Restricted Branching Programs and Hardware Verification

by

Stephen John Ponzio

Bachelor of Science in Electrical Engineering
Bachelor of Science in Mathematics
Master of Science in Electrical Engineering and Computer Science
Massachusetts Institute of Technology (1991)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science and Mathematics
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
August 1995

Signature of Author ______________________________

Department of Electrical Engineering and Computer Science
August 1995

Certified by ______________________________

Professor Mauricio Karchmer
Thesis Supervisor

Accepted by ______________________________

Frederic R. Morgenthaler
Chairman, Departmental Committee on Graduate Students

Restricted Branching Programs and Hardware Verification

by

Stephen John Ponzio

Submitted to the Department of Electrical Engineering and Computer Science
on August 8, 1995 in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
in
Computer Science and Mathematics

Abstract

Recent developments in the field of digital design and hardware verification have found great use for restricted forms of branching programs. In particular, *oblivious read-once* branching programs (also called "OBDD's") are central to a very common technique for verifying circuits. These programs are useful because they are easily manipulated and compared for equivalence. However, their utility is limited because they cannot compute in polynomial size several simple functions—most notably, integer multiplication. This limitation has prompted the consideration of alternative models, usually restricted classes of branching programs, in the hope of finding one with greater computational power but also easily manipulated and tested for equivalence.

Read-once (non-oblivious) branching programs can to some degree be manipulated and tested for equivalence, but it has been an open question whether they can compute integer multiplication in polynomial size. The main result of this thesis proves that they cannot—multiplication requires size $2^{\Omega(\sqrt{n})}$. This is the first lower bound for multiplication on non-oblivious branching programs. By defining the appropriate kind of problem reduction, which we call *read-once reductions*, we are able to show that our result implies the same asymptotic lower bound for other arithmetic functions.

We also survey known results about the various alternative models, describing the main techniques used for thinking about their computation and for proving lower bounds. These techniques are illustrated with two proofs that have not appeared in the literature. We summarize the known results by taking a structural approach of comparing the complexity classes corresponding to the various models.

Keywords: branching programs, read-once, read-k-times, oblivious, OBDD, multiplication, problem reduction, projection reduction.

Thesis Supervisor: Mauricio Karchmer
Title: Assistant Professor of Mathematics

Acknowledgments

Everyone who knows Mauricio loves him. His sincerity and integrity, his unique style and impeccable taste have been an inspiration to both me and my fellow graduate students. As his student, I have profited from his broad knowledge of mathematics and sharp intellect. He taught me how to truly understand proofs and how to present them effectively. It has been a privilege to have him as my advisor.

I would like to thank Allan Borodin for his seminar at MIT in the fall of 1993, where I first learned of this particular area of research and the problem of proving a read-once lower bound for multiplication. Thanks also to Mikael Goldmann for pointing out the reductions in [We93]; to Ravi Sundaram for helpful discussions and a careful proofreading of the lower bound; and to Ron Rivest and Mike Sipser for serving on my thesis committee.

I have benefited from the support of various people during my tenure as a graduate student, in particular Nancy Lynch, under whom I was first introduced to research in computer science; Charles Leiserson and Tom Leighton and Ravi, from whom as a TA I learned more about discrete mathematics than I thought I still could; and Albert Meyer, for his professional, sage advices.

Finally, thanks also to the excellent mathematics teachers who have inspired me over the years: Arthur Mattuck, Mike Sipser, Mauricio, Michael Artin, and Sergey Fomin; to my fellow graduate students in the Theory Group, particularly my longtime officemate Mike Klugerman; to my family—Mom, Dad, Darrell, Christa, Matthew, and Katherine; and to my teammates on the Brighton Braves, for the great years I had playing baseball, which is really all I ever wanted to do.

This research was supported by an NSF Graduate Fellowship, NSF Grant CCR95–03322, and a grant from the Siemens Corporation.

Table of Contents

1 Introduction **9**
1.1 The role of branching programs in hardware verification 9
1.2 Restricted branching programs 11

2 Related models **15**
2.1 Definitions 16
2.1.1 Reading each variable k times 16
2.1.2 Nondeterminism 17
2.2 Manipulating branching programs 19
2.2.1 Read-once programs 19
2.2.2 Nondeterministic read-once programs 20
2.2.3 k-OBDD's 20
2.2.4 k-IBDD's and read-k-times programs 21
2.3 Previous lower bounds 21
2.3.1 For oblivious programs 21
2.3.2 For read-once programs 22
2.3.3 For nondeterministic programs, read-once and read-k-times . . . 23
2.4 Comparing the models: classes and structural results 23
2.4.1 Hierarchies in k 24
2.4.2 Comparing the classes across models 25
2.5 Lower bound techniques 28
2.5.1 For OBDD's, k-OBDD's, and k-IBDD's 28

2.5.2 For nondeterministic BDD's 34
2.5.3 For arbitrary oblivious programs 35
2.5.4 For read-once programs 36
2.6 Integer multiplication 37
2.6.1 Bryant's lower bound 37
2.6.2 The decision problem **DMULT**—the graph of multiplication . . 40
2.7 Related issues 40
2.7.1 The ordering problem for OBDD's 40
2.7.2 The Fourier spectrum 41
2.7.3 Read-once programs and resolution proofs 42

3 A lower bound for multiplication with read-once programs 45
3.1 A paradigm for read-once lower bounds 45
3.2 A lower bound of $2^{\Omega(\sqrt[3]{n})}$ 47
3.3 Improving the bound to $2^{\Omega(\sqrt{n})}$ 53
3.4 Problem reductions 61
3.4.1 Reductions to other arithmetic functions 62

4 Discussion and further work 65

Bibliography 71

Chapter 1

Introduction

Branching programs have recently been found very useful in the field of hardware verification. The central problem of verification is to check whether a combinational hardware circuit has been correctly designed. One approach commonly employed today is to convert independently the circuit description and the function specification to a common intermediate representation and then test whether the two representations are equivalent (e.g., [Br92, We94]). The use of restricted forms of branching programs for the intermediate representation has made this approach feasible and very popular—several software packages are available for implementing this very strategy [Kr94, Br92]. This application raises several issues of computational complexity, renewing interest in the low-level complexity of branching programs. This thesis explores some of these issues from a computational complexity-theoretic point of view.

1.1 The role of branching programs in hardware verification

Most of the computational models considered as candidates for the intermediate representation are restricted classes of branching programs. A *branching program* is a directed acyclic graph with a distinguished root node and two sink nodes. The sink nodes are labeled 0 and 1 and each non-sink node is labeled with an input variable x_i, $i \in [n]$, and has two outgoing edges, labeled 0 and 1. A branching program computes a Boolean function $f : \{0,1\}^n \to \{0,1\}$ in the natural manner: each assignment of Boolean values to the variables x_i defines a unique path through the graph from the root to one of the sinks; the label of that sink defines the value of the function on that input. The *size* of a branching program is its number of nodes. Since branching programs are a non-uniform model of computation, asymptotic statements about size refer to families of branching programs containing one program for each input size.

The circuit to be verified is assumed to be an ordinary combinational single-output circuit, built up from a standard basis of Boolean functions such as $\{\wedge, \vee, \neg\}$. The typical algorithm for constructing the intermediate representation from the circuit is to work bottom-up through the circuit, from the inputs to the output, combining the representations appropriately at each gate. Thus, the algorithm need only compute a representation for $f \wedge g$, $f \vee g$, and $\neg f$, when given representations for f and g. In the literature, these are called the "synthesis operations". It is easy to see that arbitrary polynomial-size branching programs are closed under these operations.

This strategy for verification has several shortcomings that are immediately apparent. First, unrestricted polynomial-size branching programs compute exactly those functions in non-uniform logspace. Therefore, if the intermediate representation is a restricted form of branching program, we clearly cannot hope for a general algorithm to compute a polynomial-size representation (polynomial in the size of the original circuit) unless L/*poly* = P/*poly*. This difficulty has largely been accepted as inherent and not critical, since functions computed at level of hardware are not generally complex and are in fact in L anyway. A second observation is that efficient algorithms for the individual synthesis operations do not imply that the resulting bottom-up algorithm for computing a representation is efficient: for example, if the output of each operation has size that is the product of the input representations, the final representation will have size exponential in the size of the original circuit. Despite this problem, researchers have been content with the bottom-up algorithm as long as each synthesis operation can be performed efficiently.

Finally, there is the problem of testing whether the two branching programs, corresponding to the circuit and the specification, are equivalent. It is easy to see that this problem is co-NP-complete: Given a 3-CNF with variables $\{x_1, \ldots, x_n\}$, we may to construct a branching program on the same variables which accepts exactly when the formula is satisfied. A polynomial-time algorithm for equivalence then clearly gives a polynomial-time algorithm for **3-SAT** by comparing this program with the trivial branching program that always rejects; to say they are not equivalent is to say the formula is satisfiable.

1.2 Restricted branching programs

Because of the difficulty of comparing arbitrary branching programs for equivalence, the intermediate representation is instead chosen to be a restricted class of branching programs. These are *oblivious read-once* branching programs, or OBDD's ("ordered binary decision diagrams").

Definition 1 *A branching program is* read-once *if on every path from the source to a sink, each variable appears at most once as the label of a vertex.*

Definition 2 *A branching program is* oblivious *if on every path from the source to a sink, the variables appear in the same order.*

Our definition of *oblivious* is slightly different from the usual definition, which requires the branching program to be leveled (for each node, all paths from the sink have the same length) with each node at a given level labeled with the same variable. Our definition does not require leveling; it is easy to see that any oblivious program may be leveled at a cost in size of a factor of n, the number of variables. Since we will primarily be concerned with polynomial versus exponential growth, we will or will not assume leveled programs as convenient.

Thus, OBDD's may be thought of as non-uniform acyclic finite-state automata. Notice that the read-once property implies that an OBDD is satisfiable exactly when there *exists* a path from the source to the accepting sink—since no variable appears more than once on any path, there is a consistent assignment to the variables corresponding to that path. An OBDD for $\neg f$ is trivially constructed by exchanging the accepting and rejecting sinks. Given two OBDD's for f and g that obey the same ordering of the variables, an OBDD for $f \wedge g$ or $f \vee g$ is easily constructed using the standard product constructions for finite automata. (This last statement is not true if the two OBDD's do not obey the same ordering—see Section 2.2.1.) It follows that two OBDD's are easily tested for equivalence by testing their exclusive-or for satisfiability.

Because of the tractability of these operations on OBDD's, they have been the intermediate representation of choice. However, OBDD's are clearly a very weak model of computation, and the question arises whether they are sufficiently powerful to meet the needs at hand. The answer is yes, for the most part—OBDD's can compute in polynomial size such functions as integer addition, symmetric Boolean functions, and many of the benchmark functions used by the verification community [BF85]—but with a very important exception: exponential size is required to compute integer multiplication [Br91]. This is an serious setback to the viability of OBDD's, since the hardware to be tested typically contains circuits that perform multiplication. Today, the largest multipliers that can be checked using this method have 12-bit inputs; ideally, circuit designers would like to check multipliers of 32 or even 64 bits.

Thus, despite the success of this approach, there has also been great effort expended to find another model that is likewise manipulated, but with greater computational power [SDG94, SW95, e.g.]. Most of these models—k-OBDD's, k-IBDD's, nondeterministic OBDD's—have proven too weak to compute multiplication in polynomial size (see Chapter 2). A common feature of these models is that they are all *oblivious* branching programs. It is therefore natural to consider non-oblivious programs, the simplest of these being read-once programs.

Unfortunately, read-once programs do not enjoy quite the same degree of manipulability as OBDD's. Determining whether a read-once program is satisfiable is as simple as for an OBDD, since the read-once property implies that the program is satisfiable exactly when there is a path from the source to the accepting sink. Also, testing equivalence is reasonably tractable: although it is not known how to do so in deterministic polynomial time, there is a randomized polynomial-time algorithm with one-sided error due to Blum, Chandra, and Wegman [BCW80]. The synthesis operations, however, are provably not tractable: there exist functions f and g that each have polynomial-size read-once programs but whose conjunction $f \wedge g$ requires exponential-size read-once programs. Despite their relative recalcitrance, read-once programs have been considered by some researchers for possible use in hardware verification [GM94]. Until now, however, very little was known about the complexity of multiplication with any non-oblivious programs.

In this thesis, we prove that multiplication requires (*non-oblivious*) read-once branching programs of size $2^{\Omega(\sqrt{n})}$. This is the first superpolynomial lower bound for multiplication on non-oblivious branching programs. This result demonstrates that relaxing the ordering restriction of OBDD's is insufficient to gain the desired computational power, and thus further strengthening of the model is needed. By defining the appropriate kind of problem reduction, which we call *read-once reductions*, we are able to show that our result implies the same asymptotic lower bound for other arithmetic functions.

Chapter 2 considers in some detail the other models, all essentially generalizations of OBDD's. In addition to summarizing the lower bounds are known for functions in the various models, we compare the classes of functions that are computable in polynomial size by the models, and also describe the techniques available for proving lower bounds in the different models. Included are two simple proofs that have not appeared in the literature. Chapter 3 gives the lower bound for multiplication and the problem reductions; Chapter 4 concludes with statements of the interesting open problems.

CHAPTER 2

Related models

In the search for alternatives to OBDD's, many models have been considered. In addition to their relevance for hardware verification, they are interesting also for the questions of structural complexity that they raise.

This chapter begins by summarizing the various extensions to OBDD's and read-once programs, including adding nondeterminism and allowing variables to be read k times. These different models are compared in two respects: (1) the ease with which such programs are manipulated, and (2) their computational power.

Section 2.3 summarizes the known lower bounds. We then take a structural view of the relationships between the classes of functions computable in polynomial size for the various models. We will see that the two restrictions *obliviousness* and *restricted reading* are orthogonal to each other: With respect to polynomial size, there are functions that can be computed with read-once programs but cannot be computed by oblivious read-k-times programs for any constant k; yet at the same time, there are functions computable by oblivious read-k-times programs that cannot be computed by (non-oblivious) read-once programs. We will also consider the hierarchies with respect to k in the various models.

Section 2.5 briefly outlines the primary techniques for proving lower bounds, including two proofs that have not appeared in the literature. In Section 2.6 we discuss the problem of integer multiplication and describe the known lower bounds. Finally, in Section 2.7 we mention two related issues.

2.1 Definitions

We begin with the definitions of the various extensions to the basic models. Recall that in a read-once branching program, each variable appears at most once on every path from the source to a sink; an OBDD is an oblivious read-once branching program—each path through the program inspects the variables in the same order, each at most once.

Two recently proposed models, which we shall not consider here, are "graph-driven BDD's" [SW95] and "binary moment diagrams" [BC94]. The latter are not branching programs, and do not compute a function, but they do allow polynomial-size representation of multiplication. Also, in [S95] lower bounds are proved on branching programs in which for each path, the number of variables appearing more than once is bounded by k. In [MW95], lower bounds are proved for nondeterministic programs in which each path obeys a bound on the number of alternations between sets of variables.

2.1.1 Reading each variable k times

There are essentially three models of branching programs in which each variable may be read multiple times:

1. k-OBDD's (also known as k-BDD's [BSSW93]). On each path the variables appear at most k times each in an order that is the *same* permutation repeated k times.

2. k-IBDD's. On each path the variables appear at most k times each in an order that is the concatenation of k (possibly different) permutations.

3. Read-k-times programs. On each path the variables appear at most k times each.

We remark that our definition of read-k-times programs prevents a variable from appearing more than k times on *any* path from the source to either sink. These are sometimes referred to as *syntactic* read-k-times programs, in contrast to *semantic* read-k-times programs in which the limited reading need hold only for those paths which some input may follow—un-traversable paths need not obey the read-k-times restriction. (The two defintions are equivalent for $k = 1$.) While the "semantic" definition is perhaps more natural from the point of view of algorithms (upper bounds), the "syntactic" definition is more combinatorial and more amenable to proving lower bounds. No lower bounds (for explicit functions) are known for semantic read-k-times programs.

2.1.2 Nondeterminism

The simplest and most common way to introduce nondeterminism is to permit some nodes to be unlabeled and allow either of the two outgoing edges to be traversed on any input. Such a program is said to accept if the input may follow *some* path from the root to an accepting sink—that is, there exists a path in the subgraph induced by removing edges that are not traversable. It is not surprising that polynomial-size nondeterministic branching programs accept exactly those languages in (nonuniform) NL, nondeterministic logspace.

We may think of the unlabeled nodes of a nondeterministic branching program as being OR nodes. A standard generalization introduces nodes corresponding to other binary functions. Allowing AND nodes, for example, naturally enables polynomial-size programs to accept languages in co-NL. As NL = co-NL, it happens that allowing AND nodes results in the same power as OR nodes for polynomial-size programs[1]. Allowing both AND nodes and OR nodes enables polynomial-size programs to recognize alternating logspace, which is equal to P. By allowing parity nodes, polynomial programs recognize $\oplus$L, a logspace analogue to $\oplus$P [KW93]. Meinel [Me89] explores the range of all possibilities and concludes that allowing nodes of other binary Boolean functions does not give classes different from L, NL, P, or $\oplus$L.

[1] It is easy to see that the proof of [Im88] yields the same result in the non-uniform case: Given a polynomial-size branching program with OR nodes, that proof constructs another polynomial-size branching program with OR nodes that accepts exactly when the original program rejects. This OR-program for $\overline{f}$ is easily converted to an AND-program for f by replacing the OR nodes with AND nodes and switching the accepting and rejecting nodes.

Note that we have not introduced nondeterminism as we would with circuits, where we would allow nondeterministic variables as inputs. Defining nondeterminism in this manner immediately gives (nonuniform) NP for polynomial-size programs, since NP is characterized by polynomial-size nondeterministic formulas.

We mention that Borodin, Razborov and Smolensky [BRS93] use a different definition of nondeterminism: Nodes are unlabeled and each edge is either unlabeled or labeled with a variable and a value. Unlabeled edges are considered "free" edges which may be traversed by any input; labeled edges may of course be traversed only by inputs consistent with the label. The measure of size is number of labeled edges, rather than number of nodes. The difference in models is not of consequence for our purposes, as it is easy to see that the two size measures are within a constant factor of each other. Clearly, our nondeterministic branching programs are essentially a special case of theirs, and the number of edges in one of our programs is at most twice the number of nodes. Conversely, a program in their form is easily converted to one of our form in which the number of nodes is at most the number of edges in the original program.

