On Lower Bounds for Parity Branching Programs

Dissertation
zur Erlangung des Doktorgrades
der Mathematisch-Naturwissenschaftlichen Fakultäten
der Georg-August-Universität zu Göttingen

vorgelegt von
Matthias Homeister
aus
Hildesheim

Göttingen 2003

D 7

Referent:	Prof. Dr. St. Waack
Korreferent:	PD. Dr. Carsten Damm
Tag der mündlichen Prüfung:	15.10.2003

Acknowledgements

I would like to acknowledge all the people whose support gave me the possibility to undergo the work of this thesis.

First of all, I would like to express my thanks to my Advisor *Prof. Dr. Stephan Waack* for his extensive support. A large part of my knowledge of theoretical computer science is due to him. He introduced me to the fascinating field of branching programs and helped me with many very motivating discussions. I also have the pleasure to thank *Henrik Brosenne* who ended my time as a lone fighter. The way we worked together is what one calls team work! I would like to thank *Carsten Damm* for everything he taught me about complexity theory in general and for his helpful hints and remarks. He also gave me deeper insight into the way institutions work and a lot of information about success strategies in research. Furthermore, my thanks go to *Olaf Kuschniok* for enlightening me on the philosophical implications of my work. Even if I still have not understood them... Last but not least, I would like to thank *Conny Caspary!*

Contents

Chapter 1

Introduction

...man solle ein Verfahren angeben, nach welchem sich mittelst
einer endlichen Anzahl von Operationen entscheiden lässt,
ob die Gleichung in ganzen rationalen Zahlen lösbar ist.

David Hilbert, Paris, 1900.

Outline of this introduction. This thesis deals with a structure called *read–once parity branching programs*. This introduction tries to motivate why one should be interested in this structure and why lower bounds for it are interesting. Since branching programs are an important model of computation and restricted variants form important and widely used data structures, we begin by briefly outlining these areas. In Section 1.1 we give some basics on complexity theory. This section is written rather informally to make it comprehensible. In Section 1.2 we argue what a good data structure for Boolean functions should provide. After that we define *branching programs* in Section 1.3 and point out relations to the areas described in the foregoing sections. In Section 1.4 restricted branching programs are considered, namely read-once branching programs and the famous OBDDs. By introducing the notions of nondeterminism and parity acceptance mode in Section 1.5 we approach the model mentioned in the title of this thesis, read-once parity branching programs. We argue why it is difficult to prove lower bounds for this structure in Section 1.6. Section 1.7 summarizes this introduction and describes the progress made by this thesis.

Since the Sections 1.1 to 1.5 are written for those who are not familiar with computational complexity, a reader working in the field of branching programs could start reading at Section 1.7.

1.1 Computational complexity

Algorithms are at the heart of complexity theory. That is the dark secret of complexity theory. These are the first sentences of the *Complexity Theory Companion* by Hemaspaandra and Ogihara ([HO98]), and so first we should answer the question what an algorithm is. Quite a good definition is that given by Hilbert in the opening citation of this introduction. Associated with his tenth problem, addressed at the International Congress of Mathematicians in Paris, 1900, he asked for *a process according to which it can be determined by a finite number of operations*, whether, in that special case, a certain equation is solvable in a specified way. But that definition is still too *intuitive* to be of much help for funding something like an algorithm science. In 1936, Church and Turing proposed two different definitions of an algorithm and it turned out that both of these were equivalent. In a simplified manner we describe Turing's proposal, the Turing machine.

A *Turing machine* is equipped with an infinite tape of cells as its memory. It has a head that can read and write symbols and move around on the tape. When the computation starts the tape contains only the input string and is blank elsewhere. The machine can store information by writing it on the tape. It can move its head all over the tape and so has access to everything stored there. The machine continues computing until it stops and produces an output. Possibly, this sounds rather like the description of a computing device than that of an algorithm. But we still have to introduce the *finite state control* that determines the next action of the Turing machine dependent on the current state (described among other things by the content of the cell under consideration) the next action of the Turing machine. We restrict ourselves to problems whose solution is either 0 or 1, where we consider 1 as *true* or more commonly *accept* and 0 as *false* or *reject*.

Now one calls the set of inputs that are accepted as the *language* accepted by this Turing machine. For instance, a Turing machine that accepts each input, being a prime number, and rejects each input, being a composite number or no number at all, decides the *language consisting of all prime numbers*. So the term *language* serves as a formalization of computation problems with possible output 0 or 1.

A Turing machine can do anything that a real computer can do, where the term *real computer* refers to entities like PCs or mathematicians using pencil and paper. This has come to be called the *Church–Turing thesis.* It states that defining algorithms by Turing machines corresponds with our intuitive notion.

The objective of computational complexity theory is the investigation of the time, memory, or other resources required for solving computational problems. The *running time* of a Turing machine is the number of steps depending on the size of the input. Analogously, we define the *space complexity* to be the maximal number of tape cells (besides those cells containing the input) that are scanned by the Turing machine, again

depending on the input size. It is not possible in general and not even desirable for practical reasons to describe, for instance, the minimal time within which a given problem can be solved on a Turing machine as an exact function of the input size. Such a function necessarily depends too much on the low level details of the model of computation. In complexity theory we only try to sort problems roughly into large classes according to the resources required to solve them in the worst–case. The following complexity classes belong to the most basic ones of classical complexity theory.

The class P contains all languages decidable in *polynomial time* by Turing machines. Intuitively, polynomial time means that for inputs of size n the running time does not grow stronger than n^k for some k. Complexity theorists consider P as containing all problems that are realistically solvable on a computer. From a practical point of view this might be false, in particular when the exponent k is large. But problems requiring superpolynomial or even exponential time are a good deal less solvable and if at all, for tiny inputs. So there is a real gap between problems in P and those outsides. And having proved that a problem is in P, this result most often reveals information about the intrinsic structure of that problem, possibly inspiring further research with much practical impact. Besides this motivation the class P provides strong closure properties and normally is not affected by the chosen model of computation.

Other very elementary classes are NP and NL which are usually defined with the help of *nondeterministic Turing machines*. A nondeterministic Turing machine is a Turing machine that at any point of a computation may proceed according to several possibilities. One may think of a nondeterministic computation as a tree of possibilities. The root of the tree corresponds to the start of the computation. Every branching point in the tree corresponds to a point in the computation at which the machine has multiple choice. The machine accepts if at least one of the computation branches or paths ends in an accepting state. Now we define as the *nondeterministic running time* the length of the shortest accepting path. Intuitively, each accepting computation path (or the choice of it) refers to some *proof* for the membership of x in A. Thus, the class NP, defined to contain all languages decidable in *polynomial time* by *nondeterministic* Turing machines, is the set of languages that can be verified in polynomial time given some additional information, the *proof*. We consider an example. Let *COMPOSITES* be the set of numbers greater than 1, that are not prime numbers, i.e., $x \in \mathit{COMPOSITES}$, if there are integers $p, q > 1$ with $p \cdot q = x$. Now given some $x \in \mathit{COMPOSITES}$, and in addition as proof two integers p, q, we can multiply p and q, and check that $p \cdot q = x$, verifying x's membership in *COMPOSITES*.

L is the class of all languages decidable by deterministic Turing machines using not more than *logarithmic space*. Intuitionally, this means that for inputs of size n the running

time does not grow stronger than $k \cdot \log n$ for some k. Analogously to NP, we define NL as the class of all languages accepted by nondeterministic Turing machines using not more than *logarithmic space*. It is quite easy to see the following relation between the classes mentioned.

$$\mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{P} \subseteq \mathrm{NP}.$$

But to prove that one of these inclusions is proper belongs to the fundamental open problems of complexity theory. Most famous is the P vs. NP problem.

Why is deciding whether P=NP or P≠NP of such importance? If it is proved that these classes are equal, each polynomially verifiable problem will turn out to be decidable by polynomial time algorithms and consequently, many problems for which only superpolynomial algorithms are known will get within the range of solvability. But most researchers believe that this is not true. Proving that these classes are unequal means to show that for some problem verifiable in polynomial time, each algorithm deciding it needs superpolynomial time, i.e. to prove a superpolynomial lower bound for the running time of all deterministic algorithms solving this problem. And this would surely give much insight into the inherent reasons why some problems are easy and others are difficult to solve. The practical impact of such a result should not be underestimated. The following says something about the attention that the P vs. NP problem has attracted. The Clay Mathematics Institute of Cambridge, Massachusetts has named seven so-called Millennium Prize Problems. For the solution of each of these problems a $1 million prize was designated. Deciding whether P=NP is, besides for instance the Riemann Hypothesis, one of these very important open problems of mathematics (see [CMI00]).

