A STUDY OF WIDTH BOUNDED ARITHMETIC CIRCUITS AND THE COMPLEXITY OF MATROID ISOMORPHISM

by

Raghavendra Rao B. V.

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

September 2009

Homi Bhabha National Institute

Recommendations of the Viva Voce Board

As members of the Viva Voce Board, we recommend that the dissertation prepared by **Raghavendra Rao B. V.** entitled "A study of width bounded arithmetic circuits and the complexity of matroid isomorphism" may be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

---------- **Date :**
Chairman : V. Arvind (IMSc)

---------- **Date :**
Convener : Meena Mahajan (IMSc)

---------- **Date :**
Member : Venkatesh Raman (IMSc)

---------- **Date :**
Member : K. V. Subrahmanyam (CMI)

---------- **Date :**
Member : Sundar Vishwanathan (IIT-B)

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 : Meena Mahajan

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.

Raghavendra Rao B. V.

ACKNOWLEDGEMENTS

First of all I thank my supervisor Meena Mahajan for her kind support and for providing all the motiviation, inspiration and knowledge which led to this thesis. Also, I would like to thank Appa, Amma, Akka, Lathakka, Shubha, bhava, Meghana and Mrinal for their uncodintional kindness and support, without which I could not have pursued research as a career. I thank all the faculty members of the TCS group at imsc for offering excellent courses, which certainly has motivated me to pursue research as a career. I also thank G. Sajith, for motivating me into theoretical computer science. I would like thank my collaborators Jayalal Sarma and Maurice Jansen. Some part of the thesis is an outcome of discussion with them.

I thank all my friends Aravind, Gaurav, Jayalal, Narayanan, Pradeep, Pushkar, Soumya, Srikanth, Thakur and Ved (I am sure I have missed many) for all the moral support they have provided. Finally I would like to thank all the administrative staff for the excellent facilities provided.

Abstract

The subject of computational complexity theory deals with characterization of computational problems with respect to various resource measures. This thesis can broadly be divided into two parts: (1) Study of width bounded arithmetic circuits and (2) Computational complexity of matroid isomorphism problems.

In the first part of the thesis we further the study of various arithmetizations of boolean complexity class NC^1. In particular, we propose constant width arithmetic circuits of polynomial degree as a new possible arithmetization of NC^1. Among the candidate arithmetizations of NC^1, constant width arithmetic circuits seem to be the larger ones. In particular, it is not known if constant width arithmetic circuits of polynomial size and degree can be efficiently simulated by arithmetic formulas of polynomial size.

However, the situation changes when we restrict ourselves to syntactic multilinear circuits. We show that constant width syntactic multilinear circuits are equivalent to constant width syntactic multilinear branching programs at polynomial size and hence contained inside syntactic multilinear polynomial size formulas. This makes syntactic multilinear formulas as the stronger one among the competent classes in the syntactic multilinear world.

Moving a step further, we show that the only known technique of simulating arithmetic formulas by constant width algebraic branching programs due to Ben-Or and Cleve does not preserve syntactic multilinearity. Moreover, for a generalized version of Ben-Or and Cleve's technique, we show a linear lower bound on the number of registers if it is to preserve syntactic multilinearity.

Also, we propose width of the arithmetic circuit as a possible measure of space for arithmetic computations. This definition gives a reasonable definition of lower space complexity classes compared to the existing notions of space. We also define and propose read-once certificates as a natural model of nondeterminism for space bounded arithmetic computations.

In the second part we study the complexity of isomorphism problem on matroids. We classify the problems according to how the input matroids are represented. In the most general case, when the matroids are given as independent set oracles, the problem is in Σ_2^p, the second level of the polynomial hierarchy. For linear matroids, we show that isomorphism testing is in Σ_2^p, and is unlikely

to be Σ_2^p complete. However when the rank of the given input matroid is a constant, the problem is polynomial time many-one equivalent to the graph isomorphism problem. For the case of matroids represented by graphs, we show that the isomorphism testing problem is polynomial time equivalent to the graph isomorphism problem. Along the way, we develop colouring techniques for handling coloured instances of matroid isomorphism problems. We also prove polynomial time equivalence of isomorphism testing problem and the problem of computing automorphism groups for the case of linear and graphic matroids.

Contents

List of Figures

Chapter 1

Introduction

This thesis can be broadly divided into two parts: 1) Study of arithmetic circuits of bounded width and 2) Computational complexity of matroid isomorphism problems.

1.1 Counting classes and Arithmetic Circuits

Informally, a complexity class is a set of languages $\{\mathcal{L} \subseteq \{0,1\}^*\}$ that are accepted by Turing machines with bounded resources. A language $\mathcal{L}$ can be equivalently defined by its characteristic function: $\chi_{\mathcal{L}} : \{0,1\}^* \to \{0,1\}$, where $f(x) = 1$ if and only if $x \in \mathcal{L}$. If $\mathcal{L}$ is decided by a non-deterministic Turing machine M we can extend $\chi_{\mathcal{L}}$ to a function $f_{\mathcal{L}} : \{0,1\}^* \to \mathbb{N}$, where $f_{\mathcal{L}}(x)$ is the number of accepting paths of M on input x. This leads to the definition of counting classes. $f_{\mathcal{L}}$ is called the counting function of $\mathcal{L}$.

The class $\#\mathsf{P}$ is the most popular among the counting classes. It was introduced by Valiant ([91], see also [36]) and is defined as the class of functions computable as the number of accepting paths of a polynomial time bounded non-deterministic Turing machine. So, $\#\mathsf{P}$ can be said to be the counting class associated with the class NP. In [91], Valiant has shown that the class $\#\mathsf{P}$ exactly characterises the complexity of computing the permanent of an $n \times n$ matrix with boolean (0/1) entries.

There has been a large amount of work on the counting complexity classes since the introduction of $\#\mathsf{P}$. Counting classes were useful in providing several

structural insights into the complexity class PP and the polynomial hierarchy (PH). ([83, 85, 14, 34, 35] or see [36] for a survey on the topic.) Also, the natural counting version #L (also, GapL) of NL was introduced, and Toda [84] has shown that GapL exactly characterises the complexity of computing the determinant of an $n \times n$ matrix, where GapL is the class of functions that can be written as the difference of two #L functions.

Let $\mathcal{C}$ be a non-deterministic complexity class. The counting class $\#\mathcal{C}$ can be seen as containing all functions that compute the number of "witness" paths in a machine accepting a language $\mathcal{L}$ from $\mathcal{C}$. We can extend the definition of counting classes to complexity classes defined over boolean circuits. In a boolean circuit such a witness is a sub-tree, usually referred to as an "accepting sub-tree". Equivalently, a counting function of a boolean circuit is the function (in fact a polynomial) computed by the circuit obtained by replacing the $\wedge$ gate by a $\times$ gate and a $\vee$ gate by a $+$ gate. This later process is referred to as arithmetisation of boolean functions ([14]) and has turned out to be a very powerful tool.

Though the counting model computes polynomials over $\mathbb{N}$ or $\mathbb{Z}$, the input is restricted to be boolean. A more general model of arithmetic circuits was also introduced by Valiant ([91, 92]). The main goal of this model was in characterising the complexity of algebraic computations over arbitrary fields. In [92] Valiant introduced and studied the classes VP and VNP as candidate algebraic variants of P and NP respectively. Valiant framed the VP vs VNP question as the algebraic analogue of the celebrated P vs NP question of boolean complexity theory. Valiant proved that the problem of computing the permanent of an $n \times n$ matrix is complete for the class VNP and also that the determinant of an $n \times n$ matrix can be computed in VP. So, the VP vs VNP question is roughly equivalent to the Determinant vs Permanent question.

Since then, the main focus of algebraic complexity has been to prove lower bounds against arithmetic circuits. Though super-polynomial lower bounds for most general forms of arithmetic circuits are yet unknown, there are lower bounds for restricted models such as non-commutative arithmetic circuits ([67]), monotone arithmetic circuits ([77, 73]) and multilinear formula ([70, 71]). Also, exponential lower-bounds for the case of constant depth circuits over finite fields are known. ([38, 39].)

Also, there is a significant amount of work done on the structural aspects of

algebraic complexity classes ([24, 25]) in the literature. Here the interest is in various characterisations of complexity classes and reductions between them. In [24] Burgisser studied the structural complexity of the classes VP and VNP and gave results analogous to those of the ones in the P vs NP setting. ([16, 54].)

While setting up a new theory, it is a common practice to relate it to the existing theories. Ideally, one expects each of the algebraic complexity classes to have boolean counterparts and the comparison between algebraic classes is similar to the one among their boolean counterparts. For example, we expect comparison between VP and VNP to be similar to that between P and NP. However, this need not be true in general: the VNC hierarchy collapses to the second level ([25]) whereas in the boolean world it is not known whether the NC hierarchy collapses or not.

1.1.1 Motivation

The class of boolean functions NC^1, computed by polynomial size boolean circuits of constant fan-in and $O(\log n)$ depth, is interesting not only because it represents the most efficiently parallelizable class of boolean functions but also since it is known to have many different characterisations. On of the most prominent such characterisations is in terms of polynomial size branching programs of constant width by Barrington [17]. Barrington proved a startling result that the class NC^1 can exactly be characterised by the class of boolean functions computable by polynomial size branching programs of constant width (BWBP). In the same paper, he observed that constant width boolean circuits of polynomial size (SC^0) are also equivalent to NC^1. Apart from the above, due to the old depth reduction technique of Spira [80], a polynomial size formula can be transformed into formula of logarithmic depth and polynomial size. Hence the polynomial size formula are equivalent to NC^1 as a polynomial size circuit with $O(\log n)$ depth and constant fan-in can easily be transformed into a polynomial size formula by simply unwinding the circuit into a formula. Our interest is in the arithmetizations of these classes.

However, arithmetizations of equivalent definitions of a complexity class need not be equivalent. Hence one could ask: How well are the arithmetizations of the above mentioned equivalent definitions of the class NC^1 related? Do they coincide?

In [28], Caussinus et. al introduced the arithmetizations of some of these classes. In particular, they consider the arithmetisation of NC^1 denoted by $\#\mathsf{NC}^1$ and bounded width branching programs denoted by $\#\mathsf{BWBP}$. They have shown that counting accepting paths in a constant width branching program can be represented as counting accepting sub-trees in a logarithmic depth formula. But, it is not clear if the converse is true: i.e can the number of accepting sub-trees be represented as the number of accepting paths in a constant-width branching program with only a polynomial blow-up in size? However, in [28] it is shown that if the difference of number of accepting and rejecting paths is considered, the class of functions representable as the difference of accepting and rejecting sub-trees of an NC^1 circuit (denoted by GapNC^1) is equivalent to those representable as the difference of accepting and rejecting paths in a polynomial size branching program of constant width (denoted by $\mathsf{GapBWBP}$), *i.e.* $\mathsf{GapNC}^1 = \mathsf{GapBWBP}$. This result is based on the 3-register linear straight line program simulation of arithmetic formula given by Ben-Or and Cleve ([18]).

One natural gap that remains to be filled in the above picture is to consider arithmetizations of the remaining classes that are equivalent to NC^1. In the next subsection we describe the progress made in this direction; this constitutes the first part of this thesis.

1.1.2 Contributions of the thesis

Inspired by [28], we further the study of arithmetizations of classes that are equivalent to NC^1 with a focus on circuits of constant width. However, a straightforward arithmetisation of constant width circuits does not look interesting as they could compute exponentially large values. We propose a polynomial degree restriction of bounded width circuits denoted by sSC^0. The boolean class sSC^0 itself seems to be interesting by its own right as it is already equivalent to NC^1. We also introduce a hierarchy sSC of classes sandwiched between NC^1 and SAC^1 by extending the definition of sSC^0 to poly-logarithmic width. We show that the sSC hierarchy as a whole is closed under complementation. (Theorem 3.4.) However, it is not known if the same holds for the individual classes of the hierarchy.

We introduce the arithmetizations of sSC^0 denoted by $\#\mathsf{sSC}^0$ and GapsSC^0 and attempt to compare $\#\mathsf{NC}^1$ with $\#\mathsf{sSC}^0$. If the difference between accepting and re-

jecting paths is considered then we show that GapNC^1 is contained in GapsSC^0. The relationship between GapNC^1, $\mathsf{GapBWBP}$ and GapsSC^0 is: $\mathsf{GapNC}^1 = \mathsf{GapBWBP} \subseteq \mathsf{GapsSC}^0$. (Proposition 4.9.) However the question of whether GapsSC^0 is contained in GapNC^1 remains open.

With the hope of resolving the above question and getting more relationship among the above arithmetizations we translate the above definitions into Valiant's arithmetic circuit model. We introduce the algebraic complexity classes VSC^0 and VsSC^0, where VsSC^0 roughly corresponds to GapsSC^0. It can easily be seen that the relationship among the gap-versions also hold in this model: $\mathsf{VNC}^1 = \mathsf{VBWBP} \subseteq \mathsf{VsSC}^0$, where VBWBP denotes the class of constant width algebraic branching programs of polynomial size. However, we were not able to show the containment of VsSC^0 in VNC^1 (arithmetic version of NC^1) in Valiant's model. As a step towards breaking this wall, we consider the restriction of syntactic multilinearity on arithmetic circuits.

Roughly speaking, a syntactic multilinear circuit is one which does not multiply two sub-circuits containing a common input variable. In the syntactic multilinear world, these classes behave quite differently. We first prove that syntactic multilinear arithmetic circuits of constant width can be efficiently simulated by arithmetic formula, hence showing that syntactic multilinear version of VsSC^0 is contained in that of VNC^1 (See theorem 5.2). Further, by giving a width and size efficient simulation of syntactic multilinear constant width circuits by branching programs, we pull down the syntactic multilinear variant of VsSC^0 down to syntactic multilinear constant width branching programs (VBWBP) hence making them equivalent in the syntactic multilinear world (Theorem 5.12). However, it is not clear if a syntactic multilinear arithmetic formula can be efficiently simulated by constant width syntactic multilinear branching programs (or even circuits). So in the syntactic multilinear world poses a contrasting picture: In the general world we have $\mathsf{VNC}^1 = \mathsf{VBWBP} \subseteq \mathsf{VsSC}^0$ but in the syntactic multilinear world we have $\mathsf{sm\text{-}VBWBP} = \mathsf{sm\text{-}VsSC}^0 \subseteq \mathsf{sm\text{-}VNC}^1$. (The $\mathsf{sm\text{-}}$ prefix is used to denote the syntactic multilinear versions.)

The simulation of constant width syntactic multilinear circuit by constant width syntactic multilinear branching programs mentioned above also helps to pull out the sub-class of multiplicatively disjoint constant width arithmetic circuits (where inputs of a multiplication gate are disjoint as sub-circuits) from the class VsSC^0

and show it to be equivalent to VNC^1.

In an attempt to break the scenario in the syntactic multilinear world, we look into the limitations of the only known technique to transform a formula into a constant width branching program (Ben-Or and Cleve.). It is easy to see that the simulation of arithmetic formula by 3-register linear straight line programs does not preserve syntactic multilinearity. (A counter example is given in Chapter 6.) In order to exhibit the inherent limitation of the simulation technique of Ben-Or and Cleve in the syntactic multilinear world, we consider a generalised simulation where there are more than 3-registers available and there is a freedom of choosing the registers. We show that there exist syntactic multilinear formulas that would require at least linear number of registers, if the resulting branching program is to be syntactic multilinear. (Theorem 6.5 and corollary 6.6).

In a slightly different direction, we explore the expressive power of skew arithmetic formula (arithmetic formula where the multiplication gates are skew). We ask the question: are the class of polynomials computed by algebraic branching programs expressible as a sum of exponential many instances of a polynomial computed by a skew-arithmetic formula? We prove that this is not the case by developing an exact characterisation of polynomials computed by skew formula in terms the number of monomials in them. (See Theorem 6.12.)

Next, we make an attempt to cast the width of an arithmetic circuit as a possible measure of space for arithmetic computations. In particular our purpose is to define an arithmetic analog of the deterministic log-space, L. To begin with, we observe that all three known measures for algebraic computations ([64, 32, 52]) are either too weak or too strong to give a meaningful definition of log-space for arithmetic computations. We propose "width" of an arithmetic circuit as a possible measure of space for arithmetic computation as it does not fall into the too weak or too strong categories when restricted to log-space. We show that our definition of space coincides with the definition of $\mathsf{VPSPACE}$ introduced by Koiran and Perifel [52]. (Theorem 7.5.)

To get an arithmetic analog of NL in terms of read-once exponential sum, we introduce the notion of "read-once" certificates for width bounded arithmetic circuits. We show that in the case of polynomial width circuits the restriction of read-once does not have any impact. In the case of polynomial size algebraic branching programs, which is a natural arithmetic version of NL, we are able to

show that exponential sums over read-once certificates do not increase their power (Theorem 7.16). We also show that the polynomials in the class of polynomial size branching programs can be written as read-once exponential sums of some polynomial computed by log-width polynomial size arithmetic circuits (Theorem 7.12). We give a size upper bound of quasi-polynomial for read-once exponential sums of poly-logarithmic width circuits which are either multiplicatively disjoint or of polynomial syntactic degree. (Theorem 7.19).

1.2 Matroid Isomorphism

A matroid is a combinatorial object that generalises the notion of linear independence in vector spaces to arbitrary structures. A matroid M on a given finite set S (called ground set) is a family $\mathcal{I}$ of subsets of S which is closed under subsets and all maximal cardinality sets in $\mathcal{I}$ are of equal size. $\mathcal{I}$ is called the collection of independent sets of M. Matroid theory developed into a vast research discipline of discrete mathematics ever since it was introduced by Whitney in 1935 ([102]).

Though matroids are objects with many similarities to graphs, the progress on computational questions on matroids is not as well developed as that of graphs. Probably, one of the major difficulties is the input representation of matroids, as the number of independent sets of a matroid could be exponential in the number of elements in the ground set. However, there had been some work on the problem of testing whether a matroid is representable ([41, 53]), testing if a binary matroid is graphic ([78]) etc, when the input matroid is represented as an oracle to the independent set.

In this thesis we look into the problem of isomorphism testing of two matroids from the complexity theoretic perspective.

1.2.1 Motivation

The problem of testing if the given two input graphs are isomorphic (the graph isomorphism problem or GI for short) is an important problem for theoreticians as well as practitioners. In complexity theory, GI stands as one of those problems that is not known to be polynomial time computable and unlikely to be NP-complete ([76]) unless the polynomial hierarchy collapses. There has been a lot of research

on the graph isomorphism problem in the literature focussed on two themes: 1) Study its structural complexity ([50, 76, 9, 48, 87, 10]) 2) Come up with restrictions of GI that are polynomial time computable such as bounded degree graphs, planar graphs etc.([58, 66, 15]).

The complexity of testing isomorphism of other structures such as groups ([65, 11]), boolean formula ([2]) and rings ([75]) have also been studied in the literature. Most of these problems either are equivalent to GI or have properties almost similar to that of GI.

In [62], Mayhew studied the matroid isomorphism problem when the input matroids are given as lists of their respective independent sets. In this setting Mayhew proved that testing isomorphism of matroids is polynomial time many-one equivalent to the graph isomorphism problem (GI).

However, there are more implicit representations. The ones we consider are: 1)Matroids representable as a matrix over some field $\mathbb{F}$, called linear matroids. 2)Matroids representable as graphs, called graphic matroids. In both the above cases a complete listing of all independent sets can be exponential in the size of the representation. Hence it is worthwhile to look into the isomorphism testing problem under these input representations.

Another reason for the study of isomorphism problem on graphic matroids is its striking similarity to the graph isomorphism problem. An old result of Whitney ([100]) shows that isomorphism question for matroids and their underlying graphs are equivalent if the underlying graphs are 3-connected. (i.e, there are three edge disjoint paths between any pair of vertices.) See Theorem 8.6.

1.2.2 Contribution of the thesis

In this thesis we study the complexity of testing isomorphism between two matroids under various input representations. The three representations we consider are:

1. The input matroids are given as oracles to their respective independent sets, we denote this case by MI.

2. Matroids that are represented by matrices over a finite field $\mathbb{F}$, this instance is denoted by LMI as a short hand for linear matroid isomorphism.

3. Matroids that are represented by undirected graphs. This case is denoted by GMI.

As a starting point, we observe that it is unlikely that LMI and GMI are NP complete; if they are, the polynomial hierarchy will collapse to the third and second level respectively. (Theorem 8.10.)

Focussing further on LMI, we observe that it is co-NP-hard. As we are unable to resolve the complexity of LMI in general, we study its bounded rank restriction denoted by LMI_b. We show that if rank of the input matroids is bounded by a constant, then the isomorphism testing of such matroids is polynomial time many-one equivalent to the graph isomorphism problem, *i.e.* LMI_b is polynomial time many-one equivalent to GI. (See theorem 8.14 and corollary 8.17.)

We then consider the problem of testing isomorphism of graphic matroids, GMI. In theorem 8.19 we show that GMI is polynomial time Turing reducible to the GI. The main idea in the proof of this theorem is to decompose the given graphs into a tree of 3-connected components using the algorithm of [45] and then use queries to GMI restricted to 3-connected graphs. To implement this idea we develop a new edge-colouring scheme which preserves the isomorphism between the original matroids represented by the given graphs. By giving a many-one reduction from LMI_b to GMI (theorem 8.28), it is shown that the problems GMI, GI and LMI_b are polynomial time equivalent.

In the final chapter we study certain structural properties of GMI and LMI. We develop colouring techniques to handle colouring instances of MI, LMI and GMI (See Lemmas 9.1,9.2 and 9.4). Using these colouring techniques we show that the problem of computing automorphism groups and isomorphism testing are polynomial time equivalent. (Theorem 9.5.) Continuing with the complexity of computing automorphism groups, we propose a membership testing algorithm for the automorphism groups of graphic and binary matroids.

Finally, following [50] we define the and-functions and or-functions for GMI. We show that GMI has polynomial time computable and-functions and or-functions. (Theorem 9.13)

1.3 Organisation of the thesis

Rest of this thesis is organised as follows:

Part - I

- In chapter 2 we introduce the notions of boolean circuits and branching programs and the complexity classes that are relevant to this thesis.
- Chapter 3 is dedicated to the introduction of the boolean complexity class sSC^0 and the resulting hierarchy sSC.
- In chapter 4 we consider the arithmetisation of sSC^0 and other classes such as logarithmic width formula and a class of restricted branching programs that are equivalent to NC^1. We then translate these defintions onto the more general Valiant's arithmetic circuit model.
- In chapter 5 we consider the syntactic multilinear restrictions of the classes introduced in chapter 4.
- In chapter 6 we present limitations of the Ben-Or and Cleve's simulation technique in the syntactic multilinear world. In the same chapter, we prove that an exponential sum over skew formula cannot express polynomials computed by algebraic branching programs.
- Chapter 7 considers the small space versions in the Valiant's model. In this chapter we propose circuit width as a possible measure of space for arithmetic computations. We also introduce and discuss "Read-once" certificates and exponential sums over them

Part - II

- In chapter 8 we introduce the isomorphism problems on matroids given by various input representations. This chapter contains discussions and new results on the isomorphism testing of linear matroids and graphic matroids.
- Chapter 9 contains our results on some of the structural properties of matroid isomorphism problems viz, coloured versions, computing the automorphism group and complexity of and/or functions for GMI.

Part I

Arithmetic circuits around NC^1

Chapter 2

Preliminaries

In this chapter we briefly revise various models of computation used in this thesis. We omit the definitions of standard complexity classes such as NP, P, NL and L etc,. These can be found in any of the standard complexity theory textbooks. (see *e.g.* [8, 37, 33])

2.1 Boolean Circuits

We start with the definition of the boolean circuit computation model. Most of the definitions here are taken from [97]. Let $X = \{x_1, \ldots, x_n\}$ be a set of input variables that take values from $\{0, 1\}$. A boolean circuit C with X as its input is a directed acyclic graph with nodes labelled from $X \cup \{\wedge, \vee, \neg\} \cup \{0, 1\}$. The nodes of out-degree 1 are called *output* gates of C. Nodes labelled with variables from $X \cup \{0, 1\}$ are called *input* gates of C. The remaining nodes (those labelled from $\{\wedge, \vee, \neg\}$) are called the *internal* nodes of C. Naturally, for every gate g of C, we can associate a boolean function $f_g : \{0, 1\}^n \to \{0, 1\}$ defined inductively as follows:

Let $a \in \{0, 1\}^n$.

case 1: $\mathsf{label}(g) \in X \cup \{0, 1\}$, then $f_g(a) = 1$ if and only if $\mathsf{label}(g) = 1$ or $\mathsf{label}(g) = x_i$ and $a_i = 1$.

case 2: $g = g_1 \vee g_2$, then $f_g(a) = 1$ if and only if $f_{g_1}(a) = 1$ or $f_{g_2}(a) = 1$.

case 3 $g = g_1 \wedge g_2$, then $f_g(a) = 1$ if and only if $f_{g_1}(a) = 1$ and $f_{g_2}(a) = 1$.

The functions computed by the circuit C are the functions associated with the

output nodes of C.

Note that the circuit model is non-uniform, *i.e.* there is a different circuit for each input length.

The *fan-in* of a gate is its in-degree. Similarly, out-degree of a node is called its *fan-out*. The *Size* of a circuit is the total number of internal nodes and edges in it. However, when fan-in is bounded by a constant, the total number of nodes and nodes plus edges will differ only by a constant factor (*i.e.* the fan-in). The *Depth* of the circuit is defined as the longest path from an input gate to an output gate in C. Intuitively, if we assume that a computation at each gate takes unit time, then the depth of C is the maximum time-delay required so that the output is ready at the output gates. A less obvious measure is the *width* of a circuit. To define width of a circuit, we assume that the circuit is layered and the edges are between two consecutive layers. The *Width* of a layered circuit is defined to be the maximum number of gates in any layer.

A *formula* is a circuit where every gate has a fan-out bounded by 1, *i.e.* , if we exclude the input gates, the underlying undirected graph is a tree. Note that every circuit can be converted into a formula by duplicating the gates that have fan-out larger than 1. If the circuit has depth d and maximum fanout c then the size of the resulting formula will grow by a factor of c^d. We term this process as *unwinding* a circuit into a formula.

Let C be a circuit on input variables $X = \{x_1, \ldots, x_n\}$. Assume that C contains only one output gate. Also, without loss of generality assume that $\neg$ gates appear only at the input level. Let $a = a_1, \ldots, a_n$ be an assignment of boolean values to variables from X. Let F be the formula obtained by unwinding C into a formula. A *proving sub-tree* (or *accepting sub-tree*) T of C on input $a = a_1, \ldots, a_n$ is a sub-formula of F defined as follows:

- T contains the output gate of F.
- For every $\vee$ gate g in T, exactly one child (*i.e.* predecessor) of g is in T.
- For every $\wedge$ gate g in T, all the children of g are in T.
- For every negation gate $g = \neg x_i$ in T, where $i \in \{1, \ldots, n\}$, $a_i = 0$.
- All gates in T evaluate to 1 under the assignment a.

Syntactic degree: Let C be any boolean circuit. We define syntactic degree of a gate g in C inductively as follows:

- If $g \in X \cup \{0,1\}$, then its syntactic degree is 1.
- If $g = g_1 \vee g_2$ then syntactic degree of g is the maximum of that of g_1 and g_2.
- If $g = g_1 \wedge g_2$ then syntactic degree of g is the sum of syntactic degrees of g_1 and g_2.

It is not hard to see that syntactic degree of a circuit C of depth d and fan-in of $\wedge$ gates bounded by c is at most c^d.

Skew circuits: Let C be a boolean circuit where the $\wedge$ gates have fan-in 2. The circuit C is said to be *skew*, if every $\wedge$ gate of C has at most one internal gate as its input and all other inputs to this gate are circuit input gates. In other words, if $g = g_1 \wedge g_2$ then either $g_1 \in X \cup \{0,1\}$ or $g_2 \in X \cup \{0,1\}$. As syntactic degree of a multiplication gate in a skew circuit can be at most one plus that of one of its children, the sytactic degree of a skew circuit is bounded by its size.

2.2 Branching Programs

In this section we define the model of branching programs. A branching program (BP for short) B is a layered directed acyclic graph with the following properties:

- There are two designated nodes s and t. s has zero in-degree and t has zero out-degree.
- Edges of B are labelled from the set $\{x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n\} \cup \{0,1\}$, where $X = \{x_1, \ldots, x_n\}$ are the input variables representing the input to P.

Let $P = \langle s, v_1, \ldots, v_k, t \rangle$ be an s-t path in B. The weight of P is defined as

$$\mathsf{weight}(P) = \left(\bigwedge_{i=1}^{k-1} \mathsf{label}(v_i, v_{i+1}) \right) \wedge \mathsf{label}(s, v_1) \wedge \mathsf{label}(v_k, t)$$

Clearly, B represents a function $f_B : \{0,1\}^n \to \{0,1\}$ defined by: $f(a) = 1$ if and only if there exists and s-t path of weight 1, under the assignment $a = a_1, \ldots, a_n$ for variables $X = x_1, \ldots, x_n$ in that order.

Size of a BP is the number of vertices and edges in it. *Length* is the maximum number of layers in it. *Width* is the maximum number of nodes at any layer.

2.2.1 BPs and skew circuits

The models of branching programs and skew circuits coincide. There is an efficient and straightforward way to transform a BP to a skew circuit and vice-versa.

From BPs to skew circuits: Let B be a layered BP of width w and size s. We can convert B into a skew circuit C of width $O(w^2)$ and size $O(w^2 s)$ as follows:

- Label the nodes as $\vee$ gates except the node s which is labeled as the constant 1.
- Consider an edge (u, v) in B with label a. Replace (u, v) in C by a new gate $g = u \wedge a$ and the edge (g, v).
- The node in C that corresponds to t in B is the output node.

It is easy to see that C obtained as above is a skew circuit and also $\mathsf{width}(C) = O(w^2)$, $\mathsf{size}(C) = O(w^2 s)$.

From skew circuits to branching programs: Let C be a layered skew circuit of size s, width w and depth d. Assume without loss of generality that fan-in of every gate in C is bounded by 2. Also, assume that no $\wedge$-gate in C is a child of another $\wedge$-gate (this can be achieved by introducing dummy $\vee$ gates). We construct a BP B that computes the same function as that of C.

- Relabel all the $\vee$ gates of C as the nodes of B with their connections unchanged. For an $\vee$ gate g in C let g' denote its counterpart in B.
- Let S be a new start node and relabel the output node of C as t in B.
- Introduce a new node v_i at level i, and edges $(S, v_1), (v_1, v_2) \ldots, (v_{\ell-1}, v_\ell)$ labelled with the constant 1, where ℓ is the number of layers in C.

- Let $g = h \wedge b$ be an $\wedge$-gate, where $b \in \{x_1, \ldots, x_n\} \cup \{\neg x_1, \ldots, \neg x_n\} \cup \{0, 1\}$. Let $g_1, \ldots, g_k$ be the nodes in C, that have g as one of inputs. Now, in B we add edges $(h', g'_1), \ldots, (h', g'_k)$ with label b. Do this for all $\wedge$ gates in a bottom up fashion.

- If an $\vee$ gate g at some level i has an input $b \in X \cup \{0, 1\}$, then add an edge (v_i, g') with label b.

Note that, width of the BP B resulting from the above costruction is one more than that of the skew circuit C and size is at most double that of C.

The following proposition summarises the above two transformations:

Proposition 2.1 *A layered skew circuit of width w and size s can be transformed into an equivalent branching program of width $w + 1$ and size $O(s)$.*

Conversly a branching program of width w and size s has an equivalent skew circuit of width w and size $O(s)$.

2.2.2 Series Parallel Construction

Let C be a boolean circuit of size s and depth d and fan-in 2. Let c be the bound on the fan-out of C. We can construct a BP equivalent to C as follows:

1. Base case is when C consists of a single input gate. This case is easy.

2. $g = g_1 \vee g_2$. Let B_1 a nd B_2 be the branching programs constructed for g_1 and g_2 respectively. Let s_1, t_1 (resp. s_2, t_2) be the special nodes of B_1 (resp. B_2). $B = B_1 \cup B_2$ plus two new nodes s and t along with the edges $(s, s_1), (s, s_2), (t_1, t)$ and (t_2, t). s and t are the special nodes of B. This is called the OR-gadget. (See Figure 2.1.)

3. $g = g_1 \wedge g_2$. Let B_1 and B_2 be the BPs computing g_1 and g_2 respectively. Then B is obtained by adding an edge from t_1 to s_2, where s_1, t_1 (resp. s_2, t_2) are the special nodes of B_1 (resp. B_2). Set s_1, t_2 as the special nodes of B thus obtained. This is called the and gadget. (See Figure 2.1.)

Let g be any gate of fan-out c. Then above construction, needs c copies of the BP for g which is available via induction. Thus we can conclude that size of the

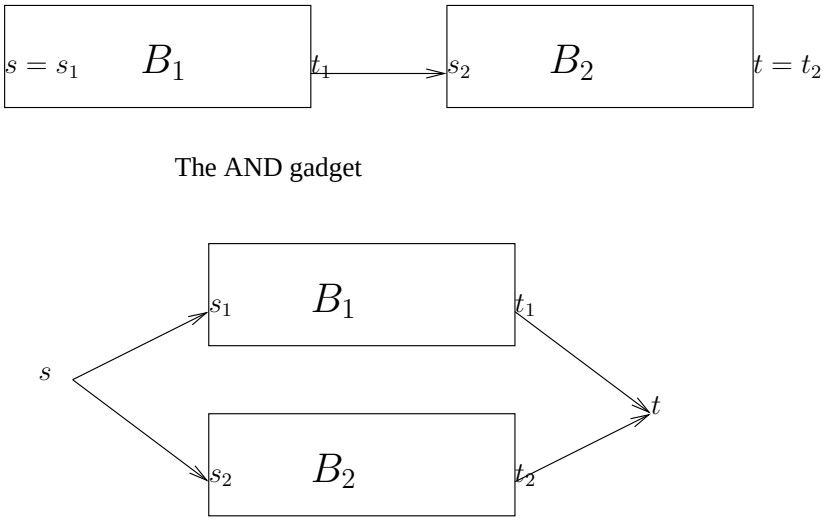


Figure 2.1: The series-parallel construction of BPs from circuits

resulting BP B is $O(c^d.s)$. If the given circuit C is a formula (i.e. $c = 1$), this value is $O(s)$. Summarising the above,

Proposition 2.2 *Every circuit of size s, depth d and fan-out c has an equivalent BP of size $O(c^d \cdot s)$. In particular, polynomial size formula have equivalent BPs of polynomial size.*

Staggering: However, the simulation of circuits/formula by BPs above need not be width efficient. To make it width efficient we need to use the idea of *staggering*, which we describe below.

In step 2 of the above simulation, we can delay the program B_2 by creating a path of no-operation nodes till the end of B_1 and then starting B_2. *i.e.* , instead of directly connecting s to s_2, we create a path $s, v_1, \ldots, v_r$ of weight 1, where r is the size of B_1. Then connect v_r to s_2. Now insert another path $t_1, u_1, \ldots, u_{r'}$ of weight 1, where r' is the size of B_2. Now connect $u_{r'}$ to t. Rest of the edges remain unchanged. In this process we incur a cost of one extra width. Notice that step 3 of the above procedure does not increase the width. Thus we can get away with 1 additional width per depth of the formula. (See Figure 2.2).

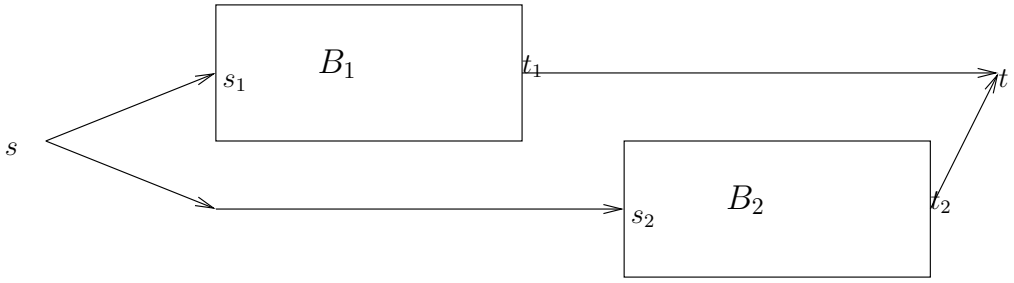


Figure 2.2: Staggering of BPs

Proposition 2.3 *A depth d formula of size s can be simulated by a width d BP of size $O(d \cdot s)$.*

2.3 Uniformity of circuits

The circuit model introduced above is non-uniform in the sense that circuit description depends on the input length. First we define direct connection language of a circuit family $\mathcal{C} = (C_n)_{n \geq 0}$ (the definition here follows the one in [97]).

Definition 2.4 *Let $\mathcal{C} = (C_n)_{n \geq 0}$ be a boolean circuit family. The direct connection language $L_{DC}(\mathcal{C})$ of $\mathcal{C}$ is defined as the set of tuples $\langle 1^n, g, p, b \rangle_{n \geq 0}$ such that,*

- *g is the index of a gate in C_n;*
- *$p \in \{0,1\}^*$;*
- *if p is the empty string, then b is the label of the gate in C_n represented by g;*
- *else, b represents the k th predecessor of gate g, where k is the binary value represented by p.*

Let $\mathcal{A}$ be any complexity class. A circuit family $\mathcal{C} = (C_n)_{n \geq 0}$ is said to be $\mathcal{A}$-uniform if $L_{DC}(\mathcal{C}) \in \mathcal{A}$. It is known that $\mathsf{P} = \mathsf{L}$-uniform circuits of polynomial size.

2.4 Circuit based complexity classes

In this section we define the complexity classes based on the circuit and branching program models that are relevant to the thesis.

We consider circuits without $\neg$ gates, where leaves take labels from $\{x_1, \ldots, x_n\} \cup \{\neg x_1, \ldots, \neg x_n\} \cup \{0, 1\}$.

We start with the definition of the NC hierarchy. First we define the class NC^i for $i \geq 0$ as follows:

$$\mathsf{NC}^i = \left\{ f : \{0,1\}^* \to \{0,1\} \;\middle|\; \begin{array}{l} \text{There exists a family } (C_n)_{n\geq 0} \text{ of boolean circuits,} \\ \text{where } C_n \text{ is of constant fan-in, polynomial size} \\ \text{and } O(\log^i n) \text{ depth such that } f \text{ restricted to in-} \\ \text{puts of length } n \text{ is } f_{C_n}. \end{array} \right\}$$

The NC hierarchy is defined as the limting point of NC^is:

$$\mathsf{NC} = \bigcup_{i \geq 0} \mathsf{NC}^i$$

The NC hierarchy represents the class of boolean functions that have efficient parallel algorithms. Some example problems inside these hierarchy are: sorting n integers, reachability in directed graphs, list ranking etc. We define FNC^i as the functional version of NC^i where the circuit contains more than one output gates.

By allowing the fan-in to be unbounded, we get a different definition of the NC hierarchy. However the individual levels of the hierarchy need not be the same.

$$\mathsf{SAC}^i = \left\{ f : \{0,1\}^* \to \{0,1\} \;\middle|\; \begin{array}{l} \text{There exists a family } (C_n)_{n\geq 0} \text{ of boolean circuits,} \\ \text{where } C_n \text{ is of unbounded } \vee \text{ fan-in and constant } \wedge \\ \text{fan-in, polynomial size and } O(\log^i n) \text{ depth such} \\ \text{that } f \text{ restricted to inputs of length } n \text{ is } f_{C_n}. \end{array} \right\}$$

For a function $\lambda = \lambda(n)$, $\mathsf{SAC}^i(\lambda(n))$ denotes the class of boolean functions computed by SAC^i type circuits, where the $\vee$ fan-in is bounded by λ.

$$\mathsf{AC}^i = \left\{ f : \{0,1\}^* \to \{0,1\} \;\; | \;\; \begin{array}{l} \text{There exists a family } (C_n)_{n\geq 0} \text{ of boolean circuits,} \\ \text{where } C_n \text{ is of unbounded fan-in, polynomial size} \\ \text{and } O(\log^i n) \text{ depth such that } f \text{ restricted to in-} \\ \text{puts of length } n \text{ is } f_{C_n}. \end{array} \right\}$$

The corresponding hierarchies are:

$$\mathsf{SAC} = \bigcup_{i\geq 0} \mathsf{SAC}^i \quad ; \quad \mathsf{AC} = \bigcup_{i\geq 0} \mathsf{AC}^i$$

These hierarchies are related as follows,

Proposition 2.5

$$\forall i \geq 0,\ \mathsf{NC}^i \subseteq \mathsf{SAC}^i \subseteq \mathsf{AC}^i \subseteq \mathsf{NC}^{i+1} \quad \textit{and hence} \quad \mathsf{NC} = \mathsf{SAC} = \mathsf{AC}$$

We define the boolean functions computable by boolean formula as follows,

$$\mathsf{F} = \left\{ f : \{0,1\}^* \to \{0,1\} \;\; | \;\; \begin{array}{l} \text{There exists a family } (C_n)_{n\geq 0} \text{ of boolean formula,} \\ \text{where } C_n \text{ is of polynomial size such that } f \text{ re-} \\ \text{stricted to inputs of length } n \text{ is } f_{C_n} \end{array} \right\}$$

$$\mathsf{LWF} = \left\{ f : \{0,1\}^* \to \{0,1\} \;\; | \;\; \begin{array}{l} \text{There exists a family } (C_n)_{n\geq 0} \text{ of boolean formula,} \\ \text{where } C_n \text{ is of polynomial size and } O(\log n) \text{ width} \\ \text{such that } f \text{ restricted to inputs of length } n \text{ is } f_{C_n} \end{array} \right\}$$

It is easy to see that constant fan-in circuits of logarithmic depth and polynomial size can be unwound into formula of polynomial size and logarithmic depth. Conversely, using the depth reduction of Spira ([80, 26]) we can convert a polynomial size formula into an equivalent formula of logarithmic depth and polynomial size. Also, it is shown in [46] that polynomial size and log-width formula are equivalent to formula of polynomial size. Thus,

Proposition 2.6

$$\mathsf{NC}^1 = \mathsf{F} = \mathsf{LWF}.$$

Another important hierarchy that stands almost parallel to the NC hierarchy is the SC hierarchy or Steve's class. The SC hierarchy characterises the class of languages that are computable by polynomial size circuits of poly-logartihmic width. Formally,

$$\mathsf{SC}^i = \left\{ f : \{0,1\}^* \to \{0,1\} \;\middle|\; \begin{array}{l} \text{There exists a family } (C_n)_{n\geq 0} \text{ of boolean circuits,} \\ \text{where } C_n \text{ is of polynomial size and } O(\log^i n) \text{ width} \\ \text{such that } f \text{ restricted to inputs of length } n \text{ is } f_{C_n}. \end{array} \right\}$$

and

$$\mathsf{SC} = \bigcup_{i\geq 0} \mathsf{SC}^i$$

It is well known that if the circuits are restricted to be uniform, then the SC hierarchy corresponds to the hierarchy of languages computed by Turing machines that run in polynomial time and poly-logarithmic space. Let L denote the class of languages (or boolean functions) decidable by logarithmic space bounded deterministic Turing machines. Let $\mathsf{DSPACE\text{-}TIME}(s,t)$ denote the languages decided by deterministic Turing machines in time $t = t(n)$ and space $s = s(n)$. We have,

Proposition 2.7 (Folklore) *For $i > 0$,*

$$\mathsf{L\text{-}Uniform\text{-}SC}^i = \mathsf{DSPACE\text{-}TIME}(\log^i n, \mathsf{poly}(n))$$

In particular, $\mathsf{L} = \mathsf{SC}^1$

Complementation and double rail construction

Let C be a boolean circuit computing a function $f : \{0,1\}^n \to \{0,1\}$. Let $\overline{f}$ denote the complement function of f (*i.e.* , $\overline{f}(x) = \neg f(x)$). We give a construction known as "double-rail construction" which constructs a circuit C' to compute $\overline{f}$ from a circuit C computing f. For every gate g of C, C' contains two gates g' and $\overline{g}$. The connections are defined inductively as follows:

- If $g = g_1 \vee g_2$ then $g' = g'_1 \vee g'_2$ and $\overline{g} = \overline{g_1} \wedge \overline{g_2}$.
- if $g = g_1 \wedge g_2$ then $g' = g'_1 \wedge g'_2$ and $\overline{g} = \overline{g_1} \vee \overline{g_2}$.

- If $g = \neg g_1$ then $g' = \overline{g_1}$ and $\bar{g} = g_1'$.

Clearly, by De Morgan's laws, it follows that g' computes the same function as g and $\overline{g}$ computes the complement of the function computed at the gate g. If f is the output gate of C then we set $\overline{f}$ as the output gate of C'. Additionally, we have:

- $\mathsf{size}(C') = 2\mathsf{size}(C)$; $\mathsf{depth}(C') = \mathsf{depth}(C)$; $\mathsf{width}(C') = 2\mathsf{width}(C)$
- $\wedge-$fan-in of C' = $\vee$-fan-in of C' = $\max\{\lambda_1, \lambda_2\}$, where λ_1 = $\vee$-fan-in of C and λ_2 = $\wedge$-fan-in of C.
- If C is a formula, then so is C'.

For a circuit complexity class $\mathcal{C}$, let co-$\mathcal{C}$ denote the set $\{\overline{f} \mid f \in \mathcal{C}\}$. The construction above shows that the classes NC^i, AC^i and SC^i are closed under complementation, *i.e.*

Proposition 2.8 (folklore)

$$\forall i \geq 0 \;\; \mathsf{NC}^1 = \; co\text{-}\mathsf{NC}^1 \;\; ; \;\; \mathsf{AC}^i = \; co\text{-}\mathsf{AC}^i \;\; ; \;\; \mathsf{SC}^i = \; co\text{-}\mathsf{SC}^i$$

However for the complementation of the classes SAC^i as the double-rail construction increases the $\wedge$ fan-in to that of $\vee$, a more sophisticated method called inductive counting technique (See [20]) is needed. We give a more detailed account of this construction in chapter 3.

Proposition 2.9 ([20])

$$\mathsf{SAC}^1 = \; co\text{-}\mathsf{SAC}^1$$

We now define complexity classes defined on the branching program models.

$$\mathsf{BP} = \left\{ f : \{0,1\}^* \to \{0,1\} \;\; | \;\; \begin{array}{l} \text{There exists a BP family } (B_n)_{n\geq 0}, \text{ where } B_n \text{ is} \\ \text{of size polynomial in n, such that for all } x \in \\ \{0,1\}^n, n \geq 0,\ f(x) = f_{B_n}(x) \end{array} \right\}$$

$$\mathsf{BPW}(w) = \left\{ f : \{0,1\}^* \to \{0,1\} \;\; | \;\; \begin{array}{l} \text{There exists a polynomial size BP family } (B_n)_{n\geq 0}, \\ \text{where } B_n \text{ has width } w = w(n) \text{ and size } \mathsf{poly}(n) \\ \text{such that for all } x \in \{0,1\}^n, n \geq 0,\ f(x) = f_{B_n}(x) \end{array} \right\}$$

$$\mathsf{BWBP} = \mathsf{BPW}(O(1)) \quad ; \quad \mathsf{LWBP} = \mathsf{BPW}(\log n)$$

We have seen that BPs can be viewed as skew-circuits hence we can define uniform BPs as in the case of boolean circuits. Let NL denote the class of boolean functions computable by non-deterministic logarithmic space bounded Turing machines. As testing reachability in a directed acyclic graph is complete for NL under logs-space many-one reductions, it is easy to see that $\mathsf{NL} = \mathsf{L\text{-}Uniform\text{-}BP}$. In his seminal work, Barrington ([17]) showed that boolean functions constant width BPs (*i.e.* BWBP) exactly characterise the functions in NC^1 (*i.e.* $\mathsf{NC}^1 = \mathsf{BWBP}$). In the same paper he observed that constant width circuits are not more powerful than constant width branching programs (*i.e.* $\mathsf{BWBP} = \mathsf{SC}^0$). Let LogCFL denote the class of languages that are log-space many-one reducible to *context free languages* (CFLs). $\mathsf{LogDCFL}$ denotes the class of languages that are log-space many-one reducible to deterministic context free languages (DCFLs).