There is another model of nondeterministic branching programs, called *rectifier-and-switching networks*, which is preferred by Razborov because of the combinatorial characterization its size measure affords (see [Ra91, Ra90]). A rectifier-and-switching network is essentially a nondeterministic branching program as [BRS93] defines them, except that the (directed) graph may contain cycles. There is no "rejecting sink" and the program accepts exactly when there exists at least one path from the source to the (accepting) sink. The measure of size is the number of labeled edges. Again, our nondeterministic programs are essentially a special case of rectifier-and-switching networks. So for a given function, our programs may be larger, but not by more than a quadratic factor, as the following transformation demonstrates. To make a network of E edges acyclic, place E copies of it in sequence redirecting original "back edges" (those edges which lead to a node that is not further from the root) to lead instead to the copy of the destination node in the subsequent copy of the graph. At most E copies are needed since any path contains at most E edges and an extra copy of the graph is needed only for each back edge in the path. Thus at a cost of squaring the size, we obtain a nondeterministic program in the sense of [BRS93]. It is not known if this measure is within a constant factor of the other two [Ra91, Open Question #1].

2.2 Manipulating branching programs

As explained in Chapter 1, OBDD's have the useful property that they are easily manipulated: Given OBDD's for f and g that obey the same ordering of the variables, it is easy to construct an OBDD for $f \wedge g$ or $f \vee g$. Since the satisfiability of an OBDD is equivalent to the reachability of the accepting sink, OBDD's are also easily tested for satisfiability and thus equivalence. We remark that although the synthesis operations of constructing OBDD's for $f \wedge g$ and $f \vee g$ are intractable if the two given OBDD's do not obey the same ordering (as shown below), this condition is *not* necessary for testing equivalence. There is a polynomial-time algorithm due to Fortune, Hopcroft, and Schmidt [FHS78] for testing whether an OBDD is equivalent to a read-once program, which can be used in this case.

2.2.1 Read-once programs

Read-once programs do not enjoy quite the same degree of manipulability as their oblivious version, OBDD's. The read-once property implies that the program is satisfiable exactly when there is a path from the source to the accepting sink. However, the synthesis operations are provably not tractable: there exist functions f and g that each have polynomial-size read-once programs but whose conjunction $f \wedge g$ requires exponential-size read-once programs. Such an example is the function π-**Matrix** of determining whether an $n \times n$ $(0,1)$-matrix is a permutation matrix—or equivalently, whether a bipartite graph on nodes $V \times W$, where $|V| = |W| = n$, is exactly a perfect matching (and no further edges). π-**Matrix** requires exponential-size read-once programs (see Section 2.3.3). On the other hand, it is easy to test that the each row has exactly one 1 or that each column has exactly one 1—in fact, these two functions are easily computed by OBDD's (with different orderings of the variables)—and π-**Matrix** is true exactly when both of these functions are true.

It is not known how to determine the equivalence of two read-once programs in polynomial time. Blum, Chandra, and Wegman [BCW80] give a co-RP algorithm (that is, it may say "equivalent" when in fact the programs are not, but never vice versa) which relies on randomly assigning to the literals values from a finite field and then computing the value of the DNF polynomial of the function.

2.2.2 Nondeterministic read-once programs

Obviously, OR nodes trivialize the synthesis operation of constructing a program for $f \vee g$. They do not help, however, with constructing $f \wedge g$: the lower bound for π-**Matrix** is actually proved for read-once programs with OR nodes, so a nondeterministic read-once program for $f \wedge g$ may require size exponential in the sizes of the programs for f and g. This result may be contrasted with NL = co-NL, which says that polynomial-size branching programs with OR nodes are equivalent to polynomial-size branching programs with AND nodes. We now see that if we restrict the programs to be read-once, OR nodes and AND nodes give different computational power [KMW91]. The same phenomenon occurs for linear-length oblivious programs [KMW92].

Determining the satisfiability of a program with AND nodes is NP-complete by the example in Section 2.2.4. The case of OR nodes is trivially as least as hard as determining the satisfiability of a deterministic read-once program, which is not known to be in P. In the case of PARITY nodes, the algorithm of [BCW80] works as long as the field used has characteristic 2 [SDG94]. In [SDG94], simple but very restrictive conditions on the use of AND and OR gates are given so that the correctness of the algorithm of [BCW80] is retained.

2.2.3 k-OBDD's

By restricting the order to be the *same* permutation repeated k times, we retain the property that two programs with obeying the same ordering are easily combined—the usual product construction works as before for OBDD's.

k-OBDD's are also testable for satisfiability though with a little more effort. Regard the program as k separate segments corresponding to the k repetitions of the permutation in which the variables are read. If the size and hence the width is polynomial in n, then there are a polynomial number of nodes at the top of each segment. The portion of a segment between a particular top node and a "bottom" node (at the top of the subsequent segment) may be viewed as an OBDD. For an input to pass through a given sequence of k "top nodes" it must satisfy the conjunction of the k corresponding OBDD's (with source and accept nodes defined appropriately). To test whether these

k OBDD's are simultaneously satisfiable, we may construct an equivalent OBDD using the synthesis operation for OBDD's (since these k OBDD's obey the same ordering) and then check it for satisfiability. There are $(poly)^k = poly$ sequences of "top nodes" that an input may follow and the k-OBDD is satisfiable if one of these paths is satisfiable. Thus, to determine whether the k-OBDD is satisfiable, we sequentially check whether any of these sequences is traversable.

Other operations on k-OBDD's are considered in detail in [BSSW93].

2.2.4 k-IBDD's and read-k-times programs

Unlike for k-OBDD's, testing the satisfiability of even 2-IBDD's, and hence read-2-times programs, is NP-complete. The reduction, from SAT, places in sequence two OBDD's, one that checks the satisfiability of the formula with each variable uniquely renamed, and another that checks whether the corresponding variables have the same value. Since it includes satisfiability as a special case, testing the equivalence of two k-IBDD's is also hard.

Since a 1-IBDD is simply an OBDD, the example **π-Matrix** implies that the synthesis operations on k-IBDD's are intractable even for $k = 1$ if the constructed program must also be a k-IBDD. Naturally, the synthesis operations on a pair of k-IBDD's are easy if we allow the constructed program to be a $2k$-IBDD. The same statements are true for read-k-times programs.

2.3 Previous lower bounds

The restriction of limited reading is severe enough that in contrast to the case of arbitrary branching programs, many exponential lower bounds have been proved for explicit functions, some of the functions quite simple.

2.3.1 For oblivious programs

Exponential lower bounds for the size of OBDD's are known for many functions, in particular the functions **Hwb** ("Hidden-Weighted-Bit"), **Ach** ("Achilles-Heel"), and integer multiplication, **Mult**, (all defined later), for which lower bounds were proved specifically for OBDD's. Krause [Kr91] proves lower bounds for other functions. We will have more to say about these lower bounds in Sections 2.4.2, 2.5, and 2.6. Of course, all other lower bounds mentioned below for stronger models imply a fortiori equally strong lower bounds for OBDD's.

Also, in a very different vein, Alon and Maass [AM88] prove lower bounds for arbitrary oblivious programs of linear length, which do not obey any restriction on the number of times a variable is read. Their lower bound is discussed in Section 2.5.3. In similar spirit, Krause and Waack [KW91] show that any oblivious program of linear length for the problem of directed s-t connectivity requires exponential size; in [KMW92], similar lower bounds are proved for such programs with nondeterminism added.

Using a lemma from [AM88], and the communication complexity arguments outlined in Section 2.5.1, Gergov [Ge94] proves that computing **Mult** requires size $2^{\Omega(n)}$ for arbitrary oblivious programs of linear length, even with nondeterministic AND, OR, or PARITY nodes.

2.3.2 For read-once programs

There has also been great success in proving lower bounds on the size of read-once programs. Many of the functions that require exponential size are very simple; some are easily computed with mere read-twice programs.

Masek [Ma76] was the first to consider read-once programs, proving a lower bound of $\Omega(m^2)$ on the size of any program determining whether $\sum_{i=1}^{n} x_i = m$. Zak [Za84] and later Wegener [We88, We87] proved lower bounds of $2^{\Omega(n)}$ for the function $\frac{n}{2}$-**Clique** of determining whether a graph on n nodes contains a clique of size $n/2$, and also for the function $\frac{n}{2}$-**Clique-Only**, of determining whether a graph on n nodes contains an $n/2$-clique and no further edges. (For comparison, there is a simple read-twice program for $\frac{n}{2}$-**Clique-Only** of size $O(n^3)$.) Dunne [Du85] proved a lower bound of $2^{\Omega(n)}$ for the problems of determining whether a graph on n nodes contains a hamiltonian cycle and determining whether it contains a perfect matching. Simon and Szegedy [SS93], in order to demonstrate their lower bound technique, proved a lower bound of $2^{\Omega(n)}$

for the problem of determining whether a graph on n nodes is $(n/2)$-regular. Note that none of these bounds is fully exponential, since the number of input variables, one for each edge, is $\binom{n}{2}$. Babai, Hajnal, Szemeredi and Turan [BHST87] proved an asymptotically optimal lower bound of $2^{\Omega(n^2)}$ for computing the parity of the number of triangles in a graph on n nodes; Simon and Szegedy [SS93] simplify and refine their analysis, improving the constant in the exponent.

2.3.3 For nondeterministic programs, read-once and read-k-times

Exponential lower bounds for explicit functions have also been proved for nondeterministic read-once branching programs. Krause, Meinel, and Waack [KMW91] (see also [Ju89]) give a lower bound of $n!/\left(\frac{n}{2}!\right)^2 = 2^{\Omega(n)}$ for the function π-**Matrix**. (It was known earlier that this function required exponential-size *deterministic* read-once programs; see [Kr91, p. 10] and [Ju86].) Also, Borodin, Razborov and Smolensky [BRS93] prove a lower bound of $2^{\Omega(n)}$ for the functions $\frac{n}{2}$-**Clique** and $\frac{n}{2}$-**Clique-Only**. Note that the complement of $\frac{n}{2}$-**Clique-Only** can be computed by nondeterministic read-once programs of polynomial size.

Okolnishnikova [Ok91] proves that computing the characteristic function of the Bose-Chaudhuri codes requires deterministic read-k-times programs of size exponential in $\Omega(\sqrt{n}/k^k)$. Borodin et. al. [BRS93] exhibit for any k, a function that requires nondeterministic read-k-times programs of size exponential in $\Omega(n/k4^k)$. Jukna [Ju92] extends the results of [BRS93] and [Ok91] to show that the function from [Ok91] requires nondeterministic read-k-times programs of size exponential in $\Omega(\sqrt{n}/k^{2k})$ even though its complement can be computed by nondeterministic read-*once* programs of polynomial size.

Also, in [MW95], lower bounds are proved for nondeterministic programs in which each path obeys a bound on the number of alternations between sets of variables.

2.4 Comparing the models: classes and structural results

In this section, we will compare the classes of functions that are computable by polynomial-size programs of the various types. We will use sans-serif font to denote the class of functions computable in polynomial size by the named model. For instance, we will use OBDD to denote the class of functions computable by OBDD's of polynomial size and $\mathsf{READ}\text{-}k$ for the functions computable by read-k-times programs of polynomial size. We will also need a notation for the union over all constants k:

Definition 3

$$
\begin{aligned}
\mathsf{C\text{-}OBDD} &\rightleftharpoons \bigcup_{k\in\mathbb{N}} k\text{-}\mathsf{OBDD} \\
\mathsf{C\text{-}IBDD} &\rightleftharpoons \bigcup_{k\in\mathbb{N}} k\text{-}\mathsf{IBDD} \\
\mathsf{READ\text{-}C} &\rightleftharpoons \bigcup_{k\in\mathbb{N}} \mathsf{READ}\text{-}k
\end{aligned}
$$

(where C *is for "constant").*

We will use $\mathsf{OBLIV\text{-}LINEAR}$ to denote the class of functions computable with oblivious programs of linear length and polynomial size. Note that

$$k\text{-}\mathsf{OBDD} \subset \mathsf{C\text{-}OBDD} \subset \mathsf{C\text{-}IBDD} \subset \mathsf{OBLIV\text{-}LINEAR}.$$

The results presented in this section are summarized in Figure 2.1, which gives the inclusion relations of these various classes.

2.4.1 Hierarchies in k

It is known that the hierarchy over k of functions computable by k-OBDD's of polynomial size is strict: $k\text{-}\mathsf{OBDD} \subsetneq (k+1)\text{-}\mathsf{OBDD}$ [BSSW95]. For the case $k = 1$, we may refer to the function **Hwb**, described below, which is in 2-OBDD but not OBDD. For k-IBDD's the hierarchy is also strict: $k\text{-}\mathsf{IBDD} \subsetneq (k+1)\text{-}\mathsf{IBDD}$ [BSSW95]. These lower bounds are based on the well-known "rounds hierarchy" for communication complexity exhibited by the "k-pointer-chasing" function, k-**Ptr**, on bipartite graphs [PS82, DGS84, Mc86, HR88, NW91] (in particular the result of [NW91]).

It is not known whether the corresponding hierarchy for read-k-times programs is strict, except for the case $k = 1$, where we have seen that π-**Matrix** $\notin \mathsf{READ}\text{-}1$

but π-**Matrix** $\in$ 2-IBDD $\subset$ READ-2. Simon and Szegedy [SS93] conjecture that the problem of testing the regularity of hypergraphs, which (for the case of ordinary graphs) they showed separates READ-1 from READ-2, will separate the levels of this hierarchy. We reconsider this question in Chapter 4.

2.4.2 Comparing the classes across models

OBDD $\subsetneq$ READ-1; OBDD $\subsetneq$ 2-OBDD

It can be shown that the inclusion OBDD $\subset$ READ-1 is proper—that is, the ordering restriction does in fact limit the computational power of read-once programs. Demonstrating this separation is the function **Hwb**(x) ("Hidden-Weighted-Bit"), which returns x_i if there are i ones in x and 0 otherwise. **Hwb** is computable in READ-1 by a clever algorithm that works its way in from the outermost bits of x; it is also easily computed in 2-OBDD. A standard lower bound argument shows that **Hwb** $\notin$ OBDD ([Br91], see Section 2.5.1).

Also, it is shown in [BHR95] (see also [BSSW93]) that **Isa** $\notin$ OBDD, where **Isa**$(x, y) : \{0,1\}^n \times \{0,1\}^{\lg n} \to \{0,1\}$ is the "Indirect-Storage-Access" function which returns x_i, where i is the integer represented by the y'th block of $\lg n$ bits of x if $0 \leq y < n/\lg n$, and returns 0 if $n/\lg n \leq y < n$. It is easy to see that **Isa** $\in$ READ-1 and **Isa** $\in$ 2-OBDD.

k-OBDD $\not\subset$ READ-1 **for** $k > 1$.

Furthermore, the classes READ-1 and k-OBDD are incomparable (for any constant $k > 1$); their models may be thought of as orthogonal restrictions of read-k-times programs. 2-OBDD is separated from READ-1 by the function **Mhwb** ("Multiple-Hidden-Weighted-Bit"), defined on 3 n-bit vectors x, y, and z as $x_{|y|+|z|} \oplus y_{|x|+|z|} \oplus z_{|x|+|y|}$ where $|x|$ is the hamming weight of x and the sums are computed modulo n. **Mhwb** has a natural read-twice algorithm where the variables may be read in order each time, so **Mhwb** $\in$ 2-OBDD. In [BHR95], it is shown that **Mhwb** $\notin$ READ-1. Krause [Kr91, Remark 5.3] gives a different function which separates 2-OBDD from READ-1.

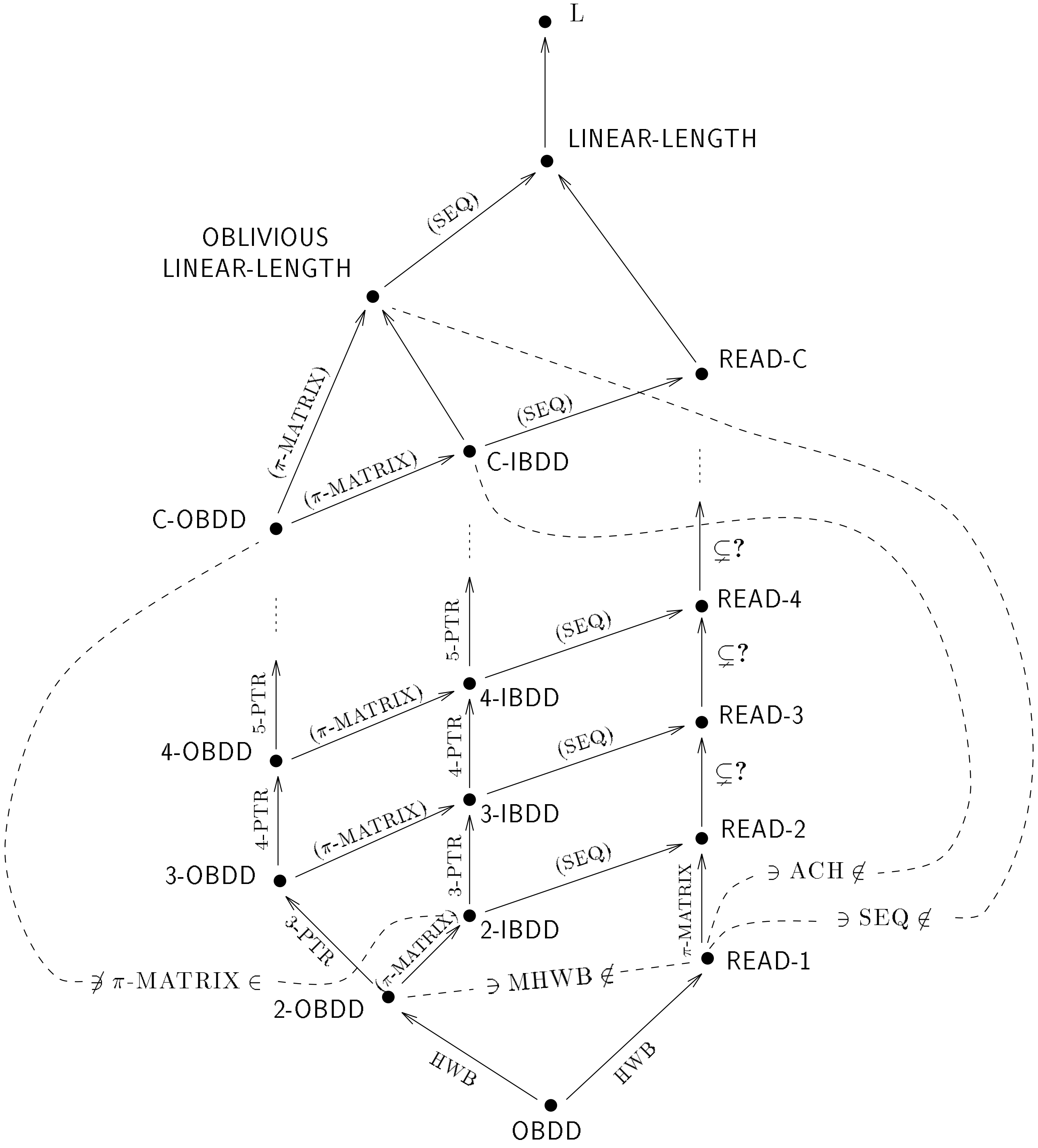

Figure 2.1: The inclusion relations among the classes. "C⟶D" means class C is contained in class D; inclusions that can be inferred by transitivity are not shown. Arrows labeled with problems denote proper inclusions, where the labeling problem separates the two classes. (Problems in parentheses denote separations that can be inferred from others.) The separations denoted with dotted lines show that further inclusions do not hold.

