. We repeat the above procedure until $i \leq n$. It is easy to note that the set Y that we finally obtain is a set of indexes such that columns corresponding to it form a circuit in one of the linear representations but not in the other. The steps given above involve retaining some set of columns of the given input matrices

and querying the ECLR oracle. Clearly, these steps are polynomial time computable, and hence the claim follows.

One of the most standard methods for computing bases in a matroid is to augment columns into the base set as long as linear independence of vectors in it is preserved. However it is unknown if there exists any such polynomial time procedure to compute the size of the smallest circuit in a matroid given by its linear representation.

6.5.2 A Hard Counting Problem related to $\overline{\text{ECLR}}$

Given linear representations $M_1, M_2 \in \mathbb{Q}^{m \times n}$ for two matroids, any set of indexes, columns corresponding to which form a circuit in one of the representations but the corresponding columns in the other matrix are linearly independent is a witness to the fact that the input matroids are different. We show that counting the number of such witnesses is #P-hard under polynomial-time Turing reductions: given as oracle the function for counting the number of witnesses for any instance of $\overline{\text{ECLR}}$, we can compute any other function in #P.

Given a simple undirected connected graph $G = (V, E)$, the problem of counting the number of cycles in G is as hard as any other problem in #P. We can arrive at this result as follows. Given a graph $G = (V, E)$, we first replace each edge in G by a path of length $|V|^3$ to obtain a new graph $G_1 = (V_1, E_1)$. Then we replace each edge $(u, v) \in E_1$ of G_1 by two paths of length 2 each. More formally, we replace each $(u, v) \in E_1$ of G_1 by the four edges: $(u, x), (x, v), (u, y), (y, v)$. Let this new graph obtained after this replacement step from G_1 be denoted by $G_2 = (V_2, E_2)$. It can be easily observed that if there exists a Hamilton cycle in the input graph G, then correspondingly there exists a cycle of length $2|V|^3$ in G_2. Also any cycle in G_2 is of length at most $2|V|^3$. It can then be observed that the newly introduced edges in G_2 create an exponential gap between the number of cycles of length $2|V|^3$ and the number of cycles of length strictly less than $2|V|^3$. As a consequence, each bit of the number of Hamilton cycles in G (which correspond to number of cycles of length $2|V|^3$ in G_2) occupies a distinct position in the number of cycles of the graph G_2. To be more precise, the leading polynomially many bits of the number of cycles in G_2 gives us the number of Hamilton cycles in G. Thus from knowing number of cycles in G_2, we can compute the number of Hamilton cycles of the original graph G. Clearly, this reduction does not produce a one-one and onto mapping from any #P-complete problem to the problem of counting cycles in a given undirected connected graph. However, it shows that if we have a procedure to count the number of cycles, then we can in fact find the number of Hamilton cycles in any input graph.

We now return back to the problem of counting witnesses for inputs in $\overline{\text{ECLR}}$. Given any simple undirected connected graph $G = (V, E)$, we can define a linear representation

$M \in \mathbb{Q}^{m \times n}$ for a matroid known as the *cycle matroid* corresponding to G (refer [Wes03] for how the cycle matroid is defined). In this representation, there is a bijection between cycles in G and circuits in M. It is easy to note that the $n \times n$ identity matrix I_n does not contain any circuit. Thus, when given a graph G as input, output the linear representation of its cycle matroid and the identity matrix. Clearly number of cycles in G equals the number of subsets of columns that form a circuit in M but is linearly independent in I_n. Thus counting the number of subsets of $\{1, \ldots, n\}$ that witness the fact that the matroids represented by M and I_n are different is also as hard as any other counting problem in #P.

6.5.3 Remarks

None of the observations obtained above reveal any clue towards classifying the complexity of ECLR. In fact problems such as perfect matching and SAT have similar properties: equivalence of the decision version and the search version, along with the #P-completeness of the counting version. While perfect matching is in P, we know that SAT is NP-complete. We leave the problem of classifying the complexity of ECLR as an open question.