The following proposition summarises the relationships among the classes defined so far (under appropriate uniformity conditions).

Proposition 2.10

$$\mathsf{BWBP} = \mathsf{SC}^0 = \mathsf{NC}^1 = \mathsf{F} = \mathsf{LWF} \subseteq \mathsf{SC}^1 = \mathsf{L} \subseteq \mathsf{NL} = \mathsf{BP} \subseteq \mathsf{SAC}^1 = \mathsf{LogCFL}$$

2.5 The Chinese Remaindering Technique

The folklore Chinese Remainder theorem (CRT) allows us to efficiently restore a large number from its residues modulo small primes. ([29], see also [3]).

Lemma 2.11 *1. The jth smallest prime p_j is bounded by $O(j \log j)$; hence p_j can be computed and represented in space $O(\log j)$. Let $P_k = \prod_{i=1}^{k} p_k$; then P_k is at least 2^k.*

2. Every integer $a \in [0, P_k)$ is uniquely defined by the residues $a_i = a \bmod p_i$ where $i \in [k]$.

3. Given $a_1, \ldots, a_k$, where $a_i \in [0, p_i)$, the unique a such that $a \equiv a_i \bmod p_i$ for each $i \in [k]$ can be found in space $O(\log k)$.

4. *Let* $f : \{0,1\}^* \to \mathbb{N}$, *with* $f(x) \in [0, 2^{q(|x|)})$ *where* q *is a polynomial. If the bits of* $f(x) \bmod p_i$ *can be computed in deterministic space* s, *for* $i = 1, \ldots, q(|x|)$, *then* f *can be computed in deterministic space* $O(\log q(|x|) + s)$.

Chapter 3

Width and degree bounded circuits

3.1 Introduction

Simultaneous width and size

The Uniform-SC hierarchy obtained by uniform polynomial size circuits of poly-logarithmic width is equivalent to the hierarchy of languages accepted by Turing machines that achieve a simultaneous space and time bound of poly-logarithm and polynomial respectively. (*i.e.* DSPACE-TIME(poly-log, poly) = Uniform-SC.) Even in the non-uniform world, the SC hierarchy is not known to be comparable with the NC hierarchy except at the first two levels. These hierarchies coincide at the lowest level (*i.e.* $\mathsf{SC}^0 = \mathsf{NC}^1$). At the higher level, SC^1 is contained in NC^2 and the class of languages that are log-space reducible to deterministic context free languages LogDCFL is the largest chunk from the NC hierarchy that is known to be contained in SC ([30]). One of the major open problems of circuit complexity is to compare these two hierarchy at higher levels.

Simultaneous degree and size

When the syntactic degree of a circuit is restricted to be a polynomial in n along with its size, the complexity of the function computed drastically comes down. As it was shown by Ruzzo [74] (also [94, 95]), polynomial size circuits of polynomial syntactic degree exactly characterize the class $\mathsf{SAC}^1 = \mathsf{LogCFL}$. *i.e.* polynomial degree restriction along with size does not take us beyond NC^2. However, is not

known if this class can be efficiently simulated by circuits in the class SC.

Putting all three together

Out of curiosity, one could ask what would happen if the width and syntactic degree are restricted simultaneously? Certainly, this restriction does not take us beyond SAC^1. In fact, such a restriction leads to a hierarchy of circuit classes sandwiched between NC^1 and SAC^1, and is by definition contained inside SC. We call this the small SC hierarchy denoted by sSC. The sSC hierarchy stands similar to the two hierarchies defined by Vinay[96]: branching programs of poly-logarithmic width and log-depth circuits with poly-logarithmic OR-fan-in.

Rest of the chapter is organized as follows. Section 3.2 contains the formal definition of sSC hierarchy. Section 3.3 contains discussion on closure properties of sSC and the chapter ends with brief discussion of other hierarchies similar to sSC and concluding remarks respectively in sections 3.4 and 3.5.

3.2 The sSC hierarchy

We start with a formal definition of the sSC hierarchy.

Definition 3.1 *For $i \geq 0$, sSC^i is the class of boolean function families which can be computed by polynomial sized boolean circuits of polynomial syntactic degree and width $O(\log^i n)$. sSC is the limiting point of all sSC^is, i.e.*

$$\mathsf{sSC} = \bigcup_{i \geq 0} \mathsf{sSC}^i$$

Let us have a look at the individual classes in the sSC hierarchy. At the base level ($i = 0$), we have the class sSC^0. Recall that branching programs are nothing but skew-circuits (see Section 2.2.1). Hence the syntactic degree of skew circuits is upper bounded by their depth (or number of layers), which in turn is bounded by its size. Thus $\mathsf{BWBP} \subseteq \mathsf{sSC}^0 \subseteq \mathsf{SC}$ and hence from proposition 2.10

Proposition 3.2 $\mathsf{sSC}^0 = \mathsf{SC}^0 = \mathsf{NC}^1 = \mathsf{BWBP}$

The situation is somewhat different when we go to the higher levels of the sSC hierarchy. As an example, consider sSC^1, which is contained in SC^1, *i.e.* $\mathsf{sSC}^1 \subseteq$

$\mathsf{SC}^1 = \mathsf{L}$. But it is not clear if this relationship can also be turned other way around, *i.e.* can we simulate a log space bounded machine with an sSC^1 circuit?.

As $\mathsf{sSC}^i \subseteq \mathsf{SC}^i$ by definition, the sSC hierarchy is another chunk of classes from the NC hierarchy that sit inside the SC hierarchy. *i.e.*

Proposition 3.3 $\mathsf{sSC} \subseteq (\mathsf{SAC}^1 \cap \mathsf{SC}) \subseteq \mathsf{NC}$

It is not known how to compare the higher members in the sSC hierarchy with other well known complexity classes between NC^1 and SAC^1. Of particular interest would be the class $\mathsf{LogDCFL}$ which is contained in SAC^1 and SC^2.

3.3 Closure properties

Once we define a boolean complexity class, the immediate question that arises is about the closure of the class under boolean operations such as complementation, AND and OR of several instances. In this section, we analyze these closure properties for the sSC hierarchy as a whole and its individual levels.

At the lowest level, as $\mathsf{sSC}^0 = \mathsf{NC}^1$ and since $\mathsf{NC}^1 =$ co-NC^1, we have co-$\mathsf{sSC}^0 = \mathsf{sSC}^0$. For $i \geq 1$, complementation seems to be highly non trivial. It is not hard to see that the standard double-rail construction (see section 2.4) does not work here as it would blow up the degree. It is not clear if sSC^i is closed under complementation for $i \geq 1$. However, in the following theorem we show that co-sSC^i is within the sSC hierarchy, hence the hierarchy as a whole is closed under complementation.

Theorem 3.4 *For each $i \geq 1$, co-sSC^i is contained in sSC^{2i}.*

Proof. Consider the proof of closure under complement for LogCFL, from [20]. This is shown by considering the characterization of LogCFL as semi-unbounded log depth circuits, and applying an inductive counting technique to such circuits. Our approach for complementing sSC^i is similar: use inductive counting as applied by [20]. However, one problem is that the construction of [20] uses monotone NC^1 circuits for threshold internally, and if we use these directly, the degree may blow up. So for the thresholds, we use the construction from [96]. A careful analysis of the parameters then yields the result.

Let C_n be a Boolean circuit of length l, width $w = O(\log^i n)$ and degree p. Without loss of generality, assume that C_n has only $\vee$ gates at odd levels and $\wedge$ gates at even levels. Also assume that all gates have fan in 2 or less. We construct a Boolean circuit C'_n, which computes $\bar{C}_n$. C'_n contains a copy of C_n. Besides, for each level k of C_n, C'_n contains the gates $cc(g|c)$ where g is a gate at level k of C_n and $0 \leq c \leq w$. These represent the conditional complement of g assuming the count at the previous level is c, and are defined as follows:

$$cc(g|c) = \begin{cases} cc(a_1|c) \vee cc(a_2|c), & \text{if } g = a_1 \wedge a_2 \\ Th^c(b_1, \cdots, b_j), & \text{if } g = a_1 \vee a_2 \end{cases}$$

where $b_1, \cdots, b_j$ range over all gates at the previous level except a_1 and a_2, and Th^c is the c-threshold value of its inputs.

C'_n also contains, for each level k of C_n and $0 \leq c \leq w$, the gates $count(c, k)$. These gates verify that the count at level k is c, and are defined as follows:

$$count(c,k) = \begin{cases} Th1(c,k) \wedge \bigvee_{d=0}^{w} [count(d, k-1) \wedge Th0(c,k,d)] & \text{if } k > 0 \\ 1 \quad \text{if } k = 0, c = \#\text{ of inputs with value 1 at level 0} \\ 0 \quad \text{otherwise} \end{cases}$$

Where $Th1(c,k) = Th^c$ of all original gates (gates from C_n) at level k, $Th0(c,k,d)$ is Th^{Z-c} of all $cc(g|d)$ at level k where Z is the number of gates in C_n at level k. Finally, the output gate of C'_n is $comp(g) = \bigvee_{c=0}^{w} count(c, l-1) \wedge cc(g|c)$, where g is the output gate of C_n, at level l. Correctness follows from the analysis in [20].

A crucial observation, used also in [20], is that any root-to-leaf path goes through at most two threshold blocks.

To achieve small width and small degree, we have to be careful about how we implement the thresholds. Since the inputs to the threshold blocks are computed in the circuit, we need monotone constructions. We do not know whether monotone NC^1 is in monotone sSC^0, for instance. But for our purpose, the following is sufficient: Lemma 4.3 of [96] says that any threshold on K bits can be computed by a monotone branching program (hence small degree) of width $O(K)$ and size $O(K^2)$. This branching program has degree $O(K)$. Note that the thresholds we use have $K = O(w)$. The threshold blocks can be staggered so that the $O(w)$ extra

width appears as an additive rather than multiplicative factor. Hence the width of C'_n is $O(w^2)$. (The conditional complement gates cause the increase in width; there are $O(w^2)$ of them at each level.)

Let q be the degree of a threshold block; $q = O(K) = O(w)$. If the inputs to a threshold block come from computations of degree p, then the overall degree is pq. Since a $cc(g|c)$ gate is a threshold block applied to gates of C_n at the previous level, and since these gates all have degree at most p, the $cc(g|c)$ gate has degree at most pq.

Also, the degree of a $count(c, k)$ gate is bounded by the sum of (1) the degree of a $count(d, k-1)$ gate, (2) the degree of a threshold block applied to gates of C_n, and (3) the degree of a threshold block applied to $cc(g|d)$ gates. Hence it is bounded by $p^{O(1)}w^{O(1)}l$, where l is the depth of C_n. Thus, the entire circuit has polynomial degree. ◇

A complexity class $\mathcal{C}$ is said to be closed under AND if for $L_1, \ldots, L_m \in \mathcal{C}$, the language $L = \{x \in \{0,1\}^* \mid x \in L_i, \forall i \geq 1\}$ is in $\mathcal{C}$, where $m = \mathsf{poly}(n)$. Closure under OR can be defined similarly. Using the standard staggering argument it is easy to see that the classes sSC^i are closed under polynomial bounded AND and OR operations. *i.e.*

Proposition 3.5 *For $i \geq 0$, sSC^i is closed under polynomial bounded* AND *and* OR *operations.*

3.4 Relations with other hierarchies

There are two hierarchies in the literature which are similar in nature to the sSC hierarchy. Vinay, in [96] studied the hierarchy of poly-log width bounded branching programs of polynomial size and log depth circuits of poly-log OR-fan-in and polynomial size. Let us denote the branching program hierarchy by $\mathsf{BP}(\mathsf{poly}\log)$ (i th level being $\mathsf{BP}(\log^i n)$) and the circuit hierarchy by $\mathsf{SAC}^1(\mathsf{poly}\log)$ (ith level being $\mathsf{SAC}^1(\log^i n)$). It is easy to see that $\mathsf{BP}(\log^i n)$ is contained in $\mathsf{SAC}^1(\log^i n)$ and sSC^i (by definition). In [96] it was shown that both these hierarchies are closed under complementation level-by-level. Lack of closure under complementation within a level (for levels above 0) makes the sSC hierarchy look somewhat odd one out when

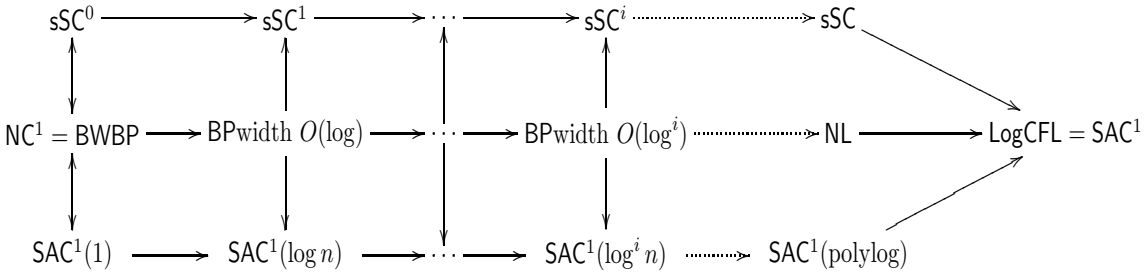


Figure 3.1: Hierarchies of classes between NC^1 and LogCFL

compared with the $\mathsf{BP}(\mathsf{poly}\log)$ and $\mathsf{SAC}^1(\mathsf{poly}\log)$ hierarchies. Figure 3.1 shows the relationship of sSC hierarchy with $\mathsf{BP}(\mathsf{poly}\log)$ and $\mathsf{SAC}^1(\mathsf{poly}\log)$ hierarchies.

3.5 Conclusion

In this chapter we introduced the hierarchy sSC of languages that are computed by polynomial size circuits of poly-logarithmic width and polynomial syntactic degree. We conclude the chapter by stating some of the open problems that are related to the sSC hierarchy.

- In the Boolean setting, exactly how much does the polynomial degree constraint restrict SC^1? In $\mathsf{NC}^1 \subseteq \mathsf{sSC}^1 \subseteq \mathsf{SC}^1 = \mathsf{L}$, are any of the containments strict? Of particular interest would be to show $\mathsf{L} \subseteq \mathsf{sSC}^1$. As a starting point for this, it would be nice to see if sSC^1 is closed under complementation.
- Compare the polynomial size branching program class (NL) with sSC.
- Relate $\mathsf{SAC}^1(\mathsf{poly}(\log))$ and sSC hierarchies.

Chapter 4

Counting and arithmetic variants of NC^1

4.1 Introduction

Given a non-deterministic complexity class $\mathcal{C}$, one can define its counting variant $\#\mathcal{C}$ as the class of all functions that can be represented as the number of accepting paths of a machine in the class $\mathcal{C}$. Counting complexity classes were introduced by Valiant ([91]) in order to study the complexity of computing the permanent. He showed that the permanent is $\#\mathsf{P}$ complete. Other popular counting classes that are extensively studied in the literature include GapP, $\#\mathsf{L}$ and GapL. (See [34] and [6].)

In the case of a Boolean circuit complexity class $\mathcal{C}$, we can define corresponding counting class $\#\mathcal{C}$ as functions that count the number of proving sub-trees of a circuit in $\mathcal{C}$. This is equivalent to arithmetizing the class $\mathcal{C}$, *i.e.* replace the $\vee$ gates by $+$ and $\wedge$ gates by $\times$ and assume that wires carry values from $\mathbb{Z}$. Arithmetic versions of various circuit complexity class have been studied in the literature. For example, see a survey on the topic by Eric Allender [4].

Our focus is on the counting versions of different but equivalent definitions of the complexity class NC^1.

Let us briefly review some of the known equivalent definitions of NC^1 (for more detailed descriptions see Chapter 2.),

- Polynomial sized formula of log depth and constant fan-in: NC^1

- Polynomial sized formula: F
- Polynomial sized circuit of log depth and constant fan in : NC^1
- Polynomial sized formula of log width: LWF
- Polynomial sized branching programs of constant width: BWBP
- Polynomial sized circuits of constant width: SC^0
- Polynomial sized circuits of constant width and polynomial syntactic degree: sSC^0

There have been studies on the counting variants of some of the above definitions. Caussinus et al.[28] introduced and studied the counting variants of NC^1 and bounded width branching programs. They showed that $\#\mathsf{BWBP} \subseteq \#\mathsf{NC}^1$, equality holds if negative constants are allowed, *i.e.* $\mathsf{GapBWBP} = \mathsf{GapNC}^1$. In [28], it is also shown that $\#\mathsf{BWBP}$ also coincides with counting number of accepting paths in a non-deterministic finite automaton (NFA) but we will not be going further into this characterization.

In this chapter, we attempt to extend the notion of counting classes to include arithmetizations of the classes LWF and SC^0. It is easy to notice that a straightforward definition of $\#\mathsf{SC}^0$ allows it to compute functions whose values can be as large as 2^{2^n}. This makes $\#\mathsf{SC}^0$ uninteresting, since it is not even contained inside $\#\mathsf{P}$ (as a polynomial time bounded non-deterministic machine can have at most exponentially many accepting paths.). To overcome this difficulty, we propose $\#\mathsf{sSC}^0$ as a reasonable arithmetic version of SC^0.

Later in the chapter, we introduce yet another new definition to the already long list of equivalent definitions of NC^1 by extending the definition of restricted grid branching programs (rGP for short) of [5]. We show that counting paths in an rGP is equivalent to counting proving sub-trees in a formula.

We now give formal definitions of the counting complexity classes that are relevant to this chapter.

Counting Classes

Let C be a boolean circuit on n variables. Define $f_C : \{0,1\}^n \to \mathbb{N}$ as $f_C(x) = \#C(x), \forall x \in \{0,1\}^n$, where $\#C(x)$ is the number of proving sub-trees of C on

x. It is not hard to see that f_C is also the function computed by the circuit C' obtained by replacing $\vee$ gates by $+$, $\wedge$ gates by $\times$ and allowing the wires to carry arbitrary integers (note that the leaf nodes of C' may still contain negations of variables). Such circuits are called as "counting-arithmetic circuits". We define the "syntactic degree" of C' to be the syntactic degree of C. We use this latter definition more often. For a boolean circuit complexity class $\mathcal{C}$ define $\#\mathcal{C} = \{f_C \mid C \text{ is a boolean circuit of type } \mathcal{C}\}$. The following are the counting circuit classes that we use in this chapter:

Definition 4.1

$$\#\mathsf{F} = \left\{ f : \{0,1\}^* \to \mathbb{N} \;\middle|\; \begin{array}{l} f \text{ can be computed by a polynomial size formula} \\ \text{over } \{+, \times, 1, 0, x_i, \overline{x_i}\}. \end{array} \right\}$$

$$\#\mathsf{NC}^1 = \left\{ f : \{0,1\}^* \to \mathbb{N} \;\middle|\; \begin{array}{l} f \text{ can be computed by a polynomial size, log depth,} \\ \text{constant fan-in circuit over } \{+, \times, 1, 0, x_i, \overline{x_i}\}. \end{array} \right\}$$

Let $(B_n)_{n\geq 0}$ be a non-deterministic branching program family accepting a language $\mathcal{L}$. For any path P between two designated nodes s and t in B_n, $\mathsf{weight}(P)$ denotes the product of all edge labels in P. For $x \in \{0,1\}^n$, let $\#B_n(x) = \sum_P \mathsf{weight}(P)$, where P ranges over all s-t paths in $B_n(x)$. The following are the counting classes in the branching program model. In the definition below, we assume that the edge labels are from $\{0, 1, x_1, \ldots, x_n, \bar{x_1}, \ldots, \bar{x_n}\}$.

Definition 4.2

$$\#\mathsf{BP} = \left\{ f : \{0,1\}^* \to \mathbb{N} \;\middle|\; \begin{array}{l} \text{There exists a polynomial size BP family } (B_n)_{n\geq 0} \\ \text{such that for all } x \in \{0,1\}^n, n \geq 0,\ f(x) = \\ \#B_n(x) \end{array} \right\}$$

$$\#\mathsf{BP}[w] = \left\{ f : \{0,1\}^* \to \mathbb{N} \;\middle|\; \begin{array}{l} \text{There exists a BP family } (B_n)_{n\geq 0} \text{ of polyno-} \\ \text{mial size and } O(w) \text{ width such that for all } x \in \\ \{0,1\}^n, n \geq 0,\ f(x) = \#B_n(x) \end{array} \right\}$$

$$\#\mathsf{LWBP} = \#\mathsf{BP}[\log n] \quad ; \quad \#\mathsf{BWBP} = \#\mathsf{BP}[O(1)]$$

We refer to the above type of branching programs as "counting branching programs", to indicate the functions computed by them.

One can extend all the above definitions to include integer valued functions with one of the following modifications:

1. Allow the constant -1 as labels of leaves (edges in the case of BPs)
2. Define f as difference of two functions in the corresponding counting class.

The classes obtained from the modification 1 above are called Gap counting classes and are denoted by the prefix Gap in place of $\#$, or more formally:

Definition 4.3

$$\mathsf{GapF} = \left\{ f : \{0,1\}^* \to \mathbb{N} \mid \begin{array}{l} f \text{ can be computed by a polynomial size formula} \\ \text{over } \{+, \times, -1, 1, 0, x_i, \overline{x_i}\}. \end{array} \right\}$$

$$\mathsf{GapBWBP} = \left\{ f : \{0,1\}^* \to \mathbb{N} \mid \begin{array}{l} \text{There exists a branching program family } (B_n)_{n\geq 0} \\ \text{of polynomial size and constant width with edge} \\ \text{labels from } \{-1, 0, 1, x_1, \ldots, x_n, \bar{x_1}, \ldots, \bar{x_n}\} \text{ such} \\ \text{that for all } x \in \{0,1\}^n, n \geq 0,\ f(x) = \#B_n(x) \end{array} \right\}$$

The classes GapNC^1, GapBP *and* $\mathsf{GapLWBP}$ *are defined in a similar fashion.*

The second type of modification leads to classes known as Difference (Diff for short) counting classes, denoted with a prefix Diff. e.g,

$$\mathsf{DiffNC}^1 = \{f : \{0,1\}^* \to \mathbb{Z} \mid \exists g, h \in \#\mathsf{NC}^1 \ f = g - h\}$$

.

The classes DiffF, DiffBP, $\mathsf{DiffLWBP}$ and $\mathsf{DiffBWBP}$ can be defined similarly. For all the above classes, it is easy to see that the Gap and Diff versions coincide.

Proposition 4.4 *For* $\mathcal{C} \in \{\mathsf{NC}^1, \mathsf{F}, \mathsf{BP}, \mathsf{LWBP}, \mathsf{BWBP}\}$, $\mathsf{Gap}\mathcal{C} = \mathsf{Diff}\mathcal{C}$.

Remark However, in the case of AC^0, proving $\mathsf{GapAC}^0 = \mathsf{DiffAC}^0$ required fairly non-trivial proof. See [7].

In the following proposition, we summarize the known relationships among the counting classes defined above:

Proposition 4.5 *[28]*

$$\#\mathsf{BWBP} \subseteq \#\mathsf{NC}^1 = \#\mathsf{F} \subseteq \#\mathsf{LWBP} \subseteq \#\mathsf{BP}$$

$$\mathsf{GapBWBP} = \mathsf{GapNC}^1 = \mathsf{GapF} \subseteq \mathsf{GapLWBP} \subseteq \mathsf{GapBP}$$

Remark Note that in the uniform setting the class #BP and GapBP coincide with the well known classes #L and GapL respectively.

4.2 Counting in a log-width formula

In this section we introduce the counting version of log-width formula. We show that the construction of [46] preserves the number of proving sub-trees. We define the classes #LWF and GapLWF formally:

Definition 4.6

$$\#\mathsf{LWF} = \left\{ f : \{0,1\}^* \to \mathbb{N} \;\middle|\; \begin{array}{l} f \text{ can be computed by a polynomial size } O(\log n) \\ \text{width formula over } \{+, \times, 1, 0, x_i, \overline{x_i}\}. \end{array} \right\}$$

$$\mathsf{GapLWF} = \left\{ f : \{0,1\}^* \to \mathbb{N} \;\middle|\; \begin{array}{l} f \text{ can be computed by a polynomial size } O(\log n) \\ \text{width formula over } \{+, \times, -1, 1, 0, x_i, \overline{x_i}\}. \end{array} \right\}$$

Theorem 4.7

$$\#\mathsf{LWF} = \#\mathsf{F} = \#\mathsf{NC}^1 \quad ; \quad \mathsf{GapLWF} = \mathsf{GapF} = \mathsf{GapNC}^1$$

Proof. Clearly, #LWF $\subseteq$ #F. It follows from [27] (see also [4]) that #F is in #NC^1. So we only need to show that #NC^1 is in #LWF.

Lemma 2 in [46] establishes that NC$^1 \subseteq$ LWF. Essentially, it starts with a $O(\log n)$ depth formula (any NC^1 circuit can be expanded into a formula of polynomial size and $O(\log n)$ depth), and staggers the computations at each level of the formula. In the process, the size blows up by a factor exponential in the original depth; this is still a polynomial. Since no other structural changes are done, it is not hard to see that the reduction preserves proof trees. $\diamond$

4.3 Counting version of sSC^o

4.3.1 Definition

In this section we propose an arithmetic variant of bounded width circuits in the form of counting proving subtrees in a width bounded circuit.

It is not hard to see that there exist width-2 circuits which can have $O(2^{2^n})$ many proving sub-trees. Hence a straightforward counting version of SC^0 cannot even be compared with much larger classes like $\#\mathsf{P}$. This is an unwanted scenario as one expects any counting version of a class below P to be at least contained within $\#\mathsf{P}$. To overcome this, we insist that the syntactic degree of the circuit be upper bounded by some $\mathsf{poly}(n)$ which in turn bounds the maximum number of proving sub-trees by $2^{\mathsf{poly}(n)}$. We give a formal definition of $\#\mathsf{sSC}^i$:

Definition 4.8 *$\#\mathsf{sSC}^0$ is the class of functions $f : \{0,1\}^* \to \mathbb{N}$ computed by circuits of polynomial size, $O(1)$ width and polynomial syntactic degree with gates labeled from $\{+, \times, 0, 1.x_1, \ldots, x_n, \bar{x_1}, \ldots, \bar{x_n}\}$, where $x_1, \ldots, x_n$ can take only boolean inputs. Equivalently, $\#\mathsf{sSC}^0$ is the class of functions counting the number of proof trees in an sSC^0 circuit. Additionally, if the constant -1 is allowed in the circuit, then the resulting class of functions is denoted by GapsSC^0.*

Now since branching programs are nothing but skew-circuits and degree of a skew circuit is bounded by its depth, we have

Proposition 4.9 *1.* $\#\mathsf{BWBP} \subseteq \#\mathsf{sSC}^0$

2. $\mathsf{GapNC}^1 = \mathsf{GapBWBP} \subseteq \mathsf{GapsSC}^0$

It is not hard to see that circuits that use the constant-1 can be written as difference of two circuits which do note use -1 by blowing up the width by a constant. Hence:

Proposition 4.10 $\mathsf{GapsSC}^0 = \mathsf{DiffsSC}^0$

Apart from the above preliminary observations, any further relationship between $\#\mathsf{sSC}^0$ and $\#\mathsf{BWBP}$ or $\#\mathsf{NC}^1$ is not known. Also, as of now, no relationship between the classes $\#\mathsf{sSC}^0$ and $\#\mathsf{NC}^1$ is known.

4.3.2 Closure Properties

In this section, we show that some of the closure properties that hold for $\#\mathsf{NC}^1$ and $\#\mathsf{BWBP}$ also hold for $\#\mathsf{sSC}^0$. The simplest closures are under addition and multiplication, and it is straightforward to see that $\#\mathsf{sSC}^0$ is closed under these. The next are weak sum and weak product: add (or multiply) the value of a two-argument function over a polynomially large range of values for the second argument. (See [28, 97] for formal definitions.) A simple staggering of computations yields:

Lemma 4.11 $\#\mathsf{sSC}^0$ *is closed under weak sum and weak product.*

$\#\mathsf{NC}^1$ and $\#\mathsf{BWBP}$ are known to be closed under decrement $f \ominus 1 = \max\{f - 1, 0\}$ and under division by a constant $\lfloor \frac{f}{m} \rfloor$. ([1] credits Barrington with this observation for $\#\mathsf{NC}^1$. See the appendix for detailed constructions.) We show that these closures hold for $\#\mathsf{sSC}^0$ as well. The following property will be useful.

Proposition 4.12 *For any f in $\#\mathsf{sSC}^0$ or $\#\mathsf{NC}^1$, and for any constant m, the value $f \mod m$ is computable in FNC^1. Further, the boolean predicates $[f > 0]$ and $[f = 0]$ are computable in $\#\mathsf{sSC}^0$.*

Proof. Consider $f \in \#\mathsf{NC}^1$. Note that, for a constant m, if $a, b \in \{0, \ldots, m-1\}$, then the values $[(a + b) \mod m]$ and $[(ab) \mod m]$ can be computed by an NC^0 circuit. Thus by induction on depth of f, $[f \mod m]$ can be computed in FNC^1. Now consider $f \in \#\mathsf{sSC}^0$. We will argue by induction on the depth of a circuit for f, that $[f \mod m] \in \mathsf{sSC}^0$. The base case is obvious. If $f = g + h$, then by the induction hypothesis, $g \mod m$, $h \mod m \in \mathsf{sSC}^0 = \mathsf{NC}^1$. Thus, $(g \mod m + h \mod m) \mod m \in \mathsf{NC}^1 = \mathsf{sSC}^0$. The case when $f = gh$ is similar. Thus $f \mod m \in \mathsf{FNC}^1$.

Clearly $[f > 0] \in \mathsf{NC}^1$. Since NC^1 is closed under complementation, $[f = 0] \in \mathsf{NC}^1$. Since NC^1 circuits have deterministic branching programs of constant width, and branching programs are nothing but skew circuits, we obtain constant width arithmetic circuits for $[f > 0]$ and $[f = 0]$. $\diamond$

Lemma 4.13 $\#\mathsf{sSC}^0$ *is closed under decrement and under division by a constant m.*

Proof. Consider $f \in \#\mathsf{sSC}^0$, witnessed by an arithmetic circuit C_n of width w, length l and degree p. Also for a fixed m, $(f \mod m)$ can be computed in FNC^1 (see Proposition 4.12). If g, h are in $\#\mathsf{sSC}^0$, then the functions t_1, t_2 defined below can be computed in FNC^1 and $\#\mathsf{sSC}^1$.

$$t_1 = \left\lfloor \frac{g \mod m + h \mod m}{m} \right\rfloor \qquad t_2 = \left\lfloor \frac{(g \mod m)(h \mod m)}{m} \right\rfloor$$

f at level l is either $g + h$ or gh. Let $op \in \{\ominus 1, \text{div } m\}$. The circuit for $op(f)$ takes values of g and h from level $(l-1)$ of C_n, and values of $op(g)$ and $op(h)$ that are inductively available at level $(l-1)$. Appropriate circuits ($\#\mathsf{sSC}^0$ circuits computing the predicates $[f > 0]$ and $[f = 0]$, or $\#\mathsf{sSC}^0$ circuits computing $(g \mod m)$, $(h \mod m)$, t_1, t_2) for each gate at level $l-1$ are explicitly substituted, contributing a multiplicative factor of width $O(w)$ and length $O(l)$ to the constructed circuit.

When $op = \ominus$, we have

$$\begin{aligned} (g+h) \ominus 1 &= (g \ominus 1 + h) \times [g > 0] + (h \ominus 1) \times [g = 0] \\ gh \ominus 1 &= [(g \ominus 1) \times h + h \ominus 1] \times [g > 0] \times [h > 0] \end{aligned}$$

When $op = \text{div } m$, we have

$$\begin{aligned} \left\lfloor \tfrac{g+h}{m} \right\rfloor &= \left\lfloor \tfrac{g}{m} \right\rfloor + \left\lfloor \tfrac{h}{m} \right\rfloor + t_1 \\ \left\lfloor \tfrac{gh}{m} \right\rfloor &= \left\lfloor \tfrac{g}{m} \right\rfloor \times h + \left\lfloor \tfrac{h}{m} \right\rfloor \times (g \mod m) + t_2 \end{aligned}$$

The constructed arithmetic circuit for $op(f)$ has width $O(w^2)$ and length $O(l^2)$. Let $p = deg(C_n)$, $q_1 = \max\{deg([f > 0]), deg([f = 0])\}$, and $q_2 = \max\{deg(g \mod m), deg(h \mod m), deg(t_1), deg(t_2)\}$. Then the circuit for $f \ominus 1$ has degree at most $p + lq_1$, while that for $\lfloor \frac{f}{m} \rfloor$ has degree at most $p + lq_2$.

Thus we have $op(f) \in \#\mathsf{sSC}^0$.

$\diamond$

4.4 Higher Width

One can extend the definition of bounded width counting arithmetic circuits to include poly-logarithmic width circuits.

Definition 4.14

$$\#\mathsf{sSC}^i = \left\{ f : \{0,1\}^* \to \mathbb{N} \;\middle|\; \begin{array}{l} f \text{ can be computed by a } \mathsf{poly} \text{ size,} \mathsf{poly} \text{ degree circuit} \\ \text{of } O(\log^i n) \text{ width over } \{+, \times, 1, 0, x_i, \overline{x_i}\}. \end{array} \right\}$$

$$\#\mathsf{sSC} = \bigcup_{i \geq 0} \#\mathsf{sSC}^i$$

We can define GapsSC^i and GapsSC in the same fashion. One interesting question for these counting classes is how hard will it be to compute bits of a $\#\mathsf{sSC}^i$ function. Using the Chinese remaindering techniques of [29], we show that the individual bits of each $\#\mathsf{sSC}^i$ function can be computed in polynomial time using $O(\log^{i+1} n)$ space. However, the Boolean circuits constructed may not have polynomial degree.

Theorem 4.15 *For all $i \geq 0$, $\#\mathsf{sSC}^i \subseteq \mathsf{GapsSC}^i \subseteq \mathsf{SC}^{i+1}$*

Proof. We show how to compute $\#\mathsf{sSC}^i$ in SC^{i+1}. The result for Diff and hence Gap follow since subtraction can be performed in SC^0.

Let $f \in \#\mathsf{sSC}^i$. Let d be the degree bound for f. Then the value of f can be represented by at most $d \in n^{O(1)}$ many bits. By the Chinese Remaindering technique (Lemma 2.11), f can be computed exactly from its residues modulo the first $O(d^{O(1)})$ primes, each of which has $O(\log d) = O(\log n)$ bits. These primes are small enough that they can be found in log space. Further, due to [29], the computation of f from its residues can also be performed in $\mathsf{L}{=}\,\mathsf{SC}^1$; see also [3]. If the residues can be computed in SC^k, then the overall computation will also be in SC^k because we can think of composing the computations in a sequential machine with a simultaneous time-space bound.

It thus remains to compute $f \bmod p$ where p is a small prime. Consider a bottom-up evaluation of the $\#\mathsf{sSC}^i$ circuit, where we keep track of the values of all intermediate nodes modulo p. The space needed is $\log p$ times the width of the circuit, that is, $O(\log^{i+1} n)$ space, while the time is clearly polynomial. Thus we have an SC^{i+1} computation. $\diamond$

In particular, bits of an $\#\mathsf{sSC}^0$ function can be computed in SC^1, which equals L. On the other hand, by an argument similar to the discussion preceding Proposition 3.2, we know that $\#\mathsf{BWBP}$ is contained in $\#\mathsf{sSC}^0$. Thus

Corollary 4.16 $\mathsf{FNC}^1 \subseteq \#\mathsf{BWBP} \subseteq \#\mathsf{sSC}^0 \subseteq \mathsf{FL} \subseteq \#\mathsf{L}$.
$\mathsf{GapNC}^1 = \mathsf{GapBWBP} \subseteq \mathsf{GapsSC}^0 \subseteq \mathsf{FL} \subseteq \mathsf{GapL}$.

4.5 Various restrictions

4.5.1 Modular Counting

Another consequence of Proposition 4.12 can be seen as follows. We have three competing arithmetizations of the Boolean class NC^1: $\#\mathsf{BWBP}$, $\#\mathsf{NC}^1$ and $\#\mathsf{sSC}^0$. The most natural one is $\#\mathsf{NC}^1$, defined by arithmetic circuits. It contains $\#\mathsf{BWBP}$, which is contained in $\#\mathsf{sSC}^1$, though we do not know the relationship between $\#\mathsf{NC}^1$ and $\#\mathsf{sSC}^0$. Applying a "> 0?" test to any yields the same class, Boolean NC^1. We show here that applying a "$\equiv 0 \bmod p$?" test to any of these arithmetic classes also yields the same language class, namely NC^1.

Definition 4.17 *Let $\mathcal{C} \in \{\mathsf{BWBP}, \mathsf{sSC}^0, \mathsf{NC}^1\}$. For $p \in \mathbb{N}$ define*

$$\mathsf{Mod}_p\mathcal{C} = \{L \in \{0,1\}^* \mid \exists f \in \#\mathcal{C},\ \forall x \in \{0,1\}^*,\ x \in L \iff f(x) \equiv 0 \mod p\}$$

Theorem 4.18 *For any fixed p, $\mathsf{Mod}_p\mathsf{BWBP} = \mathsf{Mod}_p\mathsf{sSC}^0 = \mathsf{Mod}_p\mathsf{NC}^1 = \mathsf{NC}^1$.*

Proof. The NC^1-hardness for each of these three problems is obvious. From Proposition 4.12, for $f \in \{\#\mathsf{sSC}^0, \#\mathsf{BWBP}, \#\mathsf{NC}^1\}$, and a constant m, the value $[f(x) \bmod m]$ can be computed in FNC^1. Hence the predicate $[f(x) \equiv 0 \bmod m]$ can be computed in NC^1. ◇

Remark Note that in Theorem 4.18, p needs to be a constant. If we consider the case where p is also a part of the input, then the argument in the above proof gives the upper bound $\mathsf{Mod}_p\mathsf{NC}^1 \subseteq \mathsf{L}$.

4.5.2 Arithmetic-Boolean versions

There is another natural way to produce Boolean circuits from arithmetic circuits, by allowing the circuit to perform a "test for nonzero" operation. Such circuits,

known as *Arithmetic-Boolean* circuits, were introduced by von zur Gathen, and have been studied extensively in the literature see *e.g.* [99, 98, 27, 4]. We extend this a little further, by looking at bounded width restrictions.

Definition 4.19 *Let $\mathcal{C}$ be any of the arithmetic circuit class studied above, then Arith-Bool $\mathcal{C}$, is defined to be the set of languages, which are accepted by circuits, with the following additional gates,*

$$test(f) = \begin{cases} 0 & \text{if } f = 0 \\ 1 & \text{otherwise} \end{cases} \qquad select(f_0, f_1, y) = \begin{cases} f_0 & \text{if } y = 0 \\ f_1 & \text{if } y = 1 \end{cases}$$

where y is either a constant or a literal.

Assigning $deg(select(f_0, f_1, y)) = 1 + \max\{deg(f_0), deg(f_1)\}$ and $deg(test(f)) = deg(f)$, we have the following,

Lemma 4.20 *1. Arith-Bool $\#\mathsf{NC}^1 = \#\mathsf{NC}^1$.[4]*

2. Arith-Bool $\#\mathsf{BWBP} = \#\mathsf{BWBP}$.

3. Arith-Bool $\#\mathsf{sSC}^0 = \#\mathsf{sSC}^0$

Proof. 1 and 2 are straight forward. If $f \in \#\mathsf{sSC}^0$ then the predicate $[f > 0]$ can be computed by a a deterministic BP and hence by an unambiguous skew-sSC^0 circuit. Now, given any Arith-Bool$\#\mathsf{sSC}^0$ circuit C of length l, starting from the bottom, replace every $test(f)$ gate by the sSC^0 circuit which computes $[f > 0]$, and each $select(f_0, f_1, y)$ by the circuit $\bar{y}f_0 + y.f_1$. We also stagger the resulting circuit C', so that it has width $5w$, and length ll', where l' is an upper bound on the length of the circuit for $[f > 0]$. It can also be seen that $deg(C') \leq deg(f).q$, where q is a polynomial upper bound on the degree of $[f > 0]$. $\diamond$

However, for the Gap classes, we do not have such a collapse. Analogous to the definitions of SPP and SPL, define a class SNC^1: it consists of those languages L for which there is a GapNC^1 function f satisfying

$$\forall x: \quad \begin{array}{ll} x \in L & \iff f(x) = 1 \\ x \notin L & \iff f(x) = 0 \end{array}$$

Similarly, $\mathsf{C_{=}NC^1}$ consists of languages L for which there is a $\mathsf{GapNC^1}$ function f satisfying

$$\forall x: \quad \begin{array}{ll} x \in L & \iff f(x) = 0 \\ x \notin L & \iff f(x) \neq 0 \end{array}$$

Then we have the following conditional result.

Lemma 4.21 *Arith-Bool*$\mathsf{GapNC^1}$=$\mathsf{GapNC^1}$ *if and only if* $\mathsf{SNC^1}$=$\mathsf{C_{=}NC^1}$.

Proof. If Arith-Bool$\mathsf{GapNC^1}$ = $\mathsf{GapNC^1}$, then the characteristic functions of languages in $\mathsf{C_{=}NC^1}$ can be computed in $\mathsf{GapNC^1}$. (Put a single test operation above the circuit for the $\mathsf{GapNC^1}$ function.) This implies that $\mathsf{C_{=}NC^1}$ is in $\mathsf{SNC^1}$. Conversely, if $\mathsf{SNC^1} = \mathsf{C_{=}NC^1}$, then any test operation can be performed in $\mathsf{GapNC^1}$. Select can be implemented using test and arithmetic operations anyway. ◇

4.6 Counting paths in restricted grid graphs

G-graphs are the graphs that have planar embeddings where vertices are embedded on a rectangular grid, and all the edges are between adjacent columns.In these graphs, the node s is fixed as the leftmost bottom node and t is the rightmost top node. (See Figure 4.1 for an example.) In [5], a restriction of G-graphs is considered where the width of the grid is a constant, and only certain kinds of connections are allowed between any two layers. Namely, for width $2k + 2$, the connecting pattern at any layer is represented by one of the graphs $G_{k,i}$ (see figure 4.2) for $0 \leq i \leq 2k + 2$. Let BWrGP denote the class of boolean functions accepted by constant width polynomial size branching programs that are restricted G-graphs, LWrGP the class corresponding to log width polynomial size programs that are restricted G-graphs and rGP denote polynomial size branching programs that are restricted G-graphs.

[5] characterized the bounded depth arithmetic circuits ($\#\mathsf{AC^0}$) in terms of counting number of paths in bounded width rGPs. By closely examining the parameters in [5] and extending he argument for rGPs of arbitrary wdth, we obtain a characterization for $\#\mathsf{NC^1}$ in terms of the restricted version of polynomial size grid branching programs. In the statement and proof below, we use the notion of alternation-depth: a circuit C has alternation depth a if on every root-to-leaf

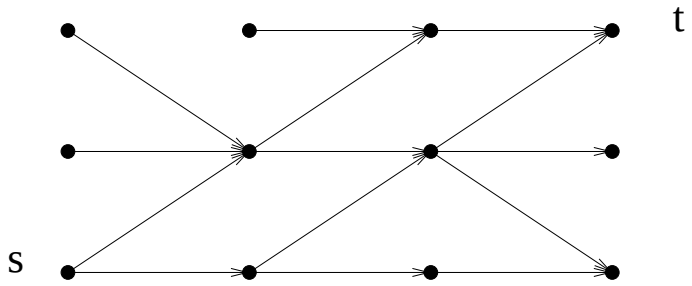


Figure 4.1: An example for a grid graph

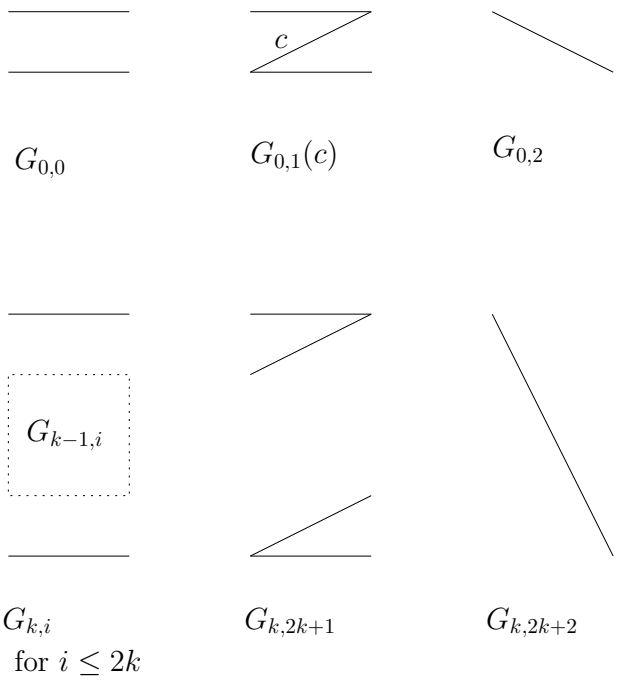


Figure 4.2: The possible patterns between two layers of rGPs

path, the number of maximal segments of gates of the same type is at most a, *i.e.* , there can be at most a switches between the types of gates in a root-to-leaf path.

Lemma 4.22 *Let Φ be a counting-arithmetic formula of size s (i.e. number of wires) and alternation-depth $2d$ over $\{0, 1, +, \times, x_1, \ldots, x_n, \bar{x_1}, \ldots, \bar{x_n}\}$ and with input variables $X = \{x_1, \ldots, x_n\}$. Then there is a restricted grid branching program P of length $s^2 + 2s$ (i.e. the number of edge layers) and width $\max\{2, 2d\}$, where the edges are labeled from $\{0, 1, x_1, \ldots, x_n, \bar{x_1}, \ldots, \bar{x_n}\}$, such that the weighted sum of s-to-t paths in P is equal to the function computed by Φ.*

Proof. The construction here is exactly the same as in [5] for showing $\#\mathsf{AC}^0 = \#BWrGP$; it is included here for completeness. Without loss of generality, assume that the formula Φ is such that all nodes in a particular layer represent the same type of gate and two successive layers have different kind of gates. Also, assume that Φ is height balanced, *i.e.* any root to leaf path in Φ is of length exactly $2d$. Further assume that the root is a $\times$ gate. If these conditions do not hold, then ensuring them will blow up the size of Φ to at most s^2, and increase the depth by at most 2. We assume that s and a are the size and alternation depth of a formula already in this normal form.

We proceed by induction on the depth of the formula Φ. The base case is when $d \leq 1$. If the depth is 0, then Φ is either a variable or a constant in the underlying ring. In this case the graph is $G_{0,1}(c)$ where $\Phi = c$. If $d = 1$, then Φ is a product of linear factors, and a suitable composition of $G_{0,1}(c)$ graphs and $G_{0,2}$ represents it.

Suppose that for any formula F with alternation depth $2d' < 2d$ and size s' (in the normal form described above), there is a restricted grid program P of width $2d'$ and length $s'^2 + 2s'$.