This result is in a sense best possible since 2-OBDD $\subset$ READ-2. In the other direction, we have

READ-1 $\not\subset$ C-OBDD.

In [BSSW93], an exponential lower bound is proved for the size of k-OBDD's for the function **Ach** ("Achilles-Heel"), defined on $2n + \lg n$ Boolean variables as

$$\mathbf{ACH}(x_0, \ldots, x_{n-1};\ y_0, \ldots, y_{n-1};\ z_1, \ldots, z_{\lg n}) = \begin{cases} \bigvee_{1 \le j \le n} (x_j \wedge y_j) & \text{if } z = 0 \\ \bigwedge_{1 \le j \le n} (x_j \vee y_{j+z}) & \text{if } z \ne 0 \end{cases}$$

where z is the integer represented in binary by $z_1 \ldots z_{\lg n}$ and the sum $j + z$ is computed modulo n. **Ach** is easily seen to be in READ-1, but a standard lower bound argument shows **Ach** is not in k-OBDD for any constant k [AGD91, BSSW93].

Krause [Kr91, Remark 5.4] gives a different function which separates C-OBDD from READ-1.

The separation READ-1 $\not\subset$ C-OBDD is subsumed by the following result:

READ-1 $\not\subset$ OBLIV-LINEAR.

This very strong separation is shown using the powerful technique of Alon and Maass [AM88]. They exhibit a function **Seq** of $4n$ bits th at is easily in READ-1, but cannot be computed by any oblivious program of length $O(n)$ (see Section 2.5.3). This result exhibits most strongly how severe a computational restriction obliviousness is.

This result is also best possible since OBLIV-LINEAR is the largest of our classes not containing READ-1.

2-IBDD $\not\subset$ C-OBDD.

Clearly, k-OBDD $\subset$ k-IBDD for each k; conversely, however, 2-IBDD $\not\subset$ C-OBDD. Again, the separating function is π-**Matrix**: π-**Matrix** $\in$ 2-IBDD easily, but π-**Matrix** $\notin$ C-OBDD. This lower bound is claimed in [Kr91, Remark 5.5], but to the best of our knowledge no proof has appeared, so we give one in Section 2.5.1 (Theorem 1).

This result indicates it really is a computational restriction to restrict the order to be the same permutation repeated k times rather than k different permutations.

Finally, we mention that some functions that are provably outside these classes are easily contained in some of their nondeterministic counterparts. **Hwb**, for example, while not in OBDD, is easily computed by a nondeterministic OBDD that initially branches into n different deterministic OBDD's, all with a common ordering.

2.5 Lower bound techniques

In this section, we describe the techniques that have been used to prove lower bounds in the various oblivious models: OBDD's, both deterministic and nondeterministic, k-OBDD's and k-IBDD's, and arbitrary oblivious programs of linear length. For completeness as well as for demonstration, we supply a proof of Theorem 1, announced in [Kr91] without proof, and also prove Theorem 2, extending in a simple way the result of [BSSW93]. We compare these methods with lower bounds for non-oblivious programs, but defer a detailed description of the latter until the presentation of our own lower bound in Chapter 3. The technique of [BRS93] for proving lower bound for read-k-times programs will bemention only briefly in Chapter 4, when we outline approaches to some open problems.

2.5.1 For OBDD's, k-OBDD's, and k-IBDD's

Lower bounds for OBDD's follow a simple strategy: Show that for any $Y \subset X$ of some fixed size (say $m = m(n)$), there are many (say $2^{\Omega(n)}$) subfunctions on Y. If the first $n - m$ variables read by an OBDD are $\overline{Y}$, clearly any two assignments to $\overline{Y}$ that induce different subfunctions on Y must lead to different nodes. Since this lower bound holds for any set Y of size m, $2^{\Omega(n)}$ is a lower bound on the number of nodes for any OBDD. Most lower bounds for OBDD's show explicitly that there are many subfunctions by exhibiting for any $\overline{Y}$ of the stated size an exponential number of settings to $\overline{Y}$ such that for any two, there is a setting to Y on which the respective subfunctions differ.

This argument may be nicely interpreted in terms of communication complexity: one party gets the values of the bits in $\overline{Y}$ and the other party the bits in Y. The second party must compute the value of the function based on a single message sent by the first party. An OBDD gives a communication protocol where $\overline{Y}$ is the first m variables in its ordering. If the program has w nodes at the level immediately following the nodes of $\overline{Y}$, then the message has $\lg w$ bits. Thus, if the one-way communication complexity is linear for every Y of size m, then the function requires OBDD's of exponential size. Bryant [Br91] uses a simple argument of this form to prove that **HWB** requires exponential-size OBDD's.

Commonly, it is proved that in fact the unlimited-round, two-way communication complexity of the function is linear for any Y of size m. This argument is sometimes made in terms of what is called a "fooling set" for the function f with respect to Y. For $Y \subseteq X$, and $x, x' \in \{0,1\}^n$, let x_Y denote the value of x on the variables in Y, and let $x_Y x'_{\overline{Y}}$ denote the n-bit input string equal to x on the variables in Y and equal to x' on the variables in $\overline{Y}$. A fooling set $F \subseteq \{0,1\}^n$ for f with respect to Y has the property that for all $x \neq x' \in F$, $f(x) = f(x') = 1$ and either $f(x_Y x'_{\overline{Y}}) = 0$ or $f(x'_Y x_{\overline{Y}}) = 0$. Thus, if an OBDD obeys an ordering in which the variables in Y are read first, the setting x_Y cannot lead to the same node at level m as the setting x'_Y since either $f(x_Y x_{\overline{Y}}) \neq f(x_Y x'_{\overline{Y}})$ or $f(x'_Y x'_{\overline{Y}}) = 1 \neq f(x'_Y x_{\overline{Y}})$. If for every Y of size m there is a fooling set of exponential size, then the function requires exponential-size OBDD's. Furthermore, the existence of a fooling set F for Y implies that the (unrestricted) communication complexity with respect to the partition $Y \mathbin{\dot\cup} \overline{Y}$ is $\lg |F|$. This is seen by inspecting the associated matrix M_{ij} where i (resp., j) ranges over all values of x_Y for $x \in F$ (resp., of $x_{\overline{Y}}$) and $M_{x_Y x'_{\overline{Y}}} = f(x_Y x'_{\overline{Y}})$. Note that the definition of F implies that $x_Y \neq x'_Y$ and $x_{\overline{Y}} \neq x'_{\overline{Y}}$ for $x \neq x'$, so M is a square matrix of dimension $|F|$. M has 1's on its diagonal because $f(x_Y x_{\overline{Y}}) = 1$, and since either $M_{ij} = 0$ or $M_{ji} = 0$, no two 1's on the diagonal can appear in the same all-1's minor. Since a communication protocol of b bits partitions the 1's of the matrix into 2^b all-1's minors, the communication complexity is at least $\lg |F|$.

For k-OBDD's

If, for any set Y of size m, there exists a fooling set of size $2^{\Omega(n)}$, then it is also easy to see that the function requires k-OBDD's of size $2^{\Omega(n)/2k}$ [Kr91]. A k-OBDD gives a communication protocol of $2k$ rounds; the total communication is $2k\lg(\mathit{width})$, which must be at least $\Omega(n)$, giving the desired bound on the width and hence the size.

For example, we give a simple proof that π-**Matrix** $\notin k$-OBDD for any constant k.

Theorem 1 π-**Matrix** $\notin$ C-OBDD

Proof: We will show that for any partition of the n^2 variables into two sets X and $\overline{X}$ of equal size, $2^{\Omega(n)}$ is a lower bound on the rank of the matrix of the communication complexity game where player I gets X and player II gets $\overline{X}$.

First notice that for certain partitions, the proof is easy. For example, consider the partition where player I gets the variables in rows $1,\ldots,n/2$ and player II gets the variables in rows $n/2+1,\ldots,n$. We may even restrict our attention to only those inputs where each row has exactly one 1 and each player gets exactly $n/2$ 1's. The inputs to the two players then correspond merely to subsets of the columns; the players accept if the subsets are disjoint and reject otherwise. It is easy to see that this problem requires $\lg\binom{n}{n/2}$ bits of communication, since the $\binom{n}{n/2}$-by-$\binom{n}{n/2}$ matrix of the communication game is diagonal.

Our proof will follow the spirit of this strategy for arbitrary partitions. Let r_i be the number of X-variables in row i. Order the rows so that $r_1 \leq r_2 \leq \cdots \leq r_n$. We have $|X| = \sum_i r_i = n^2/2$. Let rows $n/2+1,\ldots,n$ be the "top half" of the matrix.

First consider the case that the top half contains at least $3/4$ of the X-variables: $\sum_{i=n/2+1}^{n} r_i \geq \frac{3}{4}\frac{n^2}{2}$. In this case, at least $2/3$ of the columns have at least $n/8$ X-variables in their top halves: otherwise, the number of X variables in the top half is less than

$$\frac{2n}{3}\frac{n}{2} + \frac{n}{3}\frac{n}{8} = \frac{3n^2}{8},$$

a contradiction. Since the top half contains exactly half of all the variables, the "bottom half" (rows $1,\ldots,n/2$) has at least $3/4$ of the variables in $\overline{X}$. It follows that at least

2/3 of the columns have at least $n/8$ $\overline{X}$-variables in their bottom halves. Therefore at least $n/3$ columns contain at least $n/8$ X-variables in the top half and at least $n/8$ $\overline{X}$-variables in the bottom half. Let C be any subset of $n/4$ of these columns.

For any subset C' of half the columns of C, there is a setting to X in which exactly one X-variable in the top half of each column of C' is 1 and each such 1 appears in a different row. This is because $|C'| = n/8$ and there are at least $n/8$ X-variables in the top half of each column of C. Let us restrict attention to particular settings to the variables in C. On X, these settings shall be as described above (for some C') in the top half, and shall be 0 in the bottom half. On $\overline{X}$, these settings shall be 0 in the top half, and in the bottom half shall contain 1's in $n/8$ different rows and different columns C''.

If these two subsets C' and C'' of columns are complementary ($C' \cup C'' = C$), then there is a setting to the remaining variables for which the input is a permutation matrix, making the function 1. If these two subsets of columns are not complementary ($C' \cup C'' \subsetneq C$), some column in C contains both a 1 in its top half and a 1 in its bottom half, so that for all settings to the remaining variables, the function is 0. We partition these settings to X (I's inputs) into $\binom{n/4}{n/8}$ blocks, according to which subset of C contains the 1's in X. Similarly, we partition the settings to $\overline{X}$ (II's inputs). Thus the communication complexity matrix associated with these inputs is comprised of $\binom{n/4}{n/8}^2$ minors, and only the minors on the diagonal contain 1's. This matrix clearly has rank at least $\binom{n/4}{n/8} = 2^{\Omega(n)}$.

Now consider the case that $\sum_{i=n/2}^{n} r_i < \frac{3}{4}\frac{n^2}{2}$. In this case, the bottom half has at least $\frac{1}{4}\frac{n^2}{2}$ X-variables, implying that

$$r_{n/2} \ \geq \ \frac{n^2/8}{n/2} \ = \ n/4.$$

Since $r_i \geq r_{n/2}$ for $i \geq n/2$, it follows that there are at least $4n/10$ rows in the top half with at most $7n/8$ X-variables: otherwise, there are more than

$$\frac{4n}{10}\frac{7n}{8} + \frac{n}{10}\frac{n}{4} = \frac{3n^2}{8}$$

X-variables in the top half, a contradiction. Let R be these $2n/5$ rows, each containing at least $n/4$ and at most $7n/8$ X-variables.

Let C^L be columns $1, \ldots, n/2$ (the "left half") and C^R be columns $n/2+1, \ldots, n$ (the "right half"). Since each row in R has either more X-variables in C^L or C^R, at least half of the rows in R have most of their X-variables in one, say C^L. Each of these $n/5$ rows has at least $n/8$ X-variables in the left half and at most $7n/16$ X-variables in the right half. Alternatively, each of these rows has at least $n/8$ X-variables in the left half and at least $n/16$ $\overline{X}$-variables in the right half.

We now fix some $n/8$ of these rows and the rest of the proof proceeds as in the first case, yielding a lower bound of $\binom{n/8}{n/16} = 2^{\Omega(n)}$. ■

It is easy to see that π-**Matrix** has OBDD's of size $O(n2^n)$: the variables are read column-wise, easily ensuring that each column has exactly one 1; furthermore, the OBDD keeps track of the subset of the rows in which 1's have appeared, requiring width $O(2^n)$. Interestingly, for k-OBDD's, just as the lower bound degrades roughly by a factor of k in the exponent, yielding $2^{\Omega(n/k)}$, similarly the upper bound can be improved by a factor of k in the exponent. Construct a k-OBDD of width $2^{\Omega(n/k)}$ by reading the variables column-wise, but keeping track only of n/k rows at a time: Partition the rows into k sets of size n/k each, and in segment $i = 1, \ldots, k$, keep track of the subset of the ith set of rows in which 1's have appeared. Accept only if in each segment, each of the i rows is found to contain exactly one 1.

For k-IBDD's

The only lower bound that is proved specifically for IBDD's (i.e., which does not apply to linear length oblivious programs more generally) is the lower bound of [BSSW95]. They reduce the problem to one of communication complexity in the following manner. Given an IBDD, they construct two disjoint subsets of the variables by considering the levels of the IBDD one at a time. Each level disqualifies at most one-half of the variables in each set, so that after a constant number of levels, still a constant fraction 2^{-k} of the variables are retained. They argue that the problem restricted to these variables is a smaller version of the original problem, and hence the known linear lower bound on the communication complexity applies.

To demonstrate, we give an easy lower bound which has not appeared in the literature. The proof is very similar to the lower bounds of [BSSW95] and [Ge94]. Recall that [BSSW93] showed **ACH** $\notin$ C-OBDD; we will show that **ACH** $\notin$ C-IBDD.

Theorem 2 **ACH** $\notin$ C-IBDD.

Proof: Consider a k-IBDD G computing **ACH**. We will show that G has size at least $2^{n/k2^k}$. Recall from Section 2.4.2 that

ACH$(x,y,z) = \bigwedge_{1\le j\le n}(x_j \vee y_{j+z})$ if $z \neq 0$, and $\bigvee_{1\le j\le n}(x_j \wedge y_j)$ if $z=0$.

We think of G as being composed of k segments, each with n levels corresponding to a permutation of the variables. Suppose we could show that for some z there are subsets of variables

$$X_S = \{x_i : i \in S\} \subset X \quad \text{and} \quad Y_S = \{y_{i+z} : i \in S\} \subset Y$$

of size at least $n/2^{2k}$ such that for each segment of G, either all variables of X_S appear before Y_S or vice-versa. Then we may invoke the communication complexity argument in which the players get $2k$ rounds or fewer. If $z > 0$, we get a fooling set of size $2^{|X_S|}$ with respect to X_S by taking inputs ranging over all settings to X_S and where $y_{i+z} = x_i = 1$ for $i \notin S$ and $y_{i+z} = \overline{x_i}$ for $i \in S$. For each such input $w = (x,y,z)$, we have **ACH**$(w) = 1$, but for two different such inputs, $w \neq w'$, we have either **ACH**$(w_{X_S} w'_{\overline{X_S}}) = 0$ or **ACH**$(w'_{X_S} w_{\overline{X_S}}) = 0$. Similarly, if $z = 0$, letting $y_{i+z} = x_i = 0$ for $i \notin S$ and $y_{i+z} = \overline{x_i}$ for $i \in S$, for each such input $w = (x,y,z)$, we have **ACH**$(w) = 0$, but for two different such inputs $w \neq w'$ we have either **ACH**$(w_{X_S} w'_{\overline{X_S}}) = 1$ or **ACH**$(w'_{X_S} w_{\overline{X_S}}) = 1$. If we can find such a z, X_S and Y_S for any given G, then the communication complexity argument implies that the width of G is exponential in $(n/2^{2k})2k$.

We now show that there exist z, X_S, and Y_S as desired. Without loss of generality, suppose the first half of the first segment of G has more X variables than Y variables. Let $X_1 \subset X$ appear in the first half and $Y_1 \subset Y$ appear in the second half so that $|X_1| = |Y_1| \ge n/2$. Now partition the second segment of G in "half" with respect to the n variables $X_1 \cup Y_1$ only. If the first half contains more X variables than Y variables, let $X_2 \subset X_1$ appear in the first half and $Y_2 \subset Y_1$ appear in the second half, so that $|X_2| = |Y_2| \ge n/4$. Otherwise, let $X_2 \subset X_1$ appear in the second half and $Y_2 \subset Y_1$ appear in the first half. Repeating this process for the k segments, we finally obtain

X_k and Y_k of size $n/2^k$ with the desired alternation property. Since $X_k \times Y_k$ has size at least $n^2/2^{2k}$, it contains at least $n/2^{2k}$ pairs (x_i, y_{i+z}) for some value of z between 0 and $n-1$. The x_i in these pairs constitute X_S. ∎

Note that π-**Matrix** does not enjoy the same self-reducibility property: the above proof applied to the 2-IBDD for computing π-**Matrix** finds X_2 equal to the variables in one quadrant of the matrix and Y_2 equal to the variables in the diagonally opposite quadrant. Indeed, for any setting to the remaining variables, only one bit of communication between the players is necessary to compute the function: player I checks that the top rows and left columns are okay, and player II checks that the bottom rows and right columns are okay.

2.5.2 For nondeterministic BDD's

Lower bounds for nondeterministic BDD's also follow from the existence of exponential-size fooling sets: they imply that the function requires exponential-size nondeterministic OBDD's, when OR gates[2] or PARITY gates are allowed.

For example, consider an OBDD with OR nodes. We may view the corresponding communication protocol as containing nondeterministic choices by the players, giving in effect the OR of many deterministic protocols. Each such deterministic protocol determines some 1-rectangles (all-1's minors); together, the 1-rectangles of all the protocols must cover all the 1's of the matrix without covering any of the 0's. Thus the communication required is at least the logarithm of the "cover number" (the number of 1-rectangles needed), or equivalently, the logarithm of the rank[3] over the Boolean semiring $\mathbb{B}$ ($\{0,1\}$ with $\wedge$ and $\vee$; it is a *semi*ring because 1 has no additive inverse). As discussed earlier, the matrix corresponding to a fooling set of size $|F|$ has all 1's on its diagonal, no two of which may appear in the same all 1's minor, so the cover number, or the rank over $\mathbb{B}$, is $|F|$.