7

Cayley Table Group Theoretic Problems

7.1 Introduction

The goal of this chapter is to study the complexity of some group-theoretic computational problems assuming that the input group G is given by its multiplication table (i.e. its Cayley table).

Let C be an arbitrary subset of the group G and let $H = \langle C \rangle$, the group generated by the elements in C. We define the *Cayley graph* of G with respect to the set C to be $X(G, C) = (V, E)$, where $V = G$ is the set of vertices, and $E = \{(g, h) | g^{-1}h \in C\}$ is the set of edges. When C is closed under inverse, $(g, h) \in E$ if and only if $(h, g) \in E$, and hence $X(G, C)$ is undirected. But in general $X(G, C)$ is a directed Cayley graph. From the above definition we infer that the graph $X(G, C)$ is a graph-theoretic representation of the subgroup H and its left cosets in G. That is, the set of vertices in a connected component of $X(G, C)$ forms a left coset of H in G, while a directed path from a vertex g to another vertex h indicates that $h = gg'$, where $g' \in H = \langle C \rangle$. Moreover, there is a path from g to h in $X(G, C)$ if and only if there is a path from h to g. In other words, each connected component of $X(G, C)$ is in fact *strongly connected*. Therefore to check if there exists a path from g to h, we need to check if there is a path from g to h in the underlying undirected graph of $X(G, C)$. From these observations and using Reingold's result that undirected st-connectivity is in L [Rei05], it follows easily that the directed st-connectivity problem for Cayley graphs is in the complexity class L.

The precise classification of natural computational problems in terms of computational resources required by them is a central theme in complexity theory. Standard models of computation that are used for the classification of problems are usually Turing machine based, with appropriate space and time bounds. Nondeterminism or randomness are resources that play a key role in this classification. Also, it is often useful to study the circuit complexity resource bounds required for the problem, like size, depth and

uniformity conditions for a boolean circuit solving it. In particular, L, NL, RL, logspace counting classes, the NC and RNC hierarchies are typical examples of complexity classes that have arisen this way. Each of these classes contain a rich collection of natural problems from within P [All04]. Several natural problems in P [All04] that are not P-complete tend to fit into one of the above mentioned classes, in terms of completeness, with few exceptions. In [BKLM01], Barrington *et al.* study one such exception: the Cayley group membership problem (CGM) wherein the input group G, given by a Cayley table, is abelian, nilpotent or solvable. We formally define the problem CGM:

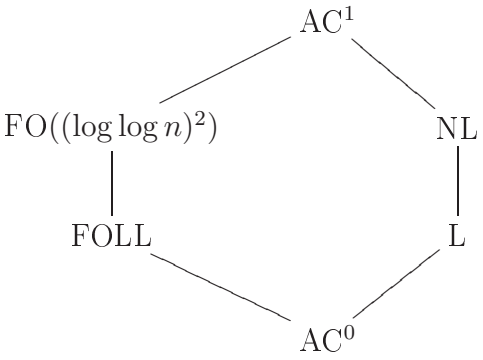
Cayley Group Membership problem (CGM): We are given a group G of order n by a Cayley table, a set $C \subseteq G$ and an element $t \in G$ as input. The problem is to determine if $t \in \langle C \rangle$.

Along with CGM, [BKLM01] also consider the problem of determining if an input group G given by its Cayley table is cyclic and similarly if G is nilpotent. We can apply Reingold's undirected *st*-connectivity result to easily show that CGM is in L: given an instance (G, C, t) of CGM, form the directed Cayley graph $X(G, C)$ and test if the identity element e and t are in the same connected component. Given any two vertices in $X(G, C)$, since G is given explicitly in terms of a Cayley table, a logspace machine can decide if there is an edge between the two vertices or not. Also, each of the connected components of X are strongly connected. Thus, we simply need to check if t is reachable from e in the underlying undirected graph $X(G, C)$.

Theorem 7.1.1. *The Cayley Group Membership problem is in* L.