Now let Φ be a normal form formula with alternation depth $2d$. Consider the root gate g of Φ. Let $g_1, \ldots, g_k$ be the children of g, where $g_i = \sum_{j=1}^{t_i} g_{i_j}$. Let s_{i_j} and $2d_{i_j} = 2d - 2$ respectively denote the size and alternation depth of the sub formula rooted at g_{i_j}. Note that $s = k + \sum_i (t_i + \sum_j s_{i_j})$. Applying induction on the sub-formula rooted at each g_{i_j}, let Q_{i_j} denote the resulting restricted grid program for the formula at g_{i_j}. Now place the Q'_{i_j}s $(1 \leq j \leq t_i)$ as in Figure 4.4 to get the program P_i, and connect the P_i's as shown in Figure 4.3 to get the desired program

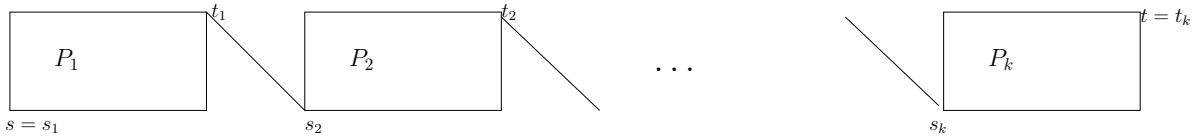


Figure 4.3: Multiplication of rGP's

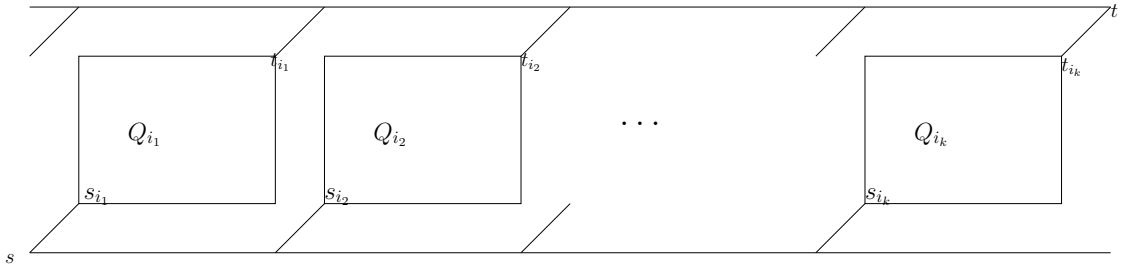


Figure 4.4: Addition of rGP's

P. By the inductive hypothesis, $\mathsf{length}(Q_{i_j}) \leq s_{i_j}^2 + 2s_{i_j}$ and $\mathsf{width}(Q_{i_j}) \leq 2d_{i_j}$. From the construction as above, we have,

$$\begin{aligned} \mathsf{length}(P_i) &= t_i + 1 + \sum_j \mathsf{length}(Q_{i_j}) \\ &\leq t_i + 1 + \sum_j (s_{i_j}^2 + 2s_{i_j}) \end{aligned}$$

and hence,

$$\begin{aligned} \mathsf{length}(P) &= k - 1 + \sum_i \mathsf{length}(P_i) \\ &\leq k - 1 + \sum_i ((t_i + 1) + \sum_j (s_{i_j}^2 + 2s_{i_j})) \\ &\leq s^2 + 2s \end{aligned}$$

Note that the construction in Figure 4.4 adds 2 to the width and the construction in Figure 4.3 does not change the width. Hence the width of P is bounded by $2 \max_{i,j} d_{i_j} + 2 = 2d$.

$\diamond$

We now establish the converse of Lemma 4.22. The proof of the converse as in

[5] is uniform and it produces a circuit rather than a formula. If we do not insist on uniformity of the circuit, then we actually get a formula. Thus it can be shown that functions computed by width $2w+2$, length l restricted grid programs can be computed (non uniformly) by formulas of depth $2w+2$ and size $O(l)$.

Lemma 4.23 *Let P be a counting rGP of length l (number of edge layers) and of width $2w+2$ with labels from $\{0,1,x_1,\ldots,x_n,\bar{x_1},\ldots,\bar{x_n}\}$. Then there exists an equivalent counting-arithmetic formula Φ over $\{0,1,+,\times,x_1,\ldots,x_n,\bar{x_1},\ldots,\bar{x_n}\}$, with alternation depth at most $2w+2$, size (number of wires) at most $2l$.*

Proof. Again, this construction the same as in [5]; The version presented here is non-uniform and results in a formula rather than a circuit as in [5].

For a program B, let $f(B)$ denote the the function computed by B. We proceed by induction on w. The base case is when $w=0$, *i.e.* we have a rGP P of width 2. Then $f(P)$ can be computed by a depth 2 circuit with one $\times$ gate as root and a number of $+$ gates as its inputs, where the $+$ gates get input from $X\cup\{0,1\}$. The total fan-in of the $+$ gates is bounded by the number of layers which contain the graph $G_{0,1}(c)$, for some c. The fan-in of the $\times$ gate is one more than the number of layers which have the graph $G_{0,2}$. (The layers having $G_{0,0}$ do not contribute to the formula.) Thus the total number of wires is bounded by $l+1\leq 2l$, and depth is 2.

Suppose that for any $w'<w$ the claim holds, *i.e.* for a rGP P' of width $2w'+2$ and length l', there is an equivalent formula Φ' of depth $2w'+2$ and size $2l'$.

Now P is the given rGP of width $2w+2$ and length l. Let P be composed as $g_1,\ldots,g_l$. Let $i_1<i_2<\ldots<i_m$ be the (uniquely defined) set of all indices where $g_{i_1},\ldots,g_{i_m}$ are the graph $G_{w,2w+2}$. Define $i_0=0$, $i_{m+1}=l+1$.

For each $0\leq j\leq m$, let P_j denote the program $g_{i_j+1},\ldots,g_{i_{j+1}-1}$ sandwiched between the jth and $(j+1)$th incidence of $G_{w,2w+2}$.

The nodes s_j and t_j for each P_j are defined accordingly. Let l_j denote the length of P_j; then $l=m+\sum l_j$. Note that these P_js do not have $G_{w,2w+2}$ at any layer, and $f(P)=\prod_j f(P_j)$.

Consider P_j for some j. Let $h_{j_1},\ldots h_{j_{r_j}}$ denote the layers of P_j which are the connecting graph $G_{w,2w+1}$. Let $Q_{j,k}$ denote the part of the program between h_{j_k} and $h_{j_{k+1}}$, and $Q_{j,0}$ denote the part between g_{i_j} and h_{j_1} and Q_{j,r_j} denote the part between h_{j_r} and $g_{i_{j+1}}$. Let $Q'_{j,k}$ denote the graph obtained from $Q_{j,k}$ be

removing the top-most and bottom-most lines and the edges connecting them. Then $\mathsf{width}(Q'_{j,k}) = \mathsf{width}(Q_{j,k}) - 2 = 2w$. Let $l_{j,k}$ denote the length of $Q'_{j,k}$; so $l_j \leq r_j + \sum_{k=1}^{r_j-1} l_{l,k}$. The nodes $s'_{j,k}$ and $t'_{j,k}$ for $Q'_{j,k}$ are defined accordingly. Now $f(P_j) = \sum_{k=1}^{r_j-1} f(Q'_{j,k})$. (Note that $Q_{j,0}$ and Q_{j,r_j}, even if non-trivial, play no role in $f(P_j)$ because there is no connection from s_j to these blocks.)

By induction, for each $Q'_{j,k}$ we obtain equivalent formula $\Phi_{j,k}$ with $\mathsf{size}(\Phi_{j,k}) = s_{j,k} = 2l_{j,k}$ and $\mathsf{depth}(\Phi_{j,k}) = d_{j,k} = 2w$. Now define $\Phi = \prod_j \sum_{k=1}^{r_j-1} \Phi_{j,k}$. Then $\mathsf{size}(\Phi) = s = m + \sum_j (r_j - 1 + \sum_k 2l_{j,k}) \leq 2l$ and $\mathsf{depth}(\Phi) = 2w + 2$ as desired. ◇

Note that if we start with a $\log n$ depth formula then we get an rGP of $O(\log n)$ width. Moreover, if we start with an rGP of polynomial width, Lemma 4.23 gives an equivalent counting arithmetic formula. Also, since a formula of polynomial size can be depth reduced to logarithmic depth ([22]), the following is an immediate consequence of the above two lemmas:

Corollary 4.24 $\#\mathsf{NC}^1 = \#\mathsf{LWrGP} = \#\mathsf{rGP}$

It is easy to see that Lemmas 4.22 and 4.23 hold even when -1 is allowed in the rGP/formula:

Corollary 4.25 $\mathsf{GapNC}^1 = \mathsf{GapLWrGP} = \mathsf{GaprGP}$

Note that the above construction also holds in the case of boolean circuits. Hence we have the following characterization for NC^1.

Corollary 4.26 $\mathsf{NC}^1 = \mathsf{LWrGP} = \mathsf{rGP}$.

Thus we get a characterization for NC^1 and $\#\mathsf{NC}^1$ in terms of a restricted class of polynomial size planar branching programs.

4.7 Translating into the Valiant's model

Suppose we allow arithmetic operations over an arbitrary ring and allow arbitrary constants in the circuit. This takes us beyond the counting classes and lands us into the algebraic computational model known as Valiant's model, introduced by Leslie G Valiant [91]. In this section we give a brief description of Valiants model and define various complexity classes in this model. Then we define width bounded

classes for this model. This sets the stage for the next chapter, where we study relationships among syntactic multilinear restrictions of theses classes.

4.7.1 Valiants' Algebraic Model

Let $\mathbb{K}$ be a fixed field. Let $X = \{x_1, \ldots, x_n\}$ be variables that take values from $\mathbb{K}$. An arithmetic circuit (or a straight line program) over $\mathbb{K}$ is a directed acyclic multigraph C, where nodes of zero in-degree are called input gates and are labeled from the set $\mathbb{K} \cup \{x_1, \ldots, x_n\}$. The nodes of zero out-degree are called as output gates. The remaining nodes of C are labeled from $\{+, \times\}$. Whenever not stated explicitly, we assume that in-degree of every node is bounded by 2. A gate f computes a polynomial p_f in $\mathbb{K}[X]$ which can be defined inductively in a natural way. *e.g.* suppose $f = g \times h$ then $p_f = p_g \times p_h$, where p_g and p_h are polynomials computed by g and h respectively (available by induction). In what follows, we may use the same symbol f for representing both gate and the polynomial represented by it. A polynomial family $(f_n)_{n\geq 0}$ is said to be computed by a circuit family $(C_n)_{n\geq 0}$ if $\forall n \geq 0$, f_n is computed by C_n.

The measure of size, depth, width and syntactic degree of a circuit is defined in the same way as in the case of boolean circuits. (see Section 2.1).

A "skew" arithmetic circuit is a circuit in which for every gate $f = g \times h$, then either $g \in \mathbb{K} \cup X$ or $h \in \mathbb{K} \cup X$.

An *algebraic branching program* (ABP) over a field $\mathbb{K}$ is a directed acyclic graph G with two designated nodes s (of zero in-degree) and t (of zero out-degree) where the edges are labeled from $\mathbb{K} \cup X$, where $X = \{x_1, \ldots, , x_n\}$. Let P be any s-t path in G, then $\mathsf{weight}(P)$ =product of labels of edges that appear in P. Then the polynomial f_G computed by G is defined as $\sum_P \mathsf{weight}(P)$, where P ranges over all s-t paths in G.

As in the case of Boolean and counting circuits, a "skew" arithmetic circuit can be transformed into an algebraic branching program and vice versa. In fact this transformation increases the width by a constant value.

4.7.2 Valiant's classes

In Valiant's model, a complexity class is a set of families of polynomials $f = (f_n)_{n\geq 0}$. Let us define the different complexity classes that are studied in Valiants' model. For all the classes, we assume that polynomials have degree bounded by $\mathsf{poly}(n)$.

Definition 4.27

$$\mathsf{VP} = \left\{ f=(f_n)_{n\geq 0} \;\middle|\; \begin{array}{l} f_n \text{ can be computed by a polynomial size arith-} \\ \text{metic circuit and } \deg(f_n) \leq \mathsf{poly}(n) \end{array} \right\}$$

$$\mathsf{VNP} = \left\{ f=(f_n)_{n\geq 0} \;\middle|\; \begin{array}{l} \exists \text{ a polynomial family } g = (g_m)_{m\geq 0} \in \mathsf{VP} \\ \text{such that } f_n(X) = \sum_{e\in\{0,1\}^{m'}} g_{m'+n}(X,e), \text{ where} \\ m', \deg(f_n) \leq \mathsf{poly}(n) \end{array} \right\}$$

$$\mathsf{VF} = \mathsf{VP}_e = \left\{ f=(f_n)_{n\geq 0} \;\middle|\; \begin{array}{l} f_n \text{ can be computed by a polynomial size arith-} \\ \text{metic formula} \end{array} \right\}$$

$$\mathsf{VP}_{skew} = \left\{ f=(f_n)_{n\geq 0} \;\middle|\; \begin{array}{l} f_n \text{ can be computed by a polynomial size skew} \\ \text{arithmetic circuit} \end{array} \right\}$$

$$\mathsf{VBP} = \left\{ f=(f_n)_{n\geq 0} \;\middle|\; \begin{array}{l} f_n \text{ can be computed by polynomial size algebraic} \\ \text{branching program} \end{array} \right\}$$

$$\mathsf{VBP}[w] = \left\{ f=(f_n)_{n\geq 0} \;\middle|\; \begin{array}{l} f_n \text{ can be computed by polynomial size algebraic} \\ \text{branching program of width } O(w) \end{array} \right\}$$

$$\mathsf{VLWBP} = \mathsf{VBP}[\log n] \quad ; \quad \mathsf{VBWBP} = \mathsf{VBP}[O(1)]$$

$$\mathsf{VNC}^i = \left\{ f=(f_n)_{n\geq 0} \;\middle|\; \begin{array}{l} f_n \text{ can be computed by polynomial size } O(\log^i n) \\ \text{depth arithmetic circuits of constant fan-in} \end{array} \right\}$$

$$\mathsf{VSAC}^1 = \left\{ f=(f_n)_{n\geq 0} \;\middle|\; \begin{array}{l} f_n \text{ can be computed by polynomial size } O(\log^i n) \\ \text{depth arithmetic circuits with constant fan-in for} \\ \times \text{ gates} \end{array} \right\}$$

$$\mathsf{VLWF} = \left\{ f=(f_n)_{n\geq 0} \;\middle|\; \begin{array}{l} f_n \text{ can be computed by a polynomial size arith-} \\ \text{metic formula of } O(\log n) \text{ width} \end{array} \right\}$$

$$\mathsf{VrGP} = \left\{ f = (f_n)_{n \geq 0} \mid \begin{array}{l} f_n \textit{ can be computed by polynomial size restricted} \\ \textit{grid algebraic branching program} \end{array} \right\}$$

Remark

- Traditionally, Valiants' classes are considered to be non-uniform. Whenever uniformity is required, we prefix the above classes with "Uniform-".
- Many a times we identify the above class of polynomials with the type of circuits which compute them, though there need not be one-to-one correspondence between the two sets. *e.g.* we consider VP as set of all polynomial size arithmetic circuits where degree of the output polynomial is bounded by $\mathsf{poly}(n)$.

We summarize the known relationships among the above classes in the following proposition.

Proposition 4.28 *[25, 18, 22, 93]*

$$\mathsf{VP}_e = \mathsf{VNC}^1 = \mathsf{VBWBP} \subseteq \mathsf{VLWBP} \subseteq \mathsf{VBP} = \mathsf{VP}_{skew} \subseteq \mathsf{VNC}^2 = \mathsf{VSAC}^1 = \mathsf{VP} \subseteq \mathsf{VNP}$$

As Lemmas 4.22 and 4.23 do not depend on the field on which arithmetic operations are performed and also on what values the input variables can take, we have:

Proposition 4.29 $\mathsf{VNC}^1 = \mathsf{VBWBP} = \mathsf{VrGP}$

Now, we translate the definition of $\#\mathsf{sSC}^0$ into the Valiants' model. We can define *syntactic degree* of an arithmetic circuit as the case of boolean circuits. *i.e.* syntactic degree of a circuit C is the formal degree of the polynomial computed by it when every leaf with a label from $\mathbb{K}$ is replaced by a new variable y. It is not hard to see that by computing homogeneous components individually, we can ensure that every polynomial in VP can be computed by a polynomial size arithmetic circuit whose syntactic degree is also bounded by $\mathsf{poly}(n)$. This process requires polynomial width, hence it is not clear if the same thing holds for width bounded arithmetic circuits. Thus the classes,

$$\mathsf{VsSC}^i = \left\{ f = (f_n)_{n \geq 0} \mid \begin{array}{l} f_n \text{ can be computed by a an arithmetic circuit of} \\ \text{polynomial size and polynomial } \textit{syntactic degree}, \\ \text{and width } O(\log^i n). \end{array} \right\}$$

$$\mathsf{VSC}^i = \left\{ f = (f_n)_{n \geq 0} \;\middle|\; \begin{array}{l} f_n \text{ can be computed by a polynomial size circuit} \\ \text{of width } O(\log^i n) \text{ and } \deg(f_n) \leq \mathsf{poly}(n) \end{array} \right\}$$

need not be the same, though we don't know how to separate them either. We define the corresponding hierarchies:

$$\mathsf{VsSC} = \bigcup_{i \geq 0} \mathsf{VsSC}^i \quad ; \quad \mathsf{VSC} = \bigcup_{i \geq 0} \mathsf{VSC}^i$$

The following relations follow from the definition:

Proposition 4.30

$$\mathsf{VNC}^1 = \mathsf{VBWBP} \subseteq \mathsf{VsSC}^0 \subseteq \mathsf{VSC}^0 \subseteq \mathsf{VSC}^1$$

These classes are further studied in Chapters 5, 6 and 7.

4.8 Conclusion and open questions

We have studied the arithmetizations of some classes that are equivalent to NC^1 in the Boolean world. Figure 4.5 shows the relationship among the classes and their counting versions. Figure 4.6 shows how these counting classes fare if we restrict to counting modulo 2. The study here is very much incomplete and throws in the following open questions which will be interesting:

1. Exactly where do the classes $\#\mathsf{sSC}^1$ and $\#\mathsf{sSC}^0$ lie? In particular, can we show that $\#\mathsf{sSC}^0$ equals $\#\mathsf{NC}^1$, or that $\#\mathsf{sSC}^1$ is in FL? What closure properties do these arithmetic classes possess?

2. A study of the language classes SNC^1 and $\mathsf{C}_=\mathsf{NC}^1$ (in particular, their closure properties) may reveal interesting connections. The exact counting and the threshold language class analogues of the classes $\#\mathsf{sSC}^0$ or $\#\mathsf{BWBP}$ may lead to potentially new complexity classes that might also turn out to be interesting.

Translating these classes into Valiant's algebraic model gives a slightly different picture. As we cannot evaluate the circuit bitwise, the containment $\mathsf{GapsSC}^0 \subset$

$\mathsf{FL} \subseteq \mathsf{GapL}$ does not hold in this setting. *i.e.* we don't know how to compare VSC^0 or VsSC^0 with VBP. Figure 4.7 shows the relationships among the classes in the Valiants' model.

The following are some of the interesting questions that arise from our study:

1. Is $\mathsf{VsSC}^0 \subseteq \mathsf{VBP}$?
2. Is the containment $\mathsf{VNC}^1 = \mathsf{VBWBP} \subseteq \mathsf{VsSC}^0$ strict?
3. Under what restrictions can the above questions be answered? (We address this in the next chapter for the case of syntactic multilinear circuits)

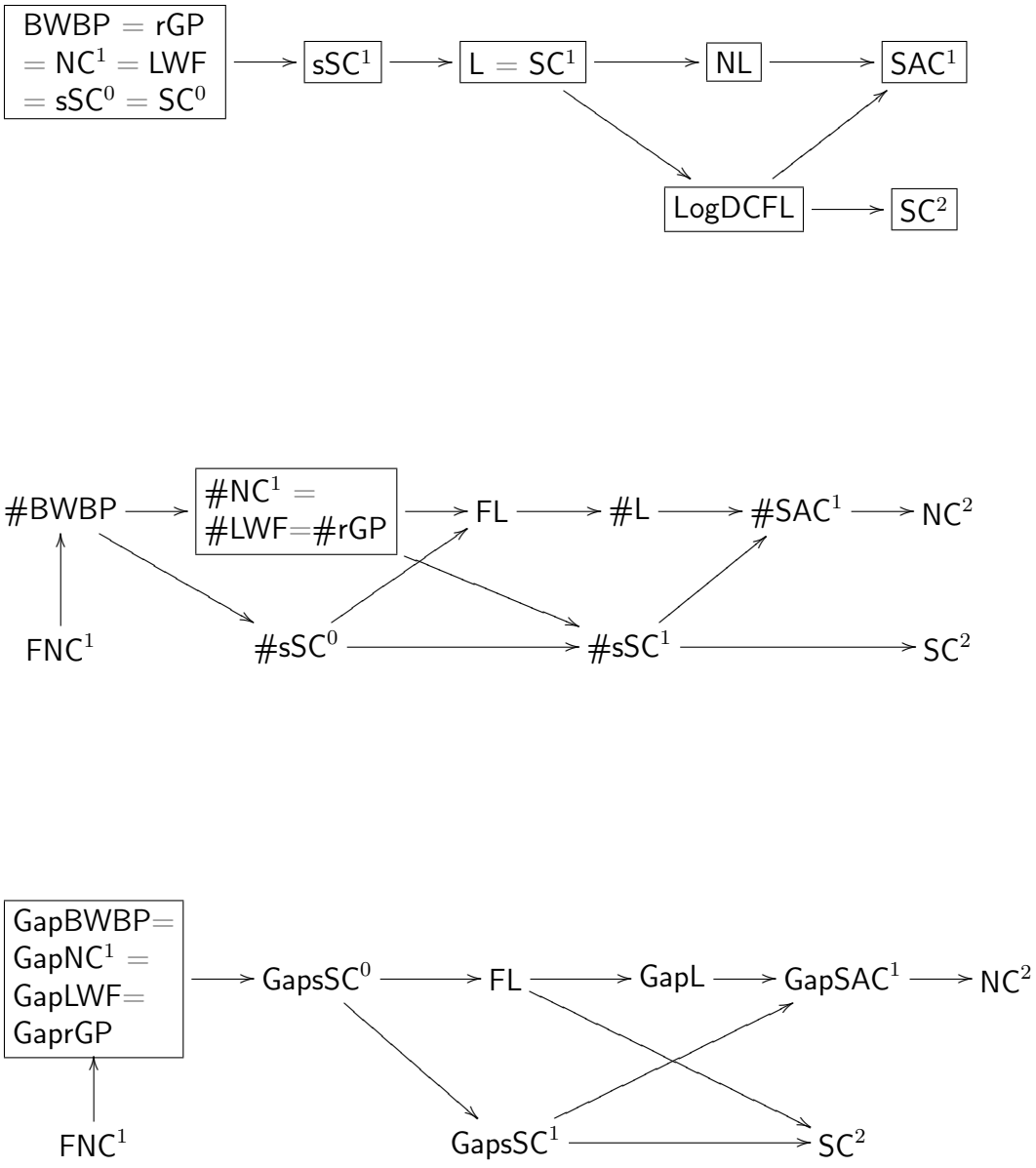


Figure 4.5: Boolean classes and their arithmetizations

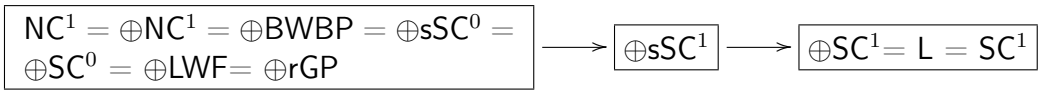


Figure 4.6: Parity Classes around NC^1

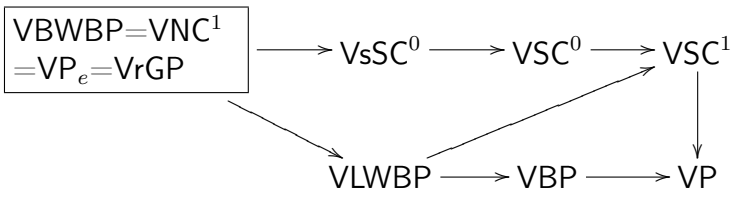


Figure 4.7: In the Valiants' model

4.9 Appendix

4.9.1 Closure Properties of $\#\mathsf{NC}^1$ and $\#\mathsf{BWBP}$

$\#\mathsf{NC}^1$ and $\#\mathsf{BWBP}$ are known to be closed under decrement $f \ominus 1 = \max\{f-1, 0\}$ and under division by a constant $\lfloor \frac{f}{m} \rfloor$. In [1], this observation for $\#\mathsf{NC}^1$ is credited to Barrington. However, we have not seen a published proof, so for completeness, we give details here.

We will repeatedly use the following fact:

Proposition 4.31 (Barrington [17], see also [28]) *For any f in $\#\mathsf{BWBP}$ or $\#\mathsf{NC}^1$, the predicates $[f(x) = 0]$ and $[f(x) > 0]$ are in $\#\mathsf{BWBP}$ and $\#\mathsf{NC}^1$. That is, they can be computed by 0-1 valued arithmetic branching programs / circuits.*

Proof. Start with $f \in \#\mathsf{NC}^1$. By replacing $+$ by $\vee$ gates and $\times$ by $\wedge$ gates, we can see that the predicates $[f > 0]$ and $[f = 0]$ are in NC^1. By [17], these predicates can be computed by deterministic branching programs. The $\#\mathsf{BWBP}$ functions computed by these programs are thus 0-1 valued, as desired. Since $\#\mathsf{BWBP} \subseteq \#\mathsf{NC}^1$, the predicates also have 0-1 valued $\#\mathsf{NC}^1$ circuits. $\diamond$

Lemma 4.32 *The classes $\#\mathsf{NC}^1$ and $\#\mathsf{BWBP}$ are closed under decrement and division by a constant m.*

Proof.

Let $f \in \#\mathsf{NC}^1$. First consider decrement. We show that $f \ominus 1 = \max\{f(x) - 1, 0\} \in \#\mathsf{NC}^1$ by induction on depth of the circuit. The base case, when depth is zero, is straightforward: $f \ominus 1 = 0$. Now consider a circuit of depth d computing f. f is either $g + h$ or gh for some g, h computed at depth $d - 1$.

$$(g + h) \ominus 1 = (g \ominus 1 + h) \times [g > 0] + (h \ominus 1) \times [g = 0]$$

$$(gh) \ominus 1 = [(g \ominus 1) \times h + h \ominus 1] \times [g > 0] \times [h > 0]$$

By induction and using Proposition 4.31, it follows that $f \ominus 1 \in \#\mathsf{NC}^1$.

Next consider division: we want to show $\lfloor \frac{f}{m} \rfloor \in \#\mathsf{NC}^1$. Note that

$$\left\lfloor \frac{g+h}{m} \right\rfloor = \left\lfloor \frac{g}{m} \right\rfloor + \left\lfloor \frac{h}{m} \right\rfloor + \left\lfloor \frac{g \mod m + h \mod m}{m} \right\rfloor$$

$$\left\lfloor \frac{gh}{m} \right\rfloor = \left\lfloor \frac{g}{m} \right\rfloor h + (g \mod m) \left\lfloor \frac{h}{m} \right\rfloor + \left\lfloor \frac{(g \mod m)(h \mod m)}{m} \right\rfloor$$

Now the required result follows from Proposition 4.12, using induction on depth.

In the case of $\#\mathsf{BWBP}$, we use induction on the length of the program. Let $f \in \#\mathsf{BWBP}$. Let w be the width and l be the length of the branching program P for f. We assume without loss of generality that all the edges in any one layer are labeled by the same variable (or a constant). The base case is branching programs of length one, in which case $f \ominus 1$ and $\lfloor \frac{f}{m} \rfloor$ are trivially 0, since $f \in \{0, 1\}$. Assume that for all branching programs P' with length at most $l-1$, $\#P' \ominus 1$ and $\lfloor \frac{\#P'}{m} \rfloor$ are in $\#\mathsf{BWBP}$. Let P be a length l branching program. Let $S = \{v_1, v_2, \ldots, v_w\}$ be the nodes at level $l-1$ of P. We also denote by v_i the value of the $\#\mathsf{BWBP}$ function computed at node v_i. Let the edges out of this level be labeled by 1, x or $\overline{x}$ for some variable x. Now f can be written as $f = \sum_{i \in S_1} v_i + x \sum_{i \in S_2} v_i + \ x \sum_{i \in S_3} v_i$, where nodes in S_1 have an edge labeled 1 to the output node, nodes in S_2 have an edge labeled x to the output node, and nodes in S_3 have an edge labeled $\overline{x}$ to the output node. Let U denote $\sum_{i \in S_2} v_i$ and Y denote $\sum_{i \in S_3} v_i$. Now

$$f \ominus 1 = [v_1 > 0](v_1 \ominus 1 + v_2 + \cdots + v_{j_1} + xU + \overline{x}Y) + [v_1 = 0][v_2 > 0](v_2 \ominus 1 + v_3 + \cdots + v_{j_1} + xU + \overline{Y}) + \cdots$$

Using Proposition 4.31 and induction, we see that $f \ominus 1$ can be computed within $\#\mathsf{BWBP}$, with width $(3w+5)w + 2w$.
Note that, in order to achieve the above width bound, we need to stagger the programs for each term in the above sum. The constant 5 is for computing the predicates like $[v_i > 0]$ and $[v_i = 0]$, which follows from Barrington's construction([17]). The length of the resulting program will be w^2lq, where q is an upper bound for the length of the branching programs which compute predicates $[v_i > 0]$ and $[v_i = 0]$.

$$\begin{aligned}\left\lfloor \tfrac{f}{m} \right\rfloor &= \sum_{i \in S_1} \left\lfloor \frac{v_i}{m} \right\rfloor + x. \sum_{i \in S_2} \left\lfloor \frac{v_i}{m} \right\rfloor + \overline{x} \sum_{i \in S_3} \left\lfloor \frac{v_i}{m} \right\rfloor \\ &\quad + \left\lfloor \frac{\sum_{i \in S_1} v_i \mod m + x \sum_{i \in S_2} v_i \mod m + x \sum_{i \in S_3} v_i \mod m}{m} \right\rfloor\end{aligned}$$

For each i, $v_i \mod m$ can be computed in NC^1. Since w is a constant, we can compute the whole sum in FNC^1 and hence in $\#\mathsf{BWBP}$. By our inductive hypothesis, all v_i's are in $\#\mathsf{BWBP}$, hence $\lfloor \frac{f}{m} \rfloor \in \#\mathsf{BWBP}$. The width of the resulting program is bounded by $2mw$, and size by mwl. $\diamond$

Chapter 5

The syntactic multilinear world

5.1 Introduction

An important question that was left open in the previous chapter: is the class of constant width polynomial size arithmetic circuits of polynomial syntactic degree contained in the class of polynomial size arithmetic formula? *i.e.* is $\mathsf{VsSC}^0 \subseteq \mathsf{VNC}^1$? An ideal result would be a bounded width version of the depth reduction given in [93], *i.e.* the resulting circuit needs to have + fan-in bounded by a function of the width of the original circuit. It is not clear how this can be achieved though. So, one of the natural ways to proceed is to look out for restrictions on the circuits where this can be achieved. The main focus of this chapter is the restriction of syntactic multilinearity on the arithmetic circuits and branching programs. We show that the classes VsSC^0, VNC^1 and VBWBP behave very differently in the syntactic multilinear world.

Syntactic multilinearity (sm for short) is a restriction on the syntactic structure of a circuit. In a syntactic multilinear circuit every multiplication gate operates on disjoint sets of variables. (A formal definition is given in Section 5.2.) The restriction of syntactic multilinearity has been studied in the literature. Also, there are many lower bounds known for syntactic multilinear circuits. Ran Raz in [70] proved a super polynomial lower bound for size of a syntactic multilinear formula computing determinant or permanent. Later in [71], he showed that the classes VNC^1 and VNC^2 are separate in the syntactic multilinear world. (This also holds for multilinear circuits, which need not necessarily be syntactic, *i.e.*

polynomials at every gate are multilinear).

In this chapter we report a "reversal" of the containments among the classes $\mathsf{VBWBP} = \mathsf{VNC}^1 \subseteq \mathsf{VsSC}^0$ in the syntactic multilinear world.

First of all, we give a depth reduction for constant width syntactic multilinear arithmetic circuits. We show that a syntactic multilinear circuit of constant width and polynomial size has an equivalent syntactic multilinear formula of logarithmic depth and polynomial size. Thus, if restricted to be syntactic multilinear, then the class VNC^1 is at least as powerful as VsSC^0 (Note that log depth formula give exactly VNC^1 even in the syntactic multilinear world). But ironically, the containment $\mathsf{VNC}^1 \subseteq \mathsf{VsSC}^0$ does not seem to translate into the syntactic multilinear world. This is mainly because the only known translation ([18]) from log-depth formula into a constant width branching program (and hence a circuit) does not preserve the syntactic multilinearity (this will be discussed with more details in Chapter 6).

Now the scenario is : $\mathsf{sm\text{-}VBWBP} \subseteq \mathsf{sm\text{-}VsSC}^0 \subseteq \mathsf{sm\text{-}VNC}^1$. (sm- prefix denotes restriction to syntactic multilinear circuits.) Looking to tie down this relationship, we obtain a somewhat surprising result: syntactic multilinear algebraic branching programs of constant width and polynomial size are as powerful as syntactic multilinear circuits of constant width and polynomial size. Thus the restriction of syntactic multilinearity pulls VsSC^0 down to VBWBP. This is in fact a reversal in the relationships among VBWBP, VNC^1 and VsSC^0: In the general world VsSC^0 is the strongest class and the other two being equal and contained in VsSC^0, *i.e.* $\mathsf{VBWBP} = \mathsf{VNC}^1 \subseteq \mathsf{VsSC}^0$. In the syntactic multilinear world $\mathsf{sm\text{-}VNC}^1$ turns out to be the strongest one, whereas the other two are equal, *i.e.* $\mathsf{sm\text{-}VBWBP} = \mathsf{sm\text{-}VsSC}^0 \subseteq \mathsf{sm\text{-}VNC}^1$.

The rest of the chapter is organized as follows: In section 5.2 we give formal definitions of syntactic multilinear circuits. Section 5.3 contains a depth reduction for syntactic multilinear constant-width circuits. In section 5.4 we give a width preserving simulation of constant width syntactic multilinear circuits by syntactic multilinear ABPs. In Section 5.5 we discuss the relationships among syntactic multilinear classes.

5.2 Syntactic Multilinear Circuits

We define multilinear and syntactic multilinear circuits as defined in [70]. Let C be an arithmetic circuit over the ring $\mathbb{K}$, and let $X = \{x_1, \ldots, x_n\}$ be its input variables. For a gate g in C, let $p_g \in \mathbb{K}[X]$ be the polynomial computed at g. Let $X_g \subseteq X$ denote the set of variables that occur in the sub-circuit rooted at g. C is called *multilinear* if for every gate $g \in C$, p_g is a multilinear polynomial. C is said to be *syntactic multilinear* if for every multiplication gate $g = h \times f$ in C, $X_h \cap X_f = \emptyset$.

In the case of formulas, the notion of multilinearity and syntactic multilinearity are (non-uniformly) equivalent ([72]).

In the case of algebraic branching programs, the notion of syntactic multilinearity coincides with that of read-once property (See [21]for more about boolean read once branching programs). An algebraic branching program P is multilinear if for every node v in P, the polynomial p_v (sum of weights of all s-v paths) is multilinear. P is syntactic multilinear if in every path of the program (not just s-to-t paths), no variable appears more than once; *i.e.* the algebraic branching program is syntactic read-once.

For any algebraic complexity class VC, we denote by $\mathsf{m\text{-}VC}$ and $\mathsf{sm\text{-}VC}$ respectively the functions computed by multilinear and syntactic multilinear versions of the corresponding circuits.

In [72] it is shown that the depth reduction of [93] preserves syntactic multilinearity; thus

Proposition 5.1 ([72]) *Any function computed by a syntactic multilinear polynomial size polynomial degree arithmetic circuit is in* $\mathsf{sm\text{-}VSAC}^1$.

5.3 Depth reduction in small width sm-circuits

This entire section is devoted to a proof of Theorem 5.2 below, which says that a circuit width bound can be translated to a circuit depth bound, provided the given small-width circuit is syntactic multilinear.

Theorem 5.2 *Let C be a* syntactic multilinear *arithmetic circuit of depth l and width w and syntactic degree d, with $X = \{x_1, \ldots, x_n\}$ as the input variables,*

and constants from the ring $\mathbb{K}$*. Then, there is an equivalent syntactic multilinear circuit* E *of depth* $O(w^2 \log l + \log d)$ *and size* $O(2^{w^2} l^{25w^2} + 4lwd)$*.*

An immediate corollary is,

Corollary 5.3 $\mathsf{sm\text{-}VsSC}^0 \subseteq \mathsf{sm\text{-}VNC}^1$.

It can also be seen that if we apply Theorem 5.2 to a syntactic multilinear arithmetic circuit of poly-logarithmic width and quasi-polynomial size and degree, then we get a poly-logarithmic depth circuit of quasi-polynomial size. Thus

Corollary 5.4

$$\mathsf{sm\text{-}Size}, \mathsf{Width}, \mathsf{Deg}(2^{\mathsf{poly}(\log)}, \mathsf{poly}(\log), 2^{\mathsf{poly}(\log)})$$

$$\subseteq \mathsf{sm\text{-}Size}, \mathsf{Depth}(2^{\mathsf{poly}(\log)}, \mathsf{poly}(\log))$$

We first give a brief outline of the technique used. The main idea is to first cut the circuit C at length $\lceil \frac{l}{2} \rceil$, to obtain circuits A (the upper part) and B (the lower part). Let $M = \{h_1, \ldots, h_w\}$ be the gates of C at level $\lceil \frac{l}{2} \rceil$. A is obtained from C by replacing the gates in M by a set $Z = \{z_1, \ldots, z_w\}$ of new variables. Each gate g of A (or B) represents a polynomial $p_g \in \mathbb{K}[X, Z]$, and can also be viewed as a polynomial in $K[Z]$, where $K = \mathbb{K}[X]$. Since A and B are circuits of length bounded by $\lceil \frac{l}{2} \rceil$, if we can prove inductively that the coefficients of the polynomials at the output gates of A and B can be computed by small depth circuits (say $O(w \log(l/2))$, then, since p_g has at most 2^w multilinear monomials in variables from Z, we can substitute for the z_i's by the value at the output gate g_i of B (*i.e.* polynomials in $\mathbb{K}[X]$). This requires an additional depth of $O(w)$. See Figure 5.1.

The first difficulty in the above argument can be seen even when $w = O(1)$. Though C is syntactic multilinear, the circuit A need not be multilinear in the new dummy variables from Z. This is because there can be gates which compute large constants from $\mathbb{K}$ (*i.e.* without involving any of the variables), and hence have large degree (bounded by the degree of the circuit). This means that the polynomials in the new variables Z at the output gates of A can have non-constant degree, and the number of monomials can be large. Thus the additional depth needed to compute the monomials will be non-constant; hence the argument fails.

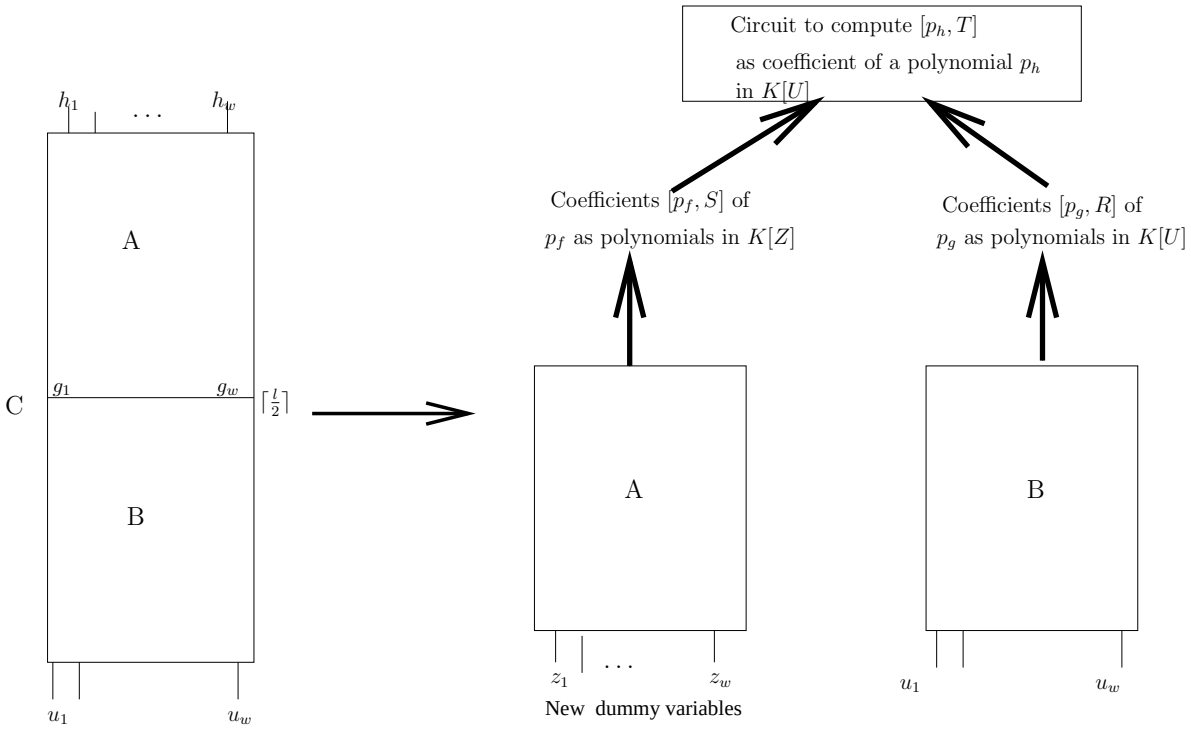


Figure 5.1: Breaking up circuit C into A and B

To overcome this difficulty, we first transform the circuit C into a new circuit C', where no gates compute "large" constants in $\mathbb{K}$. Let C be a syntactic multilinear circuit of length l and width w. Assume without loss of generality that every gate in C has a maximum fan-out of 2. For a gate $g \in C$, define the sets

$$\mathsf{leaf}(g) = \{h \in C \mid h \text{ is a leaf node in } C, \text{ and } g \text{ is reachable from } h \text{ in } C\}$$

$$G = \{g \in C \mid \mathsf{leaf}(g) \cap X = \emptyset\}$$

Thus G is exactly the nodes that syntactically compute constants. Now define C' as a new circuit which is the same as C except that, for all $g \in G$, we replace the i^{th} $(i = 1, 2)$ outgoing wire of g by a new variable y_{g_i}.[1] Note that the number of such new variables introduced is at most $4lw$. (The constants can appear anywhere in the circuit. So each gate can have two new variables on its output wires and two new variables on its input wires.) Let $Y = \{y_{g_i} \mid g \in G, 1 \leq i \leq 2\}$. We show that C' is syntactic multilinear in the variables $X \cup Y$.

Lemma 5.5 *The circuit C' constructed above is syntactic multilinear in the variables $X \cup Y$. Further, C' does not have any constants.*

Proof. The circuit C' is clearly syntactic multilinear in the variables from X. For any gate g in C', let V_g denote $\mathsf{leaf}(g) \cap Y$. Suppose $\exists h \in C'$, $h = g \times f$, such that h is not syntactic multilinear. Then a variable y from Y must be used by f and g, so $V_f \cap V_g \neq \emptyset$. Since each variable from Y occurs on exactly one wire, this implies that there is a gate $e \in C$ such that e has a path to both f and g (e could be the head of the wire carrying y), and $y \in V_e$. Choose the highest such e (closest to h); then the $e - f$ and $e - g$ paths are disjoint. Since C is syntactic multilinear, it must be the case that $e \in G$. But by the construction above, we have $y_{e_1} \in V_f$ and $y_{e_2} \in V_g$, and due to these new variables y_{e_1} and y_{e_2}, y is not in V_f and V_g at all. (In fact, none of the variables in V_e are in V_f or V_g.) $\diamond$

We now show, in Lemma 5.6, how to achieve depth reduction for syntactic multilinear bounded width circuits which have no constants. Then we complete the proof of Theorem 5.2 by explicitly plugging in the constants (*i.e.* the actual values represented by variables in Y) computed by the circuit C.

[1] Instead of the wires, if we replace each gate $g \in G$ by a new variable then the resulting circuitmay not be syntacti multilinear.

Lemma 5.6 *Let C' be a width w, length l syntactic multilinear arithmetic circuit with leaves labeled from $X \cup Y$ (no constants). Then there is an equivalent syntactic multilinear arithmetic formula C'' of size $O(2^{w^2} l^{25w^2})$ and depth $O(w^2 \log l)$ which computes the same polynomial as C'.*

To establish lemma 5.6, we use the intuitive idea sketched in the beginning of the section; namely, slice the circuit horizontally, introduce dummy variables along the slice, and proceed inductively on each part.

Now the top part has three types of variables: circuit inputs X, variables representing constants Y as introduced in Lemma 5.5, and variables along the slice Z. The variables Z appear only at the lowest level of this circuit. Note that this circuit for the top part is syntactic multilinear in Z as well (because there are no constants at the leaves).

To complete an inductive proof for Lemma 5.6, we need to show depth-reduction for such circuits. We use Lemma 5.7 below, which tells us that viewing each gate as computing a polynomial in Z, with coefficients from $K = \mathbb{K}[X, Y]$, there are small-depth circuits representing each of the coefficients. We then combine these circuits to evaluate the original circuit. The essential idea here is to compute the coefficents of polynomials at every stage and then finally produce the polynomial using these coefficients.

More formally, let D be a width w, length l, syntactic multilinear circuit, with all leaves labeled from $X \cup Y \cup Z$ (no constants), where variables from $Z = \{z_1, \ldots z_w\}$ appear only at the lowest level of the circuit. Let $h_1, \ldots, h_w$ be the set of output gates of D *i.e.* gates at level l. Let $p_{h_i} \in \mathbb{K}[X, Y, Z]$ denote the multilinear polynomial computed at h_i. Note that p_{h_i} can also be viewed as a polynomial in $K[Z]$, *i.e.* a multilinear polynomial with variables from Z and polynomials from $\mathbb{K}[X, Y]$ as its coefficients; we use this viewpoint below. For $T \subseteq \{1, \ldots, w\}$, let $[p_{h_i}, T] \in \mathbb{K}[X, Y]$ denote the coefficient of the monomial $m_T = \prod_{j \in T} z_j$ in p_{h_i}. The following lemma tells us how to evaluate these coefficients $[p_{h_i}, T]$.

Lemma 5.7 *With circuit D as above, $\forall h \in \{h_1, \ldots, h_w\}$ and $T \subseteq \{1, \ldots, w\}$, there is a bounded fan-in syntactic multilinear arithmetic formula $D^{h,T}$ of size bounded by $2^{w^2} l^{25w^2}$ and depth $O(w^2 \log l)$, with leaves labeled from $X \cup Y \cup \{0, 1\}$, such that the value computed at its output gate is exactly the coefficient $[p_h, T]$ evaluated at the input setting to $X \cup Y$.*

Proof.

We proceed by induction on the length l of the circuit.

Basis : $l = 1$. In this case, $a \in X \cup Y \cup \mathbb{K}$. The different possibilities are as follows.

$h = z_i z_j$: $[p_h, T] = 1$ for $T = \{i, j\}$ and 0 otherwise.

$h = az_i$: $[p_h, T] = a$ for $T = \{i\}$ and 0 otherwise.

$h = a$: $[p_h, T] = a$ for $T = \emptyset$ and 0 otherwise.

$h = z_i + z_j$: $[p_h, T] = 1$ for $T = \{i\}$ or $T = \{j\}$ and 0 otherwise.

$h = a + z_i$: $[p_h, \emptyset] = a$, $[p_h, \{i\}] = 1$, and $[p_h, T] = 0$ otherwise.

Hypothesis: Assume that the lemma holds for all circuits D' of length $l' < l$ and width w.

Induction Step: Let D be the given circuit of length l, syntactic multilinear in $X \cup Y \cup Z$, where variables from Z appear only at the lowest level of D and that D satisfies the conditions as in Lemma 5.5. Let $\{h_1, \ldots, h_w\}$ be the output gates of D. Let $\{g_1, \ldots, g_w\}$ be the gates of D at level $l' = \lceil \frac{l}{2} \rceil$. Denote by A the circuit resulting from replacing gates g_i with new variables z'_i for $1 \leq i \leq w$, and removing all the gates below level l', and denote by B the circuit with $\{g_1, \ldots, g_w\}$ as output gates, *i.e.* gates above the g_i's are removed. We rename the output gates of A as $\{f_1, \ldots, f_w\}$. Both A and B are syntactic multilinear circuits of length bounded by l' and width w, and of a form where the inductive hypothesis is applicable. For $i \in \{1, \ldots, w\}$, p_{f_i} is a polynomial in $K[Z']$ and p_{g_i} is a polynomial in $K[Z]$, where $K = \mathbb{K}[X, Y]$.