[2] The asymmetry with respect to OR/AND occurs because of the choice $f(x) = 1$ rather than $f(x) = 0$ in the definition of fooling sets.

[3] The *rank* of a matrix over a semiring is the fewest number of pairs of (column) vectors (v, w) such that $M = \sum_i v_i w_i^T$. This specializes to the "cover number" in the case of $\mathbb{B}$ and to the dimension of the column space in the case of a field.

Similarly, with PARITY nodes, the communication required in the corresponding communication game is the logarithm of the rank of the matrix over GF(2). Since column operations make the matrix lower triangular, it has full rank over GF(2) as well.

With AND nodes we have the dual of OR nodes: the communication complexity is equal to the nondeterministic communication complexity of the complement of the function, or the rank over $\mathbb{B}$ of the matrix with 0's and 1's reversed. Note that for a particular partition of the variables, this may be exponentially less than the case of OR nodes: the function **EQUAL?**(x, y) with respect to the partition $X \dot{\cup} Y$ requires nondeterministic complexity $|x| = |y|$ whereas its complement has nondeterministic complexity $2 \lg |x|$.

2.5.3 For arbitrary oblivious programs

Alon and Maass [AM88] prove strong lower bounds for arbitrary 3-way oblivious programs by analyzing the sequence S in which the variables are read by the levels of the program. In particular, for any two disjoint subsets of variables S and T, they consider the number of times this sequence alternates between reading variables of S and variables of T. They prove a theorem that says if for every two subsets $S \subset X$ and $T \subset Y$ with $|S| = |T| = n/2^m$ (where $|X| = |Y| = n$) there are at least m alternations between S and T, then the sequence must be of length at least $\Omega(nm)$.

They use this theorem to prove a superlinear lower bound on the length of oblivious branching programs for the "sequence equality function" **SEQ**, defined on two ternary vectors x and y of length n where each x_i and y_i may be 0, 1, or 2. **SEQ**$(x, y) = 1$ if the subsequence of x obtained by removing the 2's is equal to the subsequence of y obtained in the same manner. A standard "cut-and-paste" (or "crossing-sequence") argument shows that in any 3-way branching program[4] for **SEQ** and for any S and T as above, the number ℓ of alternations between S and T must satisfy $w^\ell \geq 2^{|S|}$ where w is the width of the program. So for $w = 2^{n/2^{2m}}$, this yields $\ell \geq 2^m$. In particular $\ell > m$, and so the theorem gives a lower bound of $\Omega(nm)$ on the length of the program.

[4] This is a branching program in which each node has 3 edges leaving it.

Thus any oblivious program for **SEQ** of size $2^{o(n)}$ must have superlinear length. This lower bound for 3-way programs clearly implies the same lower bound for ordinary branching programs, where each ternary variable x_i is represented by two binary variables. This implies, for instance, that **SEQ** $\notin$ C-IBDD. For comparison, **SEQ** has very easy read-once programs, which are non-oblivious, of length n.

In [KMW92], this lower bound for **SEQ** is extended to nondeterministic oblivious programs of linear length. At the same time, a simple co-nondeterministic oblivious program (with AND nodes) of linear length is given, showing that as for read-once programs (Section 2.2.2), the two types of nondeterminism give different computational power.

Babai, Nisan, and Szegedy [BNS92] in the same spirit improve this length/width tradeoff, using their lower bound for multiparty communication complexity to raise the lower bound on the length of polynomial-size oblivious programs (for a different function) by a factor of $\lg n$.

2.5.4 For read-once programs

Note first that the lower bound method for OBDD's is insufficient for read-once programs. Even though there may be many subfunctions arising from the settings to any $Y \subset X$ of a given size, it may also be that for each Y there is one subfunction that arises from many of the settings to Y. Since different paths may read the variables of X in different orders, different sets Y' may be the "first" ones read depending upon the values of the variables. In this case, we have not excluded the possibility that the first m input bits are read in such a way that the program needs nodes for only the "large" subfunctions on the various $\overline{Y}$ of size $n - m$.

For example, we saw that for the function **ACH**, there is a fooling set of size $2^{n/4}$ for any subset of half the X and Y variables (Theorem 2, specialized to OBDD's). However, there is a simple read-once program that reads the z variables first and then reads the X and Y variables in the appropriate order, pair by pair. Looking closely at this program, we see that there are n different subsets of half the X,Y variables that may be read first. For each subset there is a large fooling set, implying that there are many possible subfunctions on the remaining variables. However, the values of the variables (specifically, of the z variables) that give rise to these many subfunctions cause other paths to be taken through the program. For the path that leads to a given subset of X, Y, there are only *two* subfunctions (either the 0 function or the induced **ACH** function) arising from the *many* settings to the variables (in Z and the rest of X, Y) read so far.

In order to prove lower bounds for read-once programs, we must show that not only are there many subfunctions, but that each arises in very few ways. Simon and Szegedy [SS93] distill this idea into a lemma which may be considered a paradigm for proving read-once lower bounds. This technique appears implicitly in the read-once lower bounds of [We88, Za84] and explicitly in those of [Ju88, Kr88, Du85]; the generalization in [SS93] enables an easier proof of the lower bound of [BHST87] and others [We87, Du85, Ju88]. Simon and Szegedy use this technique to reprove a theorem of Babai et. al. [BHST87], that read-once programs require size $2^{\Omega(n^2)}$ to count modulo 2 the number of triangles in an n-node graph. They also give a simple proof that size $2^{n/10}$ is required to tell whether an n-node graph is $\frac{n}{2}$-regular. Since the lemma is a central part of our lower bound for multiplication, we provide a proof in Chapter 3.

2.6 Integer multiplication

By *integer multiplication*, we will refer to the Boolean function **MULT** : $\{0,1\}^{2n} \rightarrow \{0,1\}$ that computes the middle bit in the product of two n-bit integers. That is, **MULT**$(x, y) = z_{n-1}$ where $x = x_{n-1} \cdots x_0$, $y = y_{n-1} \cdots y_0$, and $z_{2n-1} \cdots z_0 = z = xy$ is the product of the integers represented in binary by x and y. The middle bit is the "hardest" bit, in the sense that if it can be computed by read-once branching programs (or most any computational model) of size $s(n)$, then any other bit can be computed with size at most $s(2n)$.

2.6.1 Bryant's lower bound

Bryant [Br91] gives the following lower bound for **MULT**; Gergov [Ge94] notices that the proof holds also for nondeterministic OBDD's, as noted the end of the proof below.

Theorem 3 **MULT** $\notin$ OBDD.

Proof: We will show that with respect to any subset $S \subset \{x_1, \ldots, x_n\}$ of size $n/2$ (corresponding to the first $n/2$ variables of X read by an OBDD), **MULT** has a fooling set of size $2^{n/8}$. The elements of the fooling set differ only in their settings to the x_i; the y_i are fixed so that the multiplication is reduced to computing the sum of two integers, one corresponding to a subsequence of $x_1, \ldots, x_{n/2}$ and the other corresponding to a subsequence of $x_{n/2+1}, \ldots, x_n$. The nth bit of the product is the high-order bit in this sum.

Choose these two subsequences so that for each i, the ith bit of one is in S and the ith bit of the other is in $\overline{S}$, and they are equally far apart in x for all i. To do this, let

$$S_L = S \cap \left\{x_1, \ldots, x_{n/2}\right\} \quad \text{and} \quad S_R = S \cap \left\{x_{n/2+1}, \ldots, x_n\right\}$$

and similarly define $\overline{S}_L$ and $\overline{S}_R$ for $\overline{S}$. It is easy to show that

$$|S_L \times \overline{S}_R| + |\overline{S}_L \times S_R| \geq n^2/8$$

and since $1 \leq |x_i - x_j| \leq n$ for each $(x_i, x_j) \in (S_L \times \overline{S}_R) \cup (\overline{S}_L \times S_R)$, we see that there is a subset of size $n/8$ with the desired property.

Exactly two bits of Y are set to 1 in such a way that these two subsequences "line up" and so that the carry out of their high-order bit corresponds to the nth bit in the product of x and y. The bits of X not contained in either subsequence are set to 0 unless they are in $\left\{x_{n/2+1}, \ldots, x_n\right\}$ and lie "in between" the bits of the subsequence. This causes carry bits of the addition to propagate as desired and thereby reduce the multiplication of x and y to the addition of the two integers determined by the subsequences. See Figure 2.2.

We may think of the addition of these two integers as the addition of an integer determined by the setting to S and an integer determined by the setting to $\overline{S}$. The fooling set ranges over all settings to the integer determined by S. Each of these two integers may take on any value between 0 and $2^{n/8} - 1$, in turn making the nth bit of the product is 1 if their sum is at least $2^{n/8}$ and 0 otherwise.

The corresponding matrix has rows indexed by all $2^{n/8}$ settings to S's integer and columns indexed by all $2^{n/8}$ settings to $\overline{S}$'s integer. After deleting the 0-column and

$$
\begin{array}{cccccccccccc|ccccccccc}
 & & 0 & 0 & x_i & 1 & x_j & 1 & 1 & x_k & 0 & x_p & 0 & x_q & 0 & 0 & x_r & 0 & 0 & 0 & = x \\
 & \times & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & = y \\
\cline{2-20}
 & 0 & 0 & x_i & 1 & x_j & 1 & 1 & x_k & 0 & x_p & 0 & x_q & 0 & 0 & x_r & 0 & 0 & 0 & & \\
\cdots & x_k & 0 & x_p & 0 & x_q & 0 & 0 & x_r & 0 & 0 & 0 & & & & & & & & & \\
\hline
 & & . & . & & . & & & . & & & & & & & & & & & & \\
 & & \uparrow & & & & & & & & & & & & & & & & & & \\
 & \multicolumn{4}{l}{\mathbf{MULT}(x,y)} & & & & & & & & & & & & & & & &
\end{array}
$$

Figure 2.2: The multiplication of x and y is reduced to computing the carry bit in the sum of the integers represented by the two subsequences $x_i x_j x_k$ and $x_p x_q x_r$. For each corresponding pair (x_i, x_p), (x_j, x_q), and (x_k, x_r), one variable is in S and the other in $\overline{S}$. The fooling set ranges over all settings to the variables in S, with each variable's "partner" getting the complementary setting. The remaining variables are set as shown in order to achieve the desired reduction.

0-row and indexing appropriately, this matrix is lower-triangular with all 1's in the lower half. It thus has full rank over $\mathbb{B}$ and over GF(2), and so does its complement. It follows that **Mult** requires exponential-size OBDD's and k-OBDD's even if OR, PARITY, or AND nodes are present. ∎

Gergov [Ge94] further generalizes Bryant's lower bound for **Mult** to arbitrary oblivious programs of linear length by using the main lemma from [AM88]. For any program of length kn, the lemma implies the existence of two "large" disjoint subsets of X (size $n/k2^{2k}$) such that there are few ($O(k)$) levels where the program changes from reading variables of one set to reading variables of the other set. Now reduced to a problem of communication complexity with $2k$ rounds, it is easy to carry though the rest of Bryant's proof to find a fooling set of size $2^{n/k^2 2^{4k}}$. Thus, the program has size at least $2^{n/2k^3 2^{4k}}$. As reasoned above, this bound holds even if nondeterministic nodes are present.

2.6.2 The decision problem DMULT—the graph of multiplication

Although it is not directly related to the issue of verification, another Boolean function that has been considered is the decision problem **DMULT** of recognizing the graph of multiplication. That is, **DMULT**$(x, y, z) = 1$ if $xy = z$. Note that it is not readily apparent which problem is "harder", **MULT** or **DMULT**. On the one hand, **DMULT** seems to require practically computing all the bits of xy; however, an algorithm for **DMULT** has the advantage of inspecting all the bits of z, the putative product. Buss [Bu92] proves that **DMULT** $\notin \mathrm{AC}^0$ by reducing it to counting the number of 1's in the input (and therefore to **MULT** and to **PARITY** by results of [CSV84]); for comparison, [FSS84] gives an easy reduction of **MULT** to **PARITY** to show **MULT** $\notin \mathrm{AC}^0$.

A simple argument [We94] shows that computing **DMULT** with read-once programs is as hard as factoring. Given a polynomial-size read-once program for **DMULT** and any integer n, the following procedure will either factor n or determine that it is prime. First instantiate n as the bits of z in the read-once program where $|z| = 2\lg n$ and $|x| = |y| = \lg n$. There is a satisfying assignment to the remaining input bits since $1z = z$. Now attempt to construct a nontrivial factor by instantiating the bits of x one at a time, maintaining the satisfiability of the program after each bit. If the only successful instantiations for x are 1 and z, then z is prime; otherwise, a nontrivial factor is determined. Since we can test the satisfiability of a read-once program in polynomial time, the entire procedure can be executed in polynomial time.

Jukna [Ju94] proves a lower bound of $2^{n^{1/4}/k^{2k}}$ for **DMULT** on non-deterministic read-k-times branching programs. His lower bound follows the framework of [BRS93], and gives a simple reduction of **DMULT** to the problem of recognizing codewords of a linear code, for which a lower bound of $2^{\sqrt{n}/k^{2k}}$ is proved in [Ju92].

2.7 Related issues

2.7.1 The ordering problem for OBDD's

When using OBDD's for verification, it is naturally desired to minimize their size. For a given function, the order in which the variables are read greatly affects the number of nodes required—it is easy to exhibit functions which have small OBDD's for good orderings but require exponential size for poor orderings. Thus, an important and interesting question is how to determine the ordering that minimizes size for a given function. The decision problem is: Given an OBDD and an integer k, determine whether there is an OBDD (possibly obeying a different ordering of the variables) with fewer than k nodes that computes the same function. This problem was recently proved to be NP-complete in [BW95], extending the work of [BW95, THY93], via a nice reduction to **Optimal-Linear-Arrangement** [GJ79].

It would be useful to find an efficient algorithm to determine an approximately optimal ordering. Many heuristics for improving an ordering can be found in the literature (see [BW95]). It is worth mentioning that the use of randomization has not been explored, either in helping to determine good variable orderings or in the verification strategy more generally.

2.7.2 The Fourier spectrum

The Fourier spectrum of Boolean functions has been widely studied over the past few years. Properties of the Fourier spectrum have been used in a variety of applications, perhaps most strikingly in deriving efficient algorithms for learning (e.g., [KM91]). Two properties of the spectrum that have proven useful for this purpose are small L_1-norm (that is, the sum of the absolute values of the coefficients) and a knowledge of which coefficients are the largest. For example, [KM91] gives an efficient algorithm for functions whose spectrum is either sparse or has polynomial L_1-norm.

It is easy to show that the L_1-norm of a function is bounded by the number of leaves in any decision tree for that function, even if the nodes may query the parity of arbitrary subsets of the variables. And [LMN89] proves that functions in AC^0 have most of the weight of their spectrum in the coefficients of small sets. These results are used to derive efficient learning algorithms for functions in AC^0 and functions with shallow decision trees.

Since OBDD's are such a constrained model of computation, perhaps interesting and useful properties can be derived about the spectrum of the functions in OBDD they compute. Some negative results are known: Bruck and Smolensky [BS90] demonstrate a function in AC^0 that has exponential L_1-norm; this function is easily computed by polynomial-size OBDD's. They also exhibit a function (inner product modulo 2), also easily computed by an OBDD, whose transform has L_∞-norm less than $1/2^{\log^{O(1)} n}$. This is an an even stronger result and further implies that any polynomial $p(x_1, \ldots, x_n)$ whose sign represents this function (i.e., whose is negative exactly when the function is 1) must have $2^{\log^{O(1)} n}$ non-zero coefficients.

Comparing OBDD's with constant-depth circuits, we note that **Parity**, though not in AC^0, is easily computed by small OBDD's, while **π-Matrix** is easily in AC^0 but requires exponential-size OBDD's.

2.7.3 Read-once programs and resolution proofs

If we consider branching programs for computing multi-valued functions, we may find a nice correspondence with resolution proofs.

A *resolution proof* for a CNF formula ϕ is a straightline program for proving that ϕ is not satisfiable. At each step, two previously obtained clauses, $(x_i \vee \alpha)$ and $(\overline{x_i} \vee \beta)$, are "resolved on x_i" to obtain a new clause $(\alpha \vee \beta)$ which is satisfiable if the previous clauses are (α and β are disjunctions of literals). The proof is complete when the empty clause is obtained. Such a proof is naturally viewed as a directed acyclic graph where the clauses correspond to the nodes of the graph: the original clauses of ϕ are "input" nodes with indegree 0, the newly obtained clauses are "internal" nodes with indegree 2, and the empty clause is the "output" node with outdegree 0. Such a resolution proof is called *regular* if on every directed path from an input node to an output node, each variable is resolved at most once.

We may consider a branching program for an unsatisfiable CNF formula ϕ that solves the following "search" problem: given an assignment x, find a clause of ϕ that is not satisfied. It is an observation of Chvatal and Szemeredi (see [LNNW95]) that read-once programs for this problem are isomorphic to regular resolution proofs. Taken together with the fact that a decision tree is a read-once branching program, [LNNW95] notes that $D(\phi) \geq \lg RRES(\phi)$, where $D(\phi)$ is the depth of the shallowest decision tree for this search problem and $RRES(\phi)$ is the fewest number of steps in a regular resolution proof of ϕ.

In general, an arbitrary resolution proof for ϕ yields a branching program for this search problem, but not vice-versa: in fact, there are formulas for which $RES(\phi)$ is exponential [CS88, e.g.], even though there is always a branching program of size $O(|\phi|)$ for the search problem.

Chapter 3

A lower bound for multiplication with read-once programs

This chapter describes a lower bound of $2^{\Omega(\sqrt{n})}$ on the size of read-once branching programs for the function **Mult**. This is the first superpolynomial lower bound for multiplication on non-oblivious branching programs. This result demonstrates that relaxing the ordering restriction of OBDD's is insufficient to gain the computational power desired for the purpose of hardware verification.

The lower bound for multiplication is motivated by the work of Simon and Szegedy [SS93], who give a basic lemma for proving lower bounds on the size of read-once branching programs. The lemma involves Neciporuk's method of counting the subfunctions that are possible when some subset of input bits is fixed. We begin by describing this lemma in Section 3.1. For ease of presentation we first prove a lower bound of $2^{\Omega(\sqrt[3]{n})}$ in Section 3.2, and then extend the proof to achieve $2^{\Omega(\sqrt{n})}$ in Section 3.3.

In Section 3.4, we define the notion of *read-once reductions* in order to deduce similar lower bounds for other arithmetic functions.