1.2 Data structures for Boolean functions

Let us consider the following fundamental task called *circuit verification*. One is given a newly designed circuit and has to prove that this circuit is correct. *Correct* means that the function computed by this circuit agrees with the function described by some specification. This is called *equivalence* of circuit and specification. For instance, such a specification can be another circuit. Normally, a circuit computes a *Boolean function*. A Boolean function in n variables is a function $f : \{0,1\}^n \to \{0,1\}$. Boolean functions are very important since every *finite function* can be considered as a sequence $(f_1, \ldots, f_m)$ of Boolean functions. A function $f : A \to B$ is called *finite* if A and B are arbitrary sets of finite size.

The idea is to transform the circuit as well as the specification into different instances of a data structure that supports the equivalence test efficiently. For circuits the equivalence

test is co-NP-complete. Most researchers believe that NP-complete problems and co-NP-complete problems are *intractable*. A problem is called intractable if it is solvable in principle, but the solutions require so much time that they can not be used in practice. Maybe, the infamous Pentium Division Bug could have been avoided if circuit verification was not so difficult.

There are many areas besides circuit verification in which it is important to store Boolean functions succinctly and manipulate them efficiently. One has to mention symbolic model checking, computer aided design, artificial intelligence, optimization, counting, genetic programming (see [Weg00], pages 313-378) and cryptography (see [Kra02]).

In the following we present some very important operations that every useful data structure for Boolean functions has to provide efficiently. A list of this type has been published in the fundamental paper of Bryant, [Bry86].

- **Evaluation.**

  Input: A representation A for a Boolean function $f : \{0,1\}^n \to \{0,1\}$, an assignment $a \in \{0,1\}^n$.

  Output: $f(a)$.

- **Boolean synthesis.**

  Input: Representations A_1, A_2 for Boolean functions f_1, f_2, resp., and a Boolean operation $\otimes : \{0,1\}^2 \to \{0,1\}$.

  Output: A representation $A^{\otimes}$ for $f_1 \otimes f_2$.

- **Minimization.**

  Input: A representation A for a Boolean function f.

  Output: A representation A' for f, that is size–minimal in the set of all representations (of a given type) for f.

- **Equivalence test.**

  Input: Representations A_1, A_2 for Boolean functions f_1, f_2, resp.

  Output: Yes, if $f_1 = f_2$; no, otherwise.

- **Satisfiability test.**

  Input: A representation A for a Boolean function f.

  Output: Yes, if there is an assignment $a \in \{0,1\}^n$ such that $f(a) = 1$; no, otherwise.

How are these operations applied for verifying Boolean circuits? First note that the input bits represent the functions $f_1 = x_1, \ldots, f_n = x_n$. We traverse the circuit in topological order. If we reach a gate computing an operation $\otimes$ we perform Boolean synthesis. To avoid a blow–up of the size we frequently use the minimization. If we have transformed both the circuit and the specification into instances of the data structure we complete our task with the help of the equivalence test. Alternatively, one can proceed by first performing Boolean synthesis of the instances with respect to the exclusive–or operation and applying the satisfiability test on the result.

1.3 Branching programs

We begin this section with the most fundamental definition of this work. It is illustrated by Figure 1.1.

Definition 1.1 *A* branching program (BP, for short) *on the variables* $\{x_1, x_2, x_n\}$ *is a directed acyclic graph with one unlabeled source* s *and two sinks labeled by the Boolean constants* 0 *or* 1. *Each node that is neither the source nor a sink is called a* branching node *and is labeled by a variable* x_i. *Each branching node has exactly two outgoing edges, one labeled by* 0 *and the other labeled by* 1.

Such BPs are called *deterministic* to distinguish them from *nondeterministic* ones that we introduce in Section 1.5. Note, that the requirement of an unlabeled source is non–standard. It is convenient for studying those nondeterministic BPs just mentioned. The *size* of a BP $\mathcal{B}$ is the number of its nodes and is denoted by $\mathrm{SIZE}(\mathcal{B})$.

Instead of *branching program* the synonymous term *binary decision diagram*, or BDD, for short, is in use. This has historical reasons and those mostly interested in data structures use the term *BDD* whenever a complexity theorist uses the term *BP*. Since this thesis is mainly about lower bounds, we prefer speaking about branching programs. Only in the case of OBDDs (see Section 1.4) this would be quite unusual. In Appendix B the reader can find a list of all variants of BPs mentioned in this thesis. This may help to avoid confusion.

By Definition 1.1 we have already described the syntax of a BP. To complete the definition we still have to describe the way a Boolean function is determined.

Definition 1.2 *A branching program computes a Boolean function as described by the following algorithm. For any assignment to the variables on that the BP is defined start at the successor of the source, follow the path determined by taking the outgoing edge from each branching node according to the value assigned to the indicated variable until a sink*

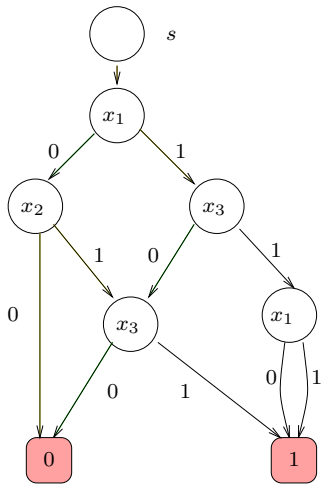

Figure 1.1: A branching program in the variables $\{x_1, x_2, x_3\}$.

is reached. This path is called the computation path *corresponding to this input. The output is the label of that sink.*

It is very helpful to see that each node of a BP for itself represents a Boolean function. Throughout this work, for a node v, Res_v denotes the function represented by the node v and is called *resulting function*. $\mathrm{Res}(\mathcal{B})$ denotes the function represented by the whole diagram $\mathcal{B}$. If v is the 1−sink (0−sink, resp.), then Res_v is the all-one (all-zero) function. If v is a branching node labeled by x_i with 1−successor v_1 and 0−successor v_0, then $\mathrm{Res}_v = x_i \wedge \mathrm{Res}_{v_1} \vee (x_i \oplus 1) \wedge \mathrm{Res}_{v_0}$, where $\oplus$ denotes the exclusive–or operation. Clearly, $\mathrm{Res}(\mathcal{B})$ is the function represented by the successor of the source. Using this inductive approach, one easily gets that the BP of Figure 1.1 represents the function $f(x_1, x_2, x_3) = (x_1 \vee x_2) \wedge x_3$.

Definition 1.2 in fact describes a linear time algorithm for evaluation. Furthermore, Boolean synthesis of two BPs $\mathcal{B}_1, \mathcal{B}_2$ is tractable in linear time, and for the size of the result $\mathcal{B}^{\otimes}$ it holds that $\mathrm{SIZE}(\mathcal{B}^{\otimes}) \leq \mathrm{SIZE}(\mathcal{B}_1) + \mathrm{SIZE}(\mathcal{B}_2)$. Let us consider the following example. To handle $\otimes = \wedge$ just identify the 1−sink of $\mathcal{B}_1$ with the source of $\mathcal{B}_2$ and merge the 0−sinks of $\mathcal{B}_1$ and $\mathcal{B}_2$. But for unrestricted branching programs all other operations mentioned in Section 1.2 are not tractable. Similarly as in the case of circuits, the satisfiability test is NP-complete and the equivalence test is co-NP-complete. Furthermore, minimization is NP-hard. Thus, after common believe these operations are intractable (see Section 1.2). To get an applicable data structure one has to restrict the definition

which we do in Section 1.4.

Now we turn to the complexity theoretical use of BPs. A branching program describes an algorithm computing a Boolean function. The complexity of this algorithm is described in terms of its size. Since the interest of complexity theory lies in studying the growth of the amount (here the size of the BP) for increasing input sizes, normally sequences of functions $(f_n)_{n\geq 1}, f_n : \{0,1\}^n \to \{0,1\}$ are under consideration. If, for instance, AND_n denotes the function $x_1 \wedge \ldots \wedge x_n$ that equals 1 if and only if all n bits of the input are 1, then by speaking about *the function AND* we refer in effect to the sequence $(\mathrm{AND}_n)_{n\geq 1}$. Consequently, one is interested in sequences of branching programs, too. In this context we introduce the following notation that is used throughout this work. If X is a model of computation, then we denote by $\mathcal{P}(X)$ the functions representable by polynomial size instances of X.

We make this explicit by an example. Let X be the computation model *BP*. Then $\mathcal{P}(\mathrm{BP})$ contains all sequences $(f_n)_{n\geq 1}, f_n : \{0,1\}^n \to \{0,1\}$ such that there is a polynomial $p(n)$ and a sequence of BPs $(\mathcal{B}_n)_{n\geq 1}$ such that the following holds. For all $n \geq 1$ the BP $\mathcal{B}_n$ represents f_n and has size $\mathrm{SIZE}(\mathcal{B}_n) \leq p(n)$.