Applying induction on A and B, for all $S, Q \subseteq \{1, \ldots, w\}$, $[p_{f_i}, S]$ and $[p_{g_i}, Q]$ have syntactic multilinear arithmetic circuits $A^{f_i,S}$ and $B^{g_i,Q}$. Note that $p_{h_i}(Z) = p_{fi}(p_{g_1}(Z), \ldots, p_{g_w}(Z))$. But due to multilinearity,

$$p_{f_i}(Z') = \sum_{S \subseteq [w]} \left([p_{f_i}, S] \prod_{j \in S} z'_j \right) \qquad p_{g_j}(Z) = \sum_{Q \subseteq [w]} \left([p_{g_j}, Q] \prod_{s \in Q} z_s \right)$$

Using this expression for p_{f_i} in the formulation for p_{h_i}, we have

$$p_{h_i}(Z) = \sum_{S \subseteq [w]} \left([p_{f_i}, S] \prod_{j \in S} p_{g_j}(Z) \right)$$

Hence, we can extract coefficients of p_{h_i} as follows. The coefficient of the monomial m_T, for any $T \subseteq [w]$ in p_{h_i} is given by

$$[p_{h_i}, T] = \sum_{S \subseteq [w]} [p_{f_i}, S] \left(\text{ coefficient of } m_T \text{ in } \prod_{j \in S} p_{g_j}(Z) \right)$$

If S has t elements, then the monomial m_T is built up in t disjoint parts (not necessarily non-empty), where the kth part is contributed by the kth polynomial p_g in the above expression. So the coefficient of m_T is the product of the corresponding coefficients. Hence

$$[p_{h_i}, T] = \sum_{S=\{j_i,\ldots,j_t\} \subseteq [w]} \left([p_{f_i}, S] \sum_{\substack{Q_1, \ldots, Q_t : \\ \text{partition of } T}} \prod_{k=1}^{t} [p_{g_{j_k}}, Q_k] \right) \tag{5.1}$$

We use this expression to compute $[p_{h_i}, T]$. We first compute $[p_{f_i}, S]$ and $[p_{g_j}, Q]$ for all $i, j \in [w]$ and all $S, Q \subseteq [w]$ using the inductively constructed sub-circuits. Then a circuit on top of these does the required combination. Since the number of partitions of T is bounded by w^w, while the number of sets S is 2^w, this additional circuitry has size at most $w^2 2^w w^w \leq 2^{w^2}$ (for $w \geq 2$) and depth $w \log w + w + \log w = O(w^2)$.

Preserving syntactic multilinearity: Clearly, the circuit obtained above need not be syntactic multilinear. To achieve this, we need to do the following modifications:

- Unwind the expression 5.1 into a formula, by creating necessary copies of $[P_{f_i}, S]$ and $[P_{g_j}, R]$ for all $S, R \subseteq \{1, \ldots, w\}$.
- Consider a term $[P_{f_i}, S][P_{g_1}, Q_1] \cdots [P_{g_1}, Q_1]$ which is conflicting, there are two cases:
  - If for some $a \neq b$, $[P_{g_a}, Q_a]$ and $[P_{g_b}, Q_b]$ share a variable, then we replace $[P_{f_i}, S]$ by 0. Note that this does not affect the output.

– For some a, sub-formula $[P_{g_a}, Q_a]$ and $[P_{f_i}, S]$ share a variable say x, where $a \in S$. We claim that either the polynomial $[P_{g_a}, Q_a]$ or $[P_{f_i}, S]$ does not depend on x. Otherwise, $[P_{h_i}, T]$ will contain non-multilinear monomial which is a contradiction to our assumption of syntactic multilinearity of the sub-circuit rooted at h_i. Now replace x with 0 in the corresponding sub-formula. (*i.e.* either$[P_{f_i}, S]$ or $[P_{g_a}, Q_a]$ depending on which polynomial x does not appear.)

Note that in the above process, we need to unwind the resulting circuit into a formula. By equation 5.1 we need to make at most 2^{w^2} copies of each $[P_{g_k}, Q_k]$ for $k \in [w]$. Hence, the size of the resulting formula will blow up by a factor of $2w2^w2^{w^2} \leq 2^{2w^2}$ at every induction step.

Let $s(l, w)$ and $d(l, w)$ denote the size and depth of the new circuit $D^{p_h,T}$. Then from the construction above, we have the recurrences

$$s(l, w) \leq 2^{2w^2} s(l', w) + 2^{w^2} \leq 2^{3w^2} s(\lceil l/2 \rceil, w)$$

$$d(l, w) \leq d(\lceil l/2 \rceil, w) + O(w^2)$$

Note that $l' = \lceil l/2 \rceil$ satisfies $l' \leq 3l/4$. Suppose that by induction, $s(l', w) \leq 2^{w^2}(l')^{cw^2}$ for some constant c to be chosen later. So

$$\begin{aligned} s(l, w) \leq \; & 2^{3w^2} 2^{w^2} (l')^{cw^2} && \leq \; 2^{4w^2} (3l/4)^{cw^2} \\ = \; & 2^{w^2} l^{cw^2} \left[2^{3w^2} (3/4)^{cw^2} \right] && \leq \; 2^{w^2} l^{cw^2} \end{aligned}$$

where the last inequality holds whenever $8(3/4)^c \leq 1$, say $c \geq 25$.

Similarly, solving the recurrence for $d(l, w)$ gives $d(l, w) = O(w^2 \log l)$. ◇

Finally, we can establish Lemma 5.6.

Proof. [of lemma 5.6] We first relabel all the nodes at the lowest level by new variables $z_1, \ldots, z_w$. Then, applying Lemma 5.7, we obtain circuits for $[p_g, T]$, where g is an output gate of C' and $T \subseteq \{1, \ldots, w\}$. Now, to compute p_g, we sum over all T the values $[p_g, T] \times \prod_{j \in T} val(z_j)$, where $val(z_j)$ denotes the original variable for which z_j was substituted. This adds $O(w)$ to the overall depth of the circuit, thus resulting an overall depth of $O((w + w^2 \log l)) = O(w^2 \log l)$. The resulting circuit size is bounded by $O(s2^w)$, where s is an upper bound on the size

of the circuits constructed in Lemma 5.7, and hence is bounded by $O(2^{w^2}l^{25w^2})$ $\diamond$

And with lemma 5.6 established, we can now get the desired depth-reduction result.

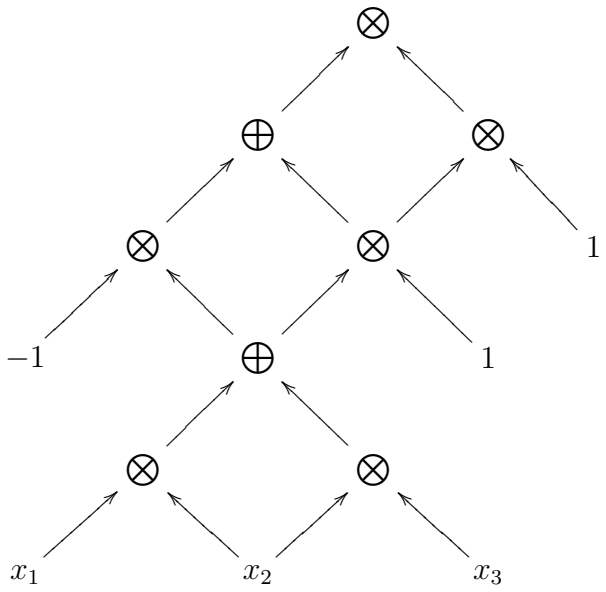
Proof. [of Theorem 5.2] Given circuit C, we construct C' as per Lemma 5.5, and then apply Lemma 5.6 to obtain an equivalent circuit C'' of depth $O(w^2 \log l)$ and size $O(2^{w^2}l^{25w^2})$, which uses variables from $X \cup Y$. To eliminate variables from Y, let $val(y_{g_i})$ denote the value of the gate g in the original circuit C. We now obtain the required circuits by substituting y_{g_i} with $val(y_{g_i}) \in \mathbb{K}$. The new circuit E thus constructed has size $O(2^{w^2}l^{25w^2})$ and depth $O(w^2 \log l)$. $\diamond$

Remark If the constant-width circuit C we start with is multilinear but not syntactic multilinear, then the circuits A as in Lemma 5.7 need not be multilinear in the slice variables Z. This is the place where the above construction crucially uses syntactic multilinearity, and does not generalize to multilinear circuits. See Figure 5.2 for an example.

5.4 Making a circuit skew

From the previous section we have $\mathsf{sm\text{-}VBWBP} \subseteq \mathsf{sm\text{-}VsSC}^0 \subseteq \mathsf{sm\text{-}VNC}^1$. The only known way of simulating a log depth formula by a bounded width ABPs is via the 3-register simulation of Ben-Or and Cleve ([18]). However, this simulation does not preserve syntactic multilinearity, *i.e.* the resulting ABP need not be syntactic multilinear even though the given formula is. (This will be dealt with more detail in Chapter 6). Hence it is not clear how to extend theorem 5.2 to show the inclusion: $\mathsf{sm\text{-}VNC}^1 \subseteq \mathsf{sm\text{-}VBWBP}$. The main purpose of this section is to give a direct simulation of width bounded syntactic multilinear circuits by syntactic multilinear ABPs to show that $\mathsf{sm\text{-}VsSC}^0 \subseteq \mathsf{sm\text{-}VBWBP}$. This implies $\mathsf{sm\text{-}VBWBP} = \mathsf{sm\text{-}VsSC}^0 \subseteq \mathsf{sm\text{-}VNC}^1$.

If we use the standard series-parallel construction (Section 2.2) on a circuit which is not a formula, the size of the resulting ABP can blow up exponentially in the depth of the original circuit (irrespective of its width). In the case of a syntactic multilinear circuit, one can assume that the circuit is multiplicatively disjoint (we will define this soon). Along with this, if we have a width bound of w then for every multiplication gate, one of its sub-circuits is of width at most $w-1$.



Circuit A below is obtained by replacing gates at level 2 by Z variables.

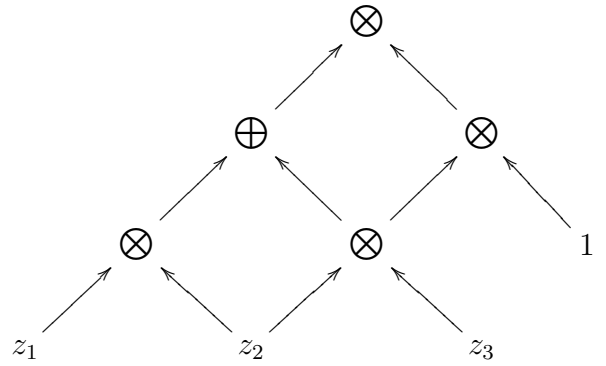


Figure 5.2: A is not multilinear in the slice variable z_2.

We exploit this fact to give a simulation of constant width syntactic multilinear circuits by syntactic multilinear ABPs of constant width, with only a polynomial blow up in the size.

The rest of this section is organized as follows: Section 5.4.1 introduces the notion of multiplicatively disjoint circuits and weakly skew circuits. In section 5.4.2 we give a width efficient simulation of weakly skew circuits by ABPs. Section 5.4.3 gives width efficient simulation of MD circuits by ABPs which preserves syntactic multilinearity.

5.4.1 Multiplicatively disjointness and Weakly skewness

Multiplicatively Disjoint circuits: Let C be an arithmetic circuit. C is said to be *multiplicatively disjoint* (MD for short) if every multiplication gate in C operates on disjoint sub-circuits, *i.e.* if $f = g \times h$ is a gate in C, then the sub-circuits rooted at g and h do not share any node (except the input nodes) between them (see [61]). We denote the multiplicatively disjoint restriction of a class by the prefix md-. *e.g.* $\mathsf{md\text{-}VSC}^i$ denotes the class family of polynomials computed by polynomial size multiplicatively disjoint arithmetic circuits of width $O(\log^i n)$. It is not hard to see that syntactic degree of a multiplicatively disjoint circuit is bounded by its size, hence we have $\mathsf{md\text{-}VsSC}^i = \mathsf{md\text{-}VSC}^i$.

An arithmetic circuit that computes polynomials of polynomial degree can be converted into an equivalent MD-circuit without significant blow up in size ([61]). Thus the three restrictions of MD, small syntactic degree and small degree of the output polynomial all coincide at polynomial size and hence are equal to the class VP. However, when the width of the circuit is bounded by $\mathsf{poly}(\log)$, all these restrictions are seemingly different, with MD circuits being the weakest among them, *i.e.* $\mathsf{md\text{-}VSC}^i \subseteq \mathsf{VsSC}^i \subseteq \mathsf{VSC}^i$.

Consider a syntactic multilinear circuit C. Now replace all the gates in C that are reachable only from leaves labeled by values from $\mathbb{K}$ by the values they represent to obtain a circuit C'. Now, it is easy to see that C' is multiplicatively disjoint and syntactic multilinear, and computes the same polynomial. Thus we can assume without loss of generality that a syntactic multilinear circuit is also multiplicatively disjoint. In particular, $\mathsf{sm\text{-}VsSC}^0 \subseteq \mathsf{md\text{-}VsSC}^0$. Also, note that if the circuit C, is multilinear, but not syntactic multilinear, then C' will not be

multiplicatively disjoint.

Weakly skew circuits: An arithmetic circuit C is said to be *weakly skew*, if for every multiplication gate $f = g \times h$ in C, either the edge (g, f) or the edge (h, f) is a bridge in the underlying graph. By definition weakly skew arithmetic circuits are also multiplicatively disjoint. We denote this restriction on a class by the prefix weaklyskew-.

In [86], Toda has shown that weakly skew circuits have equivalent skew circuits, *i.e.* weaklyskew-VP = VBP. Jansen, in [47] extended this result and showed that weakly skew circuits are equivalent to skew circuits in the syntactic multilinear world too, *i.e.* sm-weaklyskew-VP = sm-VBP. However, the simulation in [47] is not width efficient. In the next section, we present a width efficient version of the simulation in [47].

5.4.2 Weakly skew to skew

In this section we give a simulation of weakly skew syntactic multilinear constant width arithmetic circuits by syntactic multilinear ABPs of constant width. This construction serves as a simpler case of the simulation given in the next section. We include it here since we achieve slightly better size bound, which allows us to translate the result to higher width (see Corollary 5.11).

We briefly outline the overall idea: Essentially, we do the series-parallel construction. (See section 2.2.) Let C be the given weakly skew circuit of width w. All the + gates in C are left untouched. For a multiplication gate $f = g \times h$, suppose w.l.o.g the sub-circuit C_h rooted at h is not connected to rest of the circuit. If $\mathsf{width}(C_h) \leq w - 1$, then we are in good shape, since by placing the ABP $[h]$ (available by induction on the structure of C) in series with (and after) $[g]$ (again available by induction) we can obtain a width bound of $O(w^2)$. If $\mathsf{width}(C_h) = w$, then we have $\mathsf{width}(C_g) \leq w - 1$. In this case, we make a copy of $[g]$ and place it in series with (and after) $[h]$ and again can obtain a width bound of $O(w^2)$, but the size can blow up. Using a careful analysis we argue that size of the new ABP can be bounded by $O(2^w s)$, where s is the size of C. Now we state the main theorem:

Theorem 5.8 weaklyskew-sm-VsSC^0 = sm-VBWBP.

Proof. We use the following normal form for circuits:

Lemma 5.9 *Let C be an arithmetic circuit of width w and size s. Then there is an equivalent arithmetic circuit C' of width $O(w)$ and size $poly(s)$ such that fan-in and fan-out of every gate is bounded by two, and every layer has at most one $\times$ gate. Moreover, C' preserves any of the properties of syntactic multilinearity, weakly-skewness and multiplicatively disjointness.*

Proof. Let k be the bound on maximum fan-in and fan-out of C. First we can reduce the fan-in to two by staggering the circuit and keeping copies of the gates as and when needed. This blows up the width to $2w$ and size to wks. Now in a similar manner we can ensure that the fan-out of a gate is bounded by two and the size blow up will now be w^2k^2s and width will be $4w$. To ensure the second condition we need to push the gates (using staggering and dummy + gates) up by at most $4w$ levels, thus making the total width $8w$ and size $2w^2k^2s$. Since $k \leq w + n$ and $w \leq s$ we have size bounded by $poly(s, n)$. $\diamond$

We need some more definitions and notations. For an ABP B of length d with a single source s, we say B *is endowed with a mainline*, if there exist nodes $v_1, v_2, \ldots, v_{d-1}$ reachable only along the path $s, v_1, v_2, \ldots, v_{d-1}$, and if the labels on this path are all set to the field constant 1. For ABPs B_1 and B_2, *piping* the mainline of B_1 into the mainline of B_2 is the operation of removing the edge from the source of B_2 to the first node v of the mainline of B_2, and adding an edge from the last node w of the mainline of B_1 to v.

The following lemma now gives the theorem 5.8

Lemma 5.10 *Let C be a weakly skew arithmetic circuit of width $w > 1$ and size $s > 1$ in the normal form as given by Lemma 5.9. Let $f_1, \ldots, f_w$ be the output gates of C. Then there exists an equivalent ABP $[C]$ of width $w^2 + 1$, length $2^w s$ and size $(w^2 + 1)2^w s$. $[C]$ has a single start node b and terminal nodes $[f_1], \ldots, [f_w], v$ and will be endowed with a mainline ending in v. Moreover, if C is syntactically multilinear then so is $[C]$.*

Proof. We proceed by induction on $s + w$. If $s = 2$, the lemma holds trivially. If $w = 2$, C is a skew-circuit hence can be seen as an ABP of width 3 (We also need to add a mainline hence width is 3).

Let $s > 2$ and $w > 2$ be given, and assume that C has at least 2 layers. By induction hypothesis, the lemma holds for all circuits of size s' and w', where either $s' < s$ and $w' \leq w$ or $s' \leq s$ and $w' < w$.

Without loss of generality, assume that f_1 is a $\times$ gate and $f_2, \ldots, f_w$ are $+$ gates. Let C' be the circuit obtained by removing the output gates of C. Let $g_1, \ldots, g_w$ be the output gates of C'. Assume (without loss of generality) $f_1 = g_1 \times g_2$, and also that the edge (g_1, f_1) is a bridge in the circuit. We define sub-circuits D and E of C' as follows: D is obtained from C' by deleting the sub-circuit rooted at g_1, E is the sub-circuit rooted at g_1. Let $s' = size(C')$, $w' = \mathsf{width}(C')$, $s_J = \mathsf{size}(J)$ and $w_J = \mathsf{width}(J)$ for $J \in \{D, E\}$. Note that $s = s' + w$, and $s_J < s$ for $J \in \{D, E\}$.

By the induction hypothesis, we have branching programs $[D]$ and $[E]$, both endowed with a mainline. Let $[g_1], v'$ denote the output of $[E]$ and $[g_2], \ldots, [g_w], v''$ denote the output nodes of $[D]$, where v' and v'' are the last nodes on the mainlines. Let $[F]$ be the subprogram of $[D]$, which consists of all paths from the source of $[D]$ to $[g_2]$ and v''. Construct the program $[C]$ with output nodes $[f_1], \ldots, [f_w], v$ as follows:

case 1: $w_E \leq w - 1$.

Now, compose the ABP $[D]$ followed by $[E]$ as follows:

1. For $i, j \geq 2$, $[g_j]$ has an edge to $[f_i]$ iff g_j is an input to f_i.
2. For input gates f_i, draw an edge from v'' to $[f_i]$ with the appropriate label.
3. Identify $[g_2]$ with the start node of $[E]$ and relabel the output node of $[E]$ as $[f_1]$. Pipe the mainline of $[D]$ into the mainline of $[E]$.
4. Stagger the nodes $[f_2], \ldots, [f_w]$ until the last level of the new program.

Size and width analysis: By induction hypothesis we have,

$$\begin{aligned}
\mathsf{width}([E]) &\leq (w_E)^2 + 1 \leq (w-1)^2 + 1 \\
\mathsf{width}([D]) &\leq w^2 + 1 \\
\mathsf{length}([E]) &\leq 2^{w-1} size(E) \\
\mathsf{length}([D]) &\leq 2^w size(D)
\end{aligned}$$

Now,

$$\begin{aligned}
\mathsf{width}([C]) &= \max\{\mathsf{width}([D]), \mathsf{width}([E]) + w - 1\} \leq w^2 + 1 \quad \text{and} \\
\mathsf{length}([C]) &= \mathsf{length}([D]) + \mathsf{length}([E)] \\
&\leq 2^{w_D} s_D + 2^{w_E} s_E \\
&\leq 2^w s_D + 2^{w-1} s_E \leq 2^w s
\end{aligned}$$

as $s = s_D + s_E + w$.

case 2: $w_E = w$, and hence $w_F \leq w - 1$ and $w_D \leq w - 1$.

We compose ABPs $[E]$, $[F]$ and $[D]$ as follows:

1. Identify $[g_1]$ with the source of $[F]$, and pipe the mainline of $[E]$ into the mainline of $[F]$.
2. Add an edge from v' (last node of mainline of $[F]$) to the source of $[D]$,
3. Pipe the mainline of $[F]$ into the mainline of $[D]$.
4. Alongside $[D]$ stagger the output of $[F]$ (which now equals $[f_1]$).
5. For $i, j \geq 2$, $[g_j]$ has an edge to $[f_i]$ iff g_j is an input to f_i.
6. Finally, for input gates f_i, draw an edge $(v'', [f_i])$ with the appropriate label.

Size and width analysis: By induction hypothesis,

$$\begin{aligned}
\mathsf{width}([E]) &\leq w^2 + 1 \\
\mathsf{width}([D]) &\leq (w-1)^2 + 1
\end{aligned}$$

Hence, $\mathsf{width}([F]) \leq (w-1)^2 + 1$. Observe that

$$\begin{aligned}
\mathsf{width}([C]) &\leq \max(\mathsf{width}([E]), \mathsf{width}([F]), \mathsf{width}([D]) + 1) \\
&\leq w^2 + 1
\end{aligned}$$

Now,

$$\begin{aligned}\mathsf{length}([C]) &= \mathsf{length}([E]) + \mathsf{length}([F]) + \mathsf{length}([D]) + 1 \\ &\leq 2^w s_E + 2^{w-1} s_F + 2^{w-1} s_D + 1 \\ &\leq 2^w (s_D + s_E) + 1 \leq 2^w s.\end{aligned}$$

Since the size of a layered ABP is $\mathsf{length} \times \mathsf{width}$, we have the required size bound. If C was syntactic multilinear to start with, then it is easy to see that so is $[C]$. $\diamond$ This also completes the proof of Theorem 5.8 $\diamond$

By the parameters in the Lemma 5.10, it is not hard to see that if we start with a syntactic multilinear weakly skew of $O(\log n)$ width, we get a syntactic multilinear ABP of width $O(\log^2 n)$, *i.e.*

Corollary 5.11 $\mathsf{weaklyskew\text{-}sm\text{-}VsSC}^1 \subseteq \mathsf{sm\text{-}VBP}[\mathsf{width} = \log^2 n]$.

5.4.3 Multiplicatively disjoint to skew

We extend the simulation in Lemma 5.10 to include multiplicatively disjoint circuits. The idea is same as that of Lemma 5.10 but we could obtain a much weaker size bound (*i.e.* $O(s^w)$ instead of $O(2^w s)$) on the resulting ABP.

The main goal of this section is prove the following theorem:

Theorem 5.12 $\mathsf{sm\text{-}VsSC}^0 \subseteq \mathsf{sm\text{-}VBWBP}$

For proving the theorem, we establish the following lemma that states the simulation with general parameters:

Lemma 5.13 *C be a multiplicatively disjoint arithmetic circuit of width w and size s in the normal form as given by Lemma 5.9. Let $f_1, \ldots, f_w$ be the output gates of C. Then there exists an equivalent arithmetic branching program $[C]$ of width $O(w^2)$, length $O(s^w)$, and size $O(w^2 s^w)$. $[C]$ has a single start node b and terminal nodes $[f_1], \ldots, [f_w], v$, and is endowed with a mainline ending in v. Moreover, if C is syntactic multilinear then so is $[C]$.*

Proof.

The proof is similar to that of Lemma 5.10. We proceed by induction on $s+w$. If $s = 2$, the lemma holds trivially. If $w = 1$, C is a skew-circuit, and hence can be seen as a BP of width 3 (by adding a mainline).

Let $s > 2$ and $w > 2$ be given, and assume that C has at least 2 layers. Suppose, by induction hypothesis that the lemma holds for all circuits of size s' and w', where either $s' < s$ and $w' \leq w$ or $s' \leq s$ and $w' < w$.

Let C' be the sub-circuit obtained by deleting $f_1, \ldots, f_w$. Let $G = \{g_1, \ldots, g_w\}$ be the output gates of C'. Without loss of generality, let $f_1 = g_1 \times g_2$ be the only multiplication gate at the output layer of C. Let D denote the sub-circuit rooted at g_1. Since C is multiplicatively disjoint, we have either $\mathsf{width}(D) \leq w - 1$ or $\mathsf{width}(E) \leq w - 1$. Without loss of generality, assume that $\mathsf{width}(D) \leq w - 1$.

Let $s' = \mathsf{size}(C')$, $s_D = \mathsf{size}(D)$, $w' = \mathsf{width}(C')$, and $w_D = \mathsf{width}(D)$. By induction hypothesis, we obtain ABPs $[C']$ and $[D]$. $[C']$ has $w + 1$ output nodes, namely $[g_1], \ldots, [g_w], v$. $[D]$ has two output nodes $[g'_1]$ and v'.

Now construct the ABP $[C]$ with output nodes $[f_1], \ldots, [f_w], v$ by composing $[C']$ followed by $[D]$ as follows: For all $i \geq 2$, connect $[g_j]s$ to $[f_i]s$ according the edges in the circuit C, i.e edge $([g_j], [f_i])$ is in $[C]$ iff g_j is an input for f_i. In case f_i is an input gate, draw an appropriately labeled edge from v. Put an edge from $[g_2]$ to $[f_1]$. Now identify the start node of $[D]$ with $[f_1]$ and re-label the terminal node of $[D]$ as $[f_1]$. Do the necessary staggerings to carry on the values $f_2, \ldots, f_w$ to the last layer. We also pipe the mainline of $[C']$ into the mainline of $[D]$.

Analysis: By induction hypothesis, we have

$$\mathsf{length}([C']) \leq s'^{w'} \leq (s - w)^w$$

as $s' = s - w$ and $w' \leq w$. Furthermore,

$$\begin{aligned} \mathsf{width}([C']) &\leq w'^2 + 1 \leq w^2 + 1 \\ \mathsf{length}([D]) &\leq s_D^{w_D} \leq (s - w)^{w-1} \\ \mathsf{width}([D]) &\leq (w - 1)^2 + 1 \end{aligned}$$

as $s_D \leq s - w$ and $w_D \leq w - 1$.

Now by the construction,

$$\begin{aligned} \mathsf{width}([C]) &= \max\{\mathsf{width}([C'], \mathsf{width}([D]) + w - 1\} \\ &\leq \max\{w^2 + 1, (w-1)^2 + w - 1\} \leq w^2 + 1 \end{aligned}$$

And,

$$\begin{aligned} \mathsf{length}([C]) &= \mathsf{length}([C']) + \mathsf{length}([D]) \\ &\leq (s-w)^w + (s-w)^{w-1} \leq s^w \end{aligned}$$

for $w > 2$ and $w < s$. Thus, $\mathsf{size}([C]) = (w^2+1)s^w$. It is easy to see that this construction preserves the syntactic multilinearity property. ◇

We now give the proof of Theorem 5.12:

Proof. [of theorem 5.12] Given a syntactically multilinear circuit C of width w and size s, we first replace all the wires carrying only constants from $\mathbb{K}$ in C by new variables, to get a circuit D of width $w_d \leq w^2$ and size $s_d \leq ws$. Note that the circuit D is multiplicatively disjoint. By Lemma 5.13 we get a syntactic multilinear ABP $[D]$ of width $w_d^2 + 1$ and size s_d^w. Now replacing the introduced variables by the original constants they represent, we get the required syntactic multilinear ABP $[C]$. ◇

As $\mathsf{sm\text{-}VBWBP} \subseteq \mathsf{sm\text{-}VsSC}^0$, we have,

Theorem 5.14 $\mathsf{sm\text{-}VsSC}^0 = \mathsf{sm\text{-}VBWBP}$.

As Lemma 5.13 works for all multiplicatively disjoint circuits, we get $\mathsf{md\text{-}VSC}^0$ (note that $\mathsf{md\text{-}VsSC}^0 = \mathsf{md\text{-}VSC}^0$) as a largest subclass of VsSC^0 that is known to be equivalent to $\mathsf{VNC}^1 = \mathsf{VBWBP}$. *i.e.*

Corollary 5.15 $\mathsf{weaklyskew\text{-}VSC}^0 = \mathsf{md\text{-}VSC}^0 = \mathsf{VNC}^1 = \mathsf{VBWBP}$

Remark The simulation in the case of weakly skew circuits, given in Section 5.4.2 does carry over to multilinear circuits. However as a multilinear circuit need not be multiplicatively disjoint (see Section 5.4.1), Lemma 5.4.3 does not work for multilinear circuits which are not syntactic multilinear.

5.5 Big picture of the sm-world

Now we turn our attention to the overall picture of the classes defined in Section 4.7 in the syntactic multilinear world. In other words, we attempt to redraw the Figure 4.7 when all the classes are restricted to be syntactic multilinear. We consider and compare the classes sm-VP_e, sm-VNC^1, sm-VsSC^0, sm-VBWBP, and sm-VrGP.

A classical result from [22] shows that for every arithmetic formula F of size s, there is an equivalent arithmetic formula F' which has depth $O(\log s)$ and size $\mathsf{poly}(s)$. A careful observation of this proof shows that if we start with a syntactic multilinear formula F, then the depth-reduced formula F' is also syntactic multilinear.

Theorem 5.16 *Every syntactic multilinear formula with n leaves has an equivalent syntactic multilinear circuit of depth $O(\log n)$ and size $O(n)$.*
In particular, sm-$\mathsf{VP}_e \subseteq$ sm-VNC^1.

Proof. By simultaneous induction on the number of leaves in the formula, we can prove the following statements. This is exactly the construction of [22], analyzed carefully for syntactic multilinearity.

(i) If F is a syntactic multilinear formula with n leaves, then there is an equivalent syntactic multilinear circuit F' of depth $\lceil 4 \log n \rceil$ and size $2n$.

(ii) If x is any leaf in F, then we can express F as $F' = Ax + B$, where A, B are syntactic multilinear, do not depend on x and and are of depth $\lceil 4 \log n \rceil$.

In the base case, there is either a single variable or a constant, and the claim holds trivially. Let u be any node in F with L and R as its children. Let P_v denote the sub-tree of F obtained from removing v and P_L and P_R denote those rooted at L and R respectively. A *tree separator* F is a node v such that the sizes of P_v, P_L and P_R ar atmost half the size of F.

Let X be a tree separator for F, with children L, R, so that $X = L \ \mathsf{op} \ R$. Replace the whole subtree under X by a new variable x. By inductive statement (ii), we have $F' = Ax + B$ where A, B are as above (*i.e.* they are both syntactic multilinear and do not depend on X). Also by inductive statement (i), we have

syntactic multilinear formula L', R' equivalent to L, R of small depth. Thus we have $F' = A.(L' \ \mathsf{op} \ R') + B$. Since A does not depend on any variable below X, F' is syntactic multilinear. Also we can see that it has the required depth.

To prove the second half of the statement above, let x be any leaf in F. Now find a tree separator $X = L \ \mathsf{op} \ R$ such that the subtree at one of its children, say L, contains x as a leaf and is of size $< n/2$. Then, by inductive statement (ii) applied to L, $L' = Ax + B$, where A, B are independent of x, syntactic multilinear and of small depth. Now replace the subtree at X by a new variable y. Applying inductive statement (ii), we have $F' = Cy+D$, where C, D are syntactic multilinear small depth formulas which do not depend on y (*i.e.* $L \ \mathsf{op} \ R$). Applying inductive statement (i) to R, we have an equivalent small-depth R'.

Case 1: $\mathsf{op} \ = +$. Then $F' = C((Ax+B)+R')+D = CAx+(CB+CR'+D)$. This is again syntactic multilinear since C does not depend on y, *i.e.* $Ax+B+R$.

Case 2: $\mathsf{op} \ = \times$. Then $F' = C(Ax + B)R' + D = CAR'x + (CBR' + D)$. Here again F' is syntactic multilinear since C does not depend on A, B, R', and also because A and B do not share any variables with R'.

Since we are constructing a circuit and not a formula, we don't need to replicate the circuits for C and R'. For details about the size/depth, see the analysis in [22]. $\diamond$

It is easy to see that the path-preserving simulation of a constant width branching program by a log depth circuit preserves syntactic multilinearity:

Lemma 5.17 *For any syntactic multilinear branching program P of width w and size s over ring $\mathbb{K}$, there is an equivalent syntactic multilinear circuit C of depth $O(\log s)$ and size $O(s)$ with fan-in of $+$ gate bounded by w (or alternatively, depth $O(\log w \log s)$ and bounded fan-in).*
In particular, $\mathsf{sm\text{-}VBWBP} \subseteq \mathsf{sm\text{-}VNC}^1$ *and* $\mathsf{sm\text{-}VBP} \subseteq \mathsf{sm\text{-}VSAC}^1$.

Proof. Let l be the length of P ($s = lw$), and let $p_{s,t}$ denote the weighted sum of the directed paths between nodes s and t. Let $v_1, \ldots v_w$ denote the nodes at the level $l' = \lceil l/2 \rceil$ of P. Then $p_{s,t} = \sum_{i=1}^{w} p_{s,v_i} \times p_{v_i,t}$. Thus the depth and size of the inductively constructed circuit satisfy the recurrences $d(l) = 2 + d(l')$ and $s(l) = (3w)s(l')$, giving the desired bounds. It is clear that the circuit so

constructed is syntactic multilinear; if it were not, the offending $\times$ gate would pinpoint a path in P that reads some variable twice. $\diamond$

It is also straightforward to see that the construction of [46], staggering a small-depth formula into a small-width one, preserves syntactic multilinearity. Thus

Lemma 5.18 *Let Φ be any sm-formula with depth d and size s. Then there is an equivalent syntactic multilinear formula Φ' of length $2s$ and width d.*
In particular, sm-VNC$^1 \subseteq$ sm-VLWF.

Proof. For completeness we give a detailed proof here. The construction is by induction on the structure of the formula Φ. The base case is when Φ is a single variable or a constant, in which case the lemma holds trivially.

Suppose the lemma holds for any formula of depth at most $d-1$. Consider the root gate f of a formula Φ of depth d. Suppose $f = \sum_{i=1}^{k} g_i$ (respectively $f = \prod_{i=1}^{k} g_i$). As the depth of each formula g_i is bounded by $d-1$, by induction we have formulas g_i' of width $d-1$ and length bounded by s_i (the size of g_i), computing the same function as g_is. Place the node corresponding to f with two children. At one child, place the formula g_1'; at the other, place a series of no-op (*i.e.* $\times 1$ or $+0$) gates till the last level of g_1'. Then give the last no-op gate two children, place g_2' at one child, and so on. The width of the new formula Φ' thus obtained is bounded by $\max_i \mathsf{width}(g_i') + 1$, and its length is bounded by $\sum_i \mathsf{length}(g_i') + 1 \leq \sum_i s_i + 1 \leq s$. Note that in this process, for any gate g in Φ the variables it operates on are not changed in the new formula Φ', that is, the only new gates which are introduced in Φ' are the no-op gates which are used for staggering, which only multiply by the constant 1. Thus if Φ is syntactic multilinear then so is Φ. $\diamond$

From Lemma 5.18 and Theorem 5.16, we have the following equivalence.

Corollary 5.19 *Over any ring $\mathbb{K}$,*
sm-VP$_e$ = sm-VLWF= sm-NC^1 = sm-Formula-Depth,Size(log, poly).

With a careful analysis, it can be seen that the constructions in lemmas 4.22 and 4.23 also preserve syntactic multilinearity. Hence:

Corollary 5.20 sm-VNC^1 = sm-VrGP*;*

We summarize these relationships in Figure 5.3

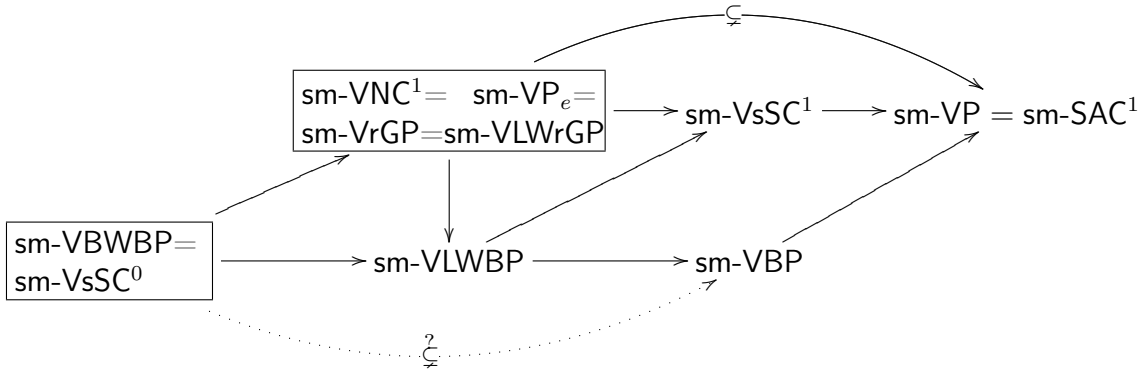


Figure 5.3: Relationship among syntactic multilinear classes

5.6 Conclusion and open questions

In this chapter we have studied the relationships among syntactic multilinear arithmetic circuit classes. In the syntactic multilinear world the relationship $\mathsf{VBWBP} = \mathsf{VNC}^1 \subseteq \mathsf{VsSC}^0$ gets reversed, *i.e.* $\mathsf{sm\text{-}VBWBP} = \mathsf{sm\text{-}VsSC}^0 \subseteq \mathsf{sm\text{-}VNC}^1$. Except the simulation from arithmetic formula to constant width branching programs ([18]) all the equivalences translate into the multilinear world. We will see more on the limitations of simulation of [18] in the next chapter.

We have, $\mathsf{sm\text{-}VsSC}^0 = \mathsf{sm\text{-}VBWBP} \subseteq \mathsf{sm\text{-}VNC}^1 \subseteq \mathsf{sm\text{-}VLWBP} \subseteq \mathsf{sm\text{-}VBP}$, can any one these containments shown to be strict? To separate $\mathsf{sm\text{-}VNC}^1$ from sm-VBP it would be sufficient to show that the full rank polynomial of [72] can be computed by syntactic multilinear ABPs. The separation of sm-VBWBP from sm-VBP would also be interesting though this will be a slightly weaker result than separating $\mathsf{sm\text{-}VNC}^1$ from sm-VBP. One of the reason why separation of sm-VBWBP and sm-VBP would be interesting and possible is that they are defined over the same model, *i.e.* algebraic branching programs. Also, as the result of [71, 72] can be seen as separation of constant + fan-in circuits from polynomial fan-in circuits at logarithmic depth and polynomial size, separating sm-VBWBP from sm-VBP can be viewed as separating constant width from polynomial width in ABPs of polynomial size.

Chapter 6

Limitations

6.1 Introduction

In the last chapter we have seen that the class of syntactic multilinear circuits and ABPs are the same when restricted to constant size and are contained in syntactic multilinear formula, *i.e.* $\mathsf{sm\text{-}VsSC}^0 = \mathsf{sm\text{-}VBWBP} \subseteq \mathsf{sm\text{-}VNC}^1$. The only way known for transforming a polynomial size formula to a constant width ABP is via the three register simulation technique of Ben-Or and Cleve ([18]). As can be seen from the following example, Ben-Or and Cleve's simulation (B-C simulation in short) does not preserve syntactic multilinearity.

Example 6.1 Consider the formula $(x \times y) \times z$. Figure 6.1 shows the ABP, that results from applying the simulation of [18]. It can be seen that any nested multiplication in the formula leads to violation of multilinearity at some of the nodes (node u in Figure 6.1 computes $p(u) = xy^2z$).

$\diamond$

One immediate thought would be: is it possible to preserve syntactic multilinearity if we allow more registers in the B-C simulation? We consider what we call a "generalized B-C simulation" wherein more registers are allowed and an arbitrary strategy for choosing the source, target and dummy registers are allowed. However the simulation steps must be as per the original B-C algorithm. In Section 6.2, we exhibit a syntactic multilinear formula of size s, for which $\Omega(s)$ many registers are necessary if the resulting branching program is to be syntactic multilinear. So, for

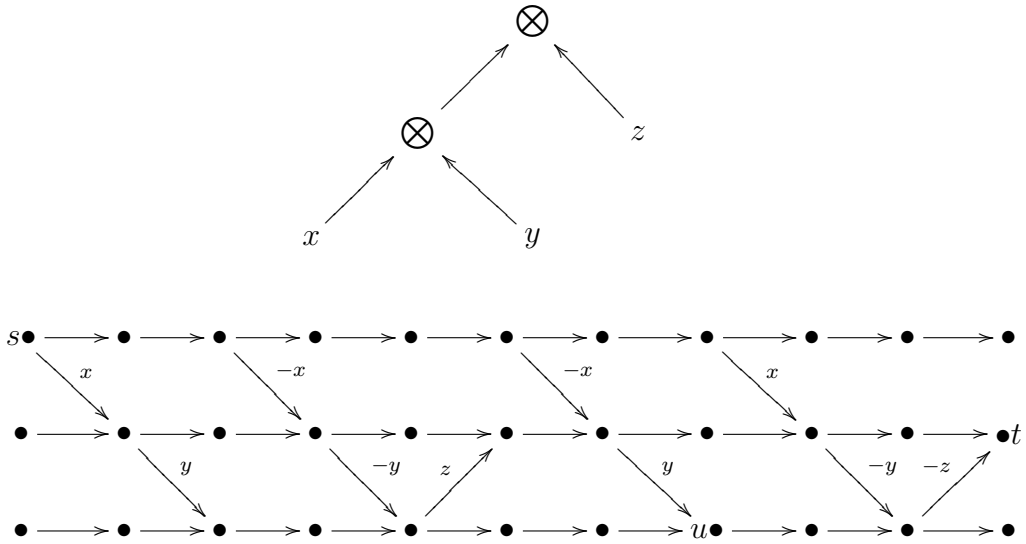


Figure 6.1: An example where the Ben-Or and Cleve construction does not preserve multilinearity as $p(u) = xy^2z$

showing that $\mathsf{sm\text{-}VNC}^1 \subseteq \mathsf{sm\text{-}VBWBP}$ one needs to come up with a new simulation technique altogether.

In the second part of the chapter we explore the expressive power of exponential sums of skew-formula. It is well known that $\mathsf{NP} = \exists \cdot \mathsf{AC}^0$. An analogous result in Valiants' model is also known: $\mathsf{VNP} = \sum \cdot \mathsf{VP} = \sum \cdot \mathsf{VP}_e$ ([92]) (We will define this formally in Section 6.3). In other words, permanent can be written as a sum of exponentially many instantiations of a polynomial in VNC^1. One could ask: what about the determinant? Can the formula be restricted further? As determinant is complete for VP_{skew}, we consider skew formula. *i.e.* can polynomials computed by skew circuits (i.e, ABPs) be written as exponential sum of a polynomial computed by a skew formula? We give a negative answer to this in Section 6.3. Essentially, we do so by observing that skew formula can be exactly characterized by sparse polynomials and show that $\sum \cdot \mathsf{VSkewF} \subseteq \mathsf{VSkewF}$. Hence determinant cannot be expressed as a sum of exponentially many instantiations of a polynomial computed by a skew formula.

Chapter organization: In Section 6.2.1 we introduce the notion of generalized B-C simulation and in section 6.2.2 show existence of formulas that require linear many registers. In Section 6.3 we discuss the limitations of skew formula.

6.2 Limitations of Ben-Or and Cleve's simulation

6.2.1 Generalized B-C simulation

We consider the following computation model of linear straightt line programs ([18]): Let $R_1, \ldots . R_k$ be k registers, which can contain values from $\mathbb{K}$. The operations allowed are, $R_i \leftarrow R_i + xR_j$ and $R_i \leftarrow R_i - xR_j$, where x can be either an input variable or a value from $\mathbb{K}$. Ben-Or and Cleve gave a 3 register simulation for arithmetic formula(see [18]). The size of the program is exponential in the depth of the formula. Since an arithmetic formula can be balanced to get logarithmic depth ([22]), the resulting program is of polynomial size. In the following we recall the simulation of [18]. The operation "offsetting R_j by $f \cdot R_i$" means at the end of the operation, the register R_j contains $R_j \pm f \cdot R_i$ and all other registers remain unchanged.

Case 1: $f = f_D \pm f_E$

Goal: Offset R_j by $\pm f \cdot R_i$.

Steps: Recursively do the following,

- Offset R_j by $\pm f_D \cdot R_i$
- Offset R_j by $\pm f_E \cdot R_i$

Case 2: $f = f_D \times f_E$

Goal: Offset R_j by $\pm f \cdot R_i$.

Steps: Let $k \in \{1, 2, 3\} \setminus \{i, j\}$. Recursively do the following,

- Offset R_k by $\pm f_D \cdot R_i$
- Offset R_j by $\pm f_E \cdot R_k$
- Offset R_k by $\mp f_D \cdot R_i$
- Offset R_j by $\mp f_E \cdot R_k$.

In the above, since the sub-formula f_E and f_D have depth one less than that of f, the construction follows by induction.

Suppose the given formula F is *syntactically multilinear*. Can it be that the corresponding ABP obtained through the above simulation is syntactically multilinear? As observed in Example 6.1, this is not the case.

This observation throws the following question: Suppose we allow more than 3 registers in the above procedure, then what is the minimum number of such extra registers that are required so that the resulting ABP is syntactic multilinear?

We give a generalized framework for the above simulation technique of Ben-Or and Cleve allowing more than 3 registers. Let $I = \{R_1, \ldots, R_k\}$ be the set of registers available. The *generalized B-C simulation* is the same as the one mentioned above except that in the case of a multiplication gate, the choice of the "dummy" register R_k can be made from the set $I \setminus \{i, j\}$. The strategy for choosing such a dummy register can be arbitrary.

For a multiplication gate v in the given formula F, let $b(v)$ denote the set of all "dummy" (i.e the registers R_k) registers that are allocated to v during the generalized B-C simulation. For a node v in F, let X_v denote the set of input variables that appear in the sub-formula rooted at v. Let u and v be any two multiplication nodes in F. The following lemma states the condition on $b(v)$ and $b(u)$ in a syntactic multilinearity preserving generalized B-C simulation:

Lemma 6.2 *Let u and v be two multiplication gates. Let $u = u_1 \times u_2$ and $v = v_1 \times v_2$ and suppose $X_{u_i} \cap X_{v_j} \neq \emptyset$, $\forall i, j \in \{1, 2\}$. In any syntactic multilinearity preserving generalized B-C simulation of F, $b(u) \cap b(v) = \emptyset$.*

Proof. Suppose $R_k \in b(u) \cap b(v)$. Without loss of generality, suppose that the first occurrence of simulation of gate u is before that of v. There will be a path (or a set of paths) reaching R_k containing the variables from X_{u_i} (for some $i \in \{1, 2\}$). In the evaluation of v one of the functions v_j will be multiplied to R_k. Since $X_{v_j} \cap X_{u_i} \neq \emptyset$, this violates the syntactic multilinearity condition. Hence we conclude that $b(u) \cap b(v) = \emptyset$. $\diamond$

Example 6.3 First of all we construct a simple formula, which requires at least 4 registers: For $d \geq 0$, define a formula F_d of depth $2k$ with $n = 2^k$ variables as follows,

$$\begin{aligned} F_d(x_1, \ldots, x_n) &= F_{d-1}(x_1, \ldots, x_{n/2}) \times F_{d-1}(x_{n/2+1}, \ldots, x_n) + \\ &\quad F_{d-1}(x_1, \ldots, x_{n/4}, x_{n/2+1}, \ldots, x_{3n/4}) \times \\ &\quad F_{d-1}(x_{n/4+1}, \ldots, x_{n/2}, x_{3n/4+1}, \ldots, x_n) \\ F_0(x) &= x \end{aligned}$$

It is easy to see that the above formula is syntactically multilinear. Now, it follows from the Lemma 6.2 that a three register simulation cannot be syntactic multilinear. ◇

6.2.2 Combinatorial Designs

Let S be the set of all the multiplication gates in the given formula F. A subset $T \subseteq S$ is said to be "good" if $\forall u = u_1 \times u_2, v = v_1 \times v_2 \in T, \forall i, j \in \{1, 2\}, X_{u_i} \cap X_{v_j} \neq \emptyset$. If F has a good T of size ℓ, then we get a lower bound of ℓ registers for the generalized B-C simulation. So the goal now is to construct a non-trivial syntactically multilinear formula which has at least one "good" T of non-constant size.