In [BKLM01], Barrington *et al.* examine a different classification of CGM using the descriptive complexity approach. They use descriptive complexity methods, pioneered by Immerman in [Imm82] (also see the monograph [Imm99]), to obtain an interesting classification of CGM depending on whether G is abelian, nilpotent or solvable. More precisely, they introduce classes FO($\log \log n$), denoted by FOLL, and FO($d \log \log n$) (where d is the length of the lower central series or the derived series of G according as G is nilpotent or solvable respectively). Then they show that CGM problem for abelian, nilpotent and solvable groups are in the above two classes, respectively [BKLM01, Theorems 3.4, 3.5 and Corollary 3.2]. The result is significant due to the relation between FOLL and the conventional circuit complexity classes. Barrington *et al.* show that FOLL does not contain any class that contains parity and hence CGM problem for abelian and nilpotent groups is unlikely to be hard for *any class* containing parity. Circuit classes sharing some relation to FOLL are AC^0 and AC^1. It is known that $AC^0 \subseteq FOLL \subseteq AC^1$. Pictorially,

we have the following.



Apart from CGM, they also consider the problem of determining if an input group is cyclic or nilpotent. The crucial ingredient in these proofs is the notion of a *power predicate:* a simple new recursive strategy for parallel computation of powers of an element in a group given by a Cayley table. In other words, given two elements $a, b \in G$ and a non-negative integer i, [BKLM01] present a recursive definition to check if $a = b^i$ with the depth of the recursion at most $O(\log\log n)$, where $n = o(G)$. That is, the power predicate can be expressed in FOLL.

Motivated by the results of [BKLM01] we examine the complexity of several group-theoretic problems — well-studied in computational group theory (see, for example, [Bab92, Luk93]) — when the input groups are given by their Cayley tables. It turns out that several of these problems such as testing nilpotence, solvability, checking if the input group is simple or not, computing the normal closure, centralizer, and so on get classified into L as a consequence of CGM being in L. Finally we show a randomized test with constant error probability, to check if an input group G given by a Cayley table is abelian. This test makes constant number of queries to the Cayley table of G.

7.2 Definitions and Notations

We start by recalling the basic group theoretic definitions and notation from Chapter 2. In addition to these we also need the following. In all the definitions and results of this chapter we deal with finite groups.

Here $G \geq H$ or $H \leq G$ denotes that H is a subgroup of G. If $X \subseteq G$ then the subgroup generated by X is denoted by $\langle X \rangle$.

Definition 7.2.1. *Given a group G, the* lower central series *of G is $G = G_0 \geq G_1 \geq \cdots \geq G_k = G_{k+1}$ where, the group G_{i+1} is $\langle \{x^{-1}y^{-1}xy | x \in G \text{ and } y \in G_i\} \rangle$, for $0 \leq i \leq (k-1)$.*

Definition 7.2.2. *A group G is* nilpotent *if the lower central series of G terminates in*

the identity element.

Let p be a prime dividing $o(G)$. We say that $g \in G$ is a p-element if the order of g is p^k, for $k \geq 0$.

Remark 6. [Hal59] It is useful to recall another characterization of nilpotent groups: G is nilpotent iff each Sylow subgroup of G is normal. Hence, nilpotent groups are a direct product of their Sylow subgroups. Let G be a finite group and, for each prime factor p of $o(G)$, let S_p denote the set of p-elements in G. Then G is nilpotent if and only if S_p is a subgroup of G for each prime factor p of $o(G)$.