Since for each input length n we have a separate branching program , BPs are a *nonuniform* model of computation, as, for instance, circuits. The next definition serves to describe the connection between nonuniform computational models to Turing machines.

Definition 1.3 *A* nonuniform Turing machine *is a Turing machine that is equipped with an additional read–only tape, the* advice tape. *On input x of length $|x| = n$ this tape is automatically loaded with the* advice string $A(n)$*, where $A : \mathbb{N} \to \{0,1\}^*$ is an arbitrary function.*

A nonuniform Turing machine is said to have polynomial advice *if there is a polynomial $p(n)$ such that $|A(n)| \leq p(n)$ for all n.*

$\mathrm{P/\,Poly}, (\mathrm{L/\,Poly})$ *is the class of all languages accepted in polynomial time (logarithmic space, resp.) by nonuniform Turing machines with polynomial advice.*

The following result is due to Cobham, [Cob66], and characterizes the connection between BPs and Turing machines.

Theorem 1.4 *The set of functions accepted by polynomial–sized branching programs is equal to the set of functions accepted by nonuniform Turing machines using not more than logarithmic space,*

$$\mathcal{P}(\mathrm{BP}) = \mathrm{L/\,Poly}\,.$$

Since nonuniform Turing machines generalize uniform ones, Theorem 1.4 implies the following. If for a function f each BP representing f is of super–polynomial size, then $f \notin \mathrm{L}$.

But the best known explicit lower bound on the size of unrestricted deterministic BPs is of order $\Omega\left(\frac{n^2}{(\log n)^2}\right)$. It was proved by Nechiporuk in 1966 (see [Nec66]). Besides the fact that unrestricted BPs are not usable as a data structure as we argued above, this is one reason more why restricted models have been studied intensively. The hope is that lower bound methods for restricted BPs may inspire lower bounds for generalized BPs and maybe even for computational models not being logarithmically space restricted.

1.4 Restricted branching programs

One of the most popular and well–examined restricted variants of BPs is the following one.

Definition 1.5 *A* read–once branching program *(*BP1*, for short) is a BP with the restriction that on each path from the source to a sink each variable is allowed to appear not more than once.*

This restriction seems to be very natural. In fact, it is possible to prove an analogue of Theorem 1.4. BP1s correspond to nonuniform Turing machines with the following restriction. After reading an input bit, this bit is never touched again. The result on the correspondence between BP1s and Turing machines can be found in [ABH$^+$86].

The first lower bounds for this model were proved by Žák, [Žák84], and Wegener, [Weg88], for certain functions testing the existence of cliques in graphs. Later lower bounds for many other functions were proved. Here we mention Ponzio's lower bound for the middle bit of integer multiplication, [Pon95], and the generalization of the different approaches by Simon and Szegedy, [SS93].

Now the question arises whether BP1s can be used as a data structure. Clearly, evaluation can be performed in time $O(n)$ where n is the number of variables. Using a depth first search approach it is possible to perform the satisfiability test in linear time where the input size is the size of the branching program. But synthesizing two polynomial sized BP1s can cause an exponential blow–up in size. This is due to results in [BW97]. Consider the function $\mathrm{ROW}_n + \mathrm{COL}_n$ defined as follows. The input is some $n \times n$ matrix X over $\{0,1\}$. We define $\mathrm{ROW}_n(X) = 1$, if X possesses one row consisting entirely of ones and, analogously, $\mathrm{COL}_n(X) = 1$, if X possesses one column constisting entirely of ones. Now $\mathrm{ROW}_n(X) + \mathrm{COL}_n(X) = 1$ if $\mathrm{ROW}_n(X) = 1$ or $\mathrm{COL}_n(X) = 1$.

It is easy to construct a linear sized branching program representing the function ROW_n testing the variables in the same rowwise manner on all paths (in Definition 1.7 such a branching program will be called an OBDD). In the same way we get a linear sized branching program representing COL_n that on all paths tests the variables in the same columnwise manner. But in [BW97] it is proved that each BP1 for $\mathrm{ROW}_n + \mathrm{COL}_n$ has size bounded below by $\Omega\left(n^{-7/2} \cdot 2^n\right)$. In Section 2.7 we generalize this result to lower bounds for restricted parity branching programs that are defined in the next section.

So BP1s cannot be used as a data structure. One possibility to cope with this problem is to introduce a *graph–ordering*. Observe that for some BP1 on $\{x_1, x_2, \ldots, x_n\}$, there is for each input $a \in \{0,1\}^n$ a variable ordering $\sigma(a)$ according to which the bits of a are queried. But not every combination of variable orderings can be implemented by deterministic BP1s. Only those resulting from *graph orderings*, independently introduced by Gergov and Meinel, [GM93], and Sieling and Wegener, [SW95], are possible.

Definition 1.6 *A* graph ordering *is a deterministic branching program with a single sink and the property that, on each path from the source to the sink, each variable is tested exactly once.*

A BP1 $\mathcal{B}$ is called graph-driven *if it is guided by such a graph-ordering G in the following sense. For each input $a \in \{0,1\}^n$ and on every computation path in $\mathcal{B}$ corresponding to a the variables are tested in the same order as in the graph-ordering G where it is allowed to leave out some variables.*

For every deterministic BP1 $\mathcal{B}$, it is easy to construct a graph ordering G that guides $\mathcal{B}$. It turns out that according to some fixed graph–ordering BP1s form a data structure. In Section 2.1 we consider graph–driven BP1s more closely and consider examples. By further strengthening the restriction to very special graph–orderings we obtain the most common used data structure, the so called *OBDDs* introduced by Bryant in [Bry86].

Definition 1.7 *A BP1 is called* oblivious *or* ordered binary decision diagram (OBDD) *if for each input $a \in \{0,1\}^n$ the variables are tested in the same ordering where it is allowed to leave out some variables.*

This data structure is used very widely and is considered as the *state of the art data structure*. The reason is that all important operations (see Section 1.2) can be performed very efficiently. For instance, the minimization according to a fixed variable ordering can be achieved in linear time. Furthermore, several implementations - so–called OBDD packages - have been developed and refined. There is an enormous amount of literature concerning different aspects of OBDDs. We refer to the overview article by Bryant, [Bry92]. The drawback of OBDDs is that the computational power is quite restricted and

many rather simple functions have only exponential size representations. For that reason several generalizations of OBDDs have been examined as candidates for data structures. Some of them make use of *nondeterminism*.

Before defining nondeterministic BPs in the next section, we would like to complement the lower bound on unrestricted BPs proved by Nechiporuk, see Section 1.3, with some important results on restricted deterministic BPs. For convenience we neglect the difference between *semantic* and *syntactic* variants.

A BP is called *read-k-times* if on every path each variable is tested at most k times. Based on results by Okol'nishnikova ([Oko97a] and [Oko97b]), in 1998 Thathachar proved that the computational power of read-$(k+1)$-times BPs is strictly stronger than that of read-k-times BPs (see [Tha98]), being the most general hierarchy result in this area. Furthermore, depth-restricted BPs have been considered, where the depth is the length of the longest path. Beame, Saks and Thathachar were the first to obtain exponential lower bounds for BPs of depth $(1+\epsilon)\cdot n$, with $\epsilon = 0.0178$, [BST98]. 1999 Ajtai proved an exponential lower bound for BPs of depth kn, k any constant, [Ajt99]. These results have been refined in some papers by Beame, Saks, Sun, and Vee (see, for instance, [BSSV00] and [BV02]). All these results are inspired by a lower bound technique of Borodin, Razborov and Smolensky ([BRS93], see also the next section).

1.5 Nondeterministic branching programs

The class NP mentioned in Section 1.1 deals with nondeterministic Turing machines. Given a potential solution it is possible to verify in polynomial time whether it is correct. The next definition generalizes branching programs to nondeterministic branching programs.

Definition 1.8 *A nondeterministic branching program is defined like a deterministic BP except that the source may have an unrestricted number of successors and each branching node may have an unrestricted number of* $0-$ *and* $1-$*successors. An input* $a \in \{0,1\}^n$ *is accepted if at least one computation path corresponding to* a *reaches the* $1-$*sink.*

Nondeterminism strengthens the computational power of many restricted variants of BPs. For instance, polynomial size nondeterministic OBDDs that are obtained by combining Definitions 1.8 and 1.7 can represent the negations of characteristic functions of linear codes (observed by Jukna in [Juk95b], see Section 3.4 for the notion of linear codes), $\overline{\mathrm{PERM}_n}$, the function accepting all Boolean matrices that are *not* permutation matrices, (see [Kra88] and Section 2.6) and $\mathrm{ROW}_n + \mathrm{COL}_n$ considered in the preceding section.