The set T as described in the previous subsection is in fact a combinatorial design (See [49] for a more detailed study on combinatorial designs) of a certain kind. We first define this formally and then show an explicit construction of such a design which is due to Srikanth Srinivasan.

Definition 6.4 *Let S be a universe of n elements. $T = \{(A_i, B_i\}_{i=1}^r$ is called an (n, ℓ, r) "design" if $\forall\ 1 \leq i \leq r$, $|A_i| = |B_i| = \ell, A_i \cap B_i = \emptyset$ and $\forall i \neq j$, $A_i \cap A_j \neq \emptyset$, $B_i \cap B_j \neq \emptyset$ and $A_i \cap B_j \neq \emptyset$.*

Let $T = \{A_i, B_i\}_{i=1}^r$ be an (n, ℓ, r) design. Let F_T be a formula defined as follows:

$$F_T = \sum_{i=1}^{r} \phi_{A_i} \times \phi_{B_i}$$

where $\phi_A = \prod_{i \in A} x_i$. It is easy to see that F_T has size $2r + 2r\ell$. Now from Lemma 6.2 it follows that the number of registers required for generalized B-C simulation is at least r.

Clearly, the set $\{(A, \bar{A}) | A \subseteq S, |A| = n/2\}$ is a $(n, n/2, r)$ design for $r \leq \binom{n}{n/2}$. This proves the following,

Theorem 6.5 *For any $0 < s(n) < \binom{n}{n/2}$, there exists a syntactic multilinear formula of size $2s(n) + 2s(n) \cdot n/2$ such that the generalized B-C simulation requires at least $s(n)/2$ registers in order to preserve syntactic multilinearity.*

6.2.3 A randomized construction

Though the construction in the previous section proves an $\Omega(s)$ lower bound on the number of registers for the syntactic multilinearity preserving B-C simulation, it can easily be seen that the hard instances in fact have width 3 branching programs which are syntactic multilinear. Our main goal is to search for hard instances which are not known to have constant width syntactic multilinear branching programs.

Though we could not explicitly construct such a formula, we show a probabilistic existence of a formula on which the generalized B-C simulation requires sub-linear number of registers. We prove the following theorem.

Theorem 6.6 *For every $n > 0$, where n is a power of 2, there exists a formula F, of size $O(n^{48})$ for which the generalized B-C simulation requires at least n^2 registers.*

Construction: Let $F = (V(F), E(F))$ is a complete binary tree of depth $32 \log n$ with the following properties:

- Root node is labeled by $+$;
- Layers are labeled alternatively as $\times$ and $+$ nodes; and
- leaf nodes of F are left unlabeled.

Let $X = \{x_1, \ldots, x_N\}$ be a set of variables, where $N = n^{16}$. Without loss of generality assume that n is a power of 2.

Let $\phi : V(F) \to 2^{\{x_1,\ldots,x_N\}}$ be an assignment of variables to nodes of F with the following properties:

- $\phi(Root) = X$;
- $\phi(f + g) = \phi(f) = \phi(g)$; and
- $\phi(f \times g)$ is partitioned into $\phi(f)$ and $\phi(g)$.

Any assignment ϕ as above defines an arithmetic formula F_ϕ where each leaf l of F is replaced by $\prod_{x \in \phi(l)} x$.

For a node $A \in V(F)$, define $\times$-depth (respectively $+$-depth) of v as the number of $\times$ (respectively $+$) nodes in the unique path from root node to A. For two nodes $A, B \in V(F)$, let $LCA(A, B)$ denote the least common ancestor of A and B in T.

Let T be a subset of nodes of F with the following properties:

- All nodes of T are labeled as $\times$ and are of multiplication depth exactly $8 \log n + 1$
- For every $A, B \in T, A \neq B$, $C_{AB} \triangleq LCA(A, B))$ is a $+$ node and the $\times$-depth of C_{AB} is in the range $[2 \log n, 4 \log n]$.

We need the following combinatorial lemma:

Lemma 6.7 *There exists a T as above with $|T| = n^2$.*

Proof. Consider a $+$ node v of F at $\times$-depth $2 \log n$. Let S be the set of $+$ nodes of F at $\times$-depth $4 \log n$ that are descendants of v. Using a bottom up construction we have a $S' \subset S$ with $|S'| \geq 2^{(2 \log n)-1} = n^2/2$ such that for every $A \neq B \in S'$, $LCA(A, B)$ is a $+$ gate. See figure 6.2 for a demonstration of this. Moreover, $LCA(A, B)$ is within the sub-tree rooted at v. We define T' as follows: For every $u \in S'$, T' contains exactly two $\times$ nodes at $\times$ depth $8 \log n + 1$ one contained in the left sub-tree of u and one in the right sub-tree of u . Clearly $|T'| = 2|S'| \geq n^2$. The properties 1) and 2) are satisfied by the definition of T'. We get the required T by removing some nodes from T' so that $|T| = n^2$. $\diamond$

Let T be the subset of nodes of F as obtained by lemma 6.7. Let $r = 8 \log n$. Then, all the nodes in T are at $\times$-depth $r + 1$. For two nodes $A = A_1 \times A_2$ and $B = B_1 \times B_2$ in T, we say that the pair $\{A, B\}$ is "good" with respect to ϕ if $\forall i, j \in \{1, 2\}$, $\phi(A_i) \cap \phi(B_j) \neq \emptyset$ and "bad" otherwise. We say that T is "good" with respect to ϕ if $\forall A, B \in T$, the pair $\{A, B\}$ is good, and T is bad otherwise.

From Lemma 6.2, if T is good with respect to ϕ then syntactic multilinearity preserving generalized B-C simulation of F_ϕ requires $\Omega(|T|)$ registers.

We show that such an assignment ϕ exists using probabilistic method. First we define a random assignment ϕ which satisfies the conditions in the preceding discussion. The root node R of F, set $\phi(R) = X$. For a node v suppose $\phi(v)$ is already defined. There are two cases:

case 1: $v = v_1 + v_2$.Then, set $\phi(v_1) = \phi(v_2) = \phi(v)$.

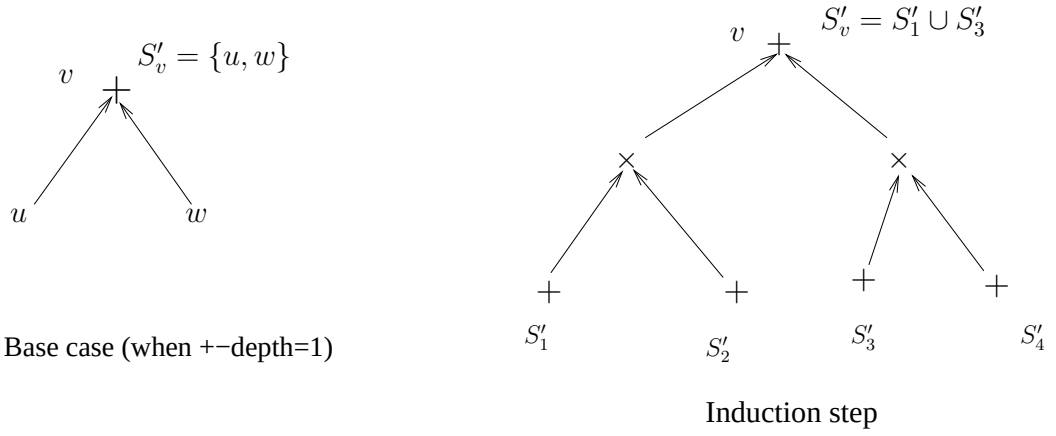


Figure 6.2: A bottom up construction of S'_v

case 2: $v = v_1 \times v_2$. For all $x \in \phi(v)$, put x into $\phi(v_1)$ with probability 1/2 and to $\phi(v_2)$ otherwise.

Now, it suffices to show that the over the choice of ϕ, the probability of T being bad is less than 1. We prove:

Lemma 6.8 *Let F, T and ϕ be as defined above. Then $Pr_\phi[$ T is bad$] < 1$.*

Assuming the above Lemma, we are now ready to prove the theorem 6.6.

Proof. [of Theorem 6.6] Let F be the formula as in the above discussion. Then, by Lemma 6.8, there exists a ϕ such that T is good with respect to ϕ. We define a formula F_ϕ from F by replacing every leaf node v by the formula $\prod_{x \in \phi(v)} x$. As T is good with respect to ϕ, from lemma 6.2, any syntactic multilinearity preserving generalized B-C simulation will require at least $|T|$ registers. Now the theorem follows. ◇

It remains to prove Lemma 6.8.

Proof. [of Lemma 6.8]

We define the following quantities:

$$\begin{aligned} m_{AB} &= |\phi(C_{AB})| \quad \text{(recall that } C_{AB} = LCA(A,B)) \\ r &= 8 \log n \\ r_{AB} &= \times\text{-depth of } C_{AB}; \quad r_{AB} \in [2 \log n + 1, 4 \log n] \end{aligned}$$

$$
\begin{aligned}
\ell_{AB} &= r - r_{AB} = \#\ \times \text{ gates between } C_{AB} \text{ and } A \text{ or } B\ ;\ \ell_{AB} \in [4\log n, 6\log n] \\
L_1 &= \frac{2}{5}\frac{n^{16}}{2^{r_{AB}}}; \quad L_2 = \frac{8}{5}\frac{n^{16}}{2^{r_{AB}}} \\
Bad_{AB} &= Pr_\phi[\{A,B\} \text{ is bad with respect to } \phi] \\
Bad'_{AB} &= Pr_\phi\left[\{A,B\}\text{is bad with respect to } \phi \mid m_{AB} \in [L_1, L_2]\right] \\
Empty_{AB} &= Pr_\phi\left[\phi(A) = \emptyset \text{ or } \phi(B) = \emptyset \mid m_{AB} \in [L_1, L_2]\right] \\
Bad''_{AB} &= Pr_\phi\left[\{A,B\} \text{ is bad w.r.t } \phi \mid m_{AB} \in [L_1, L_2],\ \ \phi(A), \phi(B) \neq \emptyset\right] \\
Bad''_{ABk} &= Pr_\phi\left[\{A,B\} \text{ is bad w.r.t } \phi \;\middle|\; \begin{array}{l} m_{AB} \in [L_1, L_2] \\ \phi(A), \phi(B) \neq \\ \emptyset | \phi(A) \cap \phi(B)| = k \end{array}\right] \\
Size_{ABk} &= Pr_\phi\left[|\phi(A) \cap \phi(B)| = k \;\middle|\; \begin{array}{l} m_{AB} \in [L_1, L_2] \\ \phi(A), \phi(B) \neq \emptyset \end{array}\right]
\end{aligned}
$$

With the above definitions, we have:

$$
\begin{aligned}
Bad''_{AB} &= \sum_{k=0}^{m_{AB}} Bad''_{ABk} \cdot Size_{ABk} \\
&\leq \sum_{k=0}^{n} Size_{ABk} + \sum_{k=n+1}^{m_{AB}} Bad''_{ABk} \qquad (6.1)
\end{aligned}
$$

$$
\begin{aligned}
Bad'_{AB} &\leq Bad''_{AB} + Empty_{AB} \qquad (6.2) \\
Bad_{AB} &\leq Bad'_{AB} + Pr_\phi\left[m_{AB} \notin [L_1, L_2]\right] \qquad (6.3) \\
Pr_\phi\left[\text{T is bad with respect to } \phi\right] &\leq \sum_{A,B \in T} Bad_{AB} \qquad (6.4)
\end{aligned}
$$

It is sufficient to prove : $\forall\ A, B \in T\ Bad_{Ab} = o(\frac{1}{|T|^2}) = o(n^{-1/4})$. It will then follow that $Pr_\phi[\text{T bad with respect } \phi] < 1$. In the following we show that each of the quantities $Pr_\phi[m_{AB} \notin [L_1, L_2]]$, $Empty_{AB}$ and Bad''_{AB} is bounded by $o(1/n^4)$.

1. Consider $Pr_\phi[m_{AB} \notin [L_1, L_2]]$. Define random variables Y_i as $Y_i = 1$ if $x_i \in \phi(C_{AB})$ and 0 otherwise. Then, $m_{AB} = \sum_{i=1}^{n^{16}} Y_i$. Also, we have, $E[Y_i] = 1/2^{r_{AB}}$. Hence the expected value of m_{AB}, $E[m_{AB}] = n^{16}/2^{r_{AB}}$ Applying

Chernoff's inequality we have:
$Pr_\phi[|m_{AB} - E[m_{AB}]| \geq (3/5)E[m_{AB}]] \leq 2 \cdot e^{-(9/100)E[m_{AB}]}$ *i.e.*

$$Pr_\phi[m_{AB}] \notin [L_1, L_2] \leq 2 \cdot e^{-9/100n^{16}/2^{r_{AB}}} = o(1/n^4)$$

2. Each $x_j \in \phi(C_{AB})$ gets into $\phi(A)$ (or $\phi(B)$) with probability $1/2^{\ell_{AB}}$. Hence,

$$\begin{aligned} Empty_{AB} &\leq 2 \cdot (1 - 1/2^{\ell_{AB}})^{m_{AB}} \\ &\leq 2(1 - 1/2^{6 \log n})^{L_1} \quad \because \ \ell_{AB} \leq 6 \log n \\ &= 2(1 - 1/n^6)^{(2/5)n^{12}} \quad \because L_1 \geq (2/5)n^{16}/n^4 \\ &\leq 2e^{-(2/5)n^6} = o(1/n^4) \end{aligned}$$

3. To compute $Size_{AB_k}$. For any $x \in \phi(C_{AB})$ and for $S \in \{A, B\}$, $Pr_\phi[x \in \phi(S)] = (1/2)^{\ell_{AB}}$. As C_{AB} is labeled $+$, we have $Pr_\phi[x \in \phi(A) \cap \phi(B)] = (1/2)^{2\ell_{AB}}$. As random choices of x's are independent of each other, for $V \subseteq \phi(C_{AB})$ with $|V| = k$. Let

$$Eq^V_{AB} \triangleq Pr_\phi \left[V = \phi(A) \cap \phi(B) \mid m_{AB} \in [L_1, L_2], \phi(A), \phi(B) \neq \emptyset \ \right]$$

Then we have,

$$\begin{aligned} Eq^V_{AB} &= Pr_\phi \left[\begin{array}{l|l} \forall x \in V, \ x \in \phi(A) \cap \phi(B) \text{ and} & m_{AB} \in [L_1, L_2] \\ \forall x \in \phi(C_{AB}) \backslash V, \ x \notin \phi(A) \cap \phi(B) & \phi(A), \phi(B) \neq \emptyset \end{array} \right] \\ &= p^k (1-p)^{m_{AB}-k}; \quad \text{where } p = (1/2)^{2\ell_{AB}} \end{aligned}$$

Now,

$$\begin{aligned} Size_{A_{ABk}} &= \sum_{V \subseteq \phi(C_{AB}), |V|=k} Eq^V_{AB} \\ &\leq \binom{m_{AB}}{k} p^k (1-p)^{m_{AB}-k} \end{aligned}$$

Therefore,

$$\sum_{k=0}^{n} Size_{A_{ABk}} \leq \sum_{k=0}^{n} \binom{m_{AB}}{k} p^k (1-p)^{m_{AB}-k}$$

The above quantity can be upper bounded by $n2^{n \log m_{AB}} e^{-\frac{m_{AB}}{2^{2\ell_{AB}}}} = o(1/n^4)$ for $m_{AB} \in [L_1, L_2]$ and the choice of ℓ_{AB}.

4. Let $A = A_1 \times A_2$ and $B = B_1 \times B_2$. Suppose that $|\phi(A) \cap \phi(B)| = k$. Recall that the pair $\{A, B\}$ is bad if for some A_i and B_j, $\phi(A_i) \cap \phi(B_j) = \emptyset$. Fix $i, j \in \{1, 2\}$. Then $\phi(A_i) \cap \phi(B_j) = \emptyset$ happens when all the k elements of $\phi(A) \cap \phi(B)$ gets out of at least one of A_i and B_j. This happens with the probability at most $(3/4)^k$. Since the pair i, j can be chosen in 4 different ways, we have

$$Bad''_{ABk} \leq 4(3/4)^k$$

Hence,

$$\sum_{k=n+1}^{m_{AB}} Bad''_{ABk} \leq 16(3/4)^{n+1} = o(1/n^4)$$

With above arguments we have shown that $Bad_{AB} \leq o(1/n^4)$ as required. $\diamond$

6.3 Skew formula

In this section we consider the expressive power of exponential sums of polynomials computed by skew formula. Firstly, let us define exponential sum:

Definition 6.9 *Let $\mathcal{C}$ be an algebraic complexity class in Valiants' model. $\sum \cdot \mathcal{C}$ is the set of families of polynomials $(f_n)_{n \geq 0}$ such that there exists a polynomial family $(g_m)_{m \geq 0}$ in $\mathcal{C}$ with $f_n(X) = \sum_{e \in \{0,1\}^{m'}} g_{m'+n}(X, e)$, where $m' \leq \mathsf{poly}(n)$. In this notation, $\mathsf{VNP} = \sum \cdot \mathsf{VP}$.*

It is well known that the complexity class NP is equivalent to $\exists \cdot \mathsf{P}$ and in fact even to $\exists \cdot \mathsf{F}$. A similar result holds in the case of Valiant's algebraic complexity classes too. Valiant has shown that $\mathsf{VNP} = \sum \cdot \mathsf{VF}$ (see [25, 23]), and thus the polynomial g in the expression above can be assumed to be computable by a formula of polynomial size and polynomial degree.

Noting that VNP is the class of polynomials which are projection equivalent to the "permanent" polynomial, a natural question arises about the polynomials which are equivalent to the determinant polynomial. Since the determinant exactly characterizes the class of polynomials which are computable by skew arithmetic circuits ([84]), the question one could ask is: can the determinant be written as an exponential sum of partial instantiations of a polynomial that can be computed by *skew formula* of poly size, VSkewF? Recall that a circuit is said to be skew if every $\times$ (or $\wedge$ in the boolean case) gate has at most one child that is not a circuit input. Skew circuits are essentially equivalent to branching programs. Thus one could ask the related question: since $\mathsf{VP} \subseteq \sum \cdot \mathsf{VP} = \sum \cdot \mathsf{VP}_e$, can we show that $\mathsf{VP}_{skew} \subseteq \sum \cdot \mathsf{VSkewF}$?

We show that this is not possible. We first give an equivalent characterization of VSkewF in terms of "sparse polynomials" (Lemma 6.10) placing it inside VAC^0, and then use it to show that $\sum \cdot \mathsf{VSkewF}$ is in fact contained in VSkewF (Theorem 6.12).

6.3.1 A characterization of VSkewF

Lemma 6.10 *Let $f \in \mathbb{K}[X]$ be computed by a skew formula Φ of size s. Then the degree and number of monomials in f are bounded by s.*
Conversely, if $f \in \mathbb{K}[X]$ is a degree d polynomial, where at most t monomials have non-zero coefficients, then f can be computed by a skew formula Φ of size $O(td)$.

Proof. Let F be a skew formula of size s. Consider a sub-tree T of F such that root of F is in T and for any gate g in T, if g is a $+$ gate then exactly one child of g is in T and if g is a $\times$ gate then both children of g are present in T. We call such a subtree T a "proving subtree" of F. Since F is skew, T looks like a path, with edges hanging out at nodes labeled $\times$. But in a tree, the number of root to leaf paths is bounded by the number of leaves in the tree. Thus the number of distinct proving subtrees of F is upper bounded by s. Let $p_F \in \mathbb{K}[X]$ be the polynomial computed by the formula F, where X is the set of input variables of F. It is easy to see that a proving subtree in F corresponds to a monomial in p_F (monomial with some value from $\mathbb{K}$ as coefficient). Thus the number of non-zero monomials in p_F is bounded by s. Since the degree of the monomial contributed by such a path is at most the length of the path, the degree of p_F is at most s.

On the other hand, if a polynomial $p \in \mathbb{K}[X]$ has t non-zero monomials $m_1, \ldots, m_t$, then we can explicitly multiply variables to get each monomial m_i and finally get the sum $\sum_i c_i m_i$, where $c_i \in \mathbb{K}$ is the coefficient of m_i in p. This formula computes p in size $O(td)$. $\diamond$

Corollary 6.11 $\mathsf{VSAC}^0 \subset \mathsf{VSkewF} \subset \mathsf{VAC}^0$.

Proof. The containments follow directly from Lemma 6.10. To see why they are proper: (1) Even over the Boolean setting, the function $\oplus_{i=1}^{\log n} x_i$ is in SkewF but not in SAC^0. Any Boolean function sensitive to only $O(\log n)$ of its n inputs is in SkewF. Functions computed by a VSAC^0 circuit have $O(1)$ degree, and so cannot equal the class of poly-degree poly-support polynomials VSkewF. (2) The function $\prod_{i=1}^{n}(x_i + y_i)$ is in VAC^0 but not in VSkewF because it has too many monomials. $\diamond$

6.3.2 An upper bound for $\sum.\mathsf{VSkewF}$

Theorem 6.12 *Let $f \in \mathbb{K}[X]$ be expressible as $f(X) = \sum_{e \in \{0,1\}^m} \phi(X, e)$, where ϕ has a poly size* skew *formula and $m \leq \mathsf{poly}(n)$. Then $f \in \mathsf{VSkewF}$.*
In other words, $\sum \cdot \mathsf{VSkewF} \subseteq \mathsf{VSkewF}$.

Proof. Since $\phi(X, Y)$ (where $X = X_1, \ldots, X_n$ and $Y = Y_1, \ldots, Y_m$) has a poly size skew formula, by Lemma 6.10 we know that the number of non-zero monomials in ϕ is bounded by some polynomial $q(n, m)$. Hence the number of non-zero monomials in $\phi(X, Y)|_X$ (*i.e.* , monomials in X with coefficients from $\mathbb{Z}[Y]$) and hence in $f(X)$, is also bounded by $q(n, m)$.

For any $\alpha \in \mathbb{N}^n$, consider the monomial $X^\alpha = \prod_{\alpha_i} X_i^{\alpha_i}$, and define the set S_α as

$$S_\alpha = \{\beta \in \{0,1\}^m \mid X^\alpha Y^\beta \text{ has a non-zero coefficient } a_{\alpha,\beta} \text{ in } \phi\}$$

Clearly, for each α, we have $|S_\alpha| \leq q(n, m)$.

Since $\phi(X, Y)$ is evaluated only at Boolean settings of Y, we can assume, without loss of generality, that it is multilinear in Y. So it can be written as

$$\phi(X, Y) = \sum_{\alpha \in \mathbb{N}^n} \sum_{\beta \in \{0,1\}^m} a_{\alpha,\beta} X^\alpha Y^\beta$$

Hence we have the following:

$$\begin{aligned} f(X) &= \sum_{e\in\{0,1\}^m} \sum_{\alpha\in\mathbb{N}^n} \sum_{\beta\in\{0,1\}^m} a_{\alpha,\beta} X^\alpha e^\beta \\ &= \sum_{\alpha\in\mathbb{N}^n} \left(X^\alpha \sum_{\beta\in S_\alpha} \left[a_{\alpha,\beta} \sum_{e\in\{0,1\}^m} e^\beta \right] \right) \end{aligned}$$

Therefore,

$$\begin{aligned} f(X) &= \sum_{\alpha\in\mathbb{N}^n} \left(X^\alpha \sum_{\beta\in S_\alpha} a_{\alpha,\beta} 2^{m-l_\beta} \right) \\ &\quad \text{where } l_\beta = \text{number of 1's in the bit vector } \beta \in \{0,1\}^m. \end{aligned}$$

Then the coefficient c_α of X^α in $f(X)$ is given by $\sum_{\beta\in S_\alpha} a_{\alpha,\beta} 2^{m-l_\beta}$. Now by hardwiring these coefficients along with X^αs (note that there are only polynomially many such αs) it is easy to see that $f(X)$ can be computed by a skew formula.

$\diamond$

Thus, it is not possible to express the determinant polynomial in $\sum$.VSkewF since it has exponentially many monomials.

6.3.3 Multilinear Versions

Here we consider the multilinear versions of the skew formula. From Lemma 6.10, we know that VSkewF is characterized by polynomials with polynomial many coefficients. The construction yields, for any multilinear polynomial computed by a skew formula, an equivalent skew formula which is syntactic multilinear. Hence the notion of multilinearity and syntactic multilinearity are the same for skew formula.

Since any multilinear polynomial that can be computed by a VSAC^0 circuit has a small number of monomials, the containments and separations of corollary 6.11 hold in the syntactic multilinear case too. Also, note that the polynomial $\prod_i (x_i + y_i)$ is multilinear, and can be computed by a $\mathsf{sm\text{-}AC}^0$ circuit.

Corollary 6.13 $\mathsf{sm\text{-}SAC}^0 \subset \mathsf{sm\text{-}VSkewF} \subset \mathsf{sm\text{-}AC}^0$

6.4 Conclusion and open questions

We introduced a notion of "generalized B-C" simulation, that allows the B-C simulation to use more registers in order to give a syntactic multilinear preserving simulation of arithmetic formula. However, we exhibited formula of size s that would require $\Omega(s)$ registers to achieve this. However these results does not imply that sm-VBWBP is strictly contained inside sm-VNC^1 as there could be other possible ways of transforming a sm-formula into a sm-constant width ABP or an sm-circuit of constant width (as sm-VBWBP = sm-VsSC^0).

In a slightly different direction, we have also shown that skew formula have weak expressive power, since they are closed under taking exponential sums.

We conclude the chapter with the following questions

- Are sm-VBWBP and sm-VNC^1 separate? It would also be interesting to extend our bounds to other simulations.
- Is there any explicit polynomial which achieves the bound of Theorem 6.5 that is not known to have a polynomial size constant width syntactic multilinear ABP?

Chapter 7

Small space analogues

7.1 Introduction

Apart from characterizing algebraic computations, an interesting task in algebraic complexity theory is to define algebraic complexity classes analogous to the ones of boolean complexity theory. As we have seen, VP,VNP, VNC, VSC, VsSC and VBP have P,NP, NC, SC, sSC and BP respectively as their boolean counterparts. Perhaps, the prominent boolean complexity classes that do not have algebraic analogues are the space complexity classes.

The main obstacle in this direction is defining a "right" measure for space. Two obvious choices are: 1) the number of arithmetic "cells" or registers used during the course of computation (i.e., the unit-space model), and 2) the size of a succinct description of the polynomials computed at each cell. A third choice is the complexity of computing the coefficient function for polynomials in the family. All three of these space measures have been studied in the literature, [64, 32, 52, 51], with varying degrees of success. In particular, the models of [64, 52, 51] when adapted to logarithmic space are too powerful to give meaningful insights into small-space classes, whereas the model of [32] as defined for log-space is too weak.

The main purpose of this chapter is to propose yet another model for describing space-bounded computations of families of polynomials. Our model is based on the width of arithmetic circuits, and captures both succinctness of coefficients and ease of evaluating the polynomials. We show that our notion of space VSPACE(s) coincides with that of [52, 51] at polynomial space with uniformity (Theorem 7.5),

and so far avoids the pitfalls of being too powerful or too weak at logarithmic space.

Continuing along this approach, we propose a way of describing non-deterministic space-bounded computation in this context. The specific motivation for this is to obtain an algebraic analogue of the class non-deterministic log-space NL as well as an analogue of the result that $\mathsf{VNP} = \Sigma \cdot \mathsf{VP}$. Again, there is a well-known model for NL that easily carries over to the arithmetic setting, namely polynomial-size branching programs BP. But we are unable to compare VBP with our version of VL. Our model here for NL is based on read-once certificates, which also provides the correct description on NL in terms of L in the Boolean world. We show that the arithmetization of this model, $\Sigma^R \cdot \mathsf{VL}$ does contain arithmetic branching programs (Theorem 7.12).

Surprisingly, we are unable to show the converse. In fact, we are unable to show a good upper bound on the complexity of read-once certified log-space polynomial families. This raises the question: Is the read-once certification procedure inherently too powerful? We show that this is not always the case; for branching programs, read-once-certification adds no power at all (Theorem 7.16). Similarly, for polylog-width circuits where the syntactic degree is bounded by a polynomial, read-once certification does not take us beyond VQP (Theorem 7.19). Further, if the circuit is multiplicatively disjoint and of constant width, then read-once certification does not take us beyond VP.

The rest of the chapter is organized as follows: Section 7.2 gives a detailed account of existing notions of space for algebraic computation and introduces circuit width as a possible measure of space. In section 7.3 we introduce the notion of read- once certificates and read-once exponential sums. Section 7.4 contains upper bounds for read-once exponential sums of some restricted circuit classes.

7.2 Notion of space for arithmetic computations?

In the case of boolean computations, the notion of "width" of a circuit captures the notion of space in the Turing machine model (under certain uniformity assumptions). In the case of arithmetic computations, defining a notion of "space bounded computation" seems to be a a hard task.

7.2.1 Previously studied notions

One possible measure for space is the number of arithmetic "cells" or registers used in the course of computation (i.e., the unit-space model). Michaux [64] showed that with this notion of space, any language that is decided by a machine in the Blum-Shub-Smale model of computation (a general model for algebraic computation capturing the idea of computation over reals, [19]; see also [25]) can also be computed using $O(1)$ registers. Hence there is no space-hierarchy theorem under this space measure.

Another possible measure is the size of a succinct description of the polynomials computed at each cell. In [32], Naurois introduced a notion of weak space in the Blum-Shub-Smale model, and introduced the corresponding log space classes $\mathsf{LOGSPACE}_W$and PSPACE_W. This is in fact a way of measuring the complexity of succinctly describing the polynomials computed by or represented at each "real" cell. Though this is a very natural notion of "succinctness" of describing a polynomial, this definition has a few drawbacks:

1. $\mathsf{LOGSPACE}_W$ seems to be too weak to contain even NC^1 over $\mathbb{R}$, which is in contrast to the situation in the Boolean world.

2. The polynomials representable at every cell have to be "sparse", i.e., the number of monomials with non-zero coefficients should be bounded by some polynomial in the number of variables.

The second condition above makes the notion of weak space very restrictive if we adapt the definition to the Valiant's algebraic computation model. This is because the corresponding log-space class in this model will be computing only sparse polynomials, but in the non-uniform setting sparse polynomials are known to be contained in a highly restrictive class called skew formula (see Section 6.3), which is in fact a proper subclass of constant depth arithmetic circuits (i.e., VAC^0).

Koiran and Perifel ([52, 51]) suggested another notion of polynomial space for Valiant's ([91, 25]) classes. The main purpose of their definition was to prove a transfer theorem over $\mathbb{R}$ and $\mathbb{C}$. Under their definition $\mathsf{Uniform\text{-}VPSPACE}$ (the non-uniform counterpart can be defined similarly) is defined as the set of families (f_n) of multivariate polynomials $f_n \in F[x_1, \ldots, x_{u(n)}]$ with integer coefficients such that

- $u(n)$ is bounded by a polynomial in n.
- Size of coefficients of f_n is bounded by $2^{poly(n)}$.
- Degree of f_n is bounded by $2^{poly(n)}$.
- Every bit of the coefficient function of f_n is computable in PSPACE.

In [52], it was observed that the class VPSPACE is equivalent to the class of polynomials computed by arithmetic circuits of polynomial depth and exponential size. Such Boolean circuits compute exactly PSPACE, hence the name VPSPACE. Thus one approach to get reasonable smaller space complexity classes is to generalize this definition. We can consider VSPACE($s(n)$) to consist of families $(f_n)_{n\geq 1}$ of polynomials satisfying the following:

- $f \in \mathbb{Z}[x_1, \ldots, x_{u(n)}]$, where $u(n)$, the number of variables in f_n, is bounded by some polynomial in n.
- Degree of f_n is bounded by $2^{s(n)}$.
- The number of bits required to represent each of the coefficients of f_n is bounded by $2^{s(n)}$, *i.e.* the coefficients of f_n are in the range $[-2^{2^{s(n)}}, 2^{2^{s(n)}}]$
- Given n in unary, an index $i \in [1, 2^{s(n)}]$, and a monomial M, the ith bit of the coefficient of M in f_n is computable in DSPACE($s(n)$).

It is easy to see that with this definition, even the permanent function PERM_n is in log-space. Thus VSPACE($\log n$) would be too big a class to be an arithmetic version of log-space. The reason here is that this definition, unlike that of [32], goes to the other extreme of considering only the complexity of coefficient functions and ignores the resource needed to compute and add the monomials with non-zero coefficients. The relationship between the complexity of coefficient functions and the polynomials themselves is explored more thoroughly in [60].

7.2.2 Defining VPSPACE in terms of circuit width

In this section we propose width of a (layered) circuit as a possible measure of space for arithmetic computations.

Definition 7.1 *Let* $\mathsf{VWIDTH}(S)$ *(with* $S = S(n)$*) be the class of polynomial families* $(f_n)_{n\geq 0}$ *with the following properties,*

- *The number of variables* $u(n)$ *in* f_n *is bounded by* $\mathsf{poly}(n)$
- $f_n \in \mathbb{Z}[x_1, \ldots, x_{u(n)}]$*, i.e* f_n *has only integer coefficients*
- $\deg(f) \leq \max\{2^{S(n)}, \mathsf{poly}(n)\}$.
- *The coefficients of* f_n *are representable using* $\max\{2^{S(n)}, \mathsf{poly}(n)\}$ *many bits.*
- f_n *is computable by an arithmetic circuit of width* $S(n)$ *and size* $\max\{2^{S(n)}, \mathsf{poly}(n)\}$.

Further, if the arithmetic circuits in the last condition are DSPACE(S)*-uniform, we call the family Uniform-*$\mathsf{VWIDTH}(S)$.

Remark For $i \geq 2$ $\mathsf{VWIDTH}(\log^i n)$ is very close to VSC^i though they are different for the following reasons:

- Polynomials in $\mathsf{VWIDTH}(\log^i n)$ can have degree $O(2^{\log^i n})$, whereas degree of polynomials in VSC^i is bounded by $\mathsf{poly}(n)$.
- Coefficients of $\mathsf{VWIDTH}(\log^i n)$ are integers and their size is bounded by $O(2^{\log^i n})$, whereas VSC^i can have arbitrary coefficients.

However, when $i = 0, 1$ they coincide, if we restrict the polynomials in VSC^0 and VSC^1 to have only integer coefficients. Also, it is easy to see that $\mathsf{VWIDTH}(\log^i n)$ and VsSC^i are different. Apart from the above two, another major difference between them is:

- The circuits in $\mathsf{VWIDTH}(\log^i n)$ need not have bound on their syntactic degree

We show in Theorem 7.5 below that with this definition, uniform $\mathsf{VWIDTH}(\mathsf{poly})$ coincides with uniform $\mathsf{VPSPACE}$ as defined in [52]; thus polynomial width indeed corresponds to polynomial space. Motivated by this equivalence, we define the following complexity classes:

Definition 7.2 $\mathsf{VSPACE}(S(n)) = \mathsf{VWIDTH}(S(n))$
Uniform-$\mathsf{VSPACE}(S(n))$$=$ *Uniform-*$\mathsf{VWIDTH}(S(n))$

We denote the log-space class by VL; thus $\mathsf{VL} = \mathsf{VWIDTH}(\log n) = \mathsf{VSC}^1$.

The following containments and equalities follow directly from known results (from [28] and Chapter 4) about width-constrained arithmetic circuits.

Lemma 7.3 $\mathsf{VBWBP} = \mathsf{VNC}^1 = \mathsf{VP}_e \subseteq \mathsf{VSPACE}(O(1)) = \mathsf{VSC}^0 \subseteq \mathsf{VL} = \mathsf{VSC}^1 \subseteq \mathsf{VP}$

Remark In the above equivalence, we need to extend $\mathsf{VWIDTH}(S(n))$ to include arbitrary constants from $\mathbb{K}$. In this case we omit the bound on the size of the coefficients in definition 7.1.

Thus VL according to this definition is in VP and avoids the trivially "too-powerful" trap; also, it contains VNC^1 and thus avoids the "too weak" trap.

The following closure property is easy to see.

Lemma 7.4 *For every $S(n) > \log n$, the classes $\mathsf{VSPACE}(S(n))$ are closed under polynomially bounded summations and constant many products.*

7.2.3 Comparing VPSPACE and VWIDTH(poly)

This subsection is devoted to proving the following equivalence,

Theorem 7.5 *The class Uniform-*VPSPACE *as defined in [52] coincides with Uniform-*$\mathsf{VWIDTH}(\mathsf{poly})$.

We use the following easy fact:

Fact 7.6 *A d degree polynomial over t variables has at most $\binom{d+t}{t}$ monomials.*

Now, Theorem 7.5 follows from the two lemmas below.

Lemma 7.7 *Uniform-*$\mathsf{VPSPACE} \subseteq$ *Uniform-*$\mathsf{VWIDTH}(\mathsf{poly})$.

Proof. Let $(f_n)_{n\geq 0}$ be a family of polynomials in VPSPACE. Then by definition, the bits of the coefficients of f_n can be computed in PSPACE and hence by exponential size polynomial width circuits. The (exponentially many) bits can be put together with appropriate weights to obtain a circuit computing the coefficient itself. The exponential-degree monomials can each be computed by an exponential-size constant-width circuit. Thus we can use the naive method of computing f_n: expand f_n into individual monomials, compute each coefficient and each monomial, and add them up sequentially. By Fact 7.6, there are only exponentially many distinct monomials. Thus we get a polynomial width exponential-size circuit computing f_n. ◇

The converse direction is a little more tedious, but essentially follows from the Lagrange interpolation formula for multivariate polynomials.

Lemma 7.8 *Uniform*-VWIDTH(poly) $\subseteq$ *Uniform*-VPSPACE.

Proof. Let $(f_n)_{n\geq 0}$ be a family of polynomials in VWIDTH(poly(n)). Let $N = u(n)$ be the number of variables in f_n, and let $q(n)$ be a polynomial such that $2^{q(n)}$ is an upper bound on both $d = \deg(f_n)$ and on the number of bits required to represent each coefficient. Let $w(n) = \mathsf{poly}(n)$ and $s(n) \in 2^{O(n^c)}$ respectively be the width and size of a witnessing circuit C.

To show that $f_n \in$ VPSPACE, we need to give a PSPACE algorithm, which computes coefficient of the monomial $\prod_{k=1}^{N} x_k^{i_k}$, given 1^n and $\langle i_1, \ldots, i_N \rangle$ as input.

We use the following notation: $S = \{0, 1, \ldots, d\}$, $T = S^N$, $\tilde{x} = \langle x_1, \ldots, x_N \rangle$, and for $\tilde{i} = \langle i_1, \ldots, i_N \rangle \in T$, the monomial $m(\tilde{i}) = \prod_{k=1}^{N} x_k^{i_k}$ is denoted $\tilde{x}^{\tilde{i}}$. We drop the subscript n for convenience.

Using Lagrangian interpolation for multivariate polynomials we have

$$f(\tilde{x}) = \sum_{\tilde{i}\in T} f(\tilde{i})\mathsf{Equal}(\tilde{x}, \tilde{i}) \;=\; \sum_{\tilde{i}\in T} f(\tilde{i}) \prod_{k=1}^{N} \mathsf{Equal}(x_k, i_k)$$

$$\text{where } \mathsf{Equal}(x, i) = \prod_{a\in S\setminus\{i\}} \left(\frac{x-a}{i-a}\right) \;=\; \frac{\prod_{a\in S\setminus\{i\}}(x-a)}{i!(d-i)!(-1)^{d-i}}$$

Thus for any $\tilde{t} \in T$, the coefficient of the monomial $m(\tilde{t})$ is given by

$$\mathsf{coeff}(m(\tilde{t})) = \sum_{\tilde{i}\in T} f(\tilde{i}) \prod_{k=1}^{N} \frac{\text{coeff of } x_k^{t_k} \text{ in } \prod_{a\in S\setminus\{i_k\}}(x_k - a)}{i_k!(d-i_k)!(-1)^{d-i_k}}$$

But we have a nice form for the inner numerator:

$$\text{coeff of } x_k^{t_k} \text{ in } \textstyle\prod_{a\in S\setminus\{i\}}(x_k - a) \text{ equals } (-1)^{d-t_k} S_{d,t_k}(0, 1, \ldots, i_k - 1, i_k + 1, \ldots, d)$$

where S_{d,t_k} denotes the elementary symmetric polynomial of degree t_k in d variables.

To compute the desired coefficient in PSPACE, we use the Chinese Remaindering technique; See Lemma 2.11 and [29] for more details. Since symmetric polynomials are easy to compute (*e.g.* [79] or Th 2.5.4 in [90]), and since $f(\tilde{i})$ is computable by a polynomial-width circuit by assumption, a PSPACE algorithm can compute the coefficient modulo a prime p, for any prime p that has an $O(d)$

bit representation. (The algorithm will require $O(w(n)\log p + \log s(n))$ space to evaluate $f(\tilde{i}) \mod p$). Reconstructing the coefficient from its residues modulo all such primes can also be performed in PSPACE. $\diamond$

Lemma 7.8 requires that the VWIDTH family be uniform (with a direct-connection uniformity condition). If the VWIDTH family is non-uniform, this problem cannot be circumvented with polynomial advice, since the circuit has exponential size.

7.3 Read-Once certificates

In general, non-deterministic complexity classes can be defined via existential quantifiers. *e.g.* , $\mathsf{NP} = \exists \cdot P$. In the algebraic setting, we know that the class VNP (algebraic counterpart of NP) is defined as an "exponential" sum of values of a polynomial size arithmetic circuit. *i.e.* , $\mathsf{VNP} = \Sigma \cdot P$. It is also known that $\mathsf{VNP} = \Sigma \cdot \mathsf{VP}_e = \Sigma \cdot \mathsf{VNC}^1$ (see [25]).

If we consider smaller classes, NL is the natural non-deterministic version of L. However to capture it via existential quantifiers, we need to restrict the use of the certificate, since otherwise $\exists \cdot L = \mathsf{NP}$. It is known that with the notion of "read once" certificates (see, *e.g.* , [8], Chapter 4) one can express NL as an existential quantification over L. Analogously, we propose a notion of "read-once" certificates in the context of arithmetic circuits so that we can get meaningful classes by taking exponential sums over classes that are below VP.

Definition 7.9 *Let C be a layered arithmetic circuit with ℓ layers. Let $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_m\}$ be the input variables of C. C is said to be "read-once certified" in Y if the layers of C can be partitioned into m blocks, such that each block reads exactly one variable from Y. That is, C satisfies the following:*

- *There is a fixed permutation $\pi \in S_m$ such that the variables of Y appear in the order $y_{\pi(1)}, \ldots, y_{\pi(m)}$ along any leaf-to-root path.*
- *There exist indices $0 = i_1 \leq \ldots \leq i_m \leq i_{m+1} = \ell$ such that the variable $y_{\pi(j)}$ appears only from layers $i_j + 1$ to i_{j+1}.*

We henceforth without loss of generality, assume that π is the identity permutation.

Now we define the the exponential sum over read-once certified circuits.

Definition 7.10 *Let $\mathcal{C}$ be any arithmetic circuit complexity class. A polynomial family $(f_n)_{n\geq 0}$ is said to be in the class $\Sigma^R \cdot \mathcal{C}$, if there is a family $(g_{m(n)})_{n\geq 0}$ such that $m(n) = n + m'(n)$, $m'(n) \leq \mathsf{poly}(n)$, $f_n(X) = \sum_{Y\in\{0,1\}^{m'(n)}} g_{m(n)}(X,Y)$ and $g_{m(n)}$ can be computed by a circuit of type $\mathcal{C}$ that is read-once certified in Y.*

We also use the term "read once exponential sum" over $\mathcal{C}$ to denote $\Sigma^R \cdot \mathcal{C}$.

For circuits of width polynomial or more, the restriction to read-once certification is immaterial: the circuit can read a variable once and carry its value forward to any desired layer via internal gates. This is equivalent to saying that for a P machine, read-once input is the same as two-way-readable input. Thus

Lemma 7.11 $\Sigma^R \cdot \mathsf{VP} = \Sigma \cdot \mathsf{VP} = \mathsf{VNP}$

Having seen that the read-once certificate definition is general enough for the case of large width circuits, we turn our focus on circuits of smaller width. Once the width of the circuit is substantially smaller than the number of bits in the certificate, the read-once property becomes a real restriction. If this restriction correctly captures non-determinism, we would expect that in analogy to $\mathsf{BP} = \mathsf{NL} = \Sigma^R \cdot \mathsf{L}$, we should be able to show that VBP equals $\Sigma^R \cdot \mathsf{VL}$. In a partial answer, we show in the following theorem one direction: read-once exponential sums over VL are indeed powerful enough to contain VBP.

Theorem 7.12 $\mathsf{VBP} \subseteq \Sigma^R \cdot \mathsf{VL}$.

In order to prove the above theorem, we consider a problem that is complete for VBP. We need the following definition:

Definition 7.13 *A polynomial $f \in \mathbb{K}[X_1, \ldots, X_n]$ is called a projection of g (denoted $f \leq g$), if*

$$f(X_1, \ldots, X_n) = g(a_1, \ldots, a_m)$$

where each $a_i \in \mathbb{K} \cup \{X_1, \ldots, X_n\}$.

Let $f = (f_n)_{n\geq 0}$ and $g = (g_m)_{m\geq 0}$ be two polynomial families. f is said to be projection reducible to g if

$$\exists n_0, \forall n \geq n_0,\ f_n \leq g_{m(n)}$$

where $m(n) \leq \mathsf{poly}(n)$.

Let $(G_n) = (V_n, E_n)$ (with $|V_n| = m = \mathsf{poly}(n)$) be a family of directed acyclic graphs and let $s = 1$ and $t = n$ denote two special nodes in G_n. We assume without loss of generality that the graph is topologically sorted; edges are from i to j only when $i < j$. Let $A = (a_{i,j})_{i,j\in\{1,\ldots,m\}}$ be an $m \times m$ matrix with variable entries, representing edge weights in G_n. For any directed $s-t$ path $P = \langle v_0, v_1, \ldots, v_\ell, v_{\ell+1}\rangle$ in G_n, let M_P denote the monomial that is the product of the variables corresponding to edges in P. Let $PATH_G^n = \sum_P M_P$, where P ranges over all the $s-t$ paths in G_n.

Definition 7.14

$$PATH = \left\{ (PATH_G^n)_{n\geq 0} \;\middle|\; \begin{array}{l} G{=}(G_n)_{n\geq 0} \text{ is a family of layered complete di-} \\ \text{rected acyclic graphs with edges of } G_n \text{ labeled from} \\ \{x_1, \ldots, x_n\} \end{array} \right\}$$

It is easy to see the following:

Proposition 7.15 *(folklore) $PATH$ is complete for* VBP *under projections.*

We prove theorem 7.12 by showing that $PATH_G^n \in \Sigma^R \cdot \mathsf{VL}$ for any layered directed acyclic graph family $G = (G_n)_{n\geq 0}$.

Proof. [of theorem 7.12] Here onwards we drop the index n from G_n.