Definition 7.2.3. *Let G be a group. An element $x \in G$ is said to be a commutator if there exists $g, h \in G$ such that $x = g^{-1}h^{-1}gh$. The* derived subgroup *of G, denoted by $[G, G]$, is the group generated by all the commutators in G. The* derived series *of G is defined as $G = G_0 \geq G_1 \geq \cdots \geq G_k = G_{k+1}$ where, the group $G_{i+1} = [G_i, G_i]$, for $0 \leq i \leq (k-1)$.*

Definition 7.2.4. *A group G is* solvable *if the derived series of G terminates in the trivial subgroup $\{e\}$.*

Definition 7.2.5. *Let G be a group and $S \subseteq G$. We define the centralizer of S in G, denoted by $C_G(S)$, to be the set of all elements $g \in G$ such that $xg = gx$ for all $x \in S$.*

7.3 Group Properties in Deterministic Logspace

In this section we present our logspace upper bound results for some well-studied problems (for example, refer [Bab92, Luk93]) in the computational group theory literature.

Theorem 7.3.1. *Given a finite group G as input by its Cayley table, and a subset $C \subseteq G$, testing nilpotence of $\langle C \rangle$ is in logspace.*

Proof. Since the group G is given by a Cayley table, the prime factorization of $o(G)$, or any of its subgroups can be computed in logspace. Let $H = \langle C \rangle$. Thus, for every prime factor p of $o(H)$, let $S_p = \{g \in H \mid o(g) = p^k \text{ for some } k\}$. Recall the group-theoretic fact from Remark 6 given above, that $H = \langle C \rangle$ is nilpotent if and only if S_p is a subgroup of H for each prime p dividing $o(H)$. Let p be a prime dividing $o(H)$. To verify that S_p is a group, it suffices to check for each pair $x, y \in H$, with $x, y \in S_p$, whether $xy \in S_p$. Using Theorem 7.1.1, we can check in logspace if $x, y \in H$. If so, we can also compute $o(x)$, and $o(y)$ and then verify if these orders are powers of p, in logspace. For every pair of elements so obtained we need to check if $o(xy)$ is also a power of p, which can once again be done in logspace. ∎

Definition 7.3.2. *Given a group G and $C \subseteq G$, the normal closure of C in G is the smallest normal subgroup of G containing C.*

Theorem 7.3.3. *Given a finite group G as input by its Cayley table, a subset $C \subseteq G$, and $g \in G$, we can check if g is in the normal closure of C in logspace.*

Proof. Since G is given by a Cayley table, we can list all elements in G of the form $g_1 h g_2^{-1}$ in logspace, where $h \in C$. Let D be the set of elements so obtained. Now, checking if g is in the normal closure of C in G is the same as checking if g is in the group generated by the elements in D. Clearly, this step is logspace computable using Theorem 7.1.1, and hence the result follows. ∎

It is also possible to test if an input group G is solvable or not in logspace. For this, we need the following result of Guralnick and Wilson [GW00, Theorem A].

Theorem 7.3.4. [GW00] *A finite group G is solvable if and only if for $x, y \in G$ picked independently and uniformly at random, the subgroup $\langle x, y \rangle$ is solvable with probability at least* $11/30$.

As a corollary we obtain the following result.

Corollary 7.3.5. *Let G be a finite non-solvable group. Then, every minimal non-solvable group F of G is generated by a pair of elements $x, y \in F$.*

Proof. Let F be a minimal non-solvable subgroup of G. In other words, F is a subgroup of G such that, there is proper of F that is also non-solvable (due to this property, it is also easy to note that the derived subgroup of F is itself). It now follows from Theorem 7.3.4, that there exists at least $(19/30)o(F)$ pairs $x, y \in F$, that generate a non-solvable subgroup of F, which can only be F again, due to its definition. This completes the proof. ∎

Theorem 7.3.6. *Let G be a group containing n elements given in terms of a Cayley table, and let $C \subseteq G$. It is possible to test if $\langle C \rangle$ is a solvable group or not in* L.

Proof. Let $H = \langle C \rangle$, and assume it is not solvable. It then follows from Corollary 7.3.5 that any minimal non-solvable subgroup of H is generated by a pair of elements in H. We use this observation to arrive at the following test for checking if H is solvable or not.