For convenience, we call a computation path reaching the 1−sink an *accepting path (corresponding to a)*. Note, that in the setting of Definition 1.8 no 0−sink is required. For a node u we denote by $\mathrm{Succ}_e(u)$ the set of e-successors of $u, e \in \{0,1\}$ and by $\mathrm{Succ}(s)$ the successors of the unlabeled source. Another mode of nondeterminism has been studied intensively. Informally, the idea is to count the number of accepting paths instead of just checking the existence of at least one such path.

Definition 1.9 *A* parity branching program (⊕–BP, for short) *is syntactically defined exactly as a nondeterministic BP. Its semantics is the following. An input $a \in \{0,1\}^n$ is accepted if the number of accepting paths is odd, and rejected in the other case.*

We say that a ⊕–BP is equipped with the *parity acceptance mode*. To characterize the function represented by a ⊕–BP it is convenient to introduce the following notations. We regard $\mathbb{B}_n$, the set of all Boolean functions in n variables, as an $\mathbb{F}_2$–algebra, where $\mathbb{F}_2$ is the prime field of characteristic 2. For $f, g \in \mathbb{B}_n$ the product $f \wedge g$ is defined as the componentwise conjunction and the sum $f \oplus g$ as the componentwise exclusive–or. For a partial assignment α to a variables, the subfunction $f|_\alpha$ results by setting these variables to the constants according to α.

Which function is represented by a node u of a ⊕–BP? In Section 1.3 we introduced the resulting function Res_u. It is possible to generalize this to ⊕–BPs in the following way. The resulting function of the target (the 1−sink) equals the all–one function. For a branching node u labeled by a variable x we get

$$\mathrm{Res}_u := (x \oplus 1) \wedge \bigoplus_{v \in \mathrm{Succ}_0(u)} \mathrm{Res}_v \;\oplus\; x \wedge \bigoplus_{v \in \mathrm{Succ}_1(u)} \mathrm{Res}_v \,.$$

If s is the source, then $\mathrm{Res}_s := \bigoplus_{v \in \mathrm{Succ}(s)} \mathrm{Res}_v$. The function $\mathrm{Res}(\mathcal{B}) : \{0,1\}^n \to \{0,1\}$ represented by the whole diagram is defined to be Res_s.

⊕–BPs provide very good computational properties and restricted ⊕–BPs also good algorithmic properties. Nondeterministic BPs and ⊕–BPs correspond to the classes NL/Poly and ⊕L/Poly, the classes of functions computable by logarithmic space restricted Turing machines with polynomial advice and the particular acceptance mode. In [Wig94], Wigderson proved that NL/Poly $\subseteq$ ⊕L/Poly. This result illustrates the strength of the parity acceptance mode in respect of logarithmic space bounded computations and can be reformulated as follows. If some function f is representable by polynomial size nondeterministic BPs, then f is also representable by polynomial size ⊕–BPs.

Next we examine the algorithmic properties of the parity acceptance mode by compairing nondeterministic OBDDs and ⊕OBDDs. ⊕OBDDs are (nondeterministic) OBDDs equipped with the parity acceptance mode and are introduced by Gergov and Meinel

in [GM96]. To test whether a nondeterministic OBDD represents the all–one function is co-NP-complete. But in [Waa01] Waack has proved that for ⊕OBDDs all operations listed in Section 1.2 are tractable in polynomial time. He algebraically characterized size–minimal ⊕OBDDs and presented cubic algorithms for Minimization and consequently, the Equivalence test based on this characterization. In addition, in [GM96] Gergov and Meinel have already proved that Boolean synthesis is tractable. In [BW98] an algorithm is presented that transforms a ⊕OBDD with respect to one variable ordering into an equivalent ⊕OBDD with respect to another preassigned variable ordering. So one may conclude that ⊕OBDDs have properties very similar to deterministic OBDDs. Indeed, on the one hand the computational power of ⊕OBDDs is strictly stronger, on the other hand the minimization algorithm is less efficient (cubic time versus linear time). Heuristics for a successful practical implementation are due to Meinel and Sack, see [MS01a], [MS01b] and [Sac01].

For ⊕OBDDs several lower bounds are known. In [Ger94], Gergov observed that the OBDD lower bound argument used by Bryant, [Bry91], for the middle bit of integer multiplication is also applicable for ⊕OBDDs. Applying the algebraic characterization of size–minimal instances proving lower bounds for ⊕OBDDs becomes a rather easy task as we outline in the next section.

But proving a superpolynomial lower bound for ⊕BP1s is still an open problem, whereas exponential lower bounds for nondeterministic BP1s have been known for a long time. In 1989 Jukna, [Juk89], and, independently Krause, Meinel and Waack, [KMW91], proved exponential lower bounds for nondeterministic BP1s and permutation matrices. In 1993, Borodin, Razborov and Smolensky, [BRS93], proved lower bounds even for nondeterministic *read-k-times BPs*, that are nondeterministic BPs where on every path each variable may be tested not more than k times.

1.6 Why proving lower bounds for ⊕BP1s is difficult

In the last section we have already mentioned the following result. Wigderson proved in [Wig94] that NL/Poly $\subseteq$ ⊕L/Poly. This illustrates the strength of the parity acceptance mode in respect of logarithmic space bounded computations and gives a first hint of the difficulties in proving lower bounds for ⊕BP1s. In addition, we wish to outline informally the common proof methods for ⊕OBDDs and nondeterministic BP1s, arguing why these methods fail when being applied to ⊕BP1s.

For nondeterministic BP1s one can prove lower bounds arguing in the following way. Let a_1 and a_2 be two assignments accepted by a nondeterministic BP1 $\mathcal{B}$. If one accepting path according to a_1 and one accepting path according to a_2 pass the same node v we are

able to construct a third accepting input b in the following way. Let α_i be the part of a_i, $i = 1, 2$ that is tested before reaching node v, excluding the variable tested in v and let β_i be the part of a_i, $i = 1, 2$ tested strictly below v. Then we can compose assignments (α_1, β_2) and (α_2, β_1) that are also accepted by $\mathcal{B}$.

Now we choose a set $\{a_1, \ldots, a_\mu\}$ of accepting assignments according to $\mathcal{B}$. If for each $1 \leq i < j \leq \mu$ it holds that (α_i, β_j) or (α_j, β_i) must not be accepted by $\mathcal{B}$ we can conclude that $\mathrm{SIZE}(\mathcal{B}) \geq \mu$. This method, called the *cut and paste* method, for instance is used in [Žák84], [Weg88] and [Pon95].

But it is not possible to prove lower bounds for $\oplus$BP1s in this way, not even for any restricted variant equipped with full parity–nondeterminism. The reason is that in the situation described above we can not conclude that some assignment (α_i, β_j) is accepted by a $\oplus$BP1. We have to mind the number of all paths for (α_i, β_j) that reach the sink.

In the case of $\oplus$OBDDs we can cope with this problem in the following way. Let us assume that the variables are tested according to the ordering $(x_1, \ldots, x_n)$. Let $\{\alpha_1, \ldots, \alpha_\mu\}$ be the set of partial assignments to $x_1, \ldots, x_k$. Using results from [Waa01] we get that the nodes of the $\oplus$OBDD have to represent a basis of the linear space spanned by $\{f|_{\alpha_1}, \ldots, f|_{\alpha_\mu}\}$, where for a partial assignment α, the subfunction $f|_\alpha$ results by setting these variables to the constants according to α (see Section 1.5). So the dimension of this space entails a lower bound for $\oplus$OBDDs with this certain variable ordering.

In the case of $\oplus$BP1s not only for each input a the variables may obey another ordering. For each single input a the variables may be tested in arbitary many different orderings. Therefore, some experts believe that methods using *subfunction arguments* are of no use for proving lower bounds for $\oplus$BP1s. In this work we see that in the case of graph–driven $\oplus$BP1s generalized subfunction arguments work (see Chapters 2 and 3). Admittedly, one has to deal with subfunctions depending on quite different sets of variables. To get lower bounds for the sum of graph–driven $\oplus$BP1s, see Definition 4.1, we do not make use of subfunctions anymore.

1.7 Summary and contributions of this thesis

A central problem of the theory of computation is to understand the *inherent complexity* of computational tasks with respect to different models of computation. Branching programs or BPs, see Definitions 1.1 and 1.2, form a well-established model of computation and various restricted variants are used as data structures for Boolean functions, then most often refered to as Binary Decision Diagrams, or BDDs for short. One reason for the interest in BPs as a computational model is that tight connection to sequential space bounded computations stated in Theorem 1.4. Unfortunately, the best known lower bound

on the size of unrestricted deterministic BPs proved by Nechiporuk ([Nec66]) in 1966 is of order $\Omega\left(\frac{n^2}{(\log n)^2}\right)$. So restricted variants are under consideration. A restricted model studied very intensively are *read–once branching programs*, BP1s for short, where on each path from the source to some sink each variable may be tested at most once, see Definition 1.5.