3.1 A paradigm for read-once lower bounds

Let f be a Boolean function, $f : \{0,1\}^n \to \{0,1\}$, and let $X = \{x_0, \ldots, x_{n-1}\}$ be its n binary input variables. Let $\mathcal{F}$ be a filter on X. (That is, $\mathcal{F} \subseteq 2^X$ and $\mathcal{F}$ is closed upward—if $S \in \mathcal{F}$, then all supersets of S are in $\mathcal{F}$.) A subset $B \subset X$ is said to be in the *boundary* of $\mathcal{F}$ if $B \notin \mathcal{F}$ but $(B \cup x_i) \in \mathcal{F}$ for some x_i. By setting the values of $\overline{B} = X \setminus B$, we naturally induce a function on B. The lemma is stated below in the form we will need it; it appears in [SS93] in slightly more generalized form.

Lemma 1 (Simon and Szegedy) *If for any B in the boundary of $\mathcal{F}$, at most $2^{|\overline{B}|}/L$ settings to $\overline{B}$ induce the same subfunction on B, then any read-once branching program computing f has size at least L.*

For completeness, we now provide a proof of this lemma.

Proof: The idea is to identify a "frontier" of edges in the branching program—a cut containing exactly one edge from each source-to-sink path—in which every edge allows only a fraction $1/L$ of the inputs in $\{0,1\}^n$ to pass through it. Since the path of every input passes through some frontier edge, there must be at least L such edges. Having fan-out 2 and only one root, the program also has at least L nodes. This is because if the endvertices of the frontier edges were distinct, they would be the leaves of an embedded binary tree which must contain $L-1$ distinct internal nodes. Since the two sinks are not among these internal nodes, there are at least $L+1$ nodes in the program.

In order to characterize a frontier, we first associate with each node of the program the set of variables appearing in the subprogram rooted there—that is, those variables appearing on nodes that are reachable from the given node. Clearly, along any path through the program, the variable-sets of later nodes are subsets of the variable-sets of earlier nodes. A frontier consists of those edges going from nodes with "large" sets of variables to nodes with "small" sets. "Large" sets are defined to be those that are in the filter $\mathcal{F}$. Clearly there is exactly one frontier edge on each source-to-sink path, as (for nontrivial filters $\mathcal{F}$) the root has the variable-set $X \in \mathcal{F}$ and the sinks have the variable-set $\emptyset \notin F$. With each frontier edge we associate a set $B \subset X$ in the boundary of $\mathcal{F}$.

Suppose boundary set B is associated with a given frontier edge. Because the program is read-once, these variables do not appear on any path from the root to this edge. In fact, the inputs $x \in \{0,1\}^n$ that reach this edge are characterized exactly by their settings to $\overline{B}$. Each setting to $\overline{B}$ that reaches this edge clearly induces the same subfunction on B, as defined by the subprogram rooted there. Since at most $2^{|\overline{B}|}/L$ settings to $\overline{B}$ give the same subfunction on B, at most $(2^{|\overline{B}|}/L) \cdot 2^{|B|} = 2^n/L$ inputs in $\{0,1\}^n$ may pass through this frontier edge. The lower bound of L then follows. ∎

3.2 A lower bound of $2^{\Omega(\sqrt[3]{n})}$

Theorem 4 *Any read-once branching program for* **MULT** *has size* $2^{\Omega(\sqrt[3]{n})}$.

Proof: Let $m = \sqrt[3]{n}/4$ and let X and Y denote the sets of variables $X = \{x_0, \ldots, x_{n-1}\}$ and $Y = \{y_0, \ldots, y_{n-1}\}$. Define the filter

$$\mathcal{F} = \{V \subset (X \cup Y) : |V \cap X| > n - m \text{ and } |V \cap Y| > n - m\}.$$

Roughly speaking this filter marks the frontier of the program where at most m bits of X and at most m bits of Y have been read.[1]

We will show that for any B in the boundary of $\mathcal{F}$, at most $2^{|\overline{B}|-m}$ settings to $\overline{B}$ give the same subfunction on B. By Lemma 1, this gives the desired lower bound of 2^m. Fix any B in the boundary of $\mathcal{F}$ and let $S = \overline{B}$. Think of S as being the variables already read by the branching program. Since B is in the boundary of $\mathcal{F}$, either $|S \cap X| = m$ or $|S \cap Y| = m$. We will show that there is a subset $S' \subset S$ of size at least m such that if two settings to S differ on S' then they induce different subfunctions on $\overline{S} = B$. Thus at most $2^{|S|-m}$ settings to $S = \overline{B}$ induce the same subfunction on $\overline{S} = B$, as desired. We will show that the two subfunctions are different by explicitly demonstrating a single setting to the bits of $\overline{S}$ where the induced subfunctions of **MULT** differ.

Suppose without loss of generality that $|S \cap X| = m$ (and $|S \cap Y| < m$). Let $i \in \{0, \ldots, n-1\}$ be the smallest index such that $y_i \notin S$. Let

$$S' = \{y_0, \ldots, y_{i-1}\} \cup \Big(S \cap \{x_0, \ldots, x_{n-1-i}\}\Big).$$

Note that because $\{y_0, \ldots, y_{i-1}\} \subseteq S$ and $|S \cap X| = m$, we have $|S'| \geq m$.

[1] In order for this notion to be strictly correct, "have been read" must be interpreted to mean "appear on any path from the root".

Let us adopt the following notation for the integers obtained from partial settings to the variables. For a setting α to $W \subseteq X \cup Y$ (i.e., $\alpha : W \rightarrow \{0,1\}$), let x_α denote the integer that is represented in binary when the variables of $X \cap W$ have the value given by α and the variables of $X \cap \overline{W}$ are each 0. Define y_α similarly. For a single variable $z \notin W$, let "$\alpha + z$" denote the setting to $W \cup \{z\}$ that further sets $z = 1$. For two settings α and τ to disjoint subsets W and V, let "$\alpha \cup \tau$" denote the setting equal to α on W and to τ on V. Finally, let $(x)_i$ denote the ith bit in the binary representation of integer x, so $x = \sum_{i=0}^{n-1} (x)_i \, 2^i$.

Let α and β be two settings to S that differ on some bit in S'. Our goal is thus to find a setting τ to the bits of $\overline{S}$ so that $(x_{\alpha\cup\tau} y_{\alpha\cup\tau})_{n-1} \neq (x_{\beta\cup\tau} y_{\beta\cup\tau})_{n-1}$.

We proceed in two stages, according to Lemmas 2 and 3. First we ensure, by setting to 1 (if necessary) a single variable z of $\overline{S}$, that the two products $x_{\alpha+z} y_{\alpha+z}$ and $x_{\beta+z} y_{\beta+z}$ differ in a "high-order" bit—a bit position in the range $[n-m-3, n-1]$ (we aren't concerned with higher bit positions). In the second stage, we set to 1 a pair of variables of $\overline{S}$, one in X and one in Y, so that the resulting product differs in a *higher* high-order bit position. We iterate this second stage, repeatedly setting a pair of variables until the resulting products differ in bit position $n-1$. It follows that α and β induce different subfunctions on $\overline{S}$—the subfunctions differ when $\overline{S}$ has z and the pairs from the second stage all set to 1 and the remaining bits of $\overline{S}$ set to 0.

Lemma 2 *If for all $i \in [n-m-3, n-1]$ we have $(x_\alpha y_\alpha)_i = (x_\beta y_\beta)_i$, then there is a single variable $z \in \overline{S}$ such that*

$$(x_{\alpha+z} y_{\alpha+z})_i \neq (x_{\beta+z} y_{\beta+z})_i$$

for some $i \in [n-m-3, n-1]$.

Lemma 3 *Let $T \subset X \cup Y$, and α and β be two settings to T. Let d be the greatest index in $[0, n-2]$ such that $(x_\alpha y_\alpha)_d \neq (x_\beta y_\beta)_d$. If $d \geq n-m-3$ and $\max(|T \cap X|, |T \cap Y|) = t \leq 3m$, then there are two variables, $x_u \in X \cap \overline{T}$ and $y_v \in Y \cap \overline{T}$, such that*

$$(x_{\alpha'} y_{\alpha'})_{d+1} \neq (x_{\beta'} y_{\beta'})_{d+1}$$

where $\alpha' = \alpha + x_u + y_v$ and $\beta' = \beta + x_u + y_v$.

Theorem 4 follows from these lemmas as outlined above. Notice that Lemma 3 is first applied with $t \leq m+1$, and since we must apply Lemma 3 at most $m+3$ times, each time setting one more variable of X and Y, we maintain $t \leq 2m+4 \leq 3m$ as required. ■

We now give the proofs of Lemmas 2 and 3.

Proof of Lemma 2: The settings α and β differ on $S' \subseteq S$; suppose first that they differ in a bit of $S' \cap X$.

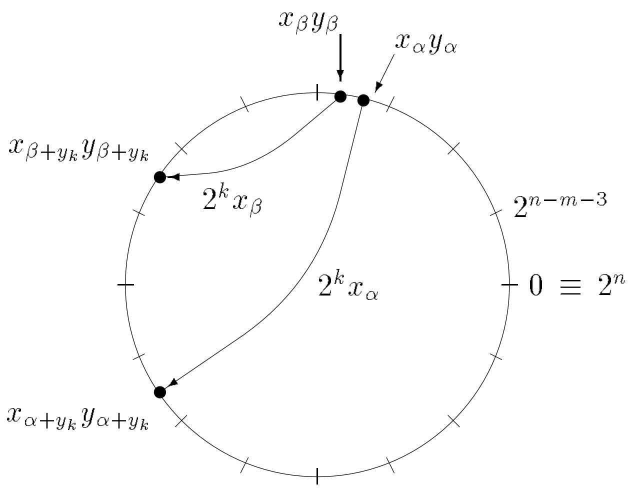

Figure 3.1: The integers modulo 2^n. In order for $x_{\beta+y_k} y_{\beta+y_k}$ and $x_{\alpha+y_k} y_{\alpha+y_k}$ to fall into different segments, we must choose k so that $2^k(x_\alpha - x_\beta)$ has large magnitude.

The proof is most easily explained by picturing the integers modulo 2^n on a circle. Partition the circle into 2^{m+3} equal-sized segments according to the values of the $m+3$ highest bits, so each segment contains 2^{n-m-3} consecutive integers, as depicted in Figure 3.1. The hypothesis of the lemma is that $x_\alpha y_\alpha$ and $x_\beta y_\beta$ fall into the same segment. If we set bit $y_k \in \overline{S} \cap Y$ to 1, we obtain the products $x_{\alpha+y_k} y_{\alpha+y_k} = x_\alpha y_\alpha + x_\alpha 2^k$ and $x_{\beta+y_k} y_{\beta+y_k} = x_\beta y_\beta + x_\beta 2^k$. The product $x_{\alpha+y_k} y_{\alpha+y_k}$ is obtained by a translation of $2^k x_\alpha$ along the circle from $x_\alpha y_\alpha$, and $x_{\beta+y_k} y_{\beta+y_k}$ is obtained by a translation of $2^k x_\beta$ from $x_\beta y_\beta$. If, modulo 2^n, their difference $2^k(x_\alpha - x_\beta)$ is at least 2^{n-m-2}, or two segments long, and at most $2^n - 2^{n-m-2}$, or "negative two" segments long, then it is clear that the translates $x_{\alpha+y_k}y_{\alpha+y_k}$ and $x_{\beta+y_k}y_{\beta+y_k}$ fall into different segments. It follows that the products $x_{\alpha+y_k}y_{\alpha+y_k}$ and $x_{\beta+y_k}y_{\beta+y_k}$ differ in a high-order bit position.

It only remains to show how to choose $y_k \in \overline{S} \cap Y$ so that $2^{n-m-2} \leq 2^k(x_\alpha - x_\beta) \leq 2^n - 2^{n-m-2}$ modulo 2^n. Let $\overline{x} = x_\alpha - x_\beta$. It is useful now to think in terms of the table generated by the usual grade-school algorithm for multiplying $\overline{x}$ by y, as shown in Figure 3.2.

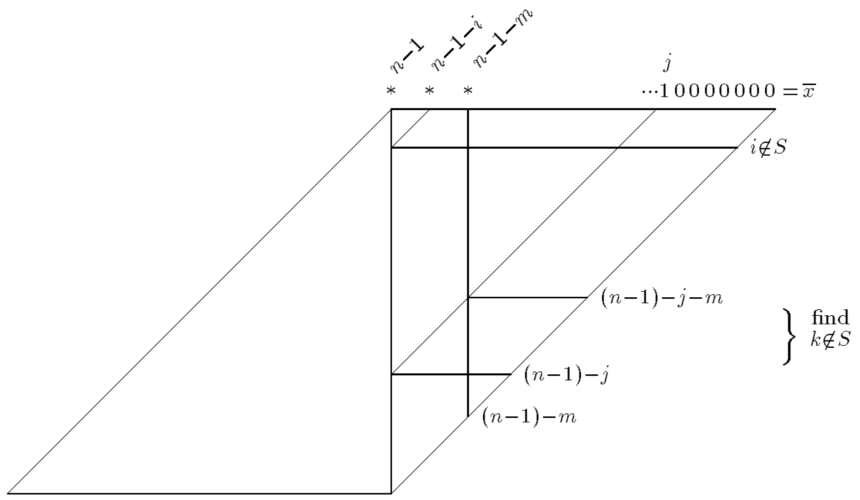

Figure 3.2: The table generated by the grade-school algorithm for multiplying $\overline{x} = x_\alpha - x_\beta$ by y. We choose a bit y_k to set to 1 so that the least significant 1 in $\overline{x}$ is shifted into a "high-order" bit position.

In this table, the rows are the partial products, indexed by $y_0, \ldots, y_{n-1}$. The diagonals are indexed by $\overline{x}_{n-1}, \ldots, \overline{x}_0$. Since α and β differ in a bit of $S' \cap X \subseteq \{x_0, \ldots, x_{n-1-i}\}$, the difference $\overline{x} = x_\alpha - x_\beta$ must have a 1 somewhere in the range of bit positions $[0, n-1-i]$. Let j be the position of the least significant 1 in $\overline{x}$, so that either there is a 0 in position $j-1$, or $j = 0$. We now choose any variable of $\overline{S} \cap Y$ with index k in the range $[(n-1)-j-m,\ (n-1)-j]$. This range must contain a variable $y_k \in \overline{S} \cap Y$ because if $j \leq n-m-1$, the range has at least $m+1$ elements but $|S \cap Y| < m$; if $j \geq n-m$, we may choose $k = i$ (by definition, $y_i \notin S$), which lies in the range $[0, n-1-j]$ since $j \leq n-i-1$. This ensures that $2^k\overline{x}$ has a 1 in position

$j+k$ and a 0 in position $j+k-1$, where $n-1-m \leq j+k \leq n-1$. It follows that modulo 2^n, we have $2^{n-m-1} \leq 2^k\overline{x} \leq 2^n - 1 - 2^{n-m-2}$, the upper bound attained if all bits except bit $j+k-1$ are 1's and $j+k = n-1-m$. This satisfies the desired bounds.

If α and β differ in a bit of $S' \cap Y \subseteq \{y_0, \ldots, y_{i-1}\}$ the proof is essentially the same. We have to choose $x_k \in \overline{S} \cap X$ so that $2^{n-m-1} \leq 2^k(y_\alpha - y_\beta) \leq 2^n - 2^{n-m-2} - 1$ modulo 2^n. In this case, we know $\overline{y} = y_\alpha - y_\beta$ has a 1 in the range $[0, i-1]$. Again letting j be the least significant 1 of $\overline{y}$ in this range, we simply choose k anywhere in the range $[n-1-j, n-1-j-m]$. Since $j \leq i-1 \leq m$ and $n \geq \sqrt[3]{n} \geq 1+j+m$, this range always has $m+1$ elements. It follows as before that $2^k\overline{y}$ satisfies the desired inequality. This completes the proof. ■

Lemma 3 *Let $T \subset X \cup Y$, and α and β be two settings to T. Let d be the greatest index in $[0, n-2]$ such that $(x_\alpha y_\alpha)_d \neq (x_\beta y_\beta)_d$. If $d \geq n-m-3$ and $\max(|T \cap X|, |T \cap Y|) = t \leq 3m$, then there are two variables, $x_u \in X \cap \overline{T}$ and $y_v \in Y \cap \overline{T}$, such that*

$$(x_{\alpha'} y_{\alpha'})_{d+1} \neq (x_{\beta'} y_{\beta'})_{d+1}$$

where $\alpha' = \alpha + x_u + y_v$ and $\beta' = \beta + x_u + y_v$.

Proof of Lemma 3: We will consider all pairs of variables (x_u, y_v) such that $u+v = d$. We want $(x_{\alpha'} y_{\alpha'})_{d+1} \neq (x_{\beta'} y_{\beta'})_{d+1}$, where

$$\begin{aligned} x_{\alpha'} y_{\alpha'} &= (x_\alpha + 2^u)(y_\alpha + 2^v) \\ &= \left(x_\alpha y_\alpha + 2^d\right) + (2^v x_\alpha + 2^u y_\alpha), \\ \text{and} \quad x_{\beta'} y_{\beta'} &= (x_\beta + 2^u)(y_\beta + 2^v) \\ &= \left(x_\beta y_\beta + 2^d\right) + (2^v x_\beta + 2^u y_\beta). \end{aligned}$$

Since d is the highest bit in which $x_\alpha y_\alpha$ and $x_\beta y_\beta$ differ, clearly $\left(x_\alpha y_\alpha + 2^d\right)_{d+1} \neq \left(x_\beta y_\beta + 2^d\right)_{d+1}$. We will choose u and v so that the addition of the "cross terms" $2^v x_\alpha + 2^u y_\alpha$ to $x_\alpha y_\alpha + 2^d$ does not affect bits d or $d+1$ of $x_\alpha y_\alpha + 2^d$ (and similarly for β). In order to do this, we choose u and v so that in each case, the cross terms have 0's in bit positions d and $d+1$ and furthermore, in the addition of the two integers, there is no carry bit into position d.

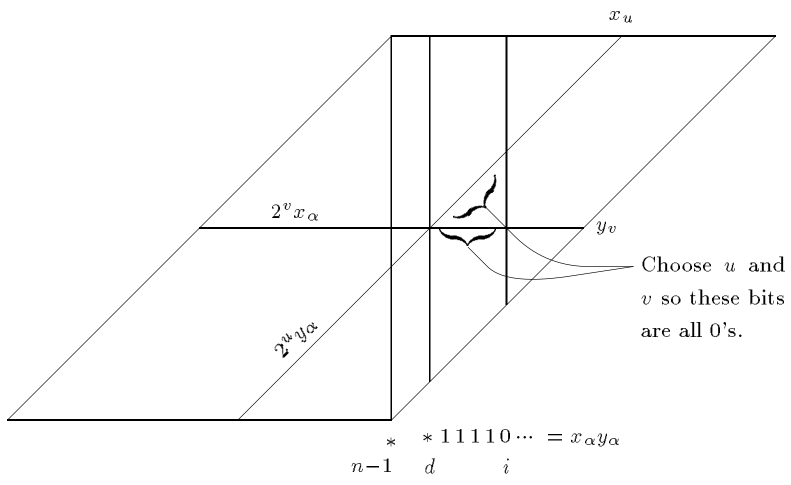

Figure 3.3: In Lemma 3, we choose x_u and y_v to set to 1 so that $u+v=d$ and also so that the products $2^u y_\alpha$ and $2^v x_\alpha$ have 0's in bit positions $d-1, \ldots, i-1$ so that when added to $x_\alpha y_\alpha + 2^d$, they do not cause a carry to propagate into position $d+1$.