For each distinct pair of elements $x, y \in G$, we can check if $x, y \in H$ in logspace using Theorem 7.1.1. Now, to test if H is solvable, we need to pick every possible pair of elements $x, y \in H$, and check if the derived subgroup of $\langle x, y \rangle$ is itself. In other words, if

both x and y are in the normal closure of the group generated by $xyx^{-1}y^{-1}$ and $x^{-1}y^{-1}xy$, which can be done in logspace using Theorem 7.3.3. ■

Guralnick and Wilson in [GW00, Theorem A] have also proved a result for nilpotent groups that is similar to Theorem 7.3.4 given above. We state this result below.

Theorem 7.3.7. [GW00] *A finite group G is nilpotent if and only if for $x, y \in G$ picked independently and uniformly at random, the subgroup $\langle x, y\rangle$ is nilpotent with probability at least $1/2$.*

Note 4. Let G be a group with n elements given by a Cayley table. It is well known that every nilpotent group is also solvable [Hal59], and moreover we also have a logspace algorithm in Theorem 7.3.6 that tests if G is solvable or not. Thus, without loss of generality assume the input group G is solvable, otherwise G is not nilpotent either. Now, similar to Corollary 7.3.5, it is easy to show that if H is a minimal non nilpotent subgroup of G, then H is generated by a pair of elements. Using this observation we describe another logspace algorithm to check if a given input group G is nilpotent.

For each pair of elements $x, y \in G$, we can compute the commutators of x and y, which are $x^{-1}y^{-1}xy$ and $xyx^{-1}y^{-1}$ in logspace. Other commutators obtained from x and y are in the group generated by these two elements. The derived subgroup of $H = \langle x, y\rangle$, denoted by H', is the normal closure of $x^{-1}y^{-1}xy$ and $xyx^{-1}y^{-1}$ in H. Using Theorem 7.3.3, we can check if any element of G is in H' in logspace. To check if H is not nilpotent, it suffices to check if both x and y are in the normal closure of the group generated by the commutators obtained from elements in H', and elements in $\{x, y\}$. Since we can list elements in H' using Theorem 7.3.3 in logspace, we can also compute the above mentioned commutators in logspace. It remains to check if x and y are in the group generated by these elements, which is also logspace computable using Theorem 7.1.1. This completes another test to check if H is not nilpotent.

We next examine the complexity of several other group-theoretic problems studied in the setting of permutation groups (and black-box groups) by Luks [Luk93], and Babai [Bab92]. However here we assume that the input group is given by a Cayley table. It turns out that all these problems are in L. We summarize these observations below.

Theorem 7.3.8. *Suppose G is a finite group given by its Cayley table. Let $B, C \subseteq G$ and $x \in G$.*

1. *Enumerating the elements of the subgroup $\langle B\rangle \cap \langle C\rangle$ is in* L. *Similarly, enumerating the elements of the coset $x\langle B\rangle \cap \langle C\rangle$ is in* L.

2. *Let $H = \langle B \rangle$. Enumerating elements in the normal closure $N_G(H)$ of the subgroup H of G is in* L. *Hence testing simplicity of H is also in* L. *Similarly, enumerating the elements in the centralizer $C_G(B)$ is also in logspace.*

3. *Checking if the groups $\langle B \rangle$ and $\langle C \rangle$ are conjugate: i.e. testing if there is $g \in G$ such that $g^{-1}\langle B \rangle g = \langle C \rangle$ is logspace computable.*

Proof.

1. Since reachability in the Cayley graphs $X(G, C)$ and $X(G, B)$ is in logspace, we can simply cycle through all elements $g \in G$ and output those g that are both reachable from x in $X(G, C)$ and e in $X(G, B)$.

2. It is easy to see that the normal closure $N_G(H)$ is the group generated by the set $S = \{ghg^{-1} \mid h \in C\}$. Using the algorithm of Theorem 7.1.1 a logspace transducer can cycle through each $g \in G$ and output g if $g \in \langle S \rangle$. Now, the group $\langle C \rangle$ is simple iff for each $x \in \langle C \rangle$ the normal closure $N_G(\langle x \rangle)$ is the entire group. Thus, simplicity testing is in logspace. The algorithm for centralizer is quite similar.