In addition, one is interested in nondeterministic branching programs and parity branching programs, see Definitions 1.8 and 1.9. Due to Wigderson we know that the computational power of $\oplus$BPs is at least so high as that of nondeterministic BPs, [Wig94]. Additionally, in 1997 Waack proved that $\oplus$OBDDs (see Section 1.5) provide good algorithmical properties. This is not the case for nondeterministic OBDDs.

So special interest arises in the study of $\oplus$BP1s. But whereas for deterministic BP1s exponential lower bounds have already been proved in 1984 (see [Žák84] and [Weg88]), for nondeterministic BP1s in 1989 (see [Juk89] and [KMW91]) and in 1993 even for nondeterministic read–k–times BPs (on each path each variables may occur not more than k times, [BRS93]) proving lower bounds for parity read–once BPs, or $\oplus$BP1s, is still an open problem, maybe one of the most challenging open problems of this field. The following two models have already been studied. In [SS00], Savický and Sieling proved lower bounds for pointer functions on the size of $(\oplus, k)$–BPs are proved. A $(\oplus, k)$–BP is a read–once BP with the source being the only nondeterministic node, where k denotes the fan–out of the source.

In [Bol00] the first exponential lower bound on the size of restricted $\oplus$BP1s with an unbounded number of nondeterministic branching nodes that generalize $\oplus$OBDDs is proven. More precisely, graph–driven $\oplus$BP1s guided by a *tree ordering* are under consideration. Lower bounds for this highly restricted model are proved by applying methods similar to those used for $\oplus$OBDDs (see Section 2.2).

In this thesis lower bounds for several restricted variants of $\oplus$BP1s are proven, in fact, the most general lower bounds for parity branching programs.

In *Chapter 2* we consider *well–structured graph–driven $\oplus$BP1s*. We have already defined graph–driven BP1s by Definition 1.6. Such a graph–driven BP1 is called well–structured if it fulfills an additional property that divides the set of nodes into different levels, determined by the graph–ordering. We show that well–structured graph–driven $\oplus$BP1s strictly generalize deterministic BP1s as well as $\oplus$OBDDs and tree–driven $\oplus$BP1s mentioned above. Well–structured graph–driven $\oplus$BP1s deserve particular interest, since size–minimal instances can be described algebraically and as a consequence this model is applicable as a data structure. We present lower bounds for permutation matrices and show that the computational power of general $\oplus$BP1s is strictly stronger than that of well–structured graph–driven $\oplus$BP1s. Both results are generalized in Chapter 3, but the

comparison of the proofs give some insight into the structure of the two different models. In addition, we derive a lower bound for the function $\mathrm{ROW}_n + \mathrm{COL}_n$ already mentioned in Section 1.4 of this introduction

In *Chapter 3* we consider graph–driven $\oplus$BP1s without the restriction being well–structured. For each deterministic BP1 there is a graph–ordering - the notion of a graph–driven BP1 is motivated in that case. This is not true for $\oplus$BP1s. But we show that a graph–ordering can be constructed if and only if for each input a there is a variable ordering $\sigma(a)$ of $\{x_1, x_2, \ldots, x_n\}$ such that on each computation path for a the bits of a are queried according to $\sigma(a)$. This shows that the condition of being guided by a graph ordering is in fact a very natural combinatorial one. Thus, being graph–driven is also for $\oplus$BP1s a natural concept. Moreover, we characterize the connection between graph–driven $\oplus$BP1s and well–structured ones. We prove lower bounds for several functions, namely for linear codes, permutation matrices, the determinant, for Hamiltonian cycles and for integer multiplication. We show that in terms of computational power unrestricted $\oplus$BP1s are strictly stronger than graph–driven ones.

In *Chapter 4* we prove the first lower bound for restricted read–once parity branching programs with unlimited parity nondeterminism where for each input the variables may be tested according to several orderings. Under consideration are sums of graph–driven $\oplus$BP1s where the graph–orderings are of polynomial size. In terms of computational power the sum of two graph–driven $\oplus$BP1s strictly generalizes graph–driven $\oplus$BP1s. In particular, the sum of two graph–driven $\oplus$BP1s driven by polynomial size graph–orderings strictly generalizes well–structured graph–driven $\oplus$BP1s. Furthermore, we show that sums of k $\oplus$BP1s driven by polynomial size graph–orderings strictly generalize $(\oplus, k)$-BPs mentioned above (examined by Savický and Sieling in [SS00]). We prove a lower bound criterion for sums of graph–driven $\oplus$BP1s and derive lower bounds for linear codes for sums of graph–driven $\oplus$BP1s guided by polynomial size graph–orderings.

The presented lower bounds for graph–driven $\oplus$BP1s and sums of graph–driven $\oplus$BP1s are the most general lower bounds for restricted $\oplus$-BPs. In particular, the methods applied in Chapter 4 for proving lower bounds for sums of graph–driven $\oplus$BP1s should be a step towards lower bounds for $\oplus$BP1s.

1.8 Publications

Some of the results presented in this thesis have already been published.

Conference proceedings.

- H. Brosenne, Homeister M., and St. Waack. Graph–driven free parity BDDs: Algorithms and lower bounds. In *Proceedings, 26th MFCS*, volume 2136 of *Lecture Notes in Computer Science*, pages 212–223. Springer Verlag, 2001.
- M. Homeister. On well–structured parity–FBDDs. In *Proceedings of the 6th International Symposium on Representations and Methodology of Future Computing Technology*, 2003.
- H. Brosenne, M. Homeister, and St. Waack. Lower bounds for general graph-driven read-once parity branching programs. In *Proceedings of the 28th symposion on Mathematical Foundations of Computer Science (MFCS)*, volume 2747 of *Lecture Notes in Computer Science*, pages 290–299. Springer Verlag, 2003.

Journal articles.

- H. Brosenne, M. Homeister, and St. Waack. Characterizing the complexity of boolean functions represented by well–structured graph–driven parity–FBDDs. *RAIRO Theoretical Informatics and Applications*, 36:229–247, 2002.

Technical Reports.

- M. Homeister. Lower bounds for the sum of graph–driven read–once parity branching programs. In *Electronic Colloquium on Computational Complexity, ECCC Report TR03-068*. `www.eccc.uni-trier.de`, 2003.

Chapter 2

Well-Structured Graph–Driven $\oplus$BP1s

Outline of this chapter. We introduce the subject of this chapter, well–structured graph–driven $\oplus$BP1s, in Section 2.1 and present upper bounds and previous results in Section 2.2. There it is shown that well–structured graph–driven $\oplus$BP1s generalize $\oplus$OBDDs as well as read–once BPs. In Section 2.3 we cite the important algebraic characterization of size–minimal instances taken from [BHW01] and mention the most important algorithms provided by this structure in Section 2.4. There it is shown that well–structured graph–driven $\oplus$BP1s generalize $\oplus$OBDDs as well as deterministic BP1s. In Section 2.5 we prove a lower bound criterion that is applied to permutation matrices in Section 2.6 and in Section 2.8 to prove that the computational power of general $\oplus$BP1s is strictly larger than that of well–structured graph–driven $\oplus$BP1s. In addition, in Section 2.7 we prove lower bounds for the function $\mathrm{ROW}_n + \mathrm{COL}_n$ already mentioned in Section 1.4 of the introduction.

The main results of this chapter are already published in [BHW01] and [Hom03b]. [BHW02] is a revised version of [BHW01]. Except from the upper bound in Section 2.2 the results of [BHW01] are stated without proof. Sections 2.5 to 2.8 describe results contained in [Hom03b].

2.1 Definitions

Deterministic BP1s are not usable as a data structure. In [GM93] and in [SW95] independently BP1s equipped with a graph ordering are introduced and their good algorithmical behaviour is described. The graph ordering plays the same role as the variable ordering

in the OBDD–case. We have introduced the notion of being graph–driven in Definition 1.6. For the sake of self–containment we restate it here.

A graph ordering *is a deterministic branching program with a single sink and the property that, on each path from the source to the sink, each variable is tested exactly once. A BP1 $\mathcal{B}$ is called* graph-driven *if it is guided by such a graph-ordering G in the following sense. For each input $b \in \{0,1\}^n$ and on every computation path in $\mathcal{B}$ according to b the variables are tested in the same order as in the graph-ordering G where it is allowed to leave out some variables.*

Figure 2.1 shows an example of this definition. Recently, Krause applied deterministic graph–driven BP1s for cryptanalysing keystream generators, [Kra02]. So, interest arises in graph–driven parity BP1s whose computational power is assuredly. Another motivation is that lower bound methods for graph–driven ⊕BP1s may inspire corresponding techniques for general ⊕BP1s. In Chapter 3 graph–driven ⊕BP1s are considered in detail.