To accomplish this, we first find the largest bit position i less than d where $x_\alpha y_\alpha$ has a 0 (so positions $i+1$ through $d-1$ are all 1's). We will choose u and v so that $2^v x_\alpha$ and $2^u y_\alpha$ each has 0's in positions $i-1$ through $d+1$. It follows that their sum then has 0's in positions i through $d+1$, and so, when added to $x_\alpha y_\alpha + 2^d$ which has a 0 in position i, causes no carry into any position $i+1$ through d (see Figure 3.3). We will choose u and v so that the same conditions hold for β as well.

A simple counting argument now shows that there exist u and v as desired. First, we claim that $x_\alpha y_\alpha$ (and $x_\beta y_\beta$) has 1's in at most t^2 bit positions, so that $(d-1)-i \leq t^2$. In general, if the binary representations of integers p and q have $w(p)$ and $w(q)$ 1's in them respectively, then clearly $p+q$ has at most $w(p)+w(q)$ 1's in it. Recall α sets at most t bits in X or Y. We may therefore view $x_\alpha y_\alpha$ as the addition of at most t shifts of x_α, and the claim follows.

We require $(2^v x_\alpha)_j = (2^v x_\beta)_j = 0$ in at most t^2+4 positions j: $j = d+1, d, d-1, \ldots, i, i-1$. There are at most t bit positions in which either x_α or x_β has a 1, and for each such 1, there are at most t^2+4 "bad" values of $v \in [0, n-1]$ that shift the 1 to a position we require to be 0. Thus, x_α and x_β rule out at most $t(t^2+4)$ values of v. Furthermore, there are up to t variables of Y that are in T, making a total of $t(t^2+4)+t$ values of v that we may not choose. Similarly, a total of at most $t(t^2+4)+t$ values of u are ruled out by y_α, y_β, and T. The number of pairs (x_u, y_v) in which either x_u or y_v has been ruled out is thus at most

$$2(t^3+5t) \leq 2\left(27m^3+15m\right) \leq 2\left(\frac{27n}{64}+\frac{15\sqrt[3]{n}}{4}\right)$$

since $t \leq 3m$ and $m = \sqrt[3]{n}/4$. There are at least $d+1 \geq n-m-2$ pairs (x_u, y_v) such that $u+v=d$. Thus we retain at least

$$n - \frac{\sqrt[3]{n}}{4} - 2 - \left(\frac{54}{64}n + \frac{30}{4}\sqrt[3]{n}\right) = \Omega(n)$$

good pairs satisfying the desired requirements for x_u and y_v. For $n \geq 378$, this expression is greater than 1, implying that there exists a pair as desired. ■

3.3 Improving the bound to $2^{\Omega(\sqrt{n})}$

We can improve the lower bound to $2^{\Omega(\sqrt{n})}$ by analyzing more closely how we iterate Lemma 3 in the proof of the theorem. We begin with the observation that we needed $m = O(\sqrt[3]{n})$ because in Lemma 3, we used $t^2 = O(m^2)$ as an upper bound on the number of consecutive 1's to the right of position d in $x_\alpha y_\alpha$ or $x_\beta y_\beta$. We then required 0's in these $O(m^2)$ positions in the cross terms $2^v x_\alpha + 2^u y_\alpha$ and $2^v x_\beta + 2^u y_\beta$. Since each of the $O(m)$ 1's in x_α may then rule out $O(m^2)$ values of v, we needed $O(m^3) < n$ in order not to rule out all values of v. In order to allow $m = O(\sqrt{n})$, we will reduce to $O(m)$ the number of positions in which we require 0's in the cross terms. **For the rest of this section, we let $m = \sqrt{n}/3$.**

For example, if we knew that $x_\alpha y_\alpha$ and $x_\beta y_\beta$ looked like[2] $\begin{array}{l} x_\alpha y_\alpha = \cdots \underset{d}{1}\,0 \cdots \\ x_\beta y_\beta = \cdots \underset{d}{0}\,0 \cdots \end{array}$, then we would need to require 0's in the cross terms in only three positions: $d+1$, d, and $d-1$. This is sufficient to ensure that the addition of cross term $2^v x_\alpha + 2^u y_\alpha$ to

[2] Here and henceforth, "$\cdots$" denotes an arbitrary string of 0's and 1's; thus $x_\alpha y_\alpha = \cdots \underset{d}{1}\,0 \cdots$ has a 1 in bit d, a 0 in bit $d-1$, and may have any values in other bit positions.

$x_\alpha y_\alpha + 2^d$ does not generate a carry into position d and does not affect bits d or $d+1$ of $x_\alpha y_\alpha + 2^d$. The same holds for β and we get $(x_{\alpha'} y_{\alpha'})_{d+1} \neq (x_{\beta'} y_{\beta'})_{d+1}$. With only these three positions required to be 0's, the total number of v's ruled out by x_β and x_α is proportional to the number of 1's they contain, which is $O(m)$. Similarly, the cases

$$\begin{aligned} x_\alpha y_\alpha &= \cdots \underset{d}{1}\,1 \cdots \\ x_\beta y_\beta &= \cdots \underset{d}{0}\,0 \cdots \end{aligned} \qquad \text{and} \qquad \begin{aligned} x_\alpha y_\alpha &= \cdots \underset{d}{1}\,1 \cdots \\ x_\beta y_\beta &= \cdots \underset{d}{0}\,1 \cdots \end{aligned}$$

can be handled with only a few constraints by choosing $u + v = d - 1$ (this will be proved in Lemma 5). In fact, there is really only one case in which we need to require $(2^v x_\beta + 2^u y_\beta)$ or $(2^v x_\alpha + 2^u y_\alpha)$ to have many 0's:

Definition 4 *Let d be the greatest index less than n in which $(x_\alpha y_\alpha)_d \neq (x_\beta y_\beta)_d$. We say that $x_\alpha y_\alpha$ and $x_\beta y_\beta$ are <u>k-bad</u> if $d \geq n - m - 4$ and the products look like*

$$\begin{aligned} x_\alpha y_\alpha &= \cdots \underset{d}{1}\,0 \cdots\cdots \\ x_\beta y_\beta &= \cdots \underset{d}{0}\,1\,1\,1\,1\,\underset{\underset{n-m-6}{\uparrow}}{1}\,\underbrace{1\,1\,1\,1\,1\,1}_{k} \cdots \end{aligned}$$

or vice versa (exchanging α and β).

In this case, say $x_\beta y_\beta = \cdots \underset{d}{0}\,1\,1\,1\,1\,\underset{\underset{n-m-6}{\uparrow}}{1}\,\underbrace{1\,1\,1\,1\,1\,1}_{k} \cdots$, we must require $2^v x_\beta + 2^u y_\beta$ to be 0 in the positions of each of these 1's in order to prevent a carry into position $d+1$ when we add it to $x_\beta y_\beta + 2^d$. In order to allow $m = O(\sqrt{n})$, we will ensure that the products are not k-bad for $k > m + 4$. Then the number of v's ruled out by each 1 of x_α and x_β is $2m + 10$, and as long as the number of 1's in x_α or x_β is $O(m)$, the total number of v's ruled out is $O(m^2)$.

We will first show that we may begin with products that differ in a high-order bit but are not 1-bad, and then prove a version of Lemma 3 in which each application allows the "badness" to grow by at most 1.

Lemma 4 *For any two settings α and β to S that differ on a bit of S', there are three (or fewer) variables $x_u, y_v, z \in \overline{S}$ such that for $\alpha' = \alpha + x_u + y_v + z$ and $\beta' = \beta + x_u + y_v + z$, the products $x_{\alpha'} y_{\alpha'}$ and $x_{\beta'} y_{\beta'}$ differ in a high-order bit (in the range $[n-m-4, n-1]$) and moreover, are not 1-bad.*

(The comment "or fewer" refers to the fact that we may not need to set some or any of these three variables.)

Lemma 5 *Let $T \subset X \cup Y$, and α and β be two settings to T. Let d be the greatest index in $[0, n-2]$ such that $(x_\alpha y_\alpha)_d \neq (x_\beta y_\beta)_d$. Suppose $d \geq n-m-4$ and $\max(|T \cap X|, |T \cap Y|) = t \leq 2m+5$ and also that $x_\alpha y_\alpha$ and $x_\beta y_\beta$ are not k-bad, for some $k \leq m+4$. Then there are two variables, $x_u, y_v \in \overline{T}$, such that*

$$(x_{\alpha'} y_{\alpha'})_{d+1} \neq (x_{\beta'} y_{\beta'})_{d+1}$$

for $\alpha' = \alpha + x_u + y_v$ and $\beta' = \beta + x_u + y_v$, and moreover, $x_{\alpha'} y_{\alpha'}$ and $x_{\beta'} y_{\beta'}$ are not $(k+1)$-bad.

We now have

Theorem 5 *Any read-once branching program for* **Mult** *has size $2^{\Omega(\sqrt{n})}$.*

Proof: The proof is exactly the same as the proof of Theorem 4 except for the lemmas. We start with products that differ in a high-order bit but are not 1-bad, as provided by Lemma 4. The number of variables in X or Y set in these products is at most $m+2$. We obtain a difference in bit $n-1$ by iterating Lemma 5 at most $m+3$ times, each time setting at most one variable in X and in Y. This maintains $t \leq (m+2)+(m+3)$ and $k \leq 1+(m+3)$ as required. ■

We now give the proofs of Lemmas 4 and 5, which we restate for convenience.

Lemma 4 *For any two settings α and β to S that differ on a bit of S', there are three (or fewer) variables $x_u, y_v, z \in \overline{S}$ such that for $\alpha' = \alpha + x_u + y_v + z$ and $\beta' = \beta + x_u + y_v + z$, the products $x_{\alpha'} y_{\alpha'}$ and $x_{\beta'} y_{\beta'}$ differ in a high-order bit (in the range $[n-m-4, n-1]$) and moreover, are not 1-bad.*

Proof: Either $x_\alpha y_\alpha$ and $x_\beta y_\beta$ differ (modulo 2^n) by at least 2^{n-m-3} or not. If they do, then they must differ in a high-order bit (in the range $[n-m-4, n-1]$). If not, we proceed just as in Lemma 2 to find a variable z such that $x_{\alpha+z} y_{\alpha+z}$ and $x_{\beta+z} y_{\beta+z}$

differ by at least 2^{n-m-3}: As in Lemma 2, when α and β differ in a bit of $S' \cap X$, it is sufficient to set to 1 a variable $y_k \in \overline{S} \cap Y$ such that $2^k(x_\alpha - x_\beta)$ is at least 2^{n-m-2}, or two segments long, and at most $2^n - 2^{n-m-2}$, or "negative two" segments long. Since $x_\alpha y_\alpha$ and $x_\beta y_\beta$ differ by less than one segment (2^{n-m-3}), the translates $x_{\alpha+y_k} y_{\alpha+y_k}$ and $x_{\beta+y_k} y_{\beta+y_k}$ differ by more than 2^{n-m-3} and must fall into different segments. The rest of the proof follows exactly as before. In order to avoid overly cumbersome notation, let us abuse it slightly by calling the products $x_\alpha y_\alpha$ and $x_\beta y_\beta$, even though they should possibly be called $x_{\alpha+z} y_{\alpha+z}$ and $x_{\beta+z} y_{\beta+z}$.

Now that we know the products differ in a high-order bit, it remains to ensure that they are not 1-bad. Assume they are. Let d be the greatest index less than n of a bit position in which $x_\alpha y_\alpha$ and $x_\beta y_\beta$ differ.

First, we claim that if the products are 1-bad, then in fact $d \geq n-m-2$. Because if, say $d = n-m-3$, then the products look like[3] $\begin{array}{rcl} x_\alpha y_\alpha & = & \cdots 1\,0 \cdots \\ x_\beta y_\beta & = & \cdots \underset{d}{0}\,1\,1\,\underset{\substack{\uparrow \\ n-m-6}}{1}\,1 \cdots \end{array}$ and therefore they differ modulo 2^n by at most $2^{n-m-7} + (2^{n-m-4} - 1)$ (since they agree in bits $d+1$ through $n-1$), but we know they differ by at least 2^{n-m-3}. Furthermore, by the same reasoning, not only is $d \geq n-m-2$, but $x_\alpha y_\alpha$ must have a 1 in some position between $d-2$ and $n-m-4$ inclusive (note that $(x_\alpha y_\alpha)_{d-1} = 0$; else the products are not 1-bad). For otherwise, the products look like $\begin{array}{rcl} x_\alpha y_\alpha & = & \cdots \underset{d}{1}\,0\,0\,0\,0\,0 \cdots \\ x_\beta y_\beta & = & \cdots \underset{d}{0}\,1\,1\,1\,1\,1\,1\,1\,\underset{\substack{\uparrow \\ n-m-6}}{1}\,1 \cdots \end{array}$ and thus they differ modulo 2^n by at most $2^{n-m-7} + 2^{n-m-4} - 1$, a contradiction.

So we are reduced to the case that the products are 1-bad, differ in position $d \geq n-m-2$, and $x_\alpha y_\alpha$ has a 1 in some position between $d-2$ and $n-m-4$. Let ℓ be the highest index of a 1 in this range: $x_\alpha y_\alpha = \cdots \underset{d}{1}\,0\,0\,0\,\underset{\ell}{1} \cdots$. We will find a pair of variables (x_u, y_v) with $u+v = n-m-6$ so that the cross terms $2^v x_\alpha, 2^v x_\beta, 2^u y_\alpha, 2^u y_\beta$ all have 0's in positions $n-m-8$ through $n-1$. Then $(2^{u+v} + 2^v x_\alpha + 2^u y_\alpha)$ and $(2^{u+v} + 2^v x_\beta + 2^u y_\beta)$ both look like $\cdots \underset{\substack{\uparrow \\ n-1}}{0}\,0\,0\,0\,0\,0\,0\,0\,\underset{\substack{\uparrow \\ n-m-6}}{1}\,0 \cdots$. We see that $x_{\alpha'} y_{\alpha'}$ looks like either $\cdots \underset{d}{1}\,0\,0\,0\,\underset{\ell}{1} \cdots$ if there is no carry into position ℓ when $2^{u+v} + 2^v x_\alpha + 2^u y_\alpha$

[3] Without loss of generality, let us assume that in position d, $x_\alpha y_\alpha$ has a 1 and $x_\beta y_\beta$ has a 0.

is added to $x_\alpha y_\alpha$, or $\cdots \underset{d}{1}\,0\,0\,1\,\underset{\ell}{0}\cdots$ if there is a carry into position ℓ. Meanwhile,

$$\begin{array}{rl} & \cdots \underset{d}{0}\,1\,1\,1\,1\,1\,1\,1\,1\cdots \\ x_{\beta'}y_{\beta'} = & \underline{+\ \ \cdots 0\,0\,0\,0\,0\,0\,0\,1\,0\ \cdots} \end{array} \text{ looks like } \cdots \underset{d}{1}\,0\,0\,0\,0\,0\,0\,0\,\underset{\underset{n-m-6}{\uparrow}}{0}\cdots \text{ or } \cdots \underset{d}{1}\,0\,0\,0\,0\,0\,0\,0\,\underset{\underset{n-m-6}{\uparrow}}{1}\cdots$$

depending on whether there is a carry into position $n-m-6$ in this addition.

Since $x_{\beta'}y_{\beta'}$ has 0's in positions $\ell \le d-2$ and $\ell - 1 \ge n-m-5$, we see that $x_{\alpha'}y_{\alpha'}$ and $x_{\beta'}y_{\beta'}$ look like

$$\begin{array}{rcl} x_{\alpha'}y_{\alpha'} & = & \cdots \underset{d}{1}\,0\,0\,0\,0\,\underset{d'}{1}\cdots \\ x_{\beta'}y_{\beta'} & = & \cdots \underset{d}{1}\,0\,0\,0\,0\,\underset{d'}{0}\,0\cdots \end{array}$$

where d' is either ℓ or $\ell+1$. Furthermore, the products agree in all higher bits up to $n-1$ because by the definition of d, $x_\alpha y_\alpha$ and $x_\beta y_\beta$ agree in bits $d+1$ through $n-1$ and we chose x_u and y_v so that the cross terms have 0's in these positions. Since $\ell \ge n-m-4$, it follows that $x_{\alpha'}y_{\alpha'}$ and $x_{\beta'}y_{\beta'}$ differ in a high-order bit and are not even 1-bad.

A counting argument like that for Lemma 3 shows that we may choose x_u and y_v as needed. We require the cross terms to have 0's in at most $m+8$ positions. Since at most $m+1$ bits are set to 1 in x_α or x_β, the total number of values v that we may not choose is $(m+1)(m+8)+(m+1)$. The same number of values u are ruled out, making a total of at most $2(m+1)(m+9) = 2\frac{n}{9} + O(\sqrt{n})$ pairs (x_u, y_v) that are ruled out. Since there are $n-m-5$ pairs to choose from initially, we retain $\Omega(n)$ pairs. ■

Lemma 5 *Let $T \subset X \cup Y$, and α and β be two settings to T. Let d be the greatest index in $[0, n-2]$ such that $(x_\alpha y_\alpha)_d \ne (x_\beta y_\beta)_d$. Suppose $d \ge n-m-4$ and $\max(|T \cap X|, |T \cap Y|) = t \le 2m+5$ and also that $x_\alpha y_\alpha$ and $x_\beta y_\beta$ are not k-bad, for some $k \le m+4$. Then there are two variables, $x_u, y_v \in \overline{T}$, such that*

$$(x_{\alpha'}y_{\alpha'})_{d+1} \ne (x_{\beta'}y_{\beta'})_{d+1}$$

for $\alpha' = \alpha + x_u + y_v$ and $\beta' = \beta + x_u + y_v$, and moreover, $x_{\alpha'}y_{\alpha'}$ and $x_{\beta'}y_{\beta'}$ are not $(k+1)$-bad.

Proof: We have four possible cases (up to switching α and β):

$$\begin{array}{rllll} x_\alpha y_\alpha = & \textbf{(1):}\ \cdots \underset{d}{1}\,0\cdots & \textbf{(2):}\ \cdots \underset{d}{1}\,1\cdots & \textbf{(3):}\ \cdots \underset{d}{1}\,1\cdots & \text{or } \textbf{(4):}\ \cdots \underset{d}{1}\,0\cdots \\ x_\beta y_\beta = & \quad\ \ \cdots \underset{d}{0}\,0\cdots & \quad\ \ \cdots \underset{d}{0}\,0\cdots & \quad\ \ \cdots \underset{d}{0}\,1\cdots & \quad\quad\ \ \cdots \underset{d}{0}\,1\,1\,1\,1\,1\,0\cdots \end{array}$$

By assumption, $d \geq n - m - 4$.