3. Given $B, C \subseteq G$, testing equality of the subgroups $\langle B \rangle$ and $\langle C \rangle$ is in L is an easy consequence of Theorem 7.1.1. Testing conjugacy of $\langle B \rangle$ and $\langle C \rangle$ amounts to testing if there is some $g \in G$ such that the groups $g\langle B \rangle g^{-1}$ and $\langle C \rangle$ are equal. Clearly, a logspace machine cycling through all $g \in G$ can test this property.

■

Remark 7. We note that several other group-theoretic objects can also be computed in logspace. For $H \leq G$, the *core* $Core_G(H) = \cap_{g \in G} H^g$ is the largest subgroup of H normalized by G. Given $x \in G$, it is easy to test in logspace the membership of x in $\cap_{g \in G} H^g$ (by cycling through $g \in G$ and testing if $x \in H^g$). Thus, $Core_G(H)$ can be listed out by a logspace computation.

However, there are other group-theoretic problems where input groups are given by Cayley tables that are computable in polynomial time, but the best space upper bound (achievable with polynomial running time seems to be $\log^2 n$). For example, we do not know if the problems of computing the Sylow subgroups of G or a composition series for G are in L. For these problems the best upper bound we know is AC^2.

7.3.1 Randomized Testing in Cayley Tables

Let G be a group of order n given by a Cayley table. In this section, we present some randomized algorithms to test if G is abelian, nilpotent or solvable. The goal is to design

randomized tests that make a sublinear number of probes to the Cayley table of the input group and decide with error probability bounded by ϵ, whether the input group satisfies the property or not, where $0 \leq \epsilon \leq 1$ is a constant. This is analogous to property testing. However, unlike the usual setting for property testing, we allow all inputs without any promise constraints.

We first take up the abelian property testing which makes queries to the Cayley table of G. We need the following lemma.

Lemma 7.3.9. *Let G be a non-abelian finite group and let h, h' be random elements of G. Then $\Pr([h, h'] \neq e) \geq 1/4$.*

Proof. As G is nonabelian its center C is a proper subgroup of G. Thus, $\Pr(h \notin C) \geq 1/2$. Furthermore, if $h \notin C$, its centralizer $C_G(h) = \{g \in G \mid gh = hg\}$ is also a proper subgroup of G. Hence, $\Pr(h' \notin C_G(h) | h \notin C) \geq 1/2$. Notice that $\Pr([h, h'] \neq e) = \Pr(h \notin C \wedge h' \notin C_G(h))$, and $\Pr(h \notin C \wedge h' \notin C_G(h)) = \Pr(h \notin C) \cdot \Pr(h' \notin C_G(h) | h \notin C) \geq 1/4$. This completes the proof. ∎

The following result is now immediate.

Theorem 7.3.10. *Let G be a group of order n given by its Cayley table and $0 < \epsilon < 1$ be a constant. Then with the probability of error bounded by ϵ it is possible to test if G is abelian with $O(\log 1/\epsilon)$ queries to the Cayley table.*

Proof. The test for abelianness is as follows:

1. Pick $O(\log 1/\epsilon)$ many pairs h_i, h_i' from G independently and uniformly at random.
2. If for some i, $[h_i, h_i'] \neq e$ then output G is nonabelian.
3. else output G is abelian and stop.

It suffices to note that the error in the test is one-sided: it can only fail when G is nonabelian. In such a case the error probability is bounded by $(3/4)^{O(\log(1/\epsilon))} = O(\epsilon)$. ∎

7.4 Discussion

Unlike arbitrary directed graph, Cayley graphs defined from finite groups given by a Cayley table, with respect to some subset that is closed under inverse, have more structure in it: a typical example is that each connected component of such a Cayley graph is in fact strongly connected. It thus prompts us to explore if properties of the underlying group

can be used to reduce the complexity of the *st*-connectivity problem in such graphs to a class contained in L, for example NC^1. We believe that it is unlikely for CGM to be complete for L.