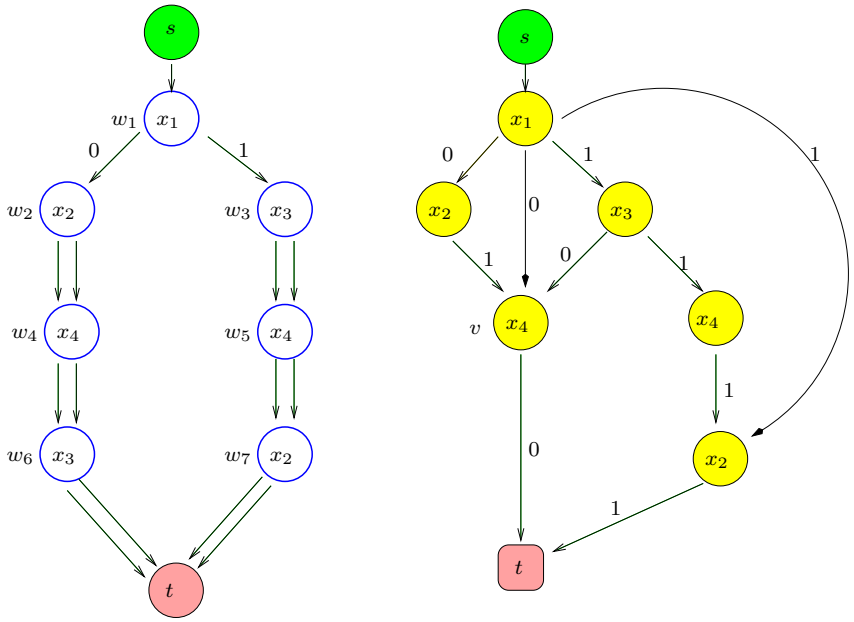

Figure 2.1: A graph ordering and a graph–driven ⊕BP1 guided by this ordering.

In this chapter we consider a restricted variant called *well-structured*. It turns out that for well-structured graph–driven ⊕BP1s all operations listed in Section 1.2 are feasible by polynomial–time algorithms. So well-structured graph–driven ⊕BP1s can be used as a data structure for Boolean functions. Moreover, size–minimal instances can be described

in quite an elegant way.

Definition 2.1 *A graph–driven $\oplus$BP1$\mathcal{B}$ with graph–ordering G is called* well-structured *if there is a function* level *mapping from the nodes of $\mathcal{B}$ to the nodes of the ordering G in the following way. For any node v that corresponding to an input is traversed on a path in $\mathcal{B}$, in G the node* level(v) *is traversed corresponding to this input and is labeled with the same variable.*

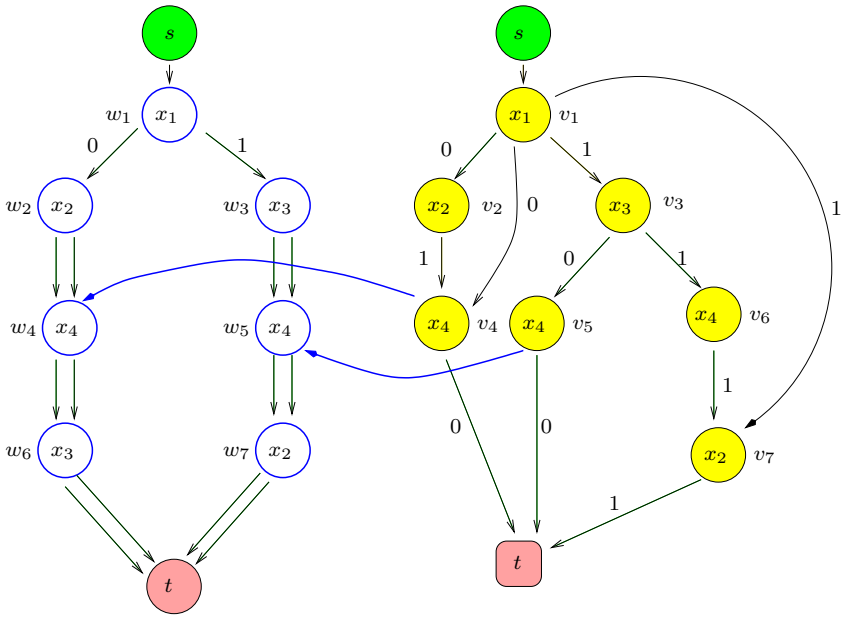

Figure 2.2: A well–structured graph–driven $\oplus$BP1 guided by a graph ordering.

Figure 2.2 shows an example of a well–structured graph–driven $\oplus$BP1. Note that the nodes v_4 and v_5 cannot be merged, since for each level function level(v_4) and level(v_5) have to differ. This is the reason why making a graph–driven $\oplus$BP1 well–structured can increase the number of nodes.

2.2 Upper bounds and previous results

We show that well-structured graph–driven $\oplus$BP1s strictly generalize parity-OBDDs as well as BP1s using results of [Sie99]. There, lower bounds for the functions MSA_n and INDEX-EQ are proved that we define in the following.

The function $\mathrm{MSA}_n \in \mathbb{B}_n$, $n = 2^k$, is defined on $x = (x_0, \ldots, x_{n-1})$. The variables are partitioned into k $s \times s$ matrices M_0, ..., M_{k-1}, where $s = \sqrt{n/k}$, and the set of remaining variables. The matrix M_i consists of the entries $x_{is^2}, \ldots, x_{(i+1)s^2-1}$. Let $a_i = 1$ iff M_i contains a row consisting of ones only, and let $|a|$ be the natural number canonically represented by a. Then

$$\mathrm{MSA}_n(x) = \begin{cases} x_0 & \text{if } |a| = 0; \\ x_{|a|} \oplus x_0 & \text{if } |a| > 0. \end{cases}$$

The function INDEX-EQ is defined on $n = 3N/2$ variables, where $N = 2^k$ and k is a power of 2. The variables $x_0, \ldots, x_{N-1}$ are interpreted as a memory and the variables $x_N, \ldots, x_{n-1}$ are interpreted as $N/(2 \log N)$ pointers each consisting of $\log N$ bits. Let $m = N/(4 \log N)$, and let $a(1), \ldots, a(m)$, $b(1), \ldots, b(m)$ denote the values of these pointers. Then INDEX-EQ$\big(x_0, \ldots, x_{n-1}\big)$ takes value 1 if and only if the following conditions are satisfied.

$$\begin{gathered} \forall i \in \{1, \ldots, m\} : \; x_{a(i)} = x_{b(i)}; \\ a(1) < \cdots < a(m) \;\text{ and }\; b(1) < \cdots < b(m); \\ a(m) < b(1) \;\text{ or }\; b(m) < a(1). \end{gathered}$$

The following result is proved by Sieling in [Sie99].

Theorem 2.2 *The function* MSA *has polynomial size ⊕OBDDs but only exponential size BP1s, whereas the function* INDEX-EQ *has polynomial size BP1s but only exponential size ⊕OBDDs.*

A ⊕OBDD is guided by an ordering for which the variables are tested in the same order for every input and a deterministic BP1 can be considered as guided by itself. In both cases the restriction being well–structured does not influence the sizes. So Theorem 2.2 immediately implies that well–structured graph–driven ⊕BP1s strictly generalize ⊕OBDDs as well as BP1s.

In [SS00] exponential lower bounds for pointer functions on the size of $(\oplus, k)$–BPs are proved. A $(\oplus, k)$–BP is a read–once BP with the source being the only nondeterministic node, where k denotes the fan–out of the source.

In [Bol00] the first exponential lower bound on the size of restricted ⊕BP1s with an unbounded number of nondeterministic branching nodes is proven. More precisely, graph–driven ⊕BP1s guided by a *tree ordering* are under consideration. A graph ordering G on $\{x_1, \ldots, x_n\}$ is said to be a tree ordering, if G becomes a tree of size $n^{\mathcal{O}(1)}$ by eliminating the sink and replacing multi–edges between nodes by simple edges. (The graph ordering shown in Figure 2.1 is in fact such a tree ordering.)

Theorem 2.3 *Let G be a tree ordering, and let $\mathcal{B}$ be a graph–driven ⊕BP1 guided by G that represents the middle bit of the integer multiplication. Then $\mathcal{B}$ has size $2^{\Omega(n/\log n)}$.*

The property being tree–driven is a very strong restriction and Theorem 2.3 is proved by methods that are very similar to the ⊕OBDD case. In fact, it is argued as follows. In a polynomial tree–ordering exists a path from the source to the sink, where for $\Omega(n - \log n)$ nodes the 0−successor is equal to the 1−successor. One has to set the variables of the $O(\log n)$ disturbing nodes to appropriately chosen constants and then gets a ⊕OBDD representing a subfunction of the function represented by the whole diagram.