Case 1:
$$\begin{aligned} x_\alpha y_\alpha &= \cdots \underset{d}{1}\, 0 \cdots \\ x_\beta y_\beta &= \cdots \underset{d}{0}\, 0 \cdots \end{aligned}$$
It is sufficient to choose (x_u, y_v) so that $u + v = d$ and each of the cross terms $2^v x_\beta$, $2^u y_\beta$, $2^v x_\alpha$, and $2^u y_\alpha$ has 0's in positions $d - 3$ through $d + 1$. Then the sums $2^v x_\beta + 2^u y_\beta$ and $2^v x_\alpha + 2^u y_\alpha$ have 0's in positions $d - 2$ through $d + 1$. Adding these to $x_\alpha y_\alpha$ and $x_\beta y_\beta$ respectively therefore causes no carry into position d and thus the addition of $2^{u+v} = 2^d$ causes a carry into bit $d+1$ for α but not for β. Since $x_\alpha y_\alpha$ and $x_\beta y_\beta$ agree in bits $d+1$ through $n - 1$, this carry bit causes them to differ in bit $d+1$ and possibly higher bits as well.

We now verify that $x_{\alpha'} y_{\alpha'}$ and $x_{\beta'} y_{\beta'}$ are not 1-bad. We know that $2^{u+v} + 2^v x_\beta + 2^u y_\beta$ looks like $\cdots 0\,\underset{d}{1}\,0\,0 \cdots$. Thus $x_{\beta'} y_{\beta'} = \begin{array}{r} \cdots \underset{d}{0}\,0 \cdots \\ + \quad \cdots 0\,1\,0\,0 \cdots \\ \hline \end{array}$ looks like either $\cdots \underset{d}{1}\,0 \cdots$ or $\cdots \underset{d}{1}\,1\,0 \cdots$, depending on whether there is a carry into position $d-1$. Thus $x_{\beta'} y_{\beta'}$ does not have a string of 1's extending past position $d - 1 \geq n - m - 5$ and cannot make the products even 1-bad. Since the products differ in position $d+1$ or higher and $x_{\alpha'} y_{\alpha'}$ has a 0 in position d, the products cannot be 1-bad due to a string of 1's in $x_{\alpha'} y_{\alpha'}$.

To see that we can choose (x_u, y_v) as desired, we argue as in the proof of Lemma 3. The number of positions required to be 0 is 5, ruling out $5t$ values of v. Of the $d + 1 = n - O(\sqrt{n})$ pairs (x_u, y_v) such that $u + v = d$, the number of pairs ruled out is at most $2(5t + t) = 12t \leq 12(2m + 5) = O(\sqrt{n})$, so there are $\Omega(n)$ remaining pairs to choose from.

Cases 2:
$$\begin{aligned} x_\alpha y_\alpha &= \cdots \underset{d}{1}\, 1 \cdots \\ x_\beta y_\beta &= \cdots \underset{d}{0}\, 0 \cdots \end{aligned}$$
and 3:
$$\begin{aligned} x_\alpha y_\alpha &= \cdots \underset{d}{1}\, 1 \cdots \\ x_\beta y_\beta &= \cdots \underset{d}{0}\, 1 \cdots \end{aligned}$$
It is sufficient to choose (x_u, y_v) as in Case 1 except that $u + v = d - 1$. Adding 2^{d-1} will cause a carry to propagate into position $d+1$ for α but not for β, causing them to differ in bit $d+1$ and possibly higher bits as well. The counting argument for choosing

(x_u, y_v) is exactly the same as in Case 1 except that there is one fewer pair (x_u, y_v) with $u + v = d - 1$.

It only remains to show that in fact $x_{\alpha'}y_{\alpha'}$ and $x_{\beta'}y_{\beta'}$ are not 1-bad. Now $2^{u+v} + 2^v x_\alpha + 2^u y_\alpha$ looks like $\cdots 0\underset{d}{0}10\cdots$ and so does $2^{u+v} + 2^v x_\beta + 2^u y_\beta$. Thus $x_{\alpha'}y_{\alpha'} =$

$$\begin{array}{r} \cdots\underset{d}{1}1\cdots \\ +\ \ \cdots 0010\cdots \\ \hline \end{array}$$

, and we see that it has a 0 in bit d.

Looking now at $x_{\beta'}y_{\beta'}$, we see that in Case 2, $x_{\beta'}y_{\beta'} =$

$$\begin{array}{r} \cdots\underset{d}{0}0\cdots \\ +\ \ \cdots 0010\cdots \\ \hline \end{array}$$

looks like either $\cdots\underset{d}{0}1\cdots$ or $\cdots\underset{d}{1}0\cdots$, depending on whether there is a carry into position $d - 1$. In Case 3, $x_{\beta'}y_{\beta'} =$

$$\begin{array}{r} \cdots\underset{d}{0}1\cdots \\ +\ \ \cdots 0010\cdots \\ \hline \end{array}$$

looks like either $\cdots\underset{d}{1}0\cdots$ or $\cdots\underset{d}{1}10\cdots$, depending on whether there is a carry into position $d - 1$. In any case, $x_{\beta'}y_{\beta'}$ does not have a string of 1's extending past $d - 2 \geq n - m - 6$, and so $x_{\alpha'}y_{\alpha'}$ and $x_{\beta'}y_{\beta'}$ are not even 1-bad.

Case 4:

$$\begin{aligned} x_\alpha y_\alpha &= \cdots \underset{d}{1}0 \cdots \\ x_\beta y_\beta &= \cdots \underset{d}{0}111\underset{\substack{\uparrow \\ n-m-6}}{1}\overbrace{1111}^{k-1}0\cdots \end{aligned}$$

Without loss of generality, let us say that $x_\beta y_\beta$ contains the maximum number, $k-1$, of consecutive 1's extending past position $n - m - 6$. We choose (x_u, y_v) so that $u + v = d$ and the cross terms $2^v x_\alpha$, $2^u y_\alpha$, $2^v x_\beta$ and $2^u y_\beta$ have 0's in positions $(n - m - 6) - k - 2$ through $n - 1$. This will ensure that from 2^d we get a carry into position $d + 1$ for α' but not for β', causing the products to differ in bit $d + 1$ and possibly higher bits as well.

The sum $2^v x_\beta + 2^u y_\beta$ has 0's in positions $(n - m - 6) - k - 1$ through $n - 1$, so $x_{\beta'}y_{\beta'} =$

$$\begin{array}{r} \cdots\underset{d}{0}1111\overbrace{1111}^{k-1}0\cdots \\ +\ \ \cdots 010000000000\cdots \\ \hline \end{array}$$

looks like either $\cdots \underset{d}{1}1111\overbrace{1111}^{k-1}0\cdots$ or

$\cdots \underset{d}{1}\,1\,1\,1\,1\,\overbrace{1\,1\,1\,1}^{k-1}\,1\,0 \cdots$, depending on whether there is a carry into position $(n - m - 6) - k$. So $x_{\beta'} y_{\beta'}$ has at most k 1's extending past position $n - m - 6$. The pair of products cannot be worse than k-bad because of a longer string of 1's in $x_{\alpha'} y_{\alpha'}$ because the products differ in position $d + 1$ or higher and $x_{\alpha'} y_{\alpha'}$ has a 0 in position d. Thus $x_{\alpha'} y_{\alpha'}$ and $x_{\beta'} y_{\beta'}$ are at worst k-bad.

The number of positions in which we require $2^v x_\alpha$ or $2^u x_\beta$ to be 0 is $m + 6 + k + 2 \leq 2m + 12$. Together, x_α and x_β may rule out $t(2m + 12)$ values v in addition to the t variables y_v already in T. Taking into account the same number of values u ruled out by y_α and y_β, there are at most $2(t(2m + 12) + t))$ pairs (x_u, y_v) that could be ruled out. Of the $d + 1 = n - O(\sqrt{n})$ possible pairs (x_u, y_v) with $u + v = d$, a total of at most

$$2(2m + 5)(2m + 13) = 8\frac{n}{9} + O(\sqrt{n})$$

pairs are ruled out, leaving $\frac{n}{9} - O(\sqrt{n}) = \Omega(n)$ pairs to choose from. For $n \geq 56,000$, we can say there is at least one pair left. ■

For preciseness, we have given explicit values of n above which our proofs hold; these numbers are most likely a reflection of our proofs rather than the true complexity, and should not be taken very seriously.

3.4 Problem reductions

We may deduce similar lower bounds for other boolean functions by the standard technique of problem reduction. In order to preserve read-once complexity, we will consider a very restrictive type of problem reduction. We begin with the notion of *projection reductions* [SV81], as defined in [CSV84]:

Definition 5 *A function $f = \{f_n\}_{n\in\mathbb{N}}$ is projection reducible to a function $g = \{g_n\}_{n\in\mathbb{N}}$, written $f \leq_{\text{proj}} g$, if there is a mapping*

$$\sigma : \{y_1, \ldots, y_{p(n)}\} \rightarrow \{0, 1, x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}\}$$

such that

$$f_n(x_1, \ldots, x_n) = g_{p(n)}(\sigma(y_1), \ldots, \sigma(y_{p(n)}))$$

for some function $p(n)$ bounded above by a polynomial in n.

In other words, $f \leq_{\text{proj}} g$ if one can use as a black box an algorithm (circuit, branching program) for $g(y_1, \ldots, y_{p(n)})$ simply by substituting the inputs to f for the inputs to g and then taking the output of the algorithm as the output for f. These reductions were used by Chandra, Stockmeyer, and Vishkin [CSV84] in their study of constant-depth reducibility—clearly, given that $f \leq_{\text{proj}} g$, if $g \in \mathrm{AC}^0$ then $f \in \mathrm{AC}^0$.

We would like a reduction $\leq'$ that allows us to deduce that if $f \leq' g$ and $g \in$ READ-1 then $f \in$ READ-1. It is easy to see that projection reductions satisfy this condition if the mapping σ is injective with respect to the x variables:

Definition 6 *A function f is read-once reducible to a function g, denoted $f \leq_{\text{r-o}} g$, if there is a projection reduction σ from f to g in which for $i \neq j$,*

$$\sigma(y_i) \neq \sigma(y_j) \quad \text{and} \quad \sigma(y_i) \neq \overline{\sigma(y_j)}.$$

It follows that a read-once branching program for $f(x_1, \ldots, x_n)$ is obtained by relabelling the nodes of a read-once program for $g(y_1, \ldots, y_n)$.

3.4.1 Reductions to other arithmetic functions

Projection reductions have also been used to deduce tight lower bounds on the depth of polynomial-size threshold circuits. It was originally proved in [HMPST93] that **Inner-Product-Modulo**-2 cannot be computed in polynomial-size by threshold circuits of depth 2. It was also noted there that the projection reduction to multiplication (first given in [FSS84], from **Parity** to **Mult**) shows that **Mult** obeys the same lower bound.

Wegener [We93] gives projection reductions from **Mult** to squaring and inversion in order to show that these functions also require depth 3 polynomial-size threshold circuits. The lower bound for the middle bit of multiplication implies a lower bound for the appropriate bit of these two functions. We phrase the reductions in [We93] in terms of the following Boolean functions:

- **Squaring** : $\{0,1\}^n \to \{0,1\}$; computes "the" middle bit (here, bit n rather than bit $n-1$ which we chose for **Mult**) in the square of an n-bit integer:
$$\textbf{Squaring}(z) = (z^2)_n.$$
- **Inversion** : $\{0,1\}^n \to \{0,1\}$; computes the ones' bit in the reciprocal of an n-bit number between 0 and 1:
$$\textbf{Inversion}(x) = y_0$$
where x represents the number $0\,.\,x_1x_2\cdots x_n = \sum_i x_i 2^{-i}$ and $y = y_n \cdots y_0$ is the integral part of $1/x$. (Note that $1 < y \leq 2^n$.) Define the function to be 0 if all x_i are 0.

Wegener actually shows that

$$\textbf{Mult} \leq_{\text{proj}} \textbf{Squaring} \leq_{\text{proj}} \textbf{Inversion},$$

except that the reductions are given for all bits of multiplication, squaring, and inversion. Though it is not noted there, we shall see that each reduction is actually a read-once reduction. The polynomial $p(n)$ of the reduction is linear in both cases, implying that if each bit of the function is computable with a read-once program of size $f(n)$, then **Mult** is computable with a read-once program of size $f(cn)$ for some constant c. This gives the following corollaries to Theorem 5:

Corollary 1 *Any read-once branching program for computing the function* **SQUARING** *has size at least* $2^{\Omega(\sqrt{n})}$.

Proof: We verify that the reduction in [We93] shows **MULT** $\leq_{\text{r-o}}$ **SQUARING** with a polynomial $p(n) = 3n + 2$. In addition to verifying $p(n)$, we must also check that the reduction is indeed between these two Boolean functions and also that the mapping σ is injective. The reduction simply maps the n-bit inputs x, y (of **MULT**) to the $(3n+2)$-bit input $z = x2^{2(n+1)} + y$ (of **SQUARING**), so that $z^2 = x^2 2^{4(n+1)} + xy2^{2(n+1)+1} + y^2$. The middle bit of the product xy is found in the middle bit of z^2: $(xy)_{n-1} = (z^2)_{3n+2}$. Thus $p(n) = 3n + 2$. It is clear that the mapping σ is injective since

$$\sigma(z_i) = \begin{cases} y_i & \text{if } 0 \leq i < n; \\ 0 & \text{if } n \leq i < 2(n+1); \\ x_{i-2(n+1)} & \text{if } 2(n+1) \leq i < 2(n+1) + n. \end{cases}$$

■

Corollary 2 *Any read-once branching program for computing the function* **INVERSION** *has size at least* $2^{\Omega(\sqrt{n})}$.

Proof: We verify that the reduction in [We93] shows **SQUARING** $\leq_{\text{r-o}}$ **INVERSION** with polynomial $p(n) = 17n + 1$.

The reduction **SQUARING**$\leq_{\text{proj}}$**INVERSION** reduces the problem of computing the square of an n-bit integer m to the problem of computing $1/(1-x) = 1+x+x^2+x^3+\cdots$ where

$$1 - x \quad = \quad 1 - m\,2^{-4n} - 2^{-10n},$$

which is a $10n$-bit number slightly less than 1. The proof in [We93] shows that the product m^2 lies in bit positions $-6n-1$ through $-8n$ in $1/(1-x)$, its middle bit being in position $-7n$. By instead computing the inverse of $2^{-7n}(1-x)$, a $17n$-bit number, we find the middle bit of m^2 in position 0.

For example, working in decimal, we may compute 5^2 (so $n = 1$) by letting $1 - x = 1 - 5 \cdot 10^{-4} - 10^{-10}$ and calculating

$$\left(1 - 5 \cdot 10^{-4} - 10^{-10}\right)^{-1} \quad = \quad 1.000500250225\cdots$$

from which we may recover $5^2 = 25$ in positions -7 and -8. By instead calculating $(10^{-7} \cdot (1 - 5 \cdot 10^{-4} - 10^{-10}))^{-1}$, we may find the middle digit, 2, of 25 in position 0.

To see that the mapping σ is injective, simply notice that $1 - x = 1 - m\,2^{-4n} - 2^{-10n}$ has 1's in all positions -1 through $-10n$, except in positions $-3n-1$ through $-4n$ where it has exactly the complements of the bits of m. The number $2^{-7n}(1-x)$ is similar, with extra 0's on the left. ∎

CHAPTER 4

Discussion and further work

In this thesis, we have proved that integer multiplication requires exponential-size read-once branching programs. This fact is important for the hardware verification community, which would like to find a simple model in which multiplication can be computed with polynomial size. It was known already that most oblivious branching programs, which are good candidates because of the ease with which they are manipulated, require exponential size to compute multiplication.

In the course of understanding the relevant lower bounds and related models, we have also assembled a survey of the structure of these low-level complexity classes, and also of the main ideas that have been brought to bear in thinking about their computation. This survey also includes a few simple proofs that have not yet appeared in the literature.

Further work

There are many open questions surrounding the topics of this thesis, some of which have already been mentioned. We will describe some of these problems that we consider to be the most important, interesting, or tractable. The oldest of these problems, open since [FHS78], is

Open Question 1 *Is there a deterministic polynomial-time algorithm for determining the equivalence of two read-once programs?*

The answer to this question loses some practical significance in light of the lower bound for multiplication and the intractability of the synthesis operations, which make read-once programs less attractive as an alternative to OBDD's in hardware verification.

Multiplication

Although perhaps not the most interesting question, there is the possibility of improving the lower bound for multiplication. We doubt that $2^{\Theta(\sqrt{n})}$ is the true read-once complexity of **Mult** (recall that Bryant's lower bound for OBDD's is $2^{n/8}$), but the simple counting technique used in our proof seems limited to this lower bound. It is curious that many of the lower bounds for read-once programs achieve only $2^{\Omega(\sqrt{n})}$ if n is the number of input bits—only the lower bound of [BHST87] achieves a fully exponential lower bound of $2^{\Omega(n)}$. This limitation is most likely an artifact of the proofs, but it is not well understood.

In addition to improving the bound, it may also be possible to extend the argument to show that a similar bound holds for nondeterministic read-once programs or for read-k-times programs.

Open Question 2 *Does* **Mult** *require superpolynomial* <u>*nondeterministic*</u> *read-once programs? ... superpolynomial read-k-times programs?*

For nondeterministic read-once programs, we may define frontier edges as before. Now, however, it is not necessary for the inputs reaching an edge to induce the same subfunction on the remaining input variables, since inputs may follow several different paths. We can say, however, that the inputs in $\textbf{Mult}^{-1}(1)$ that pass through a frontier edge are described by a function $f_1(X_1, Y_1) \wedge f_2(X_2, Y_2)$ where $X_1 \cup Y_1$ is in the boundary of the filter $\mathcal{F}$ and $X_2 \cup Y_2 = (X \cup Y) \setminus (X_1 \cup Y_1)$. Thus **Mult** can be written as the conjunction, over all frontier edges, of such functions. We would like to show that since each of these functions must reject all of $\textbf{Mult}^{-1}(0)$, it can accept only an exponentially small fraction of $\textbf{Mult}^{-1}(1)$.