On the other hand, certain other problems like computing the Sylow subgroups, and composition series are in NC but seem to elude classification into logspace counting classes. Does the Cayley table representation of the input group help us in placing any of these problems in L or in a class contained in L. If not, is it possible to arrive at hardness results for any of these problems. These problems seem natural and we leave them open.

Bibliography

[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.

[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*, volume LNCS 4393, pages 489–499, 2007.

[All04] Eric Allender. Arithmetic circuits and counting complexity classes. In Jan Krajicek, editor, *Complexity of Computations and Proofs*, volume 13 of *Quaderni di Matematica*, pages 33–72. Seconda Universita di Napoli, 2004.

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

[Apo86] Tom Apostol. *Introduction to Analytic Number Theory.* Springer, New York, 1986.

[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. In *CCC '04: Proceedings of the 19th IEEE Annual Conference on Computational Complexity*, pages 204–214, 2004.

[AV05] V. Arvind and T.C. Vijayaraghavan. The complexity of solving linear equations over a finite ring. In *STACS '05: Proceedings of the 22nd Annual Symposium on Theoretical Aspects of Computer Science*, volume LNCS 3404, pages 472–484, 2005.

[Bab92] László Babai. Bounded round interactive proofs in finite groups. *SIAM Journal on Discrete Mathematics*, 5(1):88–111, 1992.

[BDG88] José Luis Balcazar, Jose Diaz, and Joaquim Gabarro. *Structural complexity 1.* Springer-Verlag New York, Inc., New York, NY, USA, 1988.

[BDG91] José Luis Balcazar, Jose Diaz, and Joaquim Gabarro. *Structural complexity 2.* Springer-Verlag New York, Inc., New York, NY, USA, 1991.

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

[Ber84] Stuart Berkowitz. On computing the determinant in small parallel time using a small number of processors. *Information Processing Letters*, 18(3):147–150, 1984.

[BG92] Richard Beigel and John Gill. Counting classes: thresholds, parity, mods and fewness. *Theoretical Computer Science*, 103(1):3–23, 1992.

[BKLM01] David A. Mix Barrington, Peter Kadau, Klaus-Jörn Lange, and Pierre McKenzie. On the complexity of some problems on groups input as multiplication tables. *Journal of Computer and System Sciences*, 63(2):186–200, 2001.

[BL65] Garrett Birkhoff and Saunders Mac Lane. *A survey of modern algebra.* MacMillan, New York, 1965. Second edition, 1979.

[BLAS87] László Babai, Eugene Luks, and Ákos Seress. Permutation groups in NC. In *STOC '87: Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 409–420, 1987.

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

[Dam91] Carsten Damm. $DET=L^{\#L}$. Informatik-Preprint 8, Fachbereich Informatik der Humboldt-Universitat zu Berlin, 1991.

[Dic92] Leonard Dickson. *History of the theory of numbers, Volume 2: Diophantine Analysis.* Chelsea Publishing Company, 1992.

[Edm65] Jack Edmonds. Minimum partition of a matroid into independent subsets. *J. Res. National Bureau of Standards*, 69B:67–72, 1965.

[Gie95] Mark Giesbrecht. Fast computation of the Smith normal form of an integer matrix. In *ISSAC '95: Proceedings of the International Symposium on Symbolic and Algebraic Computation*, pages 110–118, 1995.

[GK87] Dima Grigoriev and Marek Karpinski. The matching problem for bipartite graphs with polynomially bounded permanents is in NC. In *FOCS '87: Proceedings of the 28th IEEE Foundations of Computer Science*, pages 166–172, 1987.

[GW00] Robert Guralnick and John Wilson. The probability of generating a finite soluble group. *Proceedings of the London Mathematical Society*, 81(3):405–427, 2000.

[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.

[Hal59] Marshall Hall. *The Theory of Groups*. MacMillan, New York, 1959.

[Her64] Ian Herstein. *Topics in Algebra*. Blaisdell Publishing Company, 1964.