Applying this technique, one can prove that any tree–driven ⊕BP1 representing $\text{INDEX-EQ}^{\vee}_{n^2}(\mathfrak{x}_1, \ldots, \mathfrak{x}_n) := \bigvee_{i=1}^{n} \text{INDEX-EQ}(\mathfrak{x}_i)$, where the $\mathfrak{x}_i$ are Boolean vectors of length n, has superpolynomial size. But there are BP1s of polynomial size representing $\text{INDEX-EQ}^{\vee}_{n^2}$. Let us define the function $\text{ite}_{n_1+n_2+1}(z, \text{INDEX-EQ}^{\vee}_{n_1}, \text{MSA}_{n_2})$ as follows. If $z = 1$, then the function value equals $\text{INDEX-EQ}^{\vee}_{n_1}(x_1, \ldots, x_{n_1})$, else $\text{MSA}_{n_2}(y_1, \ldots, y_{n_2})$. It is plain that $\text{ite}_{n_1+n_2+1}(z, \text{INDEX-EQ}^{\vee}_{n_1}, \text{MSA}_{n_2})$ has neither a polynomially bounded BP1 nor a polynomially bounded tree–driven ⊕BP1 representation. But it can be represented by well–structured graph–driven ⊕BP1 of polynomial size.

So it is easy to show that well–structured graph–driven ⊕BP1s strictly generalize tree–driven ⊕BP1s, but it is still an open problem, whether tree–driven ⊕BP1s are stronger than ⊕OBDDs.

2.3 Minimal well–structured graph–driven ⊕BP1s

In [BHW01] the size of well-structured graph-driven ⊕BP1s is exactly described as an algebraic invariant of the represented function f and the ordering graph. Our lower bound criterion in Section 2.5 makes use of these results. To present them, we need the following notation.

Let u be any node of the graph ordering. We call a node v a *descendant* of u (short: $v \in \text{Desc}(u)$), if any path from u to the target node passes through v. For a set of Boolean functions $A \subset \mathbb{B}_n$ we denote by $\text{span}_{\mathbb{F}_2} A$ the linear space spanned by these functions. We define $\mathbb{B}_u(f) = \text{span}_{\mathbb{F}_2} \bigcup_{v \in \text{Desc}(u)} \{f|_{\alpha(\pi)} ; \pi \text{ is a path in } G \text{ from } s \text{ to } v\}$, where $\alpha(\pi)$ is the partial assignment associated with the path π. A node v is u's *immediate descendant*, if $v \in \text{Desc}(u)$ and $w \in \text{Desc}(u)$ implies $w \in \text{Desc}(v)$ or $w = v$. By $\mathcal{N}_v := \text{level}^{-1}(v)$ we denote the nodes in $\mathcal{B}$ obtained from a node v of G by the level mapping.

Theorem 2.4 *Let $\mathcal{B}$ be a size–minimal well–structured graph–driven ⊕BP1 on the variables $\{x_1, \ldots, x_n\}$ guided by G representing $f \in \mathbb{B}_n$. Let u be any branching node of G,*

and let v be its immediate descendant. Then

$$\#\mathcal{N}_u = \dim_{\mathbb{F}_2} \mathbb{B}_u(f) \; - \; \dim_{\mathbb{F}_2} \mathbb{B}_v(f).$$

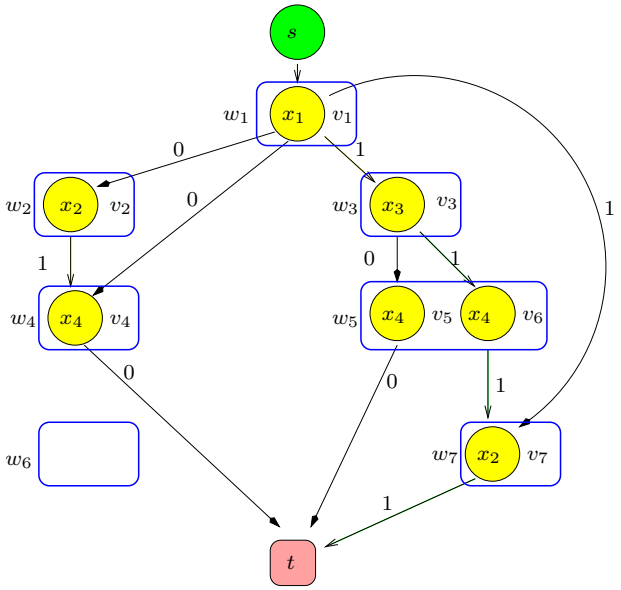

Figure 2.3: The level–structure of the well–structured graph–driven ⊕BP1 presented in Figure 2.2.

2.4 Well–structured graph–driven ⊕BP1s as a data structure

Based on the algebraic characterization presented in Theorem 2.4, in [BHW01] a minimization algorithm is described. This algorithm consists of two phases. One achieves that the subfunctions represented by nodes of $\mathcal{N}_v$ are linearly independent. The other one transforms the input in such a way that all these subfunctions associated with $\mathcal{N}_v$ are contained in $\mathcal{B}_v$.

Let us define *a feasible exponent ω of matrix multiplication* over a field k to be a real number such that multiplication of two square matrices of order h may be algorithmically achieved with $\mathcal{O}(h^\omega)$ arithmetical operations. Up to now, the best known ω is 2.376 (see

[CW90]). For practical reasons it might be best to use Gaussian elimination. Then we work with the feasible matrix exponent 3.

Theorem 2.5 *Let ω be any feasible exponent of matrix multiplication. Let $\mathcal{B}$ be a well–structured graph–driven $\oplus$BP1 guided by a fixed graph ordering G. Then there is an algorithm that computes a size–minimal one guided by the same ordering G representing the same Boolean function as $\mathcal{B}$ in time $\mathcal{O}(\mathrm{SIZE}(G) \cdot \mathrm{SIZE}(\mathcal{B})^{\omega})$ and space $\mathcal{O}(\mathrm{SIZE}(G) + \mathrm{SIZE}(\mathcal{B})^2)$.*

It is possible to test two well–structured graph–driven $\oplus$BP1s for equivalence with the help of the algorithm described in the preceding Theorem. Let $\mathcal{B}'$ and $\mathcal{B}''$ be two well–structured graph–driven $\oplus$BP1s on $\{x_1, \ldots, x_n\}$ guided by G. Using standard techniques, for example the well–known *product construction*, one can easily perform the Boolean synthesis operations in time $\mathcal{O}(\mathrm{SIZE}(G) \cdot (\mathrm{SIZE}(\mathcal{B}') \cdot \mathrm{SIZE}(\mathcal{B}''))^{\omega})$. So we can state the following.

Corollary 2.6 *It can be decided in time $\mathcal{O}(\mathrm{SIZE}(G) \cdot (\mathrm{SIZE}(\mathcal{B}') \cdot \mathrm{SIZE}(\mathcal{B}''))^{\omega})$ whether or not $\mathcal{B}'$ and $\mathcal{B}''$ represent the same function.*

2.5 A lower bound criterion

Using the algebraic characterization stated in Theorem 2.4, in [BHW01] an exponential lower bound for certain linear code functions is proved. In Section 3.4 we generalize this result for general graph–driven $\oplus$BP1s. In this section we apply Theorem 2.4 to derive a lower bound criterion that reduces the technical effort of lower bound proofs for well–structured graph–driven $\oplus$BP1s.

A function f is *called essentially* dependent on the variable x_i, if different settings to this variable result in different subfunctions, i.e. $f_{x_i=0} \neq f_{x_i=1}$. We call a set S of functions *linearly independent with respect to x_i* if each nontrivial linear combination of elements in S essentially depends on x_i.

Lemma 2.7 *Let $\mathcal{B}$ be a well-structured graph–driven $\oplus$BP1 guided by a graph ordering G representing the boolean function f. For all nodes $v \in G$ let A_v be a set of partial assignments that lead in G to v. If for all v the subfunctions $\{f|_\alpha \,;\, \alpha \in A_v\}$ are linearly independent with respect to the variable tested in v, then the size of $\mathcal{B}$ is bounded below by $\sum_{v \in G} |A_v|$.*

Proof. Let the conditions of the lemma be fulfilled. We define $B(A_v)$ to be the set $\{f|_\alpha \,;\, \alpha \in A_v\}$. By the definitions in Section 2.3 we get that $B(A_v) \subseteq \mathbb{B}_v(f)$. Now

we get $\dim_{\mathbb{F}_2} \mathbb{B}_v(f) \geq |A_v|$, since $B(A_v)$ is a linearly independent set. For linear spaces B_1, B_2 we denote by B_1/B_2 the factor space generated by $\{f \oplus B_2 \,;\, f \in B_1\}$. We get $\dim_{\mathbb{F}_2}(\mathbb{B}_v(f)/\mathbb{B}_w(f)) = \dim_{\mathbb{F}_2} \mathbb{B}_v(f) - \dim_{\mathbb{F}_2} \mathbb{B}_w(f) \geq |A_v|$, since $B(A_v)$ is linearly independent with respect to the variable tested in v. By Theorem 2.4 we conclude that $\#\mathcal{N}_v \geq |A_v|$ for all v of G and the claim follows, since by definition, the different sets $\mathcal{N}_v$ are mutually disjoint. □

2.6 A lower bound on permutation matrices

A $n \times n$ matrix over $\{0,1\}$ is called a permutation matrix, if each row and each column contains exactly one 1. The well-known function PERM_n depending on n^2 Boolean variables accepts exactly those inputs corresponding to permutation matrices. In this section we adopt ideas from [Kra88] and [KMW91].