We define function $h_G(Y, Z) : \{0,1\}^{\lceil \log m\rceil} \times \{0,1\}^{m^2} \to \{0,1\}$ as follows. Assume that the variables in $Y = \{y_1, \ldots, y_k\}$ and $Z = \{z_{1,1}, \ldots, z_{m,m}\}$ take only values from $\{0,1\}$. $h_G(Y,Z) = 1$ if and only if $Z = z_{1,1}, \ldots, z_{m,m}$ represents a directed s-t path in G of length exactly ℓ, where ℓ written in binary is $y_1 \ldots y_k$.Note that s-t paths P in G are in one-to-one correspondence with assignments to Y, Z such that $h_G(Y,Z) = 1$. Hence

$$\begin{aligned} PATH_G^n = \sum_P M_P &= \sum_{Y,Z} h_G(Y,Z) \,[\text{ weight of path specified by } Y,Z] \\ &= \sum_{Y,Z} h_G(Y,Z) \prod_{i,j} (a_{i,j} z_{i,j} + (1 - z_{i,j})) \end{aligned}$$

There is a deterministic log-space algorithm A which computes $h_G(Y,Z)$ when Y, Z is given on a "read once" input tape (see [8]). Let C be the corresponding $O(\log n)$

width boolean circuit. (without loss of generality, assume that all negation gates in C are at the leaves.) Let D the natural arithmetization of C. Since Y and Z are on a read-once input tape, it is easy to see that C, and hence D, are read-once certified in the variables from Y and Z. We can attach, parallel to D, constant-width circuitry that collects factors of the product $\prod_{i,j}(a_{i,j}z_{i,j} + (1 - z_{i,j}))$ as and when the $z_{i,j}$ variables are read, and finally multiplies this with $h_G(Y, Z)$. The resulting circuit remains $O(\log n)$-width, and remains read-once certified on Y, Z. $\diamond$

While we are unable to show the converse, we are also unable to show a reasonable upper bound on $\Sigma^R \cdot \mathsf{VL}$. It is not even clear if $\Sigma^R \cdot \mathsf{VL}$ is contained in VP. One possible interpretation is that the Σ^R operator is too powerful and can lift up small classes unreasonably. We show that this is not the case in general; in particular, it does not lift up VBP and VBWBP.

Theorem 7.16 *1.* $\Sigma^R \cdot \mathsf{VBP} = \mathsf{VBP}$

2. $\Sigma^R \cdot \mathsf{VBWBP} = \mathsf{VBWBP}$

This theorem follows from Lemma 7.18. We need the following notation:

Definition 7.17 *For $f \in \mathbb{K}[X, Y]$ with $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_m\}$, $E_Y(f)$ denotes the exponential sum of $f(X, Y)$ over all Boolean settings of Y. That is,*

$$E_Y(f)(X) = \sum_{e \subseteq \{0,1\}^m} f(X, e)$$

Lemma 7.18 *Let C be a layered skew arithmetic circuit on variables $X \cup Y$. Suppose C is* read-once *certified in Y. Let $w = \mathsf{width}(C)$, $s = \mathsf{size}(C)$ and $\ell =$ the number of layers in C. Let $f_1, \ldots, f_w$ denote the output gates (also the polynomials computed by them) of C. There exists a weakly skew circuit C', of size $O(mw^4 s)$ and width $4w$, that computes all the exponential sums $E_Y(f_1), \ldots, E_Y(f_w)$.*

Proof. We proceed by induction on $m = |Y|$. In the base case when $m = 1$, $E_Y(f_j)(X) = f_j(X, 0) + f_j(X, 1)$. Putting two copies of C next to each other, one with $y = 0$ and the other with $y = 1$ hardwired, and adding corresponding outputs

we get a circuit C' which computes the required function. Clearly $\mathsf{width}(C') \leq 2w$ and $\mathsf{size}(C') \leq 3s + w$.

Assume now that the lemma is true for all skew circuits with $m' = |Y| < m$. Let C be a given circuit where $|Y| = m$. Let Y' denote $Y \setminus \{y_m\} = \{y_1, \ldots, y_{m-1}\}$. As per definition 7.9, the layers of C can be partitioned into m blocks, with the kth block reading only y_k from Y. Let $0 = i_1 \leq i_2 \leq \ldots \leq i_m \leq \ell$ be the layer indices such that y_k is read between layers $i_k + 1$ and i_{k+1}. Let $f_1, \ldots, f_w$ be the output gates of C.

We slice C into two parts: the bottom $m - 1$ blocks of the partition together form the circuit D, and the top block forms the circuit C_m. Let $g_1, \ldots, g_w$ be the output gates of D. These are also the inputs to C_m; we symbolically relabel the non-leaf inputs at level 0 and the outputs of C_m as $Z_1, \ldots Z_w$ and $h_1, \ldots, h_w$. Clearly, C_m and D are both skew circuits of width w. Further, each h_j depends on X, y_m and Z; that is, $h_1, \ldots, h_w \in R[Z_1, \ldots, Z_w]$ where $R = \mathbb{K}[X, y_m]$. Similarly, each g_j depends on X and Y'; $g_1, \ldots, g_w \in \mathbb{K}[X, Y']$. The values computed by C can be expressed as $f_j(X, Y) = h_j\left(X, y_m, g_1(X, Y'), \ldots, g_w(X, Y')\right)$.

Since C and C_m are skew circuits, and since the variables Z_j represent non-leaf gates of C, C_m must be linear in these variables. Hence each h_j can be written as $h_j(X, y_m, Z) = c_j + \sum_{k=1}^{w} c_{j,k} Z_k$, where the coefficients $c_j, c_{j,k} \in \mathbb{K}[X, y_m]$. Combining this with the expression for f_j, we have

$$\begin{aligned} f_j(X, Y) &= h_j\left(X, y_m, g_1(X, Y'), \ldots, g_w(X, Y')\right) \\ &= c_j(X, y_m) + \sum_{k=1}^{w} c_{j,k}(X, y_m) g_k(X, Y') \qquad \text{and hence} \end{aligned}$$

$$\begin{aligned} \sum_{e \in \{0,1\}^m} f_j(X, e) &= \sum_{e \in \{0,1\}^m} \left[c_j(X, e_m) + \sum_{k=1}^{w} c_{j,k}(X, e_m) g_k(X, e') \right] \\ &= 2^{m-1} \sum_{e_m=0}^{1} c_j(X, e_m) + \sum_{k=1}^{w} \sum_{e \in \{0,1\}^m} c_{j,k}(X, e_m) g_k(X, e') \end{aligned}$$

$$= \quad 2^{m-1} \sum_{e_m=0}^{1} c_j(X, e_m) + \sum_{k=1}^{w} \left(\sum_{e_m \in \{0,1\}} c_{j,k}(X, e_m) \right) \left(\sum_{e' \in \{0,1\}^{m-1}} g_k(X, e') \right)$$

$$\text{Thus} \quad E_Y(f_j)(X) \quad = \quad 2^{m-1} E_{y_m}(c_j)(X) + \sum_{k=1}^{w} E_{y_m}(c_{j,k})(X) E_{Y'}(g_k)(X)$$

By induction, we know that there is a weakly skew circuit D' of width $4w$ and size $O((m-1)w^4 s)$ computing $E_{Y'}(g_k)(X)$ for all k simultaneously.

To compute $E_{y_m}(c_j)(X)$, note that a copy of C_m with all leaves labeled Z_k replaced by 0 computes exactly $c_j(X, y_m)$. So the sum can be computed as in the base case, in width $w+1$ and size $3(\mathsf{size}(C_m)+1)$. Multiplying this by 2^{m-1} in the standard way adds nothing to width and 2 to size, so overall width is $w+1$ and size is at most $2s+4$.

To compute $E_{y_m}(c_{j,k})(X)$, we modify C_m as follows: replace leaves labeled Z_k by the constant 1, replace leaves labeled $Z_{k'}$ for $k' \neq k$ by 0, leave the rest of the circuit unchanged, and let h_j be the output gate. This circuit computes $c_j(X, y_m) + c_{j,k}(X, y_m)$. Subtracting $c_j(X, y_m)$ (as computed above) from this gives $c_{j,k}(X, y_m)$. Now, the sum can be computed as in the base case. Again, to compute $E_{y_m}(c_{j,k})(X)$, we use two copies of the difference circuit with $y_m = 0$ and $y_m = 1$ hardwired, and add their outputs. It is easy to see that this circuit has width $w+2$ and size at most $4(w+2)\mathsf{size}(C_m) \leq 4(w+2)s$.

Putting together these circuits naively may increase width too much. So we position D' at the bottom, and carry w wires upwards from it corresponding to its w outputs. Alongside these wires, we position circuitry to accumulate the terms for each f_j and to carry forward already-computed f_k's. The width in this part is w for the wires carrying the outputs of D', w for wires carrying the values $E_Y(f_j)$, $w+2$ for computing the terms in the sum above (they are computed sequentially so the width does not add up), and 2 for computing partial sums in this process, overall at most $3w+4$. Thus the resulting circuit has width at most $\max\{\mathsf{width}(D'), 3w+4\} \leq 4w$.

To bound the size of the circuit, we bound its depth in the part above D' by

d; then size is at most $\mathsf{size}(D') + \mathsf{width} \times d$. The circuit has w modules to compute the $E_Y(f_j)$s. The depth of each module can be bounded by the depth to compute $E_{y_m}(c_j)$ plus w times the depth to compute any one $E_{y_m}(c_{j,k})$, that is, at most $(2s+4) + w \times 4(w+2)s$. So $d \leq w(2s+4+4sw(w+2)) = \theta(w^3 s)$, and the size bound follows. $\diamond$

Now we prove Theorem 7.16:

Proof. [of Theorem 7.16]

(1) Since weakly skew circuits can be transformed into skew circuits ([61]) with a constant blowup in the size([47]), the equivalence follows.

(2) From Lemma 5.10 we know that when width of the weakly skew circuit is constant, width of the resulting skew circuit will be again a constant. *i.e.* the resulting circuit now will have width $O(w^2)$ and size $O(w2^{4w}mw^4 s)$. $\diamond$

7.4 Read-Once exponential sums of some restricted circuits

In this section, we explore how far the result of Theorem 7.16 can be pushed to larger classes within VP. In effect, we ask whether the technique of Lemma 7.18 is applicable to larger classes of circuits. Such a question is relevant because we do not have any bound (better than VNP) even for $\Sigma^R \cdot \mathsf{VSC}^0$ and $\Sigma^R \cdot \mathsf{VL}$.

One generalization we consider is multiplicative disjointness. Recall from Section 5.4.1: an arithmetic circuit C is said to be multiplicatively disjoint (md) if every multiplication gate operates on sub-circuits which are not connected to each other.

A further generalization we consider is polynomial syntactic degree bounded arithmetic circuits.

Examining the proof of Lemma 7.18, we see that the main barrier in extending it to these larger classes is that when we slice C into D and C_m, C_m is no longer linear in the "slice variables" Z. However, for md-circuits, C_m is multilinear in Z. As far as computing the coefficients $c_{j,\alpha}$ goes, where α describes a multilinear monomial, this is not a problem; it can be shown that for such circuits the coefficient function can be computed efficiently. There is a cost to pay in size because the number of multilinear monomials is much larger. To handle this, we modify the inductive

step, slicing C not at the last block but at a level that halves the number of Y variables read above and below it. This works out fine for constant-width, but results in quasi- polynomial blow-up in size for larger widths.

We show the following:

Theorem 7.19 *1.* $\Sigma^R \cdot md\text{-}\mathsf{VSC}^0 \subseteq \mathsf{VP}$.

2. $\Sigma^R \cdot md\text{-}\mathsf{VSC} \subseteq \mathsf{VQP}$.

3. For all $i \geq 0$, $\Sigma^R \cdot \mathsf{VsSC}^i \subseteq \mathsf{VQP}$.

Proof strategy for Theorem 7.19.

1. Break the circuit by a horizontal cut into two parts A and B, so that each part contains approximately $m/2$ variables from Y i.e $Y_A, Y_B \leq \lceil m/2 \rceil$ and $Y_A \cup Y_B = Y$, $Y_A \cap Y_B = \emptyset$. Let A be the upper part.

2. Now express the polynomials in A as sums of monomials where the variables stand for the output gates of B and the coefficients come from $\mathbb{K}[X, Y_A]$.

3. Inductively compute the E_Y's for the coefficients of A and the monomials in terms of the output gates of B.

4. Apply equation 7.1 below to obtain the required $E_Y(f_j)$s.

This strategy is spelled out in detail for the case of multiplicative disjoint circuits in Lemma 7.21. For syntactic degree bounded by a polynomial, Step 3 above needs special treatment, spelled out in Lemma 7.23.

We need the following observation (which is already used implicitly in the proof of Lemma 7.18):

Observation 7.20 *1. If* $f = g + h$, *then* $E_Y(f) = E_Y(g) + E_Y(h)$.

2. If $f = g \times h$, *and if the variables of* Y *can be partitioned into* Y_g *and* Y_h *such that* g *depends only on* X, Y_g *and* h *depends only on* X, Y_h, *then*

$$E_Y(f) = E_{Y_g}(g) \times E_{Y_h}(h) \tag{7.1}$$

Lemma 7.21 *Let C be a layered multiplicatively disjoint circuit of width w and size s on variables $X \cup Y$. Let ℓ be the number of layers in C. Suppose C is read-once certified in Y. Let $f_1 \ldots, f_w$ be the output gates of C. Then, there is an arithmetic circuit C' of size $sm^{O(w)}$ which computes $E_Y(f_1), \ldots, E_Y(f_w)$.*

Proof. The proof is by induction on $m = |Y|$. The base case when $m = 1$ is trivial. Now assume that the statement holds for all circuits with $|Y| < m$.

Let $0 = i_1 \leq i_2 \leq \ldots \leq i_m \leq \ell$ be the level indices of C as guaranteed by definition 7.9. Consider level $\ell' = i_{\lceil m/2 \rceil}$. Let $g_1, \ldots, g_w$ be the gates at level ℓ'.

We slice C at level ℓ'; the circuit above this level is A and the circuit below it is called B. In particular, A is obtained from C by re-labeling the gates $g_1, \ldots, g_w$ with new variables $Z_1, \ldots, Z_w$ and removing all gates below level ℓ'. Let $h_1, \ldots, h_w$ denote the output gates of A. (Note that these are just relabellings of $f_1, \ldots, f_w$.) Similarly, B is obtained from C by removing all nodes above layer ℓ' and making $g_1, \ldots, g_w$ the output gates. Let s_A and s_B respectively denote their sizes. Let $Y_A \subseteq Y$ (resp Y_B) be the set of variables from Y that appear in A (resp. B). The circuits A and B have the following properties:

1. A and B are multiplicatively disjoint and are of width w.
2. A is *syntactically multilinear* in the variables $Z = \{Z_1, \ldots, Z_w\}$: at every $\times$ gate $f = g \times h$, each variable in Z has a path to g or to h or to neither, but not to both.
3. $Y_A \cap Y_B = \emptyset$, $Y_A \cup Y_B = Y$, $|Y_A| = \lfloor m/2 \rfloor$ and $|Y_B| = \lceil m/2 \rceil$.
4. For $1 \leq j \leq w$, $g_j \in \mathbb{K}[X, Y_B]$ and $h_j \in R[Z]$, where $R = \mathbb{K}[X, Y_A]$.
5. Let $v = v_1 \times v_2$ be a multiplication gate in A. If there is a path from Z_i to v_1 and there is a path from Z_j $(i \neq j)$ to v_2, then the sub-circuits of C (and hence of B) rooted at g_i and g_j are disjoint.

Since A is syntactically multilinear in Z and C is md, the monomials in $h_j \in R[Z]$ can be described by subsets of Z, where Z_i and Z_k can belong to a subset corresponding to a monomial only if the sub-circuits rooted at g_i and g_k are disjoint. Let S denote the subsets that can possibly correspond to monomials:

$$S = \left\{ R \subseteq Z \;\middle|\; \begin{array}{l} \forall Z_i, Z_k \in R \text{ with } i \neq k, \text{ the sub-circuits} \\ \text{rooted at } g_i \text{ and } g_k \text{ are disjoint} \end{array} \right\}$$

Generally, we treat S as a set of characteristic vectors instead of actual subsets; the usage will be understood from the context.

We can express the polynomials computed by A and C as follows:

$$\begin{aligned} f_j &= h_j(g_1, \ldots, g_w); \quad h_j = \sum_{\alpha \in S} c_{j,\alpha} Z^\alpha \\ \text{where } Z^\alpha &= \prod_{i=1}^{w} Z_i^{\alpha_i} \text{ and } c_{j,\alpha} \in \mathbb{K}[X, Y_A] \\ \text{Hence } f_j(X,Y) &= \sum_{\alpha \in S} c_{j,\alpha}(X, Y_A) g^\alpha(X, Y_B) \quad \text{where } g^\alpha(X, Y_B) = \prod_i g_i^{\alpha_i}(X, Y_B) \end{aligned}$$

Now using Observation 7.20 we have,

$$E_Y(f_j) = \sum_{\alpha \in S} E_Y(c_{j,\alpha} g^\alpha) = \sum_{\alpha \in S} E_{Y_A}(c_{j,\alpha}) E_{Y_B}(g^\alpha) \tag{7.2}$$

We need the following claim:

Claim 7.22 *For $1 \leq j \leq w$, and for $\alpha \in S$, the polynomial $c_{j,\alpha}(X, Y_A)$ can be computed by a multiplicatively disjoint circuit of size $w.s_A$ and width w.*

Proof. The proof is by induction on the structure of the circuit. Let $\alpha = \alpha_1 \alpha_2 \ldots \alpha_w$, where $\alpha_i \in \{0, 1\}$. The base case is when the sub-circuit rooted at g_j is a variable Z_i or $a \in \mathbb{K} \cup X \cup Y_A$. Then, $[c_{j,\alpha}]$ is set accordingly:

If $g_j = Z_i$ then

$$[c_{j,\alpha}] = \begin{cases} 1 \text{ for } \alpha_i = 1, \alpha_k = 0, \ i \neq k \\ 0 \text{ otherwise} \end{cases}$$

If $g_j = a \in \mathbb{K} \cup X \cup Y_A$ then,

$$[c_{j,\alpha}] = \begin{cases} a \text{ if } \alpha_i = 0, \forall i \\ 0 \text{ otherwise} \end{cases}$$

Induction step: case 1: $g_j = h_1 + h_2$, then $[c_{j,\alpha}] = [h_{1,\alpha}] + [h_{2,\alpha}]$.
case 2: $g_j = h_1 \times h_2$ then $[c_{j,\alpha}] = [h_{1,\alpha'}] + [h_{2,\alpha''}]$.

Where α' (respectively α'') is α restricted to the Z-variables that appear at the sub-circuit rooted at h_1 (respectively h_2). We set $[c_{j,\alpha}]$ to 0 if α' and α'' do not form a partition of α. Note that, $[h_{1,\alpha}]$, $[h_{2,\alpha}]$, $[h_{1,\alpha'}]$ and $[h_{2,\alpha''}]$ are the corresponding coefficients available from inductive hypothesis.

The size of $[c_{j,\alpha}]$ thus obtained can blow up by factor of at most w and width remains unchanged. $\diamond$

Let $[c_{j,\alpha}]$ denote the circuit obtained in the above claim.

If $\alpha \in S$, then g^α can be computed by an md-circuit of width w and size $s_B + w$. Let $[g^\alpha]$ denote this circuit. (It is an addition sub-circuit sitting on top of the relevant output gates of B.)

By the induction hypothesis, the polynomials $E_{Y_A}(c_{j,\alpha})$ for $1 \leq j \leq w$ and $\alpha \in S$ can be computed by arithmetic circuits of size $T(w, \lfloor m/2 \rfloor, ws_A)$. Also, by induction, the polynomials $E_{Y_B}(g^\alpha)$ can be computed by arithmetic circuits of size $T(w, \lceil m/2 \rceil, s_B + w)$. Now, using the expression 7.2, $E_Y(f_j)$ for each $1 \leq j \leq w$ can be computed by arithmetic circuits that compute all the $E_{Y_A}(c_{j,\alpha})$ and all the $E_{Y_B}(g^\alpha)$, and then put them together; such a circuit has size $T(w, m, s) = w \times |S| \times T(w, \lfloor m/2 \rfloor, ws_A) + |S| \times T(w, \lceil m/2 \rceil, s_B + w) + 2|S|$. As $|S| \leq 2^w$, we have,

$$\begin{aligned} T(w, m, s) &\leq w2^w T\left(w, \left\lfloor \frac{m}{2} \right\rfloor, ws_A\right) + 2^w T\left(w, \left\lceil \frac{m}{2} \right\rceil, s_B + w\right) + 2^{w+1} \\ s &= s_A + s_B \end{aligned}$$

By solving the recurrence we get the desired bound :
$T(w, m, s) = 2^{2(w+2)\log m} w^{2\log m} s + 2^{w+1} \log m = sm^{O(w)}$. $\diamond$

For VsSC circuits, the "upper half" circuit is not even multilinear. So we need to explicitly account for each monomial up to the overall degree, and compute the coefficient of each. We show that this is possible, if a quasi polynomial blow-up in size is allowed. Formally,

Lemma 7.23 *Let C be a layered arithmetic circuit size s on the variables $X \cup Y \cup Z$. Let d be the syntactic degree bound on C and w be its width. Let $f \in R[Z]$ be a polynomial computed by C, where $R = K[X, Y]$. Let $t = \langle t_1, \ldots, t_w \rangle$ be a degree sequence for variables from Z. Then $\mathsf{coeff}_f(Z^t)$ can be computed by a circuit of width $w + 2$ and size $O(s(d+1)^{2w})$, where $Z^t = \prod_{k=1}^{w} Z^{t_k}$.*

Proof.

From the proof of Lemma 7.8, we have

$$\mathsf{coeff}_f(Z^t) = \sum_{i_1,\ldots,i_w \in \{0,\ldots,d\}} f(i_1,\ldots,i_w) \prod_{k=1}^{w} \mathsf{G}(x_k, i_k)$$

where

$$\mathsf{G}(x_k, i_k) = \frac{(-1)^{d-t_k} S_{d,t_k}(0,1,\ldots,i_k-1,i_k+1,\ldots,d)}{i_k!(d-i_k)!(-1)^{(d-i_k)}}$$

The number of terms in the above sum is bounded by $(d+1)^w$. We know that each S_{d,t_k} can be computed by a depth-3 arithmetic circuit of polynomial size $\mathsf{poly}(d)$; let $p(d)$ be the size upper bound on such a circuit. This circuit can easily be transformed into a width-3 circuit of the same size. Now, to compute $\mathsf{coeff}_f(Z^t)$, we compute each term in the above expression sequentially and accumulate the sum in an internal gate along the way. To compute a term, we need to evaluate f at a certain point $i_1 \ldots, i_w$ and then multiply it by $G(x_k, i_k)$. First we compute $f(i_1,\ldots,i_w)$ by using a copy of C, and then compute the product by serially computing the corresponding symmetric polynomials and multiplying by the inverse of the denominator (note that this value only depends on d and i_k's hence can be hardwired). Thus a term can be computed within a width of $\max\{w, 5\}$. To compute the overall sum, we need an extra gate to carry the partial sum. Thus the total width needed can be bounded by $\max\{w+1, 5\}$. The number of copies of C needed is bounded by $(d+1)^w$ and the total number of circuits for S_{d,t_k} is bounded by $(d+1)^w w$. Hence the overall size can be bounded by $(d+1)^w \times s + 2 \times (d+1)^w \times w \times p(d)$. By [79], $p(d) = O(d^2)$; thus for $w \geq 2$ this is bounded by $O(s(d+1)^{(2w)})$. ◇

Proof. [of theorem 7.19]

1. This directly follows from Lemma 7.21

2. Let C be an arithmetic circuit of width w, size s and syntactic degree d. Now applying the strategy and using Lemma 7.23 for step 3, we can construct the required circuit C' through induction. Let $T(w,d,m)$ denote the size of the required circuit, then $T(w,d,m) \leq 2^w s(d+1)^{2w} T(w,d,m/2)$. Solving the recurrence, it is easy to see that $T(w,d,m) = O(2^{w \log m} s^{\log m}(d+1)^{2w \log m})$ which gives the required result. ◇

Conclusion and Open questions

We proposed a notion of "small space" for algebraic computations in terms of the circuit width. VL was defined as class of polynomials computed by log width circuits with certain degree and constraints on coefficients. However it is easy to see that our definition of $\mathsf{VWIDTH}(S(n))$ can be extended to polynomials with arbitrary coefficients from $\mathbb{K}$. Only Theorem 7.5 does not work under this definition as $\mathsf{VPSPACE}$ contains only polynomials with integer coefficients.

Having a reasonable upper bound for VL seems to be a hard task: as VBP can be seen as a natural arithmetic version of NL, we would like to have VL contained inside VBP.

Later on we introduced the notion of read-once certificates and read-once exponential sums of arithmetic circuits. It is shown that with this definition, the classes behave on the expected lines: 1) ABPs are closed under taking read once exponential sums. 2) Applying read once exponential sum to VP yields exactly the class VNP. However, in the case of VsSC^i we could prove only an upper bound of VQP, *i.e.* $\Sigma^R \cdot VsSC^i \subseteq \mathsf{VQP}$ (Theorem 7.19). For the case of $\Sigma^R \cdot \mathsf{VL}$ the best upper bound one could give is only VNP which is obvious from definition itself.

Are VNC^1 and VsSC^0 separate? The study of read once exponential sums throws in this doubt: Is VsSC^0 really more powerful than VNC^1? Since we don't have a nice definition of read once certificates for depth bounded circuits, we use the equivalence $\mathsf{VBWBP} = \mathsf{VNC}^1$ for this purpose. From Theorem 7.16, we have $\Sigma^R \cdot \mathsf{VBWBP} = \mathsf{VBWBP}$, hence we can say that $\Sigma^R \cdot \mathsf{VNC}^1 = \mathsf{VNC}^1$. On the other hand the best known upper bound for $\Sigma^R \cdot \mathsf{VsSC}^0$ is VQP. Thus on the one hand showing $\mathsf{VNC}^1 = \mathsf{VsSC}^0$ will bring $\Sigma^R \cdot \mathsf{VsSC}^0$ all the way down to VNC^1 and showing a super-polynomial formula size lower bound for VsSC^0 could separate VsSC^0 from VNC^1. However the second one is going to be much harder.

We conclude the chapter with the following questions:

- Is VL contained in VBP? *i.e.* do the class of all log width poly degree and size circuits have equivalent poly size algebraic branching programs?
- Is $\Sigma^R \cdot \mathsf{VL} \subseteq \mathsf{VP}$? Even in the case of VSC^0, it will be interesting to see an upper bound of VP, i.e, is $\Sigma^R \cdot \mathsf{VSC}^0 \subseteq \mathsf{VP}$?

- Is there any natural family of polynomials complete for VL?

7.5 Appendix

7.5.1 Blum Shub Smale (BSS) model of computation

In this section we briefly describe the model of computation over reals proposed by Blum,Shub and Smale. For more details reader is referred to [19]. The BSS model is defined for computation over the field $\mathbb{R}$ of real numbers. We present the version used in [32].

A BSS machine M has an input tape output tape and a work tape, where each cell stores a value from $\mathbb{R}$ and a set of parameters $\mathcal{A} = \{A_1, \ldots, A_k\}$, where $A_i \in \mathbb{R}$. In a single step, M can perform one of the following operations:

- *Input:* reads a value from the input tape into its work tape.
- *Computation:* Performs an arithmetic operation over values in the work tape (The number of operands is some fixed constant).
- *Output:* Writes a value on the output tape.
- *Constant:* Writes a constant $A_i \in \mathbb{R}$.
- *Branch:* Compares two real values and branches accordingly

Naturally we can associate a function $\phi_M : \mathbb{R}^* \to \mathbb{R}$ with M. We say that a real set $L \subseteq \mathbb{R}^*$ is decided by M if the characteristic function of L, χ_L equals ϕ_M. We can make the machine M above non-deterministic by allowing non-deterministic choices at every step. $\mathsf{P}_{\mathbb{R}}$ is the set of all languages from $\mathbb{R}^*$ that are decidable by polynomial time bounded BSS machines. Also, $\mathsf{NP}_{\mathbb{R}}$ is the class of languages that are computable by non-deterministic polynomial time bounded BSS machines.

In the unit space model, we count a the number of work tape cells used by the machine as the space used. Michaux ([64]) showed the following:

Proposition 7.24 ([64]) *Let $L \subseteq \mathbb{R}$ be a language computed by a machine M in time t. Then there is machine M' and a constant k such that M' computes L in unit space k.*

Part II

Complexity of Matroid Isomorphism Problems

Chapter 8

Matroid Isomorphism Problems

8.1 Introduction

Given two mathematical (algebraic or combinatorial) structures, the isomorphism question asks whether the given structures are identical modulo a bijective renaming of their elements. In the computational world we ask how hard (or easy) is it to test isomorphism of given objects.

The most prominent among isomorphism testing problems is the "Graph Isomorphism" (GI) problem: Given two graphs, test if they are isomorphic. The significance of GI arises from the fact that it is not known to be polynomial time computable and there is evidence that it is unlikely to be NP-complete([50]). In general, the study of GI has been pursued in two directions: 1) Find out restrictions on the input graphs so that GI is polynomial time solvable for such restricted graphs 2) Study the structural complexity of GI. There has been a large amount of work on both of these directions in the literature. (See *e.g.* [59], [50] and [12].)

Other structures on which isomorphism questions have been studied include: Groups, Boolean functions, Bilinear forms etc,. ([2, 11, 81, 75])

An important combinatorial structure that is missing in the above picture is matroids. A matroid M on a given finite ground set S is a collection $\mathcal{I}$ of subsets of S called "independent sets" that satisfies the following axioms: 1) $\emptyset \in \mathcal{I}$. 2) $\mathcal{I}$ is closed under taking subsets. 3) For $A, B \in \mathcal{I}$, if $|A| < |B|$, then there is some $x \in B \setminus A$ such that $A \cup \{x\} \in \mathcal{I}$. Matroids M_1 and M_2 are said to be isomorphic if there is a bijection between their corresponding ground sets that

preserves independent sets and non-independent sets. The major issue that arises in the computational aspect of testing if two given matroids are isomorphic is the representation of the input matroids.

One straightforward representation for matroids is the explicit listing of its independent sets (or the bases, *i.e.* the maximal independent sets). Dillon Mayhew ([63]) studied the matroid isomorphism problem under this input representation and showed that the problem is polynomial time equivalent to the graph isomorphism problem. However for most of the matroids the number of independent sets can be exponential in the size of the ground set.

There are matroids with more implicit representations. Among them are: graphic matroids, linear matroids, bicircular matroids etc. We are interested in the former two.

Given a matrix $A \in \mathbb{F}^{m \times n}$ we can associate a matroid $M[A]$ with A as follows: The column vectors of A is the ground set of $M[A]$ and independent sets are exactly the linearly independent columns of A. Such matroids are called "linear matroids" or "representable matroids" over $\mathbb{F}$. Given an undirected graph $= (V, E)$, we can associate a matroid $M(G)$ with E as its ground set and set of all forests in G as collection of independent set. In the literature $M(G)$ is referred to as the "graphic matroid" or "polygon matroid" of G.

It is not hard to see that graphic matroids form a sub-class of linear matroids as graphic matroid are representable over all fields (i.e, they are regular). There are linear matroids that are not graphic. (See Section 8.2 for examples.)

In this chapter we study the complexity of isomorphism testing for linear and graphic matroids. More generally we consider matroids which are represented as "independent set oracles". As a basic complexity bound, it is easy to see that isomorphism testing of two matroids represented by independent set oracles can be performed in Σ_p^2, *i.e.* the second level of polynomial hierarchy.

In the case of matroids representable over finite fields, we show that isomorphism testing is unlikely to be Σ_p^2 complete. We also show that isomorphism testing of linear matroids (LMI) is co-NP hard when the representation is over finite fields of polynomial size. Moreover, if the size of idependent sets of the linear matroids is bounded by a constant then LMI is polynomial time many-one equivalent to GI. This result essentially follows from the ideas of Babai ([13]) where he constructed linear matroids of constant rank corresponding to graphs such that the

automorphism groups of the two (the matroid and the graph) are isomorphic.

In the case of graphic matroids, the isomorphism testing problem (GMI) is polynomial time Turing equivalent to GI. Though the equivalence is expected, the reduction from GMI to GI requires a complicated colouring procedure to colour the edges so that the isomorphism is preserved while breaking the graph into its 3-connected components. Then applying the equivalence of GMI and GI for the case of 3-connected graphs the result follows([101]).

Chapter organization: Section 8.2 recollects the basic definitions of matroids and sets up the isomorphism problems. In section 8.3 we study the linear matroid isomorphism problem. Section 8.4 is devoted to proving the polynomial time equivalence of graphic matroid isomorphism and graph isomorphism problems. In section 8.5 we discuss improved upper bounds in the case of planar graphs and graphs of bounded valence and bounded genus.

8.2 Preliminaries

This section is devoted to basic definitions on matroids and the formulation of the matroid isomorphism problems.

8.2.1 Matroids

Here we recall the definition of a matroid. We follow the notations from [69].

Definition 8.1 *A matroid M is a tuple $(S, \mathcal{I})$ where S is a finite set called the ground set and $\mathcal{I}$ is a set of subsets of S satisfying the following axioms:*

- $\emptyset \in \mathcal{I}$.
- *If $A \in \mathcal{I}$, and $B \subseteq A$ then $B \in \mathcal{I}$.*
- *If $A, B \in \mathcal{I}$, and $|A| < |B|$ then $\exists x \in B \setminus A$ such that $A \cup \{x\} \in \mathcal{I}$.*

The last axiom in the above definition implies that all the maximal independent sets in M have the same size. All subsets of S that are not in $\mathcal{I}$ are called "dependent" sets. The rank function $\mathsf{rank} : 2^S \to \mathbb{N}$ is defined as, $\mathsf{rank}(A) =$ the size of the largest independent set in $\mathcal{I}$ that is contained inside $A \subseteq S$. Rank of the

matroid M is defined to be $\mathsf{rank}(S)$. A "basis" is a maximal independent set in $\mathcal{I}$. Generally we use $\mathcal{B}$ to denote the set of all bases of M. A set $A \subseteq S$ is said to be spanning if $\mathsf{rank}(A) = \mathsf{rank}(S)$, in other words A is spanning if and only if it contains a basis of M. A "circuit" $C \subseteq S$ is a minimal dependent set in M, *i.e.* $C \subseteq S$ is a circuit in M if and only if $\forall B \subset C$, $B \in \mathcal{I}$. For a matroid $M = (S, \mathcal{I})$, $\mathcal{C}$ denotes the set of all circuits in M. For any $A \subseteq S$, define closure of A as, $cl(A) = \{x \in S \mid \mathsf{rank}(A \cup \{x\}) = \mathsf{rank}(A)\}$. A set $A \subseteq S$ is said to be a "flat" if $cl(A) = A$. A "hyperplane" is a flat of rank $r - 1$, where r is the rank of M.

Note that any one of the following uniquely defines a matroid $M = (S, \mathcal{I})$:

- Set of all independent sets of M
- Set of all bases (*i.e.* maximal independent sets) of M.
- Set of all circuits (*i.e.* minimal dependent sets) of M.
- Set of all hyperplanes of M.

An *isomorphism* between two matroids $M_1 = (S_1, \mathcal{I}_1)$ and $M_2 = (S_2, \mathcal{I}_2)$ is a bijection $\phi : S_1 \to S_2$ such that $\forall I \subseteq S_1 : I \in \mathcal{I}_1 \iff \phi(I) \in \mathcal{I}_2$. Equivalently, $\forall C \subseteq S_1 : C \in \mathcal{C}_1 \iff \phi(C) \in \mathcal{C}_2$, where $\mathcal{C}_1$ and $\mathcal{C}_2$ are the family of circuits of the matroids M_1 and M_2 respectively. Two matroids M_1 and M_2 are *isomorphic* (i.e. $M_1 \cong M_2$) if and only there is an isomorphism between them.

8.2.2 Input Representations of Matroids

There are several ways of representing input matroids. The most straightforward one is a complete listing of independent sets. In fact a complete listing of all bases, all circuits or all hyperplanes will also do the job. However, the complexity of a problem need not be the same under these input representations. Mayhew in [63] (see also [62]) introduced these representations (and many more) and extensively studied relationships among these representations. In [63] it is also shown that isomorphism testing of matroids given by lists of independent sets is equivalent to the graph isomorphism problem (GI) *i.e.* ,

Theorem 8.2 ([63]) *Testing isomorphism of two matroids given by a complete list of independent sets is polynomial time many-one equivalent to the graph isomorphism problem (*GI*).*

Our focus is on more concise representations. We consider independent set oracles, linear and graphic representations.

Independent set oracle This is the most general of all the representation. Here the input matroid $M = (S, \mathcal{I})$ is given as ground set $S = \{1, \ldots, n\}$ and an oracle, which answers YES if and only if the subset A of S (given by its characteristic vector) being queried is independent in M. We assume that the oracle query operation takes unit-time.

Remark Note that given independent set oracle we can test if $A \subseteq S$ is a circuit in $M = (S, \mathcal{I})$ using $|A|$ queries.

We define the isomorphism testing problem in the independent set oracle setting as follows:

Problem [MATROID ISOMORPHISM(MI)]

Input: 1^m

Oracle: $M_1 = (S_1, \mathcal{I}_1)$ and $M_2 = (S_2, \mathcal{I}_2)$, with $m = |S_1| = |S_2|$.

Question: Are M_1 and M_2 isomorphic?

Linear Matroids

Definition 8.3 *Let $A \in \mathbb{F}^{r \times n}$ be a matrix over a field $\mathbb{F}$. The matroid $\mathcal{M}[A] = (C_A, \mathcal{I})$ is defined as follows: the set of columns C_A of A is the ground set and linearly independent columns are the independent sets of $\mathcal{M}[A]$.*

A matroid $M = (S, I)$ is said to be *linear* over $\mathbb{F}$, if there exists a matrix $A \in \mathbb{F}^{r \times n}$ such that $M = \mathcal{M}[A]$, where $|S| = n$ and $r \geq \mathsf{rank}(M)$.

Matroids that are representable over $\mathbb{F}_2$ are called *binary matroids*. *Regular matroids* are those which can be represented over any field.

Remark There are matroids that are not linear. The 8-element Vámos matroid (see [69]) is one such example.

In the following, we assume that linear matroids are given in the form of a matrix. In most of the cases we deal with finite fields. These are of two types: 1)

Fixed finite fields (*e.g.* $\mathbb{F}_q$). 2). Finite fields of varying size but bounded by polynomial in the size of the ground set. We use the straightforward input representation for fields: Addition and multiplication tables along with 0 and 1.

We define the isomorphism testing problem for linear matroids as follows:

Problem [Linear Matroid Isomorphism(LMI)]

Input: $\mathbb{F}$ with $m \leq |\mathbb{F}| \leq \mathsf{poly}(m)$ given by its addition and multiplication tables. $A, B \in \mathbb{F}^{r \times m}$.

Question: Are $\mathcal{M}[A]$ and $\mathcal{M}[B]$ isomorphic?

Graphic Matroids As mentioned in the introduction, given a graph $X = (V, E)$ ($|V| = n, |E| = m$), a classical way to associate a matroid $\mathcal{M}(X)$ with X is to treat E as ground set elements, the bases of $\mathcal{M}(X)$ are maximal forests of X. Equivalently circuits of $\mathcal{M}(X)$ are simple cycles in X. A matroid M is called *graphic* if M is the matroid $M(X)$ for some graph X. Graphic matroids can be represented over any field (see Proposition 5.1.2 in [69]). However, there are binary matroids which are not graphic: Fano matroid is one such example (see [69]).

Clearly, adding vertices to a graph X with no incident edges will not alter the matroid of the graph. So X can be assumed to be without isolated vertices. As self loops in X are singleton dependent sets (or loops) of $M(X)$, without loss of generality we can assume that X does not have self-loops. We define the corresponding isomorphism problem as follows:

Problem [Graphic Matroid Isomorphism(GMI)]

Input: Two graphs X_1 and X_2.

Output: Are $\mathcal{M}(X_1)$ and $\mathcal{M}(X_2)$ isomorphic?.

8.2.3 2-isomorphism

In this section, we introduce the terminology of 2-isomorphism for graphs which is equivalent to isomorphism of the corresponding graphic matroids.

Let us recall the definition of *isomorphism* of graphs. Two graphs $X_1 = (V_1, E_1)$ and $X_2 = (V_2, E_2)$ are said to be isomorphic if and only if there is a bijection $\psi : V_1 \to V_2$ such that for all $u, v \in V_1$, $(u, v) \in E_1$ if and only if $(\psi(u), \psi(v)) \in E_2$. We denote this by $X_1 \cong X_2$.

Now we define 2-isomorphism. Two graphs $X_1 = (V_1, E_1)$ and $X_2 = (V_2, E_2)$ are said to be 2-*isomorphic* (denoted by $X_1 \cong_2 X_2$) if their corresponding graphic matroids are isomorphic. In other words, $X_1 \cong_2 X_2$ if and only if there is a bijection $\phi : E_1 \to E_2$ such that for all $C \subseteq E_1$, C is a simple cycle in X_1 if and only if $\phi(C)$ is a simple cycle in X_2.

Whitney [101] gave a combinatorial characterization of 2-isomorphic graphs. We briefly describe it here. Let $X = (V, E)$ be an undirected graph. Whitney defined the following operations.

- *Vertex Identification:* Let v and v' be vertices of distinct components of X. We modify X by identifying v and v' as a new vertex $\bar{v}$.
- *Vertex Cleaving:* This is the reverse operation of vertex identification so that a graph can only be cleft at a cut-vertex or at a vertex incident with a loop.
- *Twisting:* Suppose that the graph X is obtained from the disjoint graphs X_1 and X_2 by identifying vertices u_1 of X_1 and u_2 of X_2 as the vertex u of X, identifying vertices v_1 of X_1 and v_2 of X_2 as the vertex v of X. In a *twisting* of X about $\{u, v\}$, we identify, instead u_1 with v_2 and u_2 with v_1 to get a new graph X'.

Theorem 8.4 (Whitney's 2-isomorphism theorem) *([101], see also [69]) Let X_1 and X_2 be two graphs having no isolated vertices. Then $\mathcal{M}(X_1)$ and $\mathcal{M}(X_2)$ are isomorphic if and only if X_1 can be transformed to a graph isomorphic to X_2 by a sequence of operations of vertex identification, cleaving and/or twisting.*

The following example demonstrates the above theorem.

Example 8.5 Consider the graphs X_1 and X_2 shown in Figure 8.1. It can be seen that we can transform X_1 into $X_1' \cong X_2$ by doing the following:

1. Identify vertices v and v'

2. Do a twist with respect to $\{w, z\}$

3. Perform cleave at u

and hence $X_1 \cong_2 X_2$. However, X_1 and X_2 are not isomorphic as graphs. $\diamond$

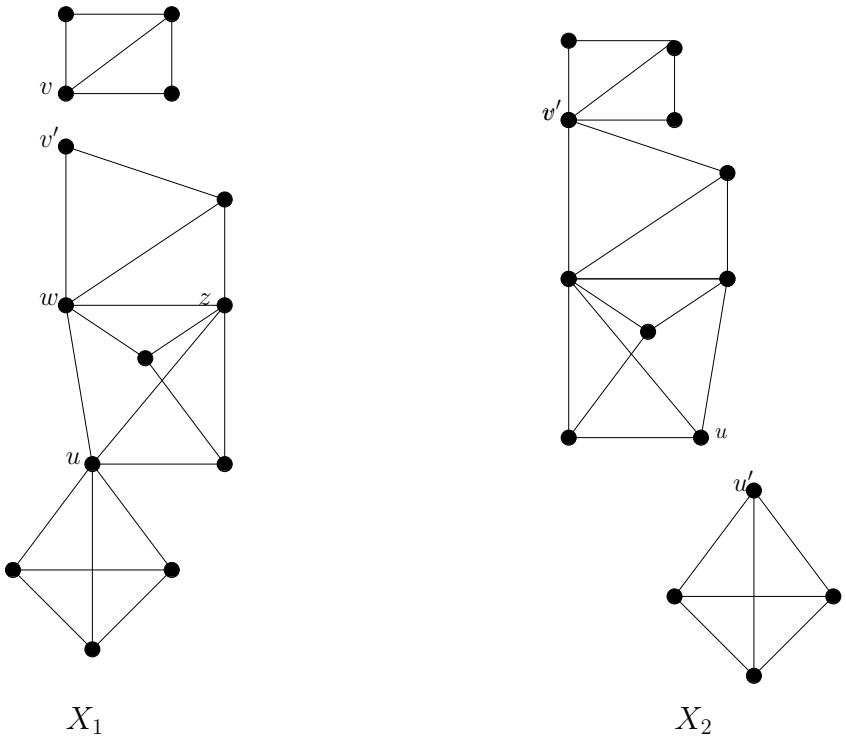


Figure 8.1: An example of 2-isomorphic graphs, that are not isomorphic

2-isomorphism of 3-connected graphs Let us recall a few definitions. A *separating pair* is a pair of vertices whose deletion increases the number of connected components. A 3-connected graph is a connected graph which does not have any separating pairs.

In another seminal paper, Whitney [100] proved that the notions of isomorphism and 2-isomorphism coincide in the case of 3-connected graphs *i.e.* ,

Theorem 8.6 (Whitney, [100]) *Let X_1 and X_2 be two graphs. If X_1 and X_2 are 3-connected then,*

$$X_1 \cong_2 X_2 \iff X_1 \cong X_2$$

8.2.4 Some complexity notions

Let A and B be two computational problems. We say that $A \leq_m^p B$ if there is a polynomial time many-one reduction from instances of A to those of B. We write $A \equiv_m^p B$ when $A \leq_m^p B$ and $B \leq_m^p A$.

We denote the polynomial time Turing reduction by $\leq_T^p$, *i.e.* $A \leq_T^p B$ if and only if there is polynomial time algorithm for A that uses B as an oracle. We write $A \equiv_T^p B$ when $A \leq_T^p B$ and $B \leq_T^p A$.

We also define the class $\mathsf{BP} \cdot \mathcal{C}$, where $\mathcal{C}$ is any complexity class, as follows:

Definition 8.7 *Let $\mathcal{C}$ be a complexity class. Then $\mathsf{BP} \cdot \mathcal{C}$ consists of all sets A for which there is a language in $B \in \mathcal{C}$ and a polynomial p such that for all $x \in \{0,1\}^*$ with $|x| = n$,*

$$Prob[(x, r) \in B \iff x \in A] \geq 3/4$$

where $r \in \{0,1\}^{p(n)}$ is uniformly randomly distributed.

The other complexity classes such as PH, Σ_i^P $(i \geq 1)$ are all standard and the reader is referred to any complexity theory text book. (*e.g.* see [8].)

8.3 Linear Matroid Isomorphism

8.3.1 General Complexity Bounds

We present some basic complexity bounds for the linear matroid isomorphism problem(LMI) . Some of these follow easily from the techniques in the literature. The purpose here is to present them in a form that is relevant to our setting.

As linear dependence over a given finite field can be computed in P, testing if two matroids are identical can be done in co-NP. By guessing a permutation π, and testing if the two matroids are identical under π, we have:

Proposition 8.8 LMI $\in \Sigma_2^p$,

In the following, we show that LMI cannot be Σ_2^p hard, unless the PH collapses to the third level. The arguments here are very similar to the arguments that prove graph isomorphism problem (GI) is unlikely to be NP-complete, given in [50]. First we show that $\overline{\text{LMI}} \in \mathsf{BP} \cdot \Sigma_2^P$. We include some details of this here.

Proposition 8.9 $\overline{\text{LMI}} \in \mathsf{BP}.\Sigma_2^{\mathsf{P}}$

Proof. Let M_1 and M_2 be the given linear matroids having m columns each. We proceed as in [50], for the case of GI. To give a $\mathsf{BP}.\Sigma_2^P$ algorithm for $\overline{\text{LMI}}$, define the following set:

$$N(M_1, M_2) = \{(N, \phi) : (N \cong M_1) \vee (N \cong M_2) \wedge \phi \in Aut(N)\}$$

where $Aut(N)$ contains all the permutations (bijections) which are isomorphisms of matroid N to itself. Automorphisms will be studied in more detail in Chapter 9. Here we only use the fact that $Aut(N)$ is a subgroup of S_m. The key property that is used in [50] has the following easy counterpart in our context.