That is, we would like to show that given $\textbf{Mult}(uv) = 1$ for all $u \in f_1^{-1}(1)$ and $v \in f_2^{-1}(1)$, it must be that $|f_1^{-1}(1) \times f_2^{-1}(1)| \le 2^{-n^k} \cdot |\textbf{Mult}^{-1}(1)|$ for some $k > 0$. (Here, u is a setting to $X_1 \cup Y_1$ and v is a setting to $X_2 \cup Y_2$.) For comparison, the proof of our lower bound (Theorem 5) in effect shows that given $\textbf{Mult}(uv) = \textbf{Mult}(u'v)$ for all $u, u' \in f_1^{-1}(1)$ and for *all* inputs v, it must be that $|f_1^{-1}(1)| \cdot 2^{|v|}$ is a fraction $2^{-\Omega(\sqrt{n})}$ of the total number of inputs, 2^{2n}.

Finally, we mention that there seem to be no nontrivial upper bounds for **Mult** in either nondeterministic or randomized read-k-times models, for $k = o(n)$. Of course, in all other models considered in this thesis—OBDD's, k-OBDD's, k-IBDD's, indeed any linear-length oblivious programs, even nondeterministic, as well as non-oblivious read-once programs—it is known that exponential size is required.

The read-k-times hierarchy

As mentioned in Section 2.4.1, it is not known whether the read-k-times hierarchy is strict:

Open Question 3 *For some $k > 2$, is there a function computable by polynomial-size read-k-times programs but not computable by polynomial-size read-$(k-1)$-times programs?*

In [SS93], it is conjectured that such a function is the problem of determining whether a k-dimensional hypergraph on n nodes is r-regular for, say, $r = n/2$. (Recall that [SS93] proves that this problem on ordinary graphs ($k = 2$), while easily computed by read-2-times programs, requires read-once programs of size $2^{\Omega(n)}$.) The function π-**Matrix** may be regarded as a special case of this problem: it is the case of determining whether a bipartite $n \times n$ graph is 1-regular. We believe that higher dimensional versions of this latter problem should separate the read-k-times hierarchy.

For example, consider the 3-dimensional version, "π-**Cube**", defined on an $n \times n \times n$ cube of boolean variables, which has the value 1 exactly when each of the n planes in each of the 3 dimensions contains exactly one 1. π-**Cube** is easily computed with read-3-times programs. Here is a possible strategy for showing it is not computable with polynomial-size read-2-times programs. According to Theorem 1 in [BRS93], a read-2-times program for π-**CUBE** enables us to express the function as

$$\pi\text{-}\mathbf{CUBE} \quad = \quad \bigvee_{i=1}^{poly(BPsize)} f_{i1}(X_{i1}) \wedge f_{i2}(X_{i2}) \wedge f_{i3}(X_{i3}) \wedge f_{i4}(X_{i4})$$

where each X_{ij} is a subset of half the n^3 variables and each variable appears in at most two of $X_{i1}, X_{i2}, X_{i3}, X_{i4}$ for each i. We would like to show that a function of the form $f_{i1}(X_{i1}) \wedge f_{i2}(X_{i2}) \wedge f_{i3}(X_{i3}) \wedge f_{i4}(X_{i4})$, which rejects all of π-$\mathbf{CUBE}^{-1}(0)$, can accept only an exponentially small fraction of π-$\mathbf{CUBE}^{-1}(1)$.

Since each variable is in two of the X_i, one of the three partitions $X_{i1} \cup X_{i2} \mid X_{i3} \cup X_{i4}$, $X_{i1} \cup X_{i3} \mid X_{i2} \cup X_{i4}$ and $X_{i1} \cup X_{i4} \mid X_{i2} \cup X_{i3}$ contains that variable on only one side of the partition ("fails to split" that variable). It follows that one of these partitions fails to split at least $1/3$ of the variables. From this, we may argue further that for one of these partitions, there are at least $1/6$ of the variables, S, that appear only on one side of the partition and at least $1/6$ of the variables, T, that appear only on the other side. Thus, we may write (if the best partition is $X_1 \cup X_2 \mid X_3 \cup X_4$)

$$\begin{aligned} f_{i1}(X_{i1}) \wedge f_{i2}(X_{i2}) \wedge f_{i3}(X_{i3}) \wedge f_{i4}(X_{i4}) &= f_i'(X_{i1} \cup X_{i2}) \wedge f_i''(X_{i3} \cup X_{i4}) \\ &= f_i'(X \setminus S) \wedge f_i''(X \setminus T). \end{aligned}$$

Since S and T each has more than $1/8$ of all the variables, there must be many coplanar pairs $(s,t) \in S \times T$. This function cannot accept two inputs x and y that have $s = 1$ and $t = 1$ respectively if x and y agree on the variables $\overline{S \cup T}$, since then it would also accept the input (which should be rejected) that looks like x on S and like y on T. Furthermore, the fraction of inputs in π-$\mathbf{CUBE}^{-1}(1)$ that have all 0's in a given $\frac{n}{c} \times \frac{n}{c} \times \frac{n}{c}$ subcube is exponentially small in n, for c constant. It should be possible to combine these facts to obtain the desired lower bound.

Read-once reductions

Read-once reductions appear to be rather limited in their utility. It is not clear, for example, how to use them even to show that directed s,t-connectivity does not have polynomial-size read-once programs. (This function, being NL-complete, is not known to have polynomial-size branching programs at all, regardless of restrictions on reading variables.) We may construct a branching program of size $O(n^3)$ for **MULT** in which there is a s,t-path if and only if **MULT** is 1, but since many edges are labelled with the same **MULT** variable, a computation that reads each edge variable once in fact reads the variables of **MULT** many times. In other words, this is a projection reduction in which the variable mapping does not have the necessary injectivity property.

The Fourier spectrum

It is an interesting question whether there is any correlation between the Fourier spectrum of a function and the size of its OBDD's.

Open Question 4 *Is there a nice correlation between some property of a functions Fourier spectrum and the size of its* OBDD*'s?*

In particular, it would be useful to know which coefficients are the largest, as this is the information that is used in the remarkable algorithms for learning functions with shallow decision trees or small constant-depth circuits. As explained in Section 2.7.2, the correlations found between such functions and the properties of their spectrums do not hold for OBDD's.

The ordering problem for OBDD's

One of the most useful research directions, as far as the hardware verification community is concerned, is further analysis of the variable ordering problem described in Section 2.7.1. Now that it is known to be NP-complete, approximation algorithms—or results demonstrating the hardness of approximability—are of most interest.

Open Question 5 *Is there a reasonable algorithm (in P, RP, or BPP) which, given an* OBDD*, finds another* OBDD *(possibly obeying a different ordering of the variables) with size that is within a bounded factor of optimal?*

Randomized algorithms for this problem should also be considered.

Bibliography

[AM88] N. Alon and W. Maass. Meanders and their applications in lower bounds arguments. *Journal of Computer and System Sciences,* 37, 1988, pp. 118–129.

[AGD91] P. Ashar, A. Ghosh, and S. Devadas. Boolean satisfiability and equivalence checking using general binary decision diagrams. *Proc. Int'l. Conference on Computer Design*, 1991, pp. 259–264.

[BHST87] L. Babai, A. Hajnal, E. Szemeredi, and G. Turan. A lower bound for read-once branching programs. *Journal of Computer and System Sciences*, No. 37 (1988), pp. 153–162.

[BNS92] L. Babai, N. Nisan, and M. Szegedy. Multiparty protocols, pseudorandom generators for logspace, and time-space tradeoffs. *Journal of Computer and System Sciences*, **45** (1992), pp. 204–232.

[BCW80] M. Blum, A. Chandra, and M. Wegman. Equivalence of free boolean graphs can be decided probabilistically in polynomial time. *Information Processing Letters,* Vol. 10, No. 2, March 1980, pp. 80–82.

[BSSW93] B. Bollig, M. Sauerhoff, D. Sieling, and I. Wegener. Read-k-times ordered binary decision diagrams—efficient algorithms in the presence of null chains. Technical Report 474, Univ. Dortmund, 1993.

[BSSW95] B. Bollig, M. Sauerhoff, D. Sieling, and I. Wegener. On the power of different types of restricted branching programs. Submitted to *Theoretical Computer Science.* Also published in the *Electronic Colloquium on Computational Complexity*, Report No. TR94-026 (1994), available via `http://www.eccc.uni-trier.de/eccc/`.

[BW95] B. Bollig and I. Wegener. Improving the variable ordering of OBDD's is NP-complete. *Dagstuhl Seminar on Computer-Aided Design and Testing*, February 1995. Available via `http://www.informatik.uni-trier.de:80/Design_and_Test/`, FB Informatik, LS II, Univ. Dortmund, 44221 Dortmund, Germany.

[BRS93] A. Borodin, A. Razborov, and R. Smolensky. On lower bounds for read-k-times branching programs. *Computational Complexity,* 3, 1993, pp. 1–18.

[BHR95] Y. Breitbart, H. B. Hunt III and D. Rosenkrantz. On the size of binary decision diagrams representing Boolean functions. *Theoretical Computer Science,* 145 (1995), pp. 45–69.

[BF85] F. Brglez and H. Fujiwara. A neutral netlist of 10 combinational circuits. *Proc. 1985 IEEE Int'l. Symposium on Circuits and Systems.*

[BS90] J. Bruck and R. Smolensky. Polynomial threshold functions, AC^0 functions and spectral norms. *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science,* 1990, pp. 632–641.

[Br91] R. Bryant. On the complexity of VLSI implementations and graph representations of boolean functions with applications to integer multiplication. *IEEE Transactions on Computers,* Vol. 40, No. 2, February 1991. pp. 205–213.

[Br92] R. Bryant. Symbolic boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys,* Vol. 24, No. 3, September 1992. pp. 293–318.

[BC94] Randal Bryant and Yirng-An Chen. Verification of arithmetic functions with binary moment diagrams. Technical Report CMU-CS-94-160, Carnegie Mellon University, May 31, 1994.

[Bu92] S. Buss. The graph of multiplication is equivalent to counting. *Information Processing Letters,* 41 (18 March 1992), pp. 199–201.

[CSV84] A. Chandra, L. Stockmeyer, and U. Vishkin. Constant depth reducibility. *SIAM Journal of Computing,* 13 (1984), pp. 423–439.

[CS88] V. Chvátal and E. Szemerédi. Many hard examples for resolution. *Journal of the ACM*, Vol. 35, No. 4 (October 1988), pp. 759–768.

[Du85] P. E. Dunne. Lower bounds on the complexity of 1-time only branching programs. *Proceedings of the FCT,* Springer-Verlag LNCS No. 199 (1985), pp. 90–99.

[DGS84] P. Duris, Z. Galil, and G. Schnitger. Lower bounds of communication complexity. *Proceedings of the 16th ACM Symposium on Theory of Computing,* (1984), pp. 81–91.

[FHS78] S. Fortune, J. Hopcroft, and E. M. Schmidt. The complexity of equivalence and containment for free single variable program schemes. Springer-Verlag LNCS No. 62, 1978, pp. 227–240.

[FSS84] M. Furst, J. B. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory,* 17 (1984), pp. 13–27.

[GJ79] M. Garey and D. Johnson. *Computers and Intractability.* W. H. Freeman and Company, New York. 1979.

[Ge94] J. Gergov. Time-space tradeoffs for integer multiplication on various types of input-oblivious sequential machines. *Information Processing Letters,* 51 (1994), pp. 265-269.

[GM94] J. Gergov and C. Meinel. Efficient boolean manipulations with OBDD's can be extended to FBDD's. *IEEE Transactions on Computers,* Vol. 43, No. 10, (October 1994), pp. 1197–1209.

[HMPST93] A. Hajnal, W. Maass, P. Pudlák, M. Szegedy, and G. Turán. Threshold circuits of bounded depth. *Journal of Computer and System Sciences*, 46(2) 1993, pp. 129–154.

[HR88] B. Halstenberg and R. Reischuk. On different modes of communication. *Proceedings of the 20th ACM Symposium on Theory of Computing* (1988), pp. 162–172.

[Im88] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal of Computing,* Vol. 17, No. 5 (October 1988), pp. 935–938.

[JABFA92] J. Jain, M. Abadir, J. Bitner, D. Fussell, and J. Abraham. IBDD's: An efficient functional representation for digital circuits. European Design Automation Conference (1992), pp. 440–446.

[Ju86] S. Jukna. Lower bounds on the complexity of local circuits. *Proceedings of the MFCS*, Springer-Verlag LNCS 233 (1986), pp. 440–448.

[Ju88] S. Jukna. Entropy of contact circuits and lower bounds on their complexity. *Theoretical Computer Science*, 47:2 (1988), pp. 113–129.

[Ju89] S. Jukna. The effect of null-chains on the complexity of contact schemes. *Proceedings of the FCT* (1989), Springer-Verlag LNCS No. 380, pp. 246–256.

[Ju92] S. Jukna. A note on read-k-times branching programs. Technical report 448, Universität Dortmund, 1992. *RAIRO Theoretical Computer Science*, to appear. Also published in the *Electronic Colloquium on Computational Complexity*, Report No. TR94-027 (1994), available via `http://www.eccc.uni-trier.de/eccc/`.

[Ju94] S. Jukna. The graph of multiplication is hard for read-k-times networks. Manuscript, April 1994.

[KW93] M. Karchmer and A. Wigderson. On span programs. *Proceedings of the 8th Structure in Complexity Theory*, (1993), pp. 102–111.

[Kr88] M. Krause. Exponential lower bounds on the complexity of real time and local branching programs. *Journal of Information Processing and Cybernetics (EIK)*, 24:3 (1988), pp. 99–110.

[Kr91] M. Krause. Lower bounds for depth-restricted branching programs. *Information and Computation,* Vol. 91, 1991. pp. 1–14.

[KMW91] M. Krause, C. Meinel, and S. Waack. Separating the eraser Turing machine classes L_e, NL_e, *co-*NL_e, and P_e. *Theoretical Computer Science,* 86 (1991), pp. 267–275.

[KMW92] M. Krause, C. Meinel, and S. Waack. Separating complexity classes related to certain input-oblivious logarithmic space-bounded Turing machines. *Theoretical Informatics and Applications,* 26 (4) (1992), pp. 345–362.

[KW91] M. Krause and S. Waack. On oblivious branching programs of linear length. *Information and Computation*, **94**:2 (1991), pp. 232–249.

[Kr94] S. Krischer. FANCY—new version (1.1). Theorynet-A announcement from `krischer@ti.uni-trier.de`, U. Trier, Germany, Nov. 17, 1994.

[KM91] E. Kushilevitz and Y. Mansour. Learning decision trees using the Fourier spectrum. *Proceedings of the 23d Annual ACM Symposium on Theory of Computing,* May 1991, pp. 455–464.

[LMN89] N. Linial, Y. Mansour, and N. Nisan. Constant-depth circuits, Fourier transform, and learnability. *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science,* October 1989, pp. 574–579.

[LNNW95] L. Lovász, M. Naor, I. Newman, and A. Wigderson. Search problems in the decision tree model. *SIAM Journal of Discrete Mathematics*, Vol. 8, No. 1 (February 1995), pp. 119–132.

[Ma76] W. Masek. *A Fast Algorithm for the String-Editing Problem and Decision Graph Complexity.* SM Thesis, MIT, 1976.

[Mc86] L. McGeoch. A strong separation between k and $k-1$ round communication complexity for a constructive language. Carnegie Mellon University technical report CMU-CS-86-157 (1986).

[Me89] C. Meinel. *Modified Branching Programs and Their Computational Power.* Springer-Verlag LNCS No. 370, 1989.

[MW95] C. Meinel and S. Waack. Separating complexity classes related to bounded alternating ω-branching programs. *Mathematical Systems Theory*, **28** (1995), pp. 21–39.

[Ne66] E. I. Neciporuk. A boolean function. *Soviet Mathematis Doklady,* 7:4 (1966), pp. 999–1000.

[NW91] N. Nisan and A. Wigderson. Rounds in communication complexity revisited. *Proceedings of the 23d ACM Symposium on Theory of Computing* (1991), pp. 419–429.

[Ok91] E. A. Okolnishnikova. Lower bounds for branching programs computing characteristic functions of binary codes. *Metody discretnogo analiza.* 51 (1991), pp. 61–83. In Russian.

[PS82] C. Papadimitriou and M. Sipser. Communication complexity. *Proceedings of the 14th ACM Symposium on Theory of Computing* (1982), pp. 330–337.

[Ra90] A. A. Razborov. Lower bounds on the size of switching-and-rectifier networks for symmetric Boolean functions. *Mathematical Notes of the Academy of Sciences of the USSR*, 48(6): 79–91, 1990.

[Ra91] A. A. Razborov. Lower bounds for deterministic and nondeterministic branching programs. *Proceedings of the 8th FCT*, Springer-Verlag LNCS 529, 1991. pp. 47–60.

[SS71] A. Schönhage and V. Strassen. Schelle multiplikation grosser zahlen. (Fast multipication of large numbers.) *Computing (Arch. Elektron. Rechen),* 7 (1971), pp. 281–292.

[SDG94] A. Shen, S. Devadas, and A. Ghosh. Probabilistic manipulation of boolean functions using free boolean diagrams. *IEEE Transactions on Computer-Aided Design*, Volume 14, Number 1 (January 1995), pp. 87-95.

[S95] D. Sieling. New lower bounds and hierarchy results for restricted branching programs. *Electronic Colloquium on Computational Complexity,* TR95–002 (January 1995), available via `http://www.eccc.uni-trier.de/eccc/`.

[SW95] D. Sieling and I. Wegener. Graph driven BDD's—a new data structure for boolean functions. *Theoretical Computer Science* 141 (1995), pp. 283–310.

[SS93] J. Simon and M. Szegedy. A new lower bound theorem for read-only-once branching programs and its applications. *Advances in Computational Com-*

plexity Theory (J. Cai, editor), DIMACS Series, Vol. 13, AMS (1993) pp. 183–193.

[SV81] S. Skyum and L. Valiant. A complexity theory based on Boolean algebra. *Proceedings of the 22nd Annual Symposium on Foundations of Computer Science*, (1981) pp. 244-253,

[THY93] S. Tani, K. Hamaguchi, and S. Yajima. The complexity of the optimal variable-ordering problem of shared binary decision diagrams. *Proceedings of the 4th Int'l. ISAAC*, (1993) Springer-Verlag LNCS 762, pp. 389–398.

[We87] I. Wegener. *The Complexity of Boolean Functions.* Wiley-Teubner Series in Computer Science. New York/Stuggart, 1987.

[We88] I. Wegener. On the complexity of branching programs and decision trees for clique functions. *Journal of the ACM,* 35(2) (1988), pp. 461-471.

[We93] I. Wegener. Optimal lower bounds on the depth of polynomial-size threshold circuits for some arithmetic functions. *Information Processing Letters,* Vol. 46, No. 2, pp. 85–87, May 17, 1993.

[We94] I. Wegener. Efficient data structures for boolean functions. *Discrete Mathematics*, 136, (1994), pp. 347–372.

[Za84] S. Zak. An exponential lower bound for one-time-only branching programs. *Proceedings of the 11th MFCT,* Springer-Verlag LNCS No. 176 (1984), pp. 562–566.