[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.

[Imm82] Neil Immerman. Upper and lower bounds for first order expressibility. *Journal of Computer and System Sciences*, 25(1):76–98, 1982.

[Imm99] Neil Immerman. *Descriptive Complexity*. Springer-Verlag, New York, 1999.

[KL86] Ravi Kannan and Richard Lipton. Polynomial-time algorithm for the orbit problem. *Journal of the ACM*, 33(4):808–821, 1986.

[KS01] Adam Klivans and Daniel Spielman. Randomness efficient identity testing of multivariate polynomials. In *STOC '01: Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 216–223, 2001.

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

[KvM02] Adam Klivans and Dieter van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM Journal on Computing*, 31(5):1501–1526, 2002.

[Luk93] Eugene M. Luks. Permutation groups and polynomial-time computation. In *DIMACS: Discrete Mathematics and Theoretical Computer Science Series*, volume 11, pages 139–175. American Mathematical Society, 1993.

[Mar77] Daniel Marcus. *Number Fields.* Springer-Verlag, Berlin, 1977.

[MC87] Pierre McKenzie and Stephen Cook. The parallel complexity of abelian permutation group problems. *SIAM Journal on Computing*, 16(5):880–909, 1987.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[MV97] Meena Mahajan and V. Vinay. Determinant: combinatorics, algorithms and complexity. *Chicago Journal of Theoretical Computer Science*, 1997(5), December 1997.

[MVV87] Ketan Mulmuley, Umesh Vazirani, and Vijay Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

[NSV94] H. Narayanan, Huzur Saran, and Vijay Vazirani. Randomized parallel algorithms for matroid union and intersection with applications to arboresences and edge-disjoint spanning trees. *SIAM Journal on Computing*, 23(2):387–397, 1994.

[NW94] Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994.

[NZM01] Ivan Niven, Herbert Zuckerman, and Hugh Montgomery. *An Introduction to the Theory of Numbers.* John Wiley and Sons, 2001.

[Pap94] Christos Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[Rei05] Omer Reingold. Undirected *st*-connectivity is in log-space. In *STOC '05: Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 376–385, 2005.

[Ros93] Arnold L. Rosenberg. Cayley graphs and direct-product graphs. In *DIMACS: Discrete Mathematics and Theoretical Computer Science Series*, volume 11, pages 245–251. American Mathematical Society, 1993.

[Sch98] Alexander Schrijver. *Theory of Linear and Integer Programming.* John Wiley, 1998.

[Ser03] Ákos Seress. *Permutation group algorithms*, volume 152 of *Cambridge Tracts in Mathematics.* Cambridge University Press, Cambridge, 2003.

[Sho28] Kenjiro Shoda. Über die automorphismen einer endlichen abelschen gruppe. *Mathematische Annalen*, 100:674–686, 1928. Source: The Göttingen State and University Library (http://www.sub.uni-goettingen.de).

[Sip01] Michael Sipser. *Introduction to the Theory of Computation.* Thomson Brooks/Cole, 2001.

[Tod91a] Seinosuke Toda. Counting problems computationally equivalent to computing the determinant. Technical report 91-07, Department of Computer Science, University of Electro-Communications, Tokyo, Japan, 1991.

[Tod91b] Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991.

[Val79] Leslie G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[Val92] Leslie G. Valiant. Why is boolean complexity theory difficult? In *Proceedings of the London Mathematical Society symposium on Boolean function complexity*, pages 84–94, New York, NY, USA, 1992. Cambridge University Press.

[Vin91] V. Vinay. Counting auxiliary pushdown automata and semi-unbounded arithmetic circuits. In *CCC '91: Proceedings of 6th Structure in Complexity Theory Conference*, pages 270–284, 1991.

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

[Wes03] Douglas West. *Introduction to Graph Theory.* Prentice-Hall of India private limited, 2003. Second edition.

[Wie64] Helmut Wielandt. *Finite Permutation Groups.* Academic Press, New York, 1964.