For any matroid M on a ground set of size m, if $Aut(M)$ denotes the automorphism group of M and $\#M$ denotes the number of different matroids isomorphic to M, then $|Aut(M)| * (\#M) = |S_m|$. Hence

$$M_1 \cong M_2 \implies |N(M_1, M_2)| = m!$$

$$M_1 \ncong M_2 \implies |N(M_1, M_2)| = 2.m!$$

As in [50], we can amplify this gap and then use a good hash family and utilize the gap to distinguish between the two cases. In the final protocol (before amplifying) the verifier chooses a hash function and sends it to the prover, the prover returns a tuple (N, ϕ) along with a proof that this belongs to $N(M_1, M_2)$. Verifier checks this claim along with the hash value of the tuple. This can be done in Σ_2^p. Hence the entire algorithm gives an upper bound of $\mathsf{BP}.\exists.\Sigma_2^P = \mathsf{BP}.\Sigma_2^P$, and thus the result follows. $\diamond$

Now, we know that [76], if $\Pi_2^p \subseteq \mathsf{BP}.\Sigma_2^p$ then $\mathsf{PH} = \mathsf{BP}.\Sigma_2^p = \Sigma_3^p$. Thus we get the following:

Theorem 8.10 LMI is Σ_2^P-hard $\implies \mathsf{PH} = \Sigma_3^P$.

Remark Note that the above theorem will not hold for linear matroids over fields such as $\mathbb{Q}$. This is because the set $N(M_1, M_2)$ will be infinite in this case.

We notice that a special case of LMI is already known to be co-NP-hard. A matroid of rank k is said to be *uniform* if all subsets of size at most k are

independent. We denote by $U_{k,m}$ the uniform matroid of rank k whose ground set is of m elements. Testing if a given linear matroid is uniform is known to be co-NP complete [68]. We observe that this can be used to prove,

Proposition 8.11 LMI *is co-*NP*-hard.*

Proof. It is known (folklore) that $U_{k,m}$ is representable over any field $\mathbb{F}$ which has at least m non-zero elements. We give some details here for completeness and also, since we have not seen an explicit description of this in the literature.

Claim 8.12 *Let* $|\mathbb{F}| > m$*,* $U_{k,m}$ *has a representation over* $\mathbb{F}$*.*

Proof. [of the claim] Let $\{\alpha_1, \ldots, \alpha_m\}$ be m distinct elements of $\mathbb{F}$, and $\{s_1, \ldots, s_m\}$ be elements of the ground set of $U_{k,m}$. Assign the vector $(1, \alpha_i, \alpha_i^2, \ldots, \alpha_i^{k-1}) \in \mathbb{F}^k$ to the element s_i. Any k subset of these vectors forms a Vandermonde matrix, and hence is linearly independent. Any larger set is dependent since the vectors are in $\mathbb{F}^k$. ◇

By [68], testing whether $M[A]$ for a given matrix A is a uniform matroid is co-NP-hard. But $M[A]$ is uniform if and only if (A, B) is in LMI, where $M[B] = U_{r,n}$, $r = \mathsf{rank}(A)$. By the above claim and the fact that rank can be computed in P, B can be computed from A in poly time. ◇

The above proposition also holds when the representation is over infinite fields. In this case, the proposition also more directly follows from a result of Hlinený [42], where it is shown that the problem of testing if a spike (a special kind of matroids) represented by a matrix over $\mathbb{Q}$ is the free spike is co-NP complete. He also derives a linear representation for spikes.

8.3.2 The bounded rank case

In this section, we consider the bounded rank variant of the problem. We denote by LMI_b (MI_b) the restriction of LMI (MI) for which the input matrices have rank bounded by b. It is easy to see that both LMI_b and MI_b are in NP. Note that the list of independent sets can be obtained in polynomial time, if the given matroid represented by independent set oracle has constant rank. Thus from Theorem 8.2 we have,

Proposition 8.13 (Implicit in [63]) *For any fixed* $b > 0$*,* $\text{MI}_b \equiv_m^P \text{GI}$*.*

However, it is not clear the hard instances of MI_b obtained in [63] are linearly representable. We extend this to include LMI_b using the ideas from [13].

In [13], Babai showed that, for a given graph X, there exists a linear matroid $St(X)$ such that the automorphism groups of X and $St(X)$ are the same. Moreover, rank of $St(X)$ can be independent of the size of X. We observe that this already gives a reduction from GI to LMI_b,

First, we briefly describe the construction of [13]. Given a graph $X = (V, E)$ ($3 \leq k \leq d$, where, d is the minimum vertex degree of X), define a matroid $M = St_k(X)$ of rank k with the ground set as E as follows: every subset of $k-1$ edges is independent in M and every subset of E with k edges is independent if and only if they do not share a common vertex. Babai proved that $Aut(X) \cong Aut(St_k(X))$ and also gave a linear representation for $St_k(X)$ (Lemma 2.1 in [13]) for all k in the above range. We use this to prove,

Theorem 8.14 *For any constant $b \geq 3$,* $\text{GI} \leq_m^P \text{LMI}_b$.

Proof. Let $X_1 = (V_1, E_1)$ and $X_2 = (V_2, E_2)$ be the given GI instance. We can assume that the minimum degree of the graph is at least 3 since otherwise we can attach cliques of size $n+1$ at every vertex. We note that from Babai's proof we can derive the following stronger conclusion.

Lemma 8.15 *The following are equivalent:*

1. $X_1 \cong X_2$
2. $\exists\ k \in [3, d],\ St_k(X_1) \cong St_k(X_2)$
3. $\forall\ k \in [3, d],\ St_k(X_1) \cong St_k(X_2)$

Proof. $3 \Rightarrow 2$ is obvious.

To show $1 \implies 3$: suppose $X_1 \cong X_2$ via a bijection $\pi : V_1 \to V_2$. (The following proof works for any $k \in [3, d]$.) Let $\sigma : E_1 \to E_2$ be the map induced by π. That is $\sigma(\{u, v\}) = \{\pi(u), \pi(v)\}$. Then σ is a bijection. Consider an independent set $I \subseteq E_1$ in $St_k(X_1)$. If $|I| \leq k-1$ then $|\sigma(I)| \leq k-1$ and hence $\sigma(I)$ is independent in $St_k(X_2)$. If $|I| = k$, and let $\sigma(I)$ be dependent. This means that the edges in $\sigma(I)$ share a common vertex w in X_2. Since π is an isomorphism which induces σ, $\pi^{-1}(w)$ must be shared by all edges in I. Thus I is independent if and only if $\sigma(I)$ is independent.

To show 2 $\implies$ 1: suppose $St_k(X_1) \cong St_k(X_2)$ via a bijection $\sigma : E_1 \to E_2$ for some $k \in [3, d]$. By definition, any subset $H \subseteq E_1$ is a hyperplane of $St_k(X_1)$ if and only if $\sigma(H)$ is a hyperplane of $St_k(X_2)$. Now we use the following claim which follows from [13].

Claim 8.16 ([13]) *For any graph X, any dependent hyperplane in $St_k(X)$ is a maximal set of edges which share a common vertex (forms a star) in X, and these are the only dependent hyperplanes.*

Now we define the graph isomorphism $\pi : V_1 \to V_2$ as follows. For any vertex v, look at the star $E_1(v)$ centered at v, we know that $\sigma(E_1(v)) = E_2(v')$ for some v'. Now set $\pi(v) = v'$. From the above claim, π is an isomorphism. $\diamond$

Thus, if X_1 and X_2 have minimum degree $d \geq 3$, then $X_1 \cong X_2$ if and only if $St_3(X_1) \cong St_3(X_2)$. This shows that GI reduces to MI_b. To complete the proof we need to show that $St_3(X)$ is a linear matroid and its linear representation can be computed from X in polynomial time.

Now we show that the representation of $St_3(X)$ given in [13] is computable in polynomial time. The representation of $St_3(X)$ is over a field $\mathbb{F}$ such that $|\mathbb{F}| \geq |V|^5$. For $e = \{u, v\} \in E$ assign a vector $b_e = [1, (x_u + x_v), (x_u x_v)] \in \mathbb{F}^k$, where x_u, x_v are distinct unknowns. To represent $St_3(X)$ we need to ensure that the 3-subsets of the columns corresponding to a basis form a linearly independent set, and all the remaining k-subsets form a dependent set. Babai [13] showed that by the above careful choice of b_e, it will be sufficient to ensure only the independence condition. He also proved the existence of a choice of values for the variables which achieves this if $|\mathbb{F}| \geq |V|^5$.

We make this constructive.Note that the number of bases is bounded by $\mathsf{poly}(m)$. We can sequentially choose the value for each variable at every step, such that on assigning this value, the resulting set of 3×3 matrices are non-singular. Since there exists a solution, this algorithm will always find one. Thus we can compute a representation for $St_3(X)$ in polynomial time. $\diamond$

Since $\mathrm{MI}_b \equiv_m^P$ GI ([63]) and LMI_b is trivially contained in MI_b we have,

Corollary 8.17 $\mathrm{LMI}_b \equiv_m^p \mathrm{MI}_b \equiv_m^p$ GI.

8.4 Graphic Matroid Isomorphism

This section is devoted to the study of complexity of GMI. Main result of this section is a polynomial time Turing reduction from GMI to GI.

Unlike in the case of the graph isomorphism problem, an NP upper bound is not so obvious for GMI. H. We start with the discussion of an NP upper bound for GMI, which also serves as an NP reduction from GMI to GI.

As stated in Theorem 8.4, Whitney gave an exact characterization of when two graphs are 2-isomorphic, in terms of three operations; twisting, cleaving and identification. Note that it is sufficient to find 2-isomorphisms between 2-connected components of X_1 and X_2. In fact, any *matching* between the sets of 2-connected components whose edges connect 2-isomorphic components will serve the purpose. This is because, any 2-isomorphism preserves simple cycles, and any simple cycle of a graph is always within a 2-connected component. Hence we can assume that both the input graphs are 2-connected and in the case of 2-connected graphs, twist is the only possible operation.

The set of separating pairs does not change under a twist operation. Despite the fact that the twist operations need not commute, Truemper [89] gave the following bound.

Lemma 8.18 ([89]) *Let X and Y be 2-connected graphs on n vertices. If X is 2-isomorphic to Y, then there is a graph X' isomorphic to Y such that X' can be obtained from X through a sequence of at most $n-2$ twist operations.*

Using this lemma we get an NP upper bound for GMI. Given two graphs, X_1 and X_2, the NP machine just guesses the sequence of $n-2$ twist operations (each twist operation consists of a separating pair and a partition of connected components corresponding to the separating pair) which corresponding to the 2-isomorphism. For each pair, guess the cut w.r.t which the twist operation is to be done, and apply each of them in sequence to the graph X_1 to obtain a graph X_1'. Now test if $X_1' \cong X_2'$ using the NP algorithm for GI. Hence the overall algorithm is in NP.

This can also be seen as an NP-reduction from GMI to GI. Now we will give a deterministic reduction from GMI to GI. Although, this does not improve the NP upper bound, it implies that it is unlikely that GMI is hard for NP (Using

methods similar to that of Theorem 8.10, one can also directly prove that if GMI is NP-hard, then PH collapses to the second level). The main theorem of this section is:

Theorem 8.19 GMI $\leq_T^p$ GI

Before giving the technical proof we discuss the idea here:

Let $X_1 = (V_1, E_1)$ and $X_2 = (V_2, E_2)$ be the given two undirected graphs. Theorem 8.6 suggests the approach similar to the well known planar graph isomorphism testing algorithm of Hopcroft and Tarjan ([45]):

1. Decompose X_1 (resp. X_2) into a tree T_1 (resp. T_2) of 3-connected components using the algorithm of [44]. (For simplicity of presentation we construct related graphs X_1' and X_2' with the same decomposition tree as X_1 and X_2)

2. Find the isomorphism classes (recall that isomorphism and 2-isomorphism are the same for 3-connected graphs) of 3-connected components, and then test if T_1 and T_2 are isomorphic preserving the classes of isomorphic 3-connected components. This step can be implemented using the tree isomorphism algorithm of [55].

However, a straightforward implementation of the above idea does not work. This is because, even if the trees T_1 and T_2 are isomorphic and the corresponding 3-connected components are isomorphic, the graphs X_1 and X_2 need not be 2-isomorphic. Figure 8.2 shows such an example.

The reason for this is the isomorphism between the 3-connected components need not respect the separating pairs. A natural attempt would be to colour the virtual edges between separating pairs and test for colour-preserving isomorphisms.

Formally, an edge-k-colouring of a graph $X = (V, E)$ is a function $f : E \to \{1, \ldots, k\}$. Given two coloured graphs $X_1 = (V_1, E_1, f_1)$ and $X_2 = (V_2, E_2, f_2)$, the Coloured-GMI problem asks for an isomorphism which preserves the colours of the edges. A more detailed study of Coloured-GMI will appear in Chapter 9. Here, we use the following fact:

Proposition 8.20 Coloured-GMI *for 3-connected graphs reduces to* GI.

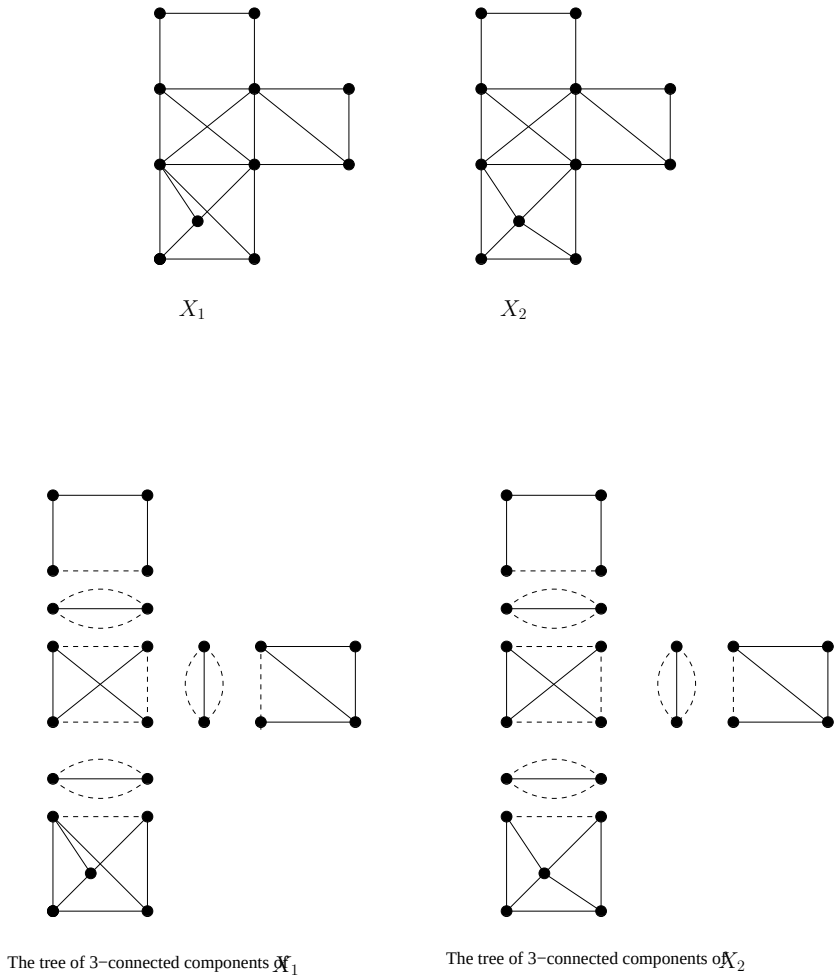


Figure 8.2: Graphs that are not 2-isomorphic but with isomorphic tree of 3-connected components

What would be a nice colouring scheme that can do the job for us? An ideal candidate is the "canonical codes" (a unique code representing a single isomorphism class) of the two sub-trees that correspond to the separating pair under consideration (we will formulate this more formally later in the section). So now the algorithm is first find the colour-preserving 2-isomorphism classes of these "edge-coloured" 3-connected components and then test for the colour-preserving tree-isomorphism.

It turns out that even this is not sufficient. This is mainly because we are not simultaneously colouring the tree and the 3-connected components hence there is

possibility of mutual mappings between separating pairs which are not compatible to each other.

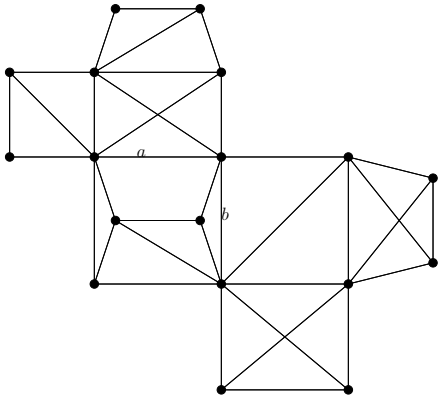
Example 8.21 The graphs X_1 and X_2 demonstrated in Figure 8.3 serve as a counter example for the above idea. Let C be the maximal 3-connected component in X_1 containing the edges a and b. The corresponding 3-connected component C' in X_2 is the one containing the edges a' and b'. Clearly $C \cong C'$, however any isomorphism between C and C' should map the edge a to b' and b to a'. Now if we look at the tree of 3-connected components, the sub-trees (two sub-trees per separating pair) obtained by removing the tree edges corresponding to a, b, a' and b' are all isomorphic. So if we use the straight forward algorithm using the idea above, the algorithm will report that X_1 and X_2 are 2-isomorphic. However it is easy to see that X_1 and X_2 are not isomorphic. $\diamond$

We circumvent the above problem by iterating the above procedure many times. *i.e.* we first find the equivalence classes of edge-coloured 3-connected components and then test isomorphism of coloured trees of 3-connected components (this colouring is defined by the class number of the 3-connected component), again repeat the procedure with these coloured trees instead of the original tree of 3-connected components.

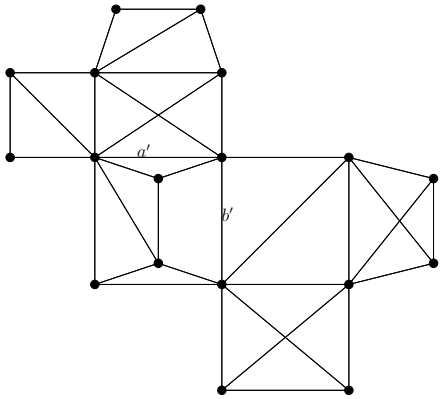
We explain the idea with the graphs X_1 and X_2 shown in Figure 8.3. Let c_1 denote the 3-connected component of X_1 containing both the edges a and b and c_1' be that of X_2 containing both the edges a' and b'.

- Color the tree of 3-connected components, such that two components get the same color if and only if they are isomorphic.
- Color the edges between sparating pairs by the canonical codes for the two subtrees obtained by deleting the corresponding edge in the tree. (See Figure 8.4 For example, color of the edge a of X_1 (see Figure 8.3) is the canonical codes of the two trees resulting from removing the edge x from T_1. It is easy to see that colour(a)=colour(a') and colour(b)=colour(b') and hence there is no isomorphism betwen c_1 and c_2 preserving the edge colours. Hence we conclude that X_1 and X_2 are not 2-isomorphic.

Now, we describe the algorithm:



X_1



X_2

Figure 8.3: A counter example where using just canonical codes is not sufficient.

INPUT: 2-connected graphs X_1 and X_2
OUTPUT: YES if $X_1' \cong_2 X_2'$, and NO otherwise.
ALGORITHM:
Notation: CODE(T) denotes the canonical label[1] for a tree T.

---

1. Obtain X_1' and X_2', T_1 and T_2 from X_1 and X_2. (Explained in detail below.)

[1]When T is coloured, CODE(T) is the code of the tree obtained after attaching the necessary gadgets to the coloured nodes. In addition, for any T, CODE(T) can be computed in L [55].

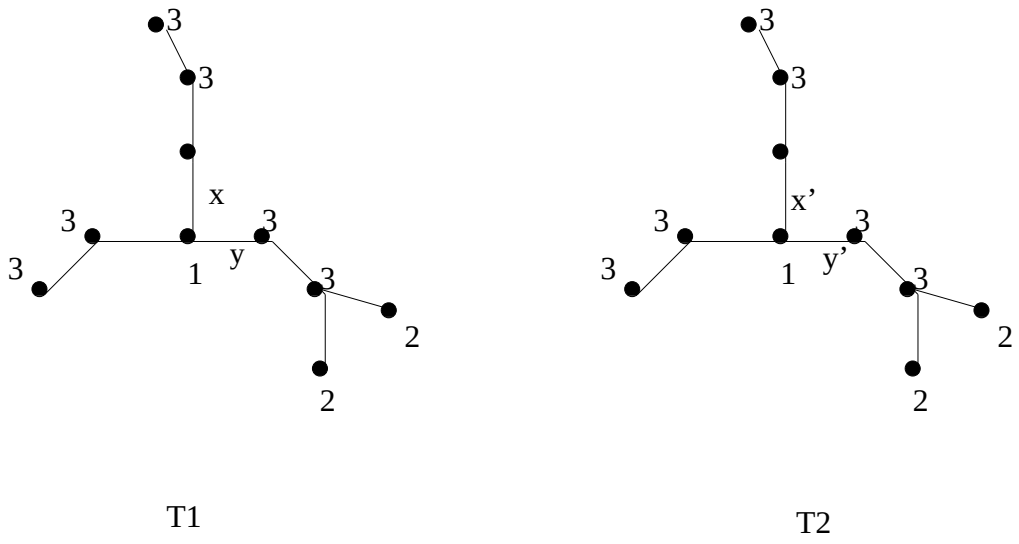


Figure 8.4: Coloured trees of 3-connected components for X_1 and X_2 of Figure 8.3

2. Initialize $T_1' = T_1$, $T_2' = T_2$.

3. REPEAT

   (a) Set $T_1 = T_1'$, $T_2 = T_2'$.

   (b) For each edge $e = (u, v) \in T_i$, $i \in \{1, 2\}$: Let $T_i(e, u)$ and $T_i(e, v)$ be subtrees of T_i obtained by deleting the edge e, containing u and v respectively. Colour the edge between the corresponding separating pairs in the components c_u and c_v with the set $\{\text{CODE}(T_i(e, u)), \text{CODE}(T_i(e, v))\}$. From now on, c_t denotes the coloured 3-connected component corresponding to node $t \in T_1 \cup T_2$.

   (c) Let S_1 and S_2 be the set of coloured 3-connected components of X_1' and X_2' and let $S = S_1 \cup S_2$. Using queries to GI (see observation 8.20) find out the isomorphism classes in S. Let $C_1, \ldots, C_q$ denote the isomorphism classes.

   (d) Colour each node $t \in T_i$, $i \in \{1, 2\}$, with colour ℓ if $c_t \in C_\ell$. (This gives two coloured trees T_1' and T_2'.)

   UNTIL $(\text{CODE}(T_i) \neq \text{CODE}(T_i'), \forall i \in \{1, 2\})$

4. Check if $T_1' \cong T_2'$ preserving the colours. Answer YES if $T_1' \cong T_2'$, and NO otherwise.

Breaking into tree of 3-connected components We describe the step 1 of the algorithm in more detail. Let us first briefly review the algorithm of [44] to break a 2-connected graph into a tree of 3-connected components in polynomial time. We will now describe some details of the algorithm which we will use for our purpose.

Let $X(V, E)$ be a 2-connected graph. Let Y be a connected component of $X \setminus \{a, b\}$, where a, b is a separating pair. Y is an *excisable* component with respect to $\{a, b\}$ if $X \setminus Y$ has at least 2 edges and is 2-connected. The operation of excising Y from X results in two graphs: $C_1 = X \setminus Y$ plus a *virtual edge* joining (a, b), and $C_2 =$ the induced subgraph on $Y \cup \{a, b\}$ plus a *virtual edge* joining (a, b). This operation may introduce multiple edges.

The decomposition of X into its 3-connected components is achieved by the repeated application of the excising operation (we call the corresponding separating pairs as *excised pairs*) until all the resulting graphs are free of excisable components. This decomposition is represented by a graph G_X with the 3-connected components of X as its vertices and two components are adjacent in G_X if and only if they share a virtual edge.

In the above construction, the graph G_X need not be a tree as the components which share a separating pair will form a clique. To make it a tree, [44] introduces new nodes in the graph G_X corresponding to every virtual edge e, which is adjacent to all the 3-connected components that contain e. Let T_X denote the tree thus obtained. In order to have the property that the vertices of T_X represent 3-connected components of X, we obtain a new graph X' with the same decomposition tree as that of X. The graph X' is obtained as follows: keep all the edges of X. For every excised pair $\{u, v\}$ of vertices in X and if (u, v) is not an edge in X, then add an edge between them. Note that T_X is also a decomposition tree for X'.

We list down the properties of the tree T_X for further reference. (1) For every node in $t \in T_X$, there is exactly one 3-connected component in X'. We denote this by c_t. (2) For every edge $e = (u, v) \in T_X$, there are exactly two virtual edges, one each in the 3-connected components c_u and c_v. We call these virtual edges the *twin edges* of each other. (3) For any given graph X, T_X is unique up to isomorphism (

since G_X is unique [44]). In addition, T_X can be obtained from G_X in polynomial time.

In the following Lemma, we prove that X' preserves the 2-isomorphism property:

Lemma 8.22 $X_1 \cong_2 X_2 \iff X_1' \cong_2 X_2'$.

Proof. Suppose $X_1 \cong_2 X_2$, via a bijection $\phi : E_1 \to E_2$. This induces a map ψ between the sets of 3-connected components of X_1 and X_2. By theorem 8.6, for every 3-connected component c of X_1, $c \cong \psi(c)$ (via say τ_c; when c is clear from the context we refer to it as τ).

We claim that ψ is an isomorphism between G_{X_1} and G_{X_2}. To see this, consider an edge $e = (u, v) \in T_{X_1}$. This corresponds to two 3-connected components c_u and c_v of X_1 which share a separating pair s_1. The 3-connected components $\psi(c_u)$ and $\psi(c_v)$ must share a separating pair say s_2; otherwise, the cycles spanning across c_u and c_v will not be preserved by ϕ which contradicts the fact that ϕ is a 2-isomorphism. Hence $(\psi(c_u), \psi(c_v))$ correspond to an edge in G_{X_2}. Therefore, ψ is an isomorphism between G_{X_1} and G_{X_2}. In fact, this also gives an isomorphism between T_{X_1} and T_{X_2}, which in turn gives a map between the excised pairs of X_1 and X_2. To define the 2-isomorphism between X_1' and X_2', we extend the map ψ to the excised edges.

To argue the reverse direction, let $X_1' \cong_2 X_2'$ via ψ. In a very similar way, this gives an isomorphism between T_{X_1} and T_{X_2}. The edge map of this isomorphism gives the map between the excised pairs. Restricting ψ to the edges of X_1 gives the required 2-isomorphism between X_1 and X_2. This is because, the cycles of X_1 (X_2) are anyway contained in X_1' (X_2'), and the excised pairs do not interfere in the mapping. ◇

Computing canonical codes of trees Let $\mathcal{G}$ denote the set of all graphs on n vertices. A canonical code (or canonization) is a map CODE: $\mathcal{G} \to \{0,1\}^*$ such that CODE(X_1) = CODE(X_2) if and only if $X_1 \cong X_2$ for every $X_1, X_2 \in \mathcal{G}$. It is known that canonical code for the class of undirected trees can be computed efficiently:

Proposition 8.23 ([55]) *Coloured tree canonization is in* L.

Running time Clearly, one iteration of the algorithm can be performed in time $\mathsf{poly}(n)$. So it is sufficient to prove that the algorithm terminates in linear number of iterations of the repeat-until loop. Let q_i denote the number of isomorphism classes of the set of the coloured 3-connected components after the i^{th} iteration. We claim that, if the termination condition is not satisfied, then $|q_i| > |q_{i-1}|$. To see this, suppose the termination is not satisfied. This means that the coloured tree T_1' is different from T_1. This can happen only when the colour of a 3-connected component c_v, $v \in T_1 \cup T_2$ changes. In addition, this can only increase the isomorphism classes. Thus $|q_i| > |q_{i-1}|$. Since q can be at most $2n$, this shows that the algorithm exits the loop after at most $2n$ steps.

Correctness Now we prove the correctness of the algorithm. Let T_1' and T_2' denote the coloured trees obtained at the termination of the algorithm. First we show that if $X_1' \cong_2 X_2'$ then the algorithm accepts, *i.e.*

Lemma 8.24 $X_1' \cong_2 X_2' \implies T_1' \cong T_2'$.

Proof. Suppose $X_1' \cong_2 X_2'$, via a bijection $\phi : E_1 \to E_2$. This induces a map ψ between the sets of 3-connected components of X_1' and X_2'. By theorem 8.6, for every 3-connected component c of X_1', $c \cong \psi(c)$ (via say τ_c; when c is clear from the context we refer to it as τ).

We claim that ψ is an isomorphism between T_1 and T_2.

To see this, consider an edge $e = (u, v) \in T_1$. This corresponds to two 3-connected components c_u and c_v of X_1' which share a separating pair s_1. The 3-connected components $\psi(c_u)$ and $\psi(c_v)$ must share a separating pair say s_2; otherwise, the cycles spanning across c_u and c_v will not be preserved by ϕ which contradicts the fact that ϕ is a 2-isomorphism. Hence $(\psi(c_u), \psi(c_v))$ correspond to an edge in T_2. Therefore, ψ is an isomorphism between T_1 and T_2. So in what follows, we interchangeably use ψ to be a map between the set of 3-connected components as well as between the vertices of the tree. Note that ψ also induces (and hence denotes) a map between the edges of T_1 and T_2.

Now we prove that ψ preserves the colours attached to T_1 and T_2 after all iterations of the repeat-until loop in step 3. To simplify the argument, we do it for the first iteration and the same can be carried forward for any number of iterations.

Let T_1' and T_2' be the coloured trees obtained after the first iteration. We argue that ψ itself is an isomorphism between T_1' and T_2'.

To this end, we prove that for any vertex u in T_1, $c_u \cong \psi(c_u)$ even after colouring as in step 3b. That is, the map preserves the colouring of the virtual edges in step 3b.

Consider any virtual edge f_1 in c_u, we know that $f_2 = \tau(f_1)$ is a virtual edge in $\psi(c_u)$. Let $e_1 = (u_1, v_1)$ and $e_2 = (u_2, v_2)$ be the tree edges in T_1 and T_2 corresponding to f_1 and f_2 respectively. We know that, $e_1 = \psi(e_2)$. Since $T_1 \cong T_2$ via ψ, we have

$$\{\text{CODE}(T_1(e_1, u_1)), \text{CODE}(T_1(e_1, v_1))\} = \{\text{CODE}(T_2(e_2, u_2)), \text{CODE}(T_2(e_2, v_2))\}\,.$$

Thus, in Step 3b, the virtual edges f_1 and f_2 get the same colour. Therefore, c_u and $\psi(c_u)$ belong to the same colour class after step 3b.

Hence ψ is an isomorphism between T_1' and T_2'. $\diamond$

Lemma 8.25 $T_1' \cong T_2' \implies X_1' \cong_2 X_2'$.

Proof. First, we recall some definitions needed in the proof. A *center* of a tree T is defined as a vertex v such that $\max_{u \in T} d(u, v)$ is minimized at v, where $d(u, v)$ is the number of edges in the unique path from u to v. It is known [40] that every tree T has a center consisting of a single vertex or a pair of adjacent vertices. The minimum achieved at the center is called the *height* of the tree, denoted by $ht(T)$. Let ψ be a colour preserving isomorphism between T_1' and T_2', and χ_t is an isomorphism between the 3-connected components c_t and $c_{\psi(t)}$. Then,

Claim 8.26 *$X_1' \cong_2 X_2'$ via a map σ such that $\forall t \in T_1'$, $\forall e \in c_t \cap E_1 : \sigma(e) = \chi_t(e)$ where E_1 is the set of edges in X_1'.*

Proof. The proof is by induction on height of the trees $h = ht(T_1') = ht(T_2')$, where the height (and center) is computed with respect to the underlying tree ignoring colours on the vertices.

Base case is when $h = 0$; that is, T_1' and T_2' have just one node (3-connected component) without any virtual edges. Simply define $\sigma = \chi$. By Theorem 8.6, this gives the required 2-isomorphism.

Suppose that if $h = ht(T_1') = ht(T_2') < k$, the above claim is true. For the induction step, suppose further that $T_1' \cong T_2'$ via ψ, and $ht(T_1') = ht(T_2') = k$. Notice that ψ should map the center(s) of T_1 to that of T_2. We consider two cases.

In the first case, T_1' and T_2' have unique centers α and β. It is clear that $\psi(\alpha) = \beta$. Let c_1 and c_2 be the corresponding coloured (as in step 3b) 3-connected components. Therefore, there is a colour preserving isomorphism $\chi = \chi_\alpha$ between c_α and c_β. Let $f_1, \ldots f_k$ be the virtual edges in c_α corresponding to the tree edges $e_1 = (\alpha, v_1), \ldots, e_k = (\alpha, v_k)$ where $v_1, \ldots, v_k$ are neighbors of α in T_1'. Denote $\psi(e_i)$ by e_i', and $\psi(v_i)$ by v_i'.

Observe that only virtual edges are coloured in the 3-connected components in step 3b while determining their isomorphism classes. Therefore, for each i, $\chi(f_i)$ will be a virtual edge in c_β, and in addition, with the same colour as f_i. That is,

$$\{\textsc{code}(T_1(e_i, \alpha)), \textsc{code}(T_1(e_i, v_i))\} = \{\textsc{code}(T_2(e_i', \beta)), \textsc{code}(T_2(e_i', v_i'))\}.$$

Since α and β are the centers of T_1' and T_2', it must be the case that in the above set equality, $\textsc{code}(T_1(e_i, v_i)) = \textsc{code}(T_2(e_i', v_i'))$. From the termination condition of the algorithm, this implies that $\textsc{code}(T_1'(e_i, v_i)) = \textsc{code}(T_2'(e_i', v_i'))$. Hence, $T_1'(e_i, v_i) \cong T_2'(e_i', v_i')$. In addition, $ht(v_i) = ht(v_i') < k$. Let X'_{f_i} and $X'_{\chi(f_i)}$ denote the subgraphs of X_1' and X_2' corresponding to $T_1'(e_i, v_i)$ and $T_2'(e_i', v_i')$ respectively. By induction hypothesis, the graphs X'_{f_i} and $X'_{\chi(f_i)}$ are 2-isomorphic via σ_i which agrees with the corresponding χ_t for $t \in T_1'(e_i, v_i)$. Define π_i as a map between the set of all edges, such that it agrees with σ_i on all edges of $X'_{f(i)}$ and with χ_t (for $t \in T_1'(e_i, v_i)$) on the coloured virtual edges.

We claim that π_i must map the twin-edge of f_i to twin-edge of $\tau(f_i)$. Suppose not. By the property of the colouring, this implies that there is a subtree of $T_1'(e_i, v_i)$ isomorphic to $T_1' \setminus T_1'(e_i, v_i)$. This contradicts the assumption that c_α is the center of T_1'.

For each edge $e \in E_1$, define $\sigma(e)$ to be $\chi(e)$ when $e \in c_\alpha$ and to be $\pi_i(e)$ when $e \in E_{f_i}$ (edges of X_{f_i}). From the above argument, $\chi = \chi_\alpha$ and σ_i indeed agrees on where it maps f_i to. This ensures that every cycle passing through the separating pairs of c_α gets preserved. Thus σ is a 2-isomorphism between X_1' and X_2'.

For the second case, let T_1' and T_2' have two centers (α_1, α_2) and (β_1, β_2) respectively. It is clear that $\psi(\{\alpha_1, \alpha_2\}) = \{\beta_1, \beta_2\}$. Without loss of generality, we assume that $\psi(\alpha_1) = \beta_1$, $\psi(\alpha_2) = \beta_2$. Therefore, there are colour preserving

isomorphisms χ_1 from c_{α_1} to c_{β_1} and χ_2 from c_{α_2} and c_{β_2}. Define $\chi(e)$ as follows:

$$\chi(e) = \begin{cases} \chi_1(e) & e \in c_{\alpha_1} \\ \chi_2(e) & e \in c_{\alpha_2} \end{cases}$$

$$c_\alpha = \cup_i c_{\alpha_i}, \quad c_\beta = \cup_i c_{\beta_i}$$

With this notation, we can appeal to the proof in the case 1, and construct the 2-isomorphism σ between X_1' and X_2'.

This completes the proof of the claim. ◇

This also completes the proof of Lemma 8.25. ◇

Now, Lemmas 8.24,8.25 combined with Propositions 8.20 and 8.23 completes the proof of Theorem 8.19.

To complete the equivalence of GI, MI_b, LMI_b and GMI, we give a polynomial time many-one reduction from MI_b to GMI. Let M_1 and M_2 be two matroids of rank b over the ground set S_1 and S_2. Let $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively denote the set of circuits of M_1 and M_2. Note that $|\mathcal{C}_1|, |\mathcal{C}_2| \leq m^{b+1}$.

We define graphs $X_1 = (V_1, E_1)$ (respectively for $X_2 = (V_2, E_2)$) as follows. For each circuit $c = \{s_1, \ldots, s_\ell\} \subseteq S_1$ in M_1, let G_c be the undirected graph (V_c, E_c) where $V_c = \{u_i, x_i, y_i \mid 1 \leq i \leq \ell\}$ and

$$E_c = \bigcup_{i=1}^{\ell} \{(u_i, u_{(i+1 \mod \ell)+1}), (x_i, y_i), (u_i, x_i), (u_{(i+1 \mod \ell)+1}, y_i)\}$$

We say that x_i and y_i are the vertices corresponding to s_i in G_c. Colour the edges $(u_i, u_{(i+1 \mod \ell)+1})$ as BLUE for $1 \leq i \leq \ell$. The edges (x_i, y_i) and (u_i, x_i) are colored YELLOW and (x_i, y_i) are coloured GREEN for $1 \leq i \leq \ell$. Now, X_1 contains the disjoint union of G_c for all $c \in \mathcal{C}_1$ and additionally the following edges: For every $s \in S_1$, consider all the circuits $c \in \mathcal{C}_1$ that contain s. Let $x_{s,c}$ and $y_{s,c}$ denote the vertices that correspond to s in G_c. Then add all the edges necessary so that the set $\{x_{s,c}, y_{s,c} \mid \text{s is contained in } c\}$ is a clique in X_1; call this clique R_s. The new edges added to complete the clique are coloured as RED.

We list down the properties of X_1 for further reference:

1. For every circuit $c \in \mathcal{C}_1$, there is a unique BLUE cycle in X_1 that is disjoint from all other BLUE cycles.

2. All the cliques with at least four vertices in X_1 are formed by edges coloured RED and GREEN. Moreover, there is a one-one map from the set of all cliques of size at least four in X_1 to the ground set S_1.

3. For every circuit $c \in \mathcal{C}_1$, the union of all the cliques of X_1 corresponding to the elements of c defines a unique blue cycle whose associated GREEN edges are in the cliques.

Now we claim the following:

Lemma 8.27 *$M_1 \cong M_2$ if and only if $X_1 \cong_2 X_2$.*

Proof. Suppose $M_1 \cong M_2$, via a map $\phi : S_1 \to S_2$. This gives a map ψ between the BLUE edges of the graphs X_1 and X_2 which preserves BLUE cycles. Now it is not hard to see that we can extend this map to include the remaining edges.

Conversely, suppose $X_1 \cong_2 X_2$ via $\psi : E_1 \to E_2$. Define $\phi : S_1 \to S_2$ as follows: For $s \in S_1$ let R_s denote the clique in X_1 corresponding to s. R_s is either a single GREEN edge or a clique on at least 4 vertices (in the latter case it is 3-connected). Thus, by the property 2 of X_1 we can see that ψ maps R_s to $R'_{s'}$ for some s' in S_2. Define $\phi(s) = s'$. Now we argue that ψ is an isomorphism between M_1 and M_2. Let $c =\subseteq S_1$ be a circuit in M_1. Now using the property 2 of X_1, we have:

$$\begin{aligned} c \in \mathcal{C}_1 &\iff \bigcup_i \psi(R_{s_i}) \text{ defines a unique BLUE cycle in } X_1 \\ &\iff \bigcup_i \psi(R'_{s'_i}) \text{ defines a unique BLUE cycle in } X_2 \\ &\iff \phi(c) \in \mathcal{C}_2 \end{aligned}$$

◇

From the above construction, we have the following theorem.

Theorem 8.28 $\mathrm{MI}_b \leq^p_m \mathrm{GMI}$.

The following theorem summarizes all the reductions proved so far in this chapter:

Theorem 8.29 $\mathrm{GMI} \leq^p_T \mathrm{GI} \equiv^p_m \mathrm{LMI}_b \equiv^p_m \mathrm{MI}_b \leq^p_m \mathrm{GMI}$

In [88], Toran showed that GI is hard for the log space classes NL and GapL. To extend this to GMI we need to show the following:

1. Reduction from GI to MI_b can be done in log-space.

2. Reduction from MI_b to GMI (Theorem 8.28) can be done in log-space.

It is easy to see that the gadgets used in Theorem 8.28 can be computed in log-space. To show 1) above, let us look at the proof of Lemma 8.15. All that we need to do here is to show that given a graph $X = (V, E)$ and $k > 0$, implement an independent set oracle for $St_k(X)$. Recall that a set $A \subseteq E$ is independent if and only if (1) $|A| \leq k-1$ or (2) $|A| = k$ and edges in A does not share a common vertex. It is straightforward to devise a log-space algorithm to check these two conditions and hence implement an independent set oracle for $St_k(X)$. This essentially shows the following:

Corollary 8.30 $\mathrm{GI} \leq_m^{\mathsf{L}} \mathrm{GMI}$

and hence we have,

Corollary 8.31 GMI *is hard for* NL *and* GapL *under* log*-space many-one reductions.*

In the next section, we observe some improved upper bounds for GMI for some specific types of graphs.

8.5 Improved upper bounds for special cases of GMI

In this section we give improved upper bounds for special cases such as planar graphic matroids, matroids of graphs of bounded genus and bounded eigen value.

8.5.1 Planar Matroids

Recall that a graph is said to be planar if it can be drawn on a plane without any crossings. A matroid is called a planar matroid if it is the graphic matroid of a planar graph. Let PMI denote the computational problem of isomorphism testing for planar matroid. Observing that the construction used in the proof of theorem 8.19 does not use any non-planar gadgets and the fact that isomorphism testing of planar graphs can be done in P ([43]), we get the following.

Corollary 8.32 PMI *is in* P.

Using the recent developments on the planar graph isomorphism problem, we improve the above bound to show that PMI $\in \mathsf{L}$. We adapt the log-space canonization procedure of [31] to the setting of planar matroids to obtain a log space algorithm for PMI. The idea used in [31] is to build canonization using the 3-connected component decomposition of the given 2-connected planar graph. We briefly describe the modifications to this procedure.

Theorem 8.33 PMI $\in \mathsf{L}$. *Moreover, a canonical encoding for planar matroids can be obtained in* log*-space.*

Proof. As observed in section 8.4, it is sufficient to consider the case of 2-connected graphs. Let $X_1 = (V_1, E_1)$ and $X_2 = (G_2, V_2)$ be the given 2-connected planar graphs. Let T_1 and T_2 be the unique decompositions of X_1 and X_2 into 3-connected components respectively.(This can be done in log space [31]). Suppose T_1 (resp. T_2) is rooted ar r_1 (resp. r_2). We proceed as in [31], the only difference being we ignore the orientations of the virtual edges.

The modified definition of ordering of the 3-connected component tree is as follows:

$T_1 <_T T_2$ if one of the following holds,

1) $|T_1| < |T_2|$

2) $|T_1| = |T_2|$ and # of subtrees of r_1 is less than that of r_2 or

3) $|T_1| = |T_2|$ and # of subtrees of r_1 is equal to that of r_2 and $(T_{1,1}, \ldots, T_{1,l}) < (T_{2,1}, \ldots, T_{2,l})$ where $T_{1,1} \leq \ldots \leq T_{1,l}$ (resp. $T_{1,1} \leq \ldots \leq T_{1,l}$) are subtrees of of T_1 (resp. T_2) rooted at the children of r_1 (resp. r_2).

Here is an outline of the algorithm:

1) Compute T_1 (resp. T_2) rooted at r_1 (resp. r_2).

2) Check if $T_1 =_T T_2$ using the algorithm of [31].

By Whitney's theorem (see Theorem 8.4), twist operations on G do not change the underlying matroid, and so we get the required correctness of the algorithm. The space complexity bound follows from the arguments in [31].

The canonization of planar matroids can also done in a similar fashion following [31]. ◇

8.5.2 Matroids of bounded genus and bounded degree graphs

The *genus* of a graph is the minimum number k of handles that are required so that the graph can be drawn on a plane with k handles without any crossings of the edges. If we are given the guarantee that the input instances of GMI are graphs of bounded genus (resp. bounded degree), then in the decomposition of the graphs into 3-connected components the components obtained are themselves graphs of bounded genus (resp. bounded degree). Hence the queries made to GI are that of bounded genus (resp. bounded degree) instances which are known to be in P (see [58, 66]). Thus, as a corollary of theorem 8.19, we have:

Corollary 8.34 *Isomorphism testing of matroids of graphs of bounded genus/degree can be done in* P

8.6 Conclusion and open problems

In this chapter we studied the computational complexity of isomorphism problems for matroids on different input representations. The input representations considered in this chapter are: independent set oracle representation, linear representation and graphic representation. We have shown that isomorphism testing of bounded rank linear matroids (LMI_b) is polynomial time many-one equivalent to the graph isomorphism problem. The graphic matroid isomorphism problem GMI is shown to be polynomial time Turing reducible to GI. Conversely, GI is polynomial time many one reducible to GMI.

In addition, we find it interesting that in the bounded rank case, MI_b and LMI_b are equivalent, though there exist matroids of bounded rank which are not representable over any field. *e.g.* Lindström in [56] shows existence of an infinite class of rank-3 matroids that are not even algebraic.

Some of the open questions that arise in the context are:

- It will be interesting to prove a co-NP upper bound for LMI_b, this will imply a similar upper bound for GI.
- Are there special cases of GMI (other than what is translated from the bounds for GI) that can be solved in polynomial time?

- Note that, in the definition of LMI the field needs to have size at least m and at most $\mathsf{poly}(m)$, where m is the size of the ground set of the matroid. This is crucially needed for the observation of co-NP-hardness. One could ask if the problem is easier over fixed finite fields independent on the input. However, we note that, by our results, it follows that this problem over $\mathbb{F}_2$ is already hard for GI. It will still be interesting to give a better (than the trivial Σ_2) upper bound for linear matroids represented over fixed finite fields. For $\mathbb{F}_2$, we show an NP upper bound in the next chapter.

- Can we make the reduction from GMI to GI many-one? Can we improve the complexity of this reduction in the general case?

- Can we improve the upper bound for LMI in the case of linear matroids of poly-logarithmic rank?

- A more general project would be to study the complexity of isomorphism testing of matroids with implicit representations such as transversal matroids, bi-circular matroids etc,.

Chapter 9

Structural Complexity of Matroid Isomorphism Problems

9.1 Introduction

In this chapter we put together some of the structural results on graphic and linear matroid isomorphism problems. This is in line with the structural studies on the graph isomorphism problems ([50]).

Colouring is an important tool in the study of isomorphism problems. One of the major applications of colouring is in proving the equivalence of computing automorphism groups and testing isomorphism problems. We exhibit polynomial time computable "gadgets" so that the resultant matroids are isomorphic if and only if there exists a colour-preserving isomorphism between the original matroids. Though the idea is the same, we need different implementations for all the three types of representations: independent set oracles, linear matroids and graphic matroids.

Next we turn our attention to the complexity of computing automorphism groups for graphic and linear matroids. A "good" comparison between the problem of computing automorphism groups and isomorphism testing gives access to the well studied permutation group techniques. (see e.g., [59].) Using our colouring techniques for matroids, we show that the isomorphism testing problem and the problem of computing automorphism groups are polynomial time equivalent for linear and graphic matroids.

Ideally, as in the case of graphs, one expects efficient membership testing algorithms for automorphism groups of matroids. However, they seem to be highly non-trivial. For the case of graphic matroids, we show that this can be done in polynomial time, using the idea of "cycle basis" of graphs. As "circuit bases" exist for binary matroids (matroids representable over $\mathbb{F}_2$) we can extend this idea to get membership test for automorphism groups of binary matroids. As a consequence of this, we can show that isomorphism testing for binary matroids is in NP.

As done in [50] for the graph isomorphism problem, we define "and-functions" and "or-functions" for graphic matroid isomorphism problem (GMI). We show that, and/or-functions of GMI can be computed in polynomial time.

9.2 Colouring Techniques

Vertex or edge colouring is a classical tool used extensively in proving various results about the graph isomorphism problem. We develop similar techniques for matroid isomorphism problems too.

Recall that an edge-k-colouring of a graph $X = (V, E)$ is a function $f : E \to \{1, \ldots, k\}$. Given two coloured graphs $X_1 = (V_1, E_1, f_1)$ and $X_2 = (V_2, E_2, f_2)$, the COLOURED-GMI problem asks for an isomorphism which preserves the colours of the edges. We assume that the colours are given in unary. Not surprisingly, we can prove the following.

Lemma 9.1 COLOURED-GMI *is* AC^0 *many-one reducible to* GMI.

Proof. Let $X_1 = (V_1, E_1, f_1)$ and $X_2 = (V_2, E_2, f_2)$, be the two k-coloured graphs at the input, with $n = |V_1| = |V_2|$. For every edge $e = (u, v) \in E_1$ (respectively E_2), add a path $P_e = \{(u, v_{e,1}), (v_{e,1}, v_{e,2}), \ldots, (v_{e,n+f_1(e)}, v)\}$ of length $n + f_1(e)$ (respectively $n + f_2(e)$)where $v_{e,1}, \ldots v_{e,n+f_1(e)}$ are new vertices. Let X_1' and X_2' be the two new graphs thus obtained. By definition, any 2-isomorphism between X_1' and X_2' can only map cycles of equal length to themselves. There are no simple cycles of length more than n in the original graphs. Thus, given any 2-isomorphism between X_1' and X_2', we can recover a 2-isomorphism between X_1 and X_2 which preserves the colouring and vice versa. ◇

Now we generalize the above construction to the case of linear matroid isomorphism. COLOURED-LMI denotes the variant of LMI where the inputs are the

linear matroids M_1 and M_2 along with colour functions $c_i : \{1, \ldots, m\} \to \mathbb{N}, i \in \{1, 2\}$. The problem is to test if there is an isomorphism between M_1 and M_2 which preserves the colours of the column indices. We have,

Lemma 9.2 COLOURED-LMI *is* AC^0 *many-one reducible to* LMI.

Proof. Let M_1 and M_2 be two coloured linear matroids represented over a field $\mathbb{F}$. We illustrate the reduction where only one column index of M_1 (resp. M_2) is coloured. Without loss of generality, we assume that there are no two vectors in M_1 (resp.M_2) which are scalar multiples of each other.

We transform M_1 and M_2 to get two matroids M_1' and M_2'. In the transformation, we add more columns to the matrix (vectors to the ground set) and create dependency relations in such a way that any isomorphism between the matroids must map these new vectors in M_1 to the corresponding ones M_2.

We describe this transformation in a generic way for a matroid M. Let $\{e_1, \ldots, e_m\}$ be the column vectors of M, where $e_i = \langle e_{i,1}, \ldots e_{i,n} \rangle \in \mathbb{F}^n$. Let $e = e_1$ be the coloured vector in M.

Choose $m' > m$, we construct $\ell = m + m'$ vectors $f_1, \ldots f_\ell \in \mathbb{F}^{n+m'}$ as the columns of the following $(n + m') \times \ell$ matrix. The i^{th} column of the matrix represents f_i.

$$\left[\begin{array}{cccc|cccccccc}
e_{11} & e_{21} & \ldots & e_{m1} & e_{11} & 0 & \ldots & 0 & 0 & \ldots & 0 \\
e_{12} & e_{22} & \ldots & e_{m2} & 0 & e_{12} & \ldots & 0 & 0 & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
e_{1m} & e_{2m} & \ldots & e_{mm} & 0 & 0 & \ldots & e_{1m} & 0 & \ldots & 0 \\
\hline
0 & 0 & \ldots & 0 & 1 & -1 & 0 & 0 & \ldots\ldots & & 0 \\
\vdots & \vdots & \ldots & \vdots & 0 & 1 & -1 & 0 & \ldots\ldots & & 0 \\
\vdots & \vdots & \ldots & \vdots & \vdots & \vdots & \ddots & \ddots & \ldots\ldots & & \vdots \\
0 & 0 & \ldots & 0 & 0 & 0 & \ldots & & 0 & 1 & -1 \\
0 & 0 & \ldots & 0 & -1 & 0 & \ldots & & 0 & 0 & 1
\end{array}\right]$$

where -1 denotes the additive inverse of 1 in $\mathbb{F}$. Denote the above matrix as $M' = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$. Let $S = \{f_{m+1}, \ldots, f_{m+m'}\}$. We observe the following:

1. Columns of B generate e_1. Since C is a 0-matrix $f_1 \in Span(S)$.

2. Columns of D are minimal dependent. Any proper subset of columns of D will split the 1, -1 pair in at least a row and hence will be independent.

3. S is linearly independent. Suppose not. Let $\sum_{i=m}^{m+m'} \alpha_i f_i = 0$. Restricting this to the columns of B gives that $\alpha_j = 0$ for first j such that $e_{1j} \neq 0$. Thus this gives a linearly dependent proper subset of columns of D, and contradicts the above observation.

4. If for any $f \in Span(f_1, \ldots, f_m)$, $f = \sum_{f_i \in S} \alpha_i f_i$, then α_i's must be the same.

Now we claim that the newly added columns respect the circuit structure involving e_1. Let $\mathcal{C}$ and $\mathcal{C}'$ denote the sets of circuits of M and M' respectively. Fix any subset $E = \{e_1, e_{i_2}, \ldots, e_{i_k}\}$. Define $G = \{f_{i_2}, \ldots, f_{i_k}\}$. $F_1 = \{f_1\} \cup G$ and $F_2 = G \cup S$.

Claim 9.3 *1.* $E \in \mathcal{C} \iff F_1 \in \mathcal{C}'$

2. $F_1 \in \mathcal{C}' \iff F_2 \in \mathcal{C}'$

Proof. Clearly, $E \in \mathcal{C}$ if and only if $F_1 \in \mathcal{C}'$. Now we prove 2. Suppose $F_1 \in \mathcal{C}'$. Then F_2 is dependent as $f_1 \in Span(S)$. Suppose F_2 is not minimally dependent and let $F' \subset F_2$ be dependent.

Then clearly, $F' \not\subseteq S$ and $F' \not\subseteq G$. Also, since no proper subset of S can generate the all 0s vector when restricted to the matrix D, we have $S \subseteq F'$ and $G \cap F' \subset G$. Hence, we have

$$0 = \sum_{g \in G \cap F'} \gamma_g g + \sum_{s \in S} \delta_s s$$

Now, by property 4, all the δ_ss are the same and hence we have

$$f_1 = \sum_{g \in G \cap F'} \gamma_g g$$

which is a contradiction since $G \cap F' \subset G$. The converse direction can also be argued in a similar manner. $\diamond$

From the above observations and the fact that there is no other column in M which is a multiple of e, the set $f(e) = \{f_1, f_{m+1}, \ldots, f_{m+m'}\}$ is a unique circuit of size $m' + 1$ in M', where e is the column which is coloured.

Now we argue about the isomorphism between M_1' and M_2' obtained from the above operation. Note that there is a unique circuit of length $m'+1 > m$ in both M_1' and M_2' corresponding to two vectors $e \in M_1$ and $e' \in M_2$. Hence any matroid isomorphism should map these sets to each other. From such an isomorphism, we can recover an isomorphism between M_1 and M_2 that pairs off e and e', thus preserving the colours. Indeed, if there is a matroid isomorphism between M_1 and M_2, that can easily be extended to M_1' and M_2'.

For the general case, let k be the number of different colour classes and c_i denote the size of the ith colour class. Then for each vector e in the color class i, we add $l_i = m+m'+i$ many new vectors, which also increases the dimension of the space by l_i. Thus the total number of vectors in the new matroid is $\sum c_i(l_i) \leq m^3$. Similarly, the dimension of the space is bounded by m^3. This completes the proof of Lemma 9.2. ◇

We can further generalize the above idea to matroids given in the form of independent set oracles. We define COLOURED-MI as the variant of MI where the inputs are matroids $M_1 = (S_1, \mathcal{I}_1)$ and $M_2 = (S_2, \mathcal{I}_2)$ given as independent set oracles along with colour functions $c_i : \{1, \ldots, m\} \to \{1, \ldots, m\}, i \in \{1, 2\}$. (Here $m = |S_1| = |S_2|$.) We assume that the colour functions are part of the input and not in the oracle. The problem is to test if there is an isomorphism between M_1 and M_2 which preserves the colours of the ground set elements. We have,

Lemma 9.4 COLOURED-MI *is polynomial time many-one reducible to* MI

Proof. Let $M_1 = (S_1, \mathcal{I}_1)$ and $(S_2, \mathcal{I}_2)$ be the given matroids, c_1 and c_2 be their colour classes. Let $m = |S_1| = |S_2|$. We demonstrate colouring for a singleton colour class. Suppose $c_1(e) = i$. Let $m' = m + i$. As done in lemma 9.1, we need to introduce a "large" (new) circuit C that contains e. We construct matroid M_1' (resp. M_2') as follows.

1. Let $F_1 = \{f_1, \ldots, f_{m'}\}$ be new ground set elements. Let $S_1' = S_1 \cup F_1$.
2. All circuits of M_1 remain to be so in M_1'.
3. Let $\{f_1, \ldots, f_{m'}, e\}$ be a circuit in M_1'.
4. If C is a circuit in M_1 containing e, then $(C \setminus \{e\}) \cup F_1$ is a circuit in M_1'

To see that M_1' is a matroid, we need a circuit based characterization of matroids. A set $\mathcal{C}$ of subsets of S defines circuits of a matroid on S if and only if it satisfies the *circuit elimination axioms*, which are:

- $\emptyset \notin \mathcal{C}$;
- If $A \in \mathcal{C}$ then for all $B \subset A$, $B \notin \mathcal{C}$; and
- For all $C_1 \neq C_2 \in \mathcal{C}$ and $e \in C_1 \cap C_2$, the set $(C_1 \cup C_2) \setminus \{e\}$ contains a circuit.

It is known that the set of circuits uniquely defines a matroid. (See [69] for more details.) Now by doing a case analysis it is not hard to see that the sets of circuits of M_1' defined above satisfy the above properties. Hence M_1' is a matroid. We construct M_2' analogously. Now using the arguments from Lemma 9.1 it follows that M_1' and M_2' satisfy the required property: $M_1' \cong M_2' \iff$ there is a colour-preserving isomorphism between M_1 and M_2. However we need to show how to implement independent set oracles for M_1' and M_2' in polynomial time using access to those of M_1 and M_2 respectively. This can be done by the following algorithm (this is shown for M_1', the case of M_2' can be handled analogously):
Input: $A \subseteq S_1'$
Output: YES if and only if A is independent in M_1'

1. If $A \subseteq S_1$, then return YES if and only if $A \in \mathcal{I}_1$.
2. If $F_1 \cup \{e\} \subseteq A$, then return NO,
3. If $F_1 \subseteq A$ but $e \notin A$, then return YES if and only if $(A \setminus F_1) \cup \{e\} \in \mathcal{I}_1$.
4. If $F_1 \cap A \neq \emptyset$ and F_1 is not contained in A, then return YES if and only if $(A \setminus F_1) \in \mathcal{I}_1$.

$\diamond$

9.3 Complexity of computing automorphism groups

With any isomorphism problem, there is an associated automorphism problem i.e, to find a generating set for the automorphism group of the underlying object.

Relating the isomorphism problem to the corresponding automorphism problem gives access to algebraic tools associated with the automorphism groups. In the case of graphs, studying automorphism problem has been fruitful.(e.g. see [58, 15, 10].) In this section we turn our attention to Matroid automorphism problems on the different input settings.

Recall that an automorphism of a matroid $M = (S, \mathcal{C})$ (where S is the ground set and $\mathcal{C}$ is the set of circuits) is a permutation ϕ of elements of S such that $\forall C \subseteq S,\ C \in \mathcal{C} \iff \phi(C) \in \mathcal{C}$. $Aut(M)$ denotes the group of automorphisms of the matroid M. When the matroid is graphic we denote by $Aut(X)$ and $Aut(M(X))$ the automorphism group of the graph and the graphic matroid respectively.

9.3.1 Relationship with isomorphism testing

As in the case of graphs, we can define automorphism problems for matroids.

Matroid Automorphism(MAuto)*: Given a matroid M as independent set oracle, compute a generating set for $Aut(M)$.*

We define GMAuto and LMAuto as the corresponding automorphism problems for graphic and linear matroids, when the input is a graph and matrix respectively. Using the colouring techniques from Section 9.2, we prove the following equivalence.

Theorem 9.5 LMI $\equiv_T^p$ LMAuto*, and* GMI $\equiv_T^p$ GMAuto.

Proof. This proof follows a standard idea due to Luks [59]. We show the forward direction as follows. Given two matrices M_1 and M_2, form the new matrix M as,

$$M = \begin{bmatrix} M_1 & 0 \\ 0 & M_2 \end{bmatrix}$$

Now using queries to LMAuto construct the generating set of $Aut(M)$. Check if at least one of the generators map the columns in M corresponding to columns of M_1 to those corresponding to the columns of M_2.

To see the other direction, we use the colouring idea, and the rest of the details is standard. The idea is to find the orbits of each element of the ground set as follows: For every element of $e \in S$, for each $f \in S$, colour e and f by the same colour to obtain coloured matroids M_1 and M_2. Now by querying to LMI we can

check if f is in the orbit of e. Thus we can construct the orbit structure of $Aut(M)$ and hence compute a generating set.

Using similar methods we can prove GMI $\equiv_T^p$ GMAUTO using the colouring techniques from Section 9.2. $\diamond$

9.3.2 Membership tests

Here we discuss algorithms for testing memberships in the automorphism groups of graphic matroids and then extend it to include binary matroids.

Given a graph X, and a permutation $\pi \in S_m$, it is not clear a priori how to check if $\pi \in Aut(M(X))$ efficiently. This is because we need to ensure that π preserves all the simple cycles, and there could be exponentially many of them. Note that such a membership test (given a $\pi \in S_n$) for $Aut(X)$ can be done easily by testing whether π preserves all the edges. We provide an efficient membership test for $Aut(M(X))$ here.

We use the notion of a cycle bases of X. A *cycle basis* of a graph X is a minimal set $\mathcal{B}$ of cycles of X such that every cycle in X can be written as a linear combination (viewing every cycle as a vector in $\mathbb{F}_2^m$) of the cycles in $\mathcal{B}$. Let $\mathscr{B}$ denote the set of all cycle basis of the graph X.

Lemma 9.6 *For any $\pi \in S_m$, $\exists \mathcal{B} \in \mathscr{B} : \pi(\mathcal{B}) \in \mathscr{B} \implies \forall \mathcal{B} \in \mathscr{B} : \pi(\mathcal{B}) \in \mathscr{B}$.*

Proof. Let $\mathcal{B} = \{b_1, \ldots b_\ell\} \in \mathscr{B}$ such that $\pi(\mathcal{B}) = \{\pi(b_1), \ldots, \pi(b_\ell)\}$ is a cycle basis. Then for all cycles b, $\pi(b)$ is a cycle. Now consider any other cycle basis $\mathcal{B}' = \{b'_1, \ldots, b'_k\} \in \mathscr{B}$. Thus, $b_i = \sum_j \alpha_j b'_j$. This implies,

$$\pi(b_i) = \sum_j \alpha_j \pi(b'_j).$$

Thus, $\pi(B') = \{\pi(b'_1), \ldots, \pi(b'_\ell)\}$ forms a cycle basis. $\diamond$

Lemma 9.7 *Let $\pi \in S_m$, and let $\mathcal{B} \in \mathscr{B}$, then $\pi \in Aut(M(X)) \iff \pi(\mathcal{B}) \in \mathscr{B}$.*

Proof. Let $\mathcal{B} = \{b_1, \ldots, b_\ell\}$ be the given cycle basis.

For the forward direction, suppose $\pi \in Aut(M(X))$. That is, $C \subseteq E$ is a cycle in X if and only if $\pi(C)$ is also a cycle in X. Let C be any cycle in X, and let

$D = \pi^{-1}(C)$. Since $\mathcal{B} \in \mathscr{B}$, we can write, $D = \sum_i \alpha_i b_i$, and hence $C = \sum_i \alpha_i \pi(b_i)$. Hence $\pi(\mathcal{B})$ forms a cycle basis for X.

For the reverse direction, suppose $\pi(\mathcal{B})$ is a cycle basis of X. Let C be any cycle in X. We can write $C = \sum_i \alpha_i b_i$. Hence, $\pi(C) = \sum_i \alpha_i \pi(b_i)$. As π is a bijection, we have $\pi(b_i \cap b_j) = \pi(b_i) \cap \pi(b_j)$. Thus $\pi(C)$ is also a cycle in X. Since π extends to a permutation on the set of cycles, we conclude that C is a cycle if and only if $\pi(C)$ is a cycle. Hence π is an automorphism. $\diamond$

Using Lemmas 9.6 and 9.7 it follows that, given a permutation π, to test if $\pi \in Aut(M(X))$ it suffices to check if for a cycle basis $\mathcal{B}$ of X, $\pi(\mathcal{B})$ is also a cycle basis. Given a graph X a cycle basis $\mathcal{B}$ can be computed in polynomial time, e.g, by taking all the fundamental circuits of a spanning forest. Now it suffices to show:

Lemma 9.8 *Given a permutation $\pi \in S_m$, and a cycle basis $\mathcal{B} \in \mathscr{B}$, testing whether $\pi(\mathcal{B})$ is a cycle basis, can be done in polynomial time.*

Proof. To check if $\pi(\mathcal{B})$ is a cycle basis, it is sufficient to verify that every cycle in $\mathcal{B} = \{b_1, \ldots, b_\ell\}$ can be written as a $\mathbb{F}_2$-linear combination of the cycles in $\mathcal{B}' = \{b'_1, \ldots, b'_\ell\} = \pi(\mathcal{B})$. This can be done as follows.

For $b_i \in \mathcal{B}$, let $\pi(b_i) = b'_i$. View b_i and b'_i as vectors in $\mathbb{F}_2^m$. Let b_{ij} (resp. b'_{ij}) denote the j^{th} component of b_i (resp. b'_i). Construct the set of linear equations, $b'_{ij} = \sum_{b_k \in \mathcal{B}} x_{ik} b_{kj}$ where x_{ik} are unknowns. There are exactly ℓ b'_is and each of them gives rise to exactly m equations like this. This gives a system I of ℓm linear equations in ℓ^2 unknowns such that, $\pi(B)$ is a cycle basis if and only if I has a non-trivial solution. This test can indeed be done in polynomial time. $\diamond$

This gives us the following:

Theorem 9.9 *Given a graph X and a $\pi \in S_m$, testing if $\pi \in Aut(M(X))$ is in* P.

Let M be a binary matroid on $S = v_1, \ldots, v_n \in \mathbb{F}_2^r$. For $A \subseteq S$, define the incidence vector of A as $\alpha \in \mathbb{F}_2^n$, where $\alpha_i = 1$ if and only if $v_i \in A$. The "circuit space" of M is the sub-space of $\mathbb{F}_2^n$ spanned by the incidence vectors of all circuits of M.

For a graph X, the simple cycles of X are in one-to-one correspondence with the circuits of $M(X)$. Thus clearly, cycle space of X corresponds to the circuit space of $M(X)$ when $M(X)$ is viewed as a binary matroid. It is not hard to see that the Lemmas 9.6,9.7 and 9.8 can be extended to any matroid that has circuit

bases. So, in order to extend theorem 9.9 to binary matroids, we need to show that a circuit basis of a binary matroid can be computed efficiently. However, using the proposition 9.2.2 and corollary 9.2.3 in [69], we can do this efficiently. (See appendix for more details.) Thus we have,

Theorem 9.10 *Let M be a binary matroid with n column vectors.. Given any $\pi \in S_n$, we can test if $\pi \in Aut(M)$ in polynomial time.*

As a consequence of the above theorem, we show that isomorphism testing of binary matroids can be done in NP.

Corollary 9.11 *Testing if two binary matroids are isomorphic is in* NP

Proof. Let M_1 and M_2 be given binary matroids. Define,

$$M = \begin{bmatrix} M_1 & 0 \\ 0 & M_2 \end{bmatrix}$$

Clearly $M_1 \cong M_2$ if and only if there is a $\pi \in Aut(M)$ that maps vectors from M_1 to those of M_2 and vice versa. The NP algorithm is: guess a permutation $\pi \in S_n$ check if $\pi \in Aut(M)$ and if π maps column vectors of M_1 to those of M_2 and vice-versa. (The second part can be done in polynomial time using Theorem 9.10.) $\diamond$

9.4 Closure Properties

In this section we consider taking and-function and or-functions of polynomial many instances of GMI. Following [50], we formally define and-functions and or-functions as follows:

Definition 9.12 *(see [50, 57]) Let A be any language in $\{0,1\}^*$. An* or-function *for A is a function $f : \{0,1\}^* \to \{0,1\}^*$ such that for every sequence $x_1, \ldots, x_\ell \in \{0,1\}^*$ we have,*

$$f(x_1, \ldots, x_\ell) \in A \iff \exists i \in [\ell],\ x_i \in A$$

Similarly, an and-function for A is a function $g : \{0,1\}^* \to \{0,1\}^*$ such that for all $x_1, \ldots, x_\ell \in \{0,1\}^*$ the following holds:

$$g(x_1, \ldots, x_\ell) \in A \iff \forall i \in [\ell],\ x_i \in A$$

We show that GMI restricted to 2-connected graphs is closed under polynomial bounded and-functions and or-functions. The restriction of the input graphs is necessary as the reduction from the case of general GMI to 2-connected GMI is not many-one. We prove:

Theorem 9.13 GMI *restricted to 2-connected graphs has polynomial time computable and-functions and or-functions.*

Proof. Our proof follows closely the proof of closure properties of and/or-functions for GI given in [50].

and-function: Let $(G_1, H_1), \ldots, (G_\ell, H_\ell)$ be ℓ different instances of GMI where all the graphs are 2-connected. We demonstrate the construction for $\ell = 2$.

Let $G_1 = (V_1, E_1), G_2 = (V_2, E_2), H_1 = (V_1', E_1'), H_2 = (V_1', E_1')$, $|V_1| = |V_1'| = n_1$, $|V_2| = |V_2'| = n_2$ and $|E_1| = |E_1'| = m_1$, $|E_2| = |E_2'| = m_2$. We construct two graphs $G = \langle G_1, G_2 \rangle$ and $H = \langle H_1, H_2 \rangle$ such that $G \cong_2 H \iff (G_1 \cong_2 H_1$ and $G_2 \cong_2 H_2)$. Intuitively, the vertex set V of G consists of G_1 and G_2 with four additional vertices u^1, u^2, v^1, v^2. Add (u^1, u^2) and (v^1, v^2) as edges. Now for every edge $e = (a, b) \in E_1$, add new edges so that the subgraph induced by $\{u^1, u^2, a, b\}$ is a 4-vertex clique. Similarly for every $e = (a, b) \in E_1 \cup E_2$, add new edges to G so that the subgraph induced by $\{v^1, v^2, a, b\}$ forms a 4-vertex clique.

Define $G = (V, E)$ (resp. $H = (V', E')$) as follows;

$$V = V_1 \cup V_2 \cup \{u^1, u^2, v^1, v^2\}$$

$$\begin{aligned} E \;=\; & E_1 \cup E_2 \cup \{(u^1, u^2), (v^1, v^2)\} \\ & \cup \{(u^i, a), (u^i, b) \mid (a, b) \in E_1, i \in \{1, 2\}\} \\ & \cup \{(v^i, a), (v^i, b) \mid (a, b) \in E_1 \cup E_2,\ i \in \{1, 2\}\} \end{aligned}$$

We define H in a similar fashion using H_1 and H_2 instead of G_1 and G_2 respectively. We denote the four new vertices thus introduced in H by $\bar{u}^1, \bar{u}^2, \bar{v}^1, \bar{v}^2$.

Now the following claim completes the proof for and-function:

Claim 9.14 *($G_1 \cong_2 H_1$ and $G_2 \cong H_2$) $\iff$ $G \cong_2 H$*

Proof. [of claim] The forward direction is easy to see. To prove the converse, suppose $G \cong_2 H$ via a bijection $\phi : E \to E'$. Let $e_{m+1} = (u^1, u^2), e_{m+2} = (v^1, v^2)$ and $\bar{e}_{m+1} = (\bar{u}^1, \bar{u}^2), e_{m+2} = (\bar{v}^1, \bar{v}^2)$. Now, as e_{m+1} (resp. $\bar{e}_{m+1}$) is the unique edge in G that intersects with n_1 many 4-vertex cliques, we have $\phi(e_{m+1}) = \bar{e}_{m+1}$. Similarly we can argue that $\phi(e_{m+2}) = \bar{e}_{m+2}$. Also, all the newly introduced edges of G get mapped to those of H. Thus we can recover the required 2-isomorphisms between G_1, H_1 and G_2, H_2 respectively. ◇

Now notice that we introduced only 8 (4 each for G and H) new vertices. In the case of $\ell > 2$ we do the above process iteratively. At each iteration we add 8 new vertices, hence the final graphs will have number of vertices bounded by $n + 8\ell$ (where n is the total number of vertices in the graphs we began with). As the graphs obtained are always simple, the number of edges is also bounded by $O((n + 8\ell)^2)$. Also, it is straightforward to see that the computation of the resulting graphs can be done in polynomial time.

or-function: Let (G_1, H_1) and (G_2, H_2) be two instances of GMI. Now define the function f as:

$$f((G_1, H_1), (G_2, H_2)) = (\langle G_1, G_2 \rangle \cup \langle H_1, H_2 \rangle, \langle G_1, H_2 \rangle \cup \langle H_1, G_2 \rangle)$$

From the arguments in the above paragraphs, it is easy to see that f represents the or-function of (G_1, H_1) and (G_2, H_2). However, extending this directly for polynomial many instances will cause an exponential blow up in size. We use the divide and conquer approach as done in Theorem 1.42 of [50].

Let $x_i = (G_i, H_i)$, $1 \leq i \leq \ell$ be the given sequence of instances of GMI. We define the function $\bar{f}$ as follows:

$$\bar{f}(x_1, \ldots, x_\ell) = \begin{cases} x_1 & if\ \ell = 1 \\ f(\bar{f}(x_1, \ldots, x_{\lceil \ell/2 \rceil}), \bar{f}(x_{\lceil \ell/2 \rceil + 1}, \ldots, x_\ell)) & otherwise \end{cases}$$

From the definition, the depth of recursion is $O(\log \ell)$. At each step the application of f blows up the size by a constant factor. Thus the size of the graph $\bar{f}(x_1, \ldots, x_\ell)$ is bounded by $\mathsf{poly}(\ell)$. Now using the arguments similar to the one in the proof of Theorem 1.42 of [50] we get the desired result. $\diamond$

Remark Note that the theorem 9.13 cannot be directly applied to graphs that are not 2-connected. This is mainly because our reduction from the connected GMI instance to 2-connected instance is a Turing reduction and not a many-one reduction. (See discussions preceding lemma 8.18.)

9.5 Conclusion and open problems

In this chapter we introduced techniques to handle coloured matroid isomorphism problem and shown that coloured matroid isomorphism is polynomial time reducible to standard isomorphism questions for all three types of representations: independent set oracle representation, linear matroids and graphic matroids. As an application of this, we have shown that in the case of graphic and linear matroids, isomorphism testing is polynomial time equivalent to finding generating sets for automorphism groups. We have also exhibited polynomial time membership test for graphic and binary matroids. Finally we studied and/or-functions for graphic matroids and have shown that they are polynomial time computable.

We conclude the chapter with the following open questions:

- Can the membership testing algorithm in Section 9.3 be extended to linear matroids represented over fixed finite field $\mathbb{F}_q$ for $q > 2$? Note that this would imply an NP upper bound for the corresponding isomorphism problem.
- Can the and/or-functions of linear matroids and general matroids given by independent set oracles be computed in polynomial time? Note that an obvious extension of the ideas in Section 9.4 does not work, as the size of the ground set will grow by a polynomial factor (rather than constant increase in the number of vertices the case of graphic matroids) and hence the overall size even for and-function will be exponential.

9.6 Appendix

9.6.1 B-Fundamental circuit incidence matrix

In this section we briefly discuss the construction of circuit basis of a binary matroid. First we introduce the notion of B-fundamental circuit for a basis B of a linear matroid. Let M be a matroid of rank r, without loss of generality, assume that M is represented as the matrix $A \in \mathbb{F}^{r\times n}$, where $A = [I_r|D]$, I_r is the $r \times r$ identity matrix. Let the columns of A be labeled as $v_1, \ldots, v_n$, in that order. Let B be the basis $\{v_1, \ldots, v_r\}$ of M. Let $D^{\#}$ be the matrix obtained from D by replacing every non-zero element by 1. The matrix $D^{\#}$ is called the *B-fundamental-circuit incidence matrix* of M.

In the following, A^T denotes the transpose of the matrix A.

Let N be an n-element binary matroid of rank r and B be a basis of N, where $r \leq n-1$. Then,

Proposition 9.15 (Corollary 9.2.3 in [69]) *Let X be the B-fundamental circuit matrix of N. Then the subspace spanned by the rows of the matrix $[X^T|I_{n-r}]$ is the circuit space of M.*

By the above proposition, given a binary matroid N, one can compute its circuit basis as follows:

- Choose a basis B of N, without loss of generality, we can assume that $B = I_r$.
- Compute the B-fundamental circuit matrix X of N.
- Output the rows of the matrix $[X^T|I_{n-r}]$ as the circuit basis.

It is not hard to see that the above procedure runs in polynomial time.

Bibliography

[1] Manindra Agrawal, Eric Allender, and Samir Datta. On TC^0, AC^0 and arithmetic circuits. In *Proc 12th IEEE Computational Complexity*, CCC'97, pages 134–148. IEEE Computer Society, 1997.

[2] Manindra Agrawal and Thomas Thierauf. The Formula Isomorphism Problem. *SIAM J. Comput.*, 30(3):990–1009, 2000.

[3] E. Allender. The division breakthroughs. *BEATCS: Bulletin of the European Association for Theoretical Computer Science*, 74, 2001.

[4] E. Allender. Arithmetic circuits and counting complexity classes. In Jan Krajicek, editor, *Complexity of Computations and Proofs*, Quaderni di Matematica Vol. 13, pages 33–72. Seconda Universita di Napoli, 2004. An earlier version appeared in the Complexity Theory Column, SIGACT News 28, 4 (Dec. 1997) pp. 2-15.

[5] Eric Allender, Andris Ambainis, David A.Mix Barrington, Samir Datta, and Huong LêThanh. Bounded depth arithmetic circuits: Counting and closure. In *International Colloquium on Automata, Languages, and Programming ICALP*, ICALP'99, pages 149–158, 1999.

[6] Eric Allender and Mitsunori Ogihara. Relationships among pl, #l, and the determinant. In *Structure in Complexity Theory Conference*, pages 267–278, 1994.

[7] Andris Ambainis, David A. Mix Barrington, and Huong LeThanh. On counting AC^0 circuits with negative constants. In *MFCS*, pages 409–417, 1998.

[8] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 2009.

[9] Vikraman Arvind and Johannes Köbler. Graph isomorphism is low for zpp(np) and other lowness results. In *STACS*, pages 431–442, 2000.

[10] Vikraman Arvind and Piyush P. Kurur. Graph isomorphism is in SPP. In *FOCS*, pages 743–750, 2002.

[11] Vikraman Arvind and Jacobo Torán. Solvable group isomorphism. In *IEEE Conference on Computational Complexity*, pages 91–103, 2004.

[12] Vikraman Arvind and Jacobo Torán. Isomorphism testing: Perspective and open problems. *Bulletin of the EATCS*, 86:66–84, 2005.

[13] Làszlò Babai. Vector Representable Matroids of Given Rank with Given Automorphism Group. *Discrete Math.*, 24:119–125, 1978.

[14] László Babai and Lance Fortnow. Arithmetization: A new method in structural complexity theory. *Computational Complexity*, 1:41–66, 1991.

[15] László Babai, D. Yu. Grigoryev, and David M. Mount. Isomorphism of graphs with bounded eigenvalue multiplicity. In *STOC*, pages 310–324, 1982.

[16] Theodore P. Baker, John Gill, and Robert Solovay. Relativizatons of the p =? np question. *SIAM J. Comput.*, 4(4):431–442, 1975.

[17] David.A.Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38(1):150–164, 1989.

[18] Michael Ben-Or and Richard Cleve. Computing algebraic formulas using a constant number of registers. *SIAM J. Comput.*, 21(1):54–58, 1992.

[19] Lenore Blum, Felipe Cucker, Mike Shub, and Steve Smale. *Complexity and Real Computation*. Springer, 1997.

[20] A. Borodin, S. Cook, P. Dymond, W. Ruzzo, and M. Tompa. Two applications of inductive counting for complementation problems. *SIAM Journal of Computing*, 18(3):559–578, 1989.

[21] A. Borodin, A. Razborov, and R. Smolensky. On lower bounds for read-k-times branching programs. *Comput. Complex.*, 3(1):1–18, 1993.

[22] Richard P. Brent. The parallel evaluation of arithmetic expressions in logarithmic time. In *Complexity of sequential and parallel numerical algorithms (Proc. Sympos., Carnegie-Mellon Univ., Pittsburgh, Pa., 1973)*, pages 83–102. Academic Press, New York, 1973.

[23] P Bürgisser, M. Clausen, and M.A. Shokrollahi. *Algebraic Complexity Theory*. Springer-Verlag, 1997.

[24] Peter Bürgisser. On the structure of valiant's complexity classes. In Michel Morvan, Christoph Meinel, and Daniel Krob, editors, *STACS*, volume 1373 of *Lecture Notes in Computer Science*, pages 194–204. Springer, 1998.

[25] Peter Bürgisser. *Completeness and Reduction in Algebraic Complexity Theory*. Algorithms and Computation in Mathematics. Springer-Verlag, 2000.

[26] S. Buss. The Boolean formula value problem is in ALOGTIME. In *Proceedings of the ACM Symposium on Theory of Computing STOC*, pages 123–131, 1987.

[27] S. Buss, S. Cook, A. Gupta, and V. Ramachandran. An optimal parallel algorithm for formula evaluation. *SIAM Journal on Computing*, 21(4):755–780, 1992.

[28] Hervé Caussinus, Pierre McKenzie, Denis Thérien, and Heribert Vollmer. Nondeterministic NC^1 computation. *Journal of Computer and System Sciences*, 57:200–212, 1998.

[29] A Chiu, G Davida, and B Litow. Division in logspace-uniform NC^1. *RAIRO Theoretical Informatics and Applications*, 35:259–276, 2001.

[30] Stephen A. Cook. Deterministic CFL's are accepted simultaneously in polynomial time and log squared space. In *Proceedings of the ACM Symposium on Theory of Computing STOC*, pages 338–345, 1979.

[31] Samir Datta, Nutan Limaye, Prajakta Nimbhorkar, Thomas Thierauf, and Fabian Wagner. A log-space algorithm for canonization of planar graphs. In *IEEE comference on Computational Complexity, CCC*, 2009.

[32] Paulin Jacobé de Naurois. A Measure of Space for Computing over the Reals. In *CiE*, pages 231–240, 2006.

[33] Ding-Zhu Du and Ker-I Ko. *Theory of Computational Complexity.* Wiley International, 2000.

[34] Stephen A. Fenner, Lance Fortnow, and Stuart A. Kurtz. Gap-definable counting classes. In *Structure in Complexity Theory Conference*, pages 30–42, 1991.

[35] Stephen A. Fenner, Lance Fortnow, and Lide Li. Gap-definability as a closure property. In Patrice Enjalbert, Alain Finkel, and Klaus W. Wagner, editors, *STACS*, volume 665 of *Lecture Notes in Computer Science*, pages 484–493. Springer, 1993.

[36] Lancs Fortnow. Counting complexity. In *Complexity Theory Retrospective II*, pages 81–107. Springer, 1997. Survey.

[37] Oded Goldreich. *Computational Complexity: A Conceptual Perspective.* Cambridge University Press, 2008.

[38] Dima Grigoriev and Marek Karpinski. An exponential lower bound for depth 3 arithmetic circuits. In *STOC*, pages 577–582, 1998.

[39] Dima Grigoriev and Alexander A. Razborov. Exponential complexity lower bounds for depth 3 arithmetic circuits in algebras of functions over finite fields. In *FOCS*, pages 269–278, 1998.

[40] F. Harary. *Graph theory.* Addison Wesley, 1969.

[41] Petr Hlinený. On matroid representability and minor problems. In Rastislav Kralovic and Pawel Urzyczyn, editors, *MFCS*, volume 4162 of *Lecture Notes in Computer Science*, pages 505–516. Springer, 2006.

[42] Petr Hlinený. Some hard problems on matroid spikes. *Theory of Computing Systems*, 41(3):551–562, 2007.

[43] J. E. Hopcroft and J. K. Wong. Linear time algorithm for isomorphism of planar graphs (preliminary report). In *STOC '74: Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 172–184, New York, NY, USA, 1974. ACM.

[44] J.E. Hopcroft and R.E. Tarjan. Dividing a graph into triconnected components. *SIAM Journal of Computing*, 2(3):135–158, 1973.

[45] John E. Hopcroft and Robert Endre Tarjan. A v^2 algorithm for determining isomorphism of planar graphs. *Inf. Process. Lett.*, 1(1):32–34, 1971.

[46] Sorin Istrail and Dejan Zivkovic. Bounded width polynomial size Boolean formulas compute exactly those functions in AC^0. *Information Processing Letters*, 50:211–216, 1994.

[47] Maurice J. Jansen. Lower bounds for syntactically multilinear algebraic branching programs. In *MFCS*, pages 407–418, 2008.

[48] Birgit Jenner, Johannes Köbler, Pierre McKenzie, and Jacobo Torán. Completeness results for graph isomorphism. *J. Comput. Syst. Sci.*, 66(3):549–566, 2003.

[49] Stasys Jukna. *Extremal Combinatorics: with Applications in Computer Science*. Springer, 2001.

[50] Johannes Köbler, Uwe Schöning, and Jacobo Torán. *The graph isomorphism problem: its structural complexity*. Birkhauser Verlag, Basel, Switzerland, Switzerland, 1993.

[51] Pascal Koiran and Sylvain Perifel. VPSPACE and a Transfer Theorem over the Complex Field. In *MFCS*, pages 359–370, 2007.

[52] Pascal Koiran and Sylvain Perifel. VPSPACE and a Transfer Theorem over the Reals. In Thomas and Weil [82], pages 417–428.

[53] Daniel Král. Computing representations of matroids of bounded branch-width. In Thomas and Weil [82], pages 224–235.

[54] Richard E. Ladner. On the structure of polynomial time reducibility. *J. ACM*, 22(1):155–171, 1975.

[55] Steven Lindell. A logspace algorithm for tree canonization (extended abstract). In *STOC*, pages 400–404, 1992.

[56] Bernt Lindström. A class of nonalgebraic matroids of rank three. *Geom. Dedicata.*, 23(3):255–258, 1987.

[57] Antoni Lozano and Jacobo Torán. On the nonuniform complexity of the graph isomorphism problem. In *Structure in Complexity Theory Conference*, pages 118–129, 1992.

[58] Eugene M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. In *FOCS*, pages 42–49, 1980.

[59] Eugene.M Luks. *Permutation groups and polynomial-time computation*, volume 11 of *DIMACS*, pages 139–175. American Mathematical Society, 1993.

[60] Guillaume Malod. The complexity of polynomials and their coefficient functions. In *IEEE Conference on Computational Complexity*, pages 193–204, 2007.

[61] Guillaume Malod and Natacha Portier. Characterizing Valiant's algebraic complexity classes. In *MFCS*, pages 704–716, 2006.

[62] Dillon Mayhew. *Matroids and Complexity*. PhD thesis, University of Oxford, 2005.

[63] Dillon Mayhew. Matroid complexity and nonsuccinct descriptions. *SIAM Journal of Discrete Mathematics*, 22(2):455–466, 2008.

[64] Christian Michaux. Une remarque à propos des machines sur $\mathbb{R}$ introduites par Blum, Shub et Smale. *Comptes Rendus de l'Académie des Sciences de Paris*, 309(7):435–437, 1989.

[65] Gary L. Miller. On the n^{log} isomorphism technique: A preliminary report. In *STOC*, pages 51–58, 1978.

[66] Gary L. Miller. Isomorphism testing for graphs of bounded genus. In *STOC*, pages 225–235, 1980.

[67] Noam Nisan. Lower bounds for non-commutative computation. In *STOC '91: Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 410–418, New York, NY, USA, 1991. ACM.

[68] James Oxley and Dominic Welsh. Chromatic, flow and reliability polynomials: The complexity of their coefficients. *Combinatorics, Probability and Computing*, 11:403–426, 2002.

[69] James G. Oxley. *Matroid theory*. Oxford University Press, New York, 1992.

[70] Ran Raz. Multi-linear formulas for permanent and determinant are of super-polynomial size. In *STOC*, pages 633–641, 2004.

[71] Ran Raz. Multilinear-$NC^1 \neq$ multilinear-NC^2. In *FOCS*, pages 344–351, 2004.

[72] Ran Raz and Amir Yehudayoff. Balancing syntactically multilinear arithmetic circuits. *Computational Complexity*, 17(4):515–535, 2008.

[73] Ran Raz and Amir Yehudayoff. Multilinear formulas, maximal-partition discrepancy and mixed-sources extractors. In *FOCS*, pages 273–282. IEEE Computer Society, 2008.

[74] W.L. Ruzzo. Tree-size bounded alternation. *Journal of Computer and System Sciences*, 21:218–235, 1980.

[75] Nitin Saxena. *Morphisms of Rings and Applications to Complexity*. PhD thesis, IIT Kanpur, 2006.

[76] Uwe Schöning. Graph isomorphism is in the low hierarchy. In Franz-Josef Brandenburg, Guy Vidal-Naquet, and Martin Wirsing, editors, *STACS*, volume 247 of *Lecture Notes in Computer Science*, pages 114–124. Springer, 1987.

[77] Rimli Sengupta and H. Venkateswaran. A lower bound for monotone arithmetic circuits computing 0-1 permanent. *Theor. Comput. Sci.*, 209(1-2):389–398, 1998.

[78] Paul D. Seymour. Recognizing of graphic matroids. *Combinatorica*, 1:75–78, 1981.

[79] Amir Shpilka and Avi Wigderson. Depth-3 arithmetic formulae over fields of characteristic zero. In *IEEE Conference on Computational Complexity*, pages 87–, 1999.

[80] Philip M. Spira. On time hardware complexity tradeoffs for boolean functions. In *Fourth Hawaii International Symposium on System Sciences*, pages 525–527, 1971.

[81] Thomas Thierauf. The isomorphism problem for read-once branching programs and arithmetic circuits. *Chicago J. Theor. Comput. Sci.*, 1998, 1998.

[82] Wolfgang Thomas and Pascal Weil, editors. *STACS 2007, 24th Annual Symposium on Theoretical Aspects of Computer Science, Aachen, Germany, February 22-24, 2007, Proceedings*, volume 4393 of *Lecture Notes in Computer Science*. Springer, 2007.

[83] Seinosuke Toda. On the computational power of pp and (+)p. In *SFCS '89: Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 514–519, Washington, DC, USA, 1989. IEEE Computer Society.

[84] Seinosuke Toda. Counting problems computationally equivalent to the determinant. Technical Report CSIM 91-07, Dept. Comp. Sci. and Inf. Math., Univ. of Electro-Communications, Tokyo, 1991.

[85] Seinosuke Toda. PP is as Hard as the Polynomial-Time Hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991.

[86] Seinosuke Toda. Classes of arithmetic circuits capturing the complexity of computing the determinant. *IEICE Transactions on Informations and Systems*, E75-D:116–124, 1992.

[87] Jacobo Torán. On the hardness of graph isomorphism. In *FOCS*, pages 180–186, 2000.

[88] Jacobo Toran. On the Hardness of Graph Isomorphism. *SIAM Jl. of Comp.*, 33(5):1093–1108, 2004.

[89] Klaus Truemper. On Whitney's 2-isomorphism theorem for graphs. *Jl. of Graph Theory*, pages 43–49, 1980.

[90] Iddo Tzamaret. *Studies in Algebraic and Propositional Proof Complexity.* PhD thesis, Tel Aviv University, 2008.

[91] Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.

[92] Leslie G. Valiant. Reducibility by algebraic projections. *Logic and Algorithmic: an International Symposium held in honour of Ernst Specker*, 30:365–380, 1982.

[93] Leslie G. Valiant, Sven Skyum, S. Berkowitz, and Charles Rackoff. Fast parallel computation of polynomials using few processors. *SIAM J. Comput.*, 12(4):641–644, 1983.

[94] H. Venkateswaran. Properties that characterize LogCFL. *Journal of Computer and System Sciences*, 42:380–404, 1991.

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

[96] V. Vinay. Hierarchies of circuit classes that are closed under complement. In *Proceedings of the 11th Annual IEEE Conference on Computational Complexity CCC*, pages 108–117, Washington, DC, USA, 1996. IEEE Computer Society.

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

[98] Joachim von zur Gathen. Parallel linear algebra. In J. H. Reif, editor, *Synthesis of Parallel Algorithms*, pages 573–617. Morgan Kaufmann, 1993.

[99] Joachim von zur Gathen and Gadiel Seroussi. Boolean circuits versus arithmetic circuits. *Information and Computation*, 91(1):142–154, 1991.

[100] Hassler Whitney. Congruent graphs and connectivity of graphs. *American Journal of Mathematics*, 54(1):150–168, 1932.

[101] Hassler Whitney. 2-isomorphic graphs. *American Journal of Mathematics*, 55:245–254, 1933.

[102] Hassler Whitney. On the abstract properties of linear independence. *American Journal of Mathematics*, 57:509–533, 1935.

List of Publications

The materials presented in this thesis are based on the following publications:

Conferences

- **Arithmetizing Classes around NC1 and L** Nutan Limaye and Meena Mahajan and Raghavendra Rao B.V , Proceedings of 24th International Symposium on Theoretical Aspects of Computer Science STACS, 22-24 Feb 2007, Aachen (Germany). Springer-Verlag LNCS Volume 4393 pp 477–488.

- **Arithmetic circuits, syntactic multilinearity, and the limitations of skew formulae.** Meena Mahajan and Raghavendra Rao B V , Proceedings of 33rd International Symposium on Mathematical Foundations of Computer Science MFCS, Aug 25–29, 2008, Torun (Poland). Springer-Verlag LNCS Volume 5162: 144-155, The full version is a Technical report of the ECCC: TR08-048, 2008

- **On the Complexity of Matroid Isomorphism Problem.** Raghavendra Rao B V and Jayalal M.N. Sarma. Proceedings of 4th International Computer Science Symposium in Russia CSR, August 18-23, 2009 Novosibirsk (Russia). Springer-Verlag LNCS Volume 5675: 286-298.

- **Simulation of Arithmetical Circuits by Branching Programs Preserving Constant Width and Syntactic Multilinearity.** Maurice Jansen and Raghavendra Rao B.V. Proceedings of 4th International Computer Science Symposium in Russia CSR, August 18-23, 2009 Novosibirsk (Russia), Springer-Verlag LNCS Volume 5675:179-190.

- **Small space analogues of Valiant's classes**

Meena Mahajan and Raghavendra Rao B.V. Proceedings of 17th International Symposium on Fundamentals of Computation Theory FCT, 2-4 Sep 2009, Wroclaw (Poland). Springer-Verlag LNCS Volume 5699: 250-261

Journal

- **Arithmetizing Classes around NC1 and L** Nutan Limaye and Meena Mahajan and Raghavendra Rao B.V , Accepted for the Theory of Computing Systems (STACS special issue).