Theorem 2.8 *Each well-structured graph–driven ⊕BP1 representing $PERM_n$ has size bounded below by $\Omega\left(n^{-1/2}2^n\right)$.*

Proof. Let $\mathcal{B}$ be a graph–driven ⊕BP1 guided by the graph ordering G that represents $f = \mathrm{PERM}_n$. According to the lemma we examine sets A_v of partial assignments that lead in G to a node v.

We consider the $n!$ inputs that correspond to permutation matrices and the corresponding paths in the graph ordering and truncate these paths after exactly $n/2$ variables have been tested 1. If such a truncated path leads from the source to a node v, we define the corresponding partial assignment α to be a member of A_v^*. Note, that for two different nodes v and w the sets A_v^* and A_w^* are disjoint. For $\alpha \in A_v^*$ let $R(\alpha)$ be the set of row indices for which by α a variable is tested 1. Analogously, let $C(\alpha)$ be the indices of columns that according to α already contain a 1. By construction $|C(\alpha)| = |R(\alpha)| = n/2$ for each $\alpha \in A_v^*$.

Now for each node v of G we choose $A_v \subseteq A_v^*$ such that for $\alpha \neq \beta$ in A_v it holds that $R(\alpha) \neq R(\beta)$ or $C(\alpha) \neq C(\beta)$. We can estimate the possible size of these sets A_v, using ideas from [Kra88]. For a fixed α there are $(n/2)!$ bijections from $R(\alpha)$ to $C(\alpha)$ and furthermore $\frac{n}{2}!$ bijections from $\{1,\ldots,n\} \setminus R(\alpha)$ to $\{1,\ldots,n\} \setminus C(\alpha)$. Consequently only $\frac{n}{2}! \cdot \frac{n}{2}!$ permutations have the same sets C and R and it follows that it is possible to choose the sets A_v such that $\sum |A_v| \geq \frac{n!}{(n/2)! \cdot (n/2)!}$.

Now we prove that for each v in the ordering the set $\{f|_\alpha \,;\, \alpha \in A_v\}$ is linear independent with respect to the variable tested in v. Then by Lemma 2.7 we get that the

number of nodes of $\mathcal{B}$ is bounded below by $\sum |A_v|$ and the claim follows with Stirling's formula. Let a set A_v consist of the elements $\alpha_1, \ldots, \alpha_k$. Then there are assignments $\alpha'_1, \ldots, \alpha'_k$ such that (α_i, α'_i) is a permutation matrix for $i = 1, \ldots, k$. But on the other hand for $i \neq j$ the assignment (α_i, α'_j) gets rejected, since by definition $R(\alpha_i) \neq R(\alpha_j)$ or $C(\alpha_i) \neq C(\alpha_j)$. Let moreover, α^*_i be obtained from α'_i by switching the variable tested in v. Then in α^*_i either $n/2-1$ or $n/2+1$ variables are tested 1 and so for $i, j \leq k$ the input (α_i, α^*_j) is not a permutation matrix. Let $I \subseteq A_v$ and $\alpha \in I$. Then the linear combination $\sum_{\beta \in I} f|_\beta$ takes value 1 for α' and 0 for α^* and the claim follows. □

2.7 The function $\mathrm{ROW}_n + \mathrm{COL}_n$

The next lower bound we present concerns the function $\mathrm{ROW}_n + \mathrm{COL}_n$ that we have already mentioned in Section 1.4 of the introduction. We consider this function because it is well–examined (see [BW97]) and closely related to PERM_n.

The input is a $n \times n$ matrix X over $\{0, 1\}$. We define $\mathrm{ROW}_n(X) = 1$, if X possesses one row constisting entirely of ones and, analogously, $\mathrm{COL}_n(X) = 1$, if X possesses one column constisting entirely of ones. Now $\mathrm{ROW}_n(X) + \mathrm{COL}_n(X) = 1$ if $\mathrm{ROW}_n(X) = 1$ or $\mathrm{COL}_n(X) = 1$.

We like to derive a lower bound using an observation due to Sauerhoff, stated in [Weg00], page 140. We denote by E_{n,n^2} the function outputting 1 if and only if the number of 1s of X equals n and by $\overline{X}$ the matrix resulting from switching each entry of X. Then it is easy to see that

$$\mathrm{PERM}_n(X) = \big(1 \oplus (\mathrm{ROW}_n(\overline{X}) + \mathrm{COL}_n(\overline{X})) \big) \wedge \mathrm{E}_{n,n^2}.$$

But first we have to examine how to complete a well–structured graph–driven $\oplus$BP1. Recall that a BP1 is called *complete* if on each path each variable is tested exactly once.

Lemma 2.9 *Let $\mathcal{B}$ be a well–structured graph–driven $\oplus$BP1 guided by a graph–ordering G. Then there exists a complete well–structured graph–driven $\oplus$BP1 $\mathcal{B}'$ guided by G representing the same function such that* $\mathrm{SIZE}(\mathcal{B}') \leq \mathrm{SIZE}(\mathcal{B}) \cdot \mathrm{SIZE}(G)$.

Proof. The claim follows with the well–known product construction that is also used for Boolean synthesis (see [Weg00], for instance). □

Now we can prove the following corollary.

Corollary 2.10 *Each graph–driven ⊕BP1 representing $ROW_n + COL_n$ has size bounded below by $\Omega\left(2^{n/2}\right)$.*

Proof. Let $\mathcal{B}$ be a well–structured graph–driven ⊕BP1 guided by G representing the function $\mathrm{ROW}_n(X) + \mathrm{COL}_n(X)$ with size s. Then one can transform $\mathcal{B}$ into a G–guided graph–driven ⊕BP1 of the same size representing $g = 1 \oplus (\mathrm{ROW}_n(\overline{X}) + \mathrm{COL}_n(\overline{X}))$. Applying Lemma 2.9 we get a complete G–guided graph–driven ⊕BP1 $\mathcal{B}'$ for g of size bounded above by $s \cdot \mathrm{SIZE}(G)$. ¿From $\mathcal{B}'$ we can construct a well–structured G–driven ⊕BP1 $\mathcal{B}''$ representing $g \wedge \mathrm{E}_{n,n^2}$ with size $\mathrm{SIZE}(\mathcal{B}'') \leq \mathrm{SIZE}(\mathcal{B}') \cdot (n+1)$. To this end, we observe that it is sufficient to store for each node v of $\mathcal{B}'$ the information, whether up to v, $0, 1, \ldots, n-1$ or n variables have been tested 1. Moreover, paths on that more than n variables are tested 1 are not permitted to lead into the sink. This is possible by multiplying each node v not more than n times. We can achieve that for $\mathcal{B}''$ the following holds. For an input a with $|a| = n$, in $\mathcal{B}''$ the same number of paths reach the sink as in $\mathcal{B}'$ and for an input a' with $|a'| \neq n$, in $\mathcal{B}''$ no path reaches the sink.

Thus, we get that $\mathrm{PERM}_n = g \wedge \mathrm{E}_{n,n^2}$ is representable by a well–structured graph–driven ⊕BP1 of size $s \cdot \mathrm{SIZE}(G) \cdot (n+1)$. In [BWW02] the following is shown. Given a polynomial–sized well–structured graph–driven ⊕BP1 guided by an ordering. Then there is an ordering G such that $\mathcal{B}$ is guided by G with $\mathrm{SIZE}(G) \leq 2 \cdot n \cdot \mathrm{SIZE}(\mathcal{B})$ and the condition being well–structured is fulfilled. Since by Theorem 2.8 we know that each well–structured graph–driven ⊕BP1 representing PERM_n has size bounded below by $\Omega(2^n)$, we get for s, the size of a well–structured graph–driven ⊕BP1 representing $\mathrm{ROW}_n(X) + \mathrm{COL}_n(X)$,

$$s^2 \cdot 2n(n+1) = \Omega(2^n)$$

and the claim follows. □

2.8 General ⊕BP1s are stronger than well–structured ones

In this section we prove that the computational power of general ⊕BP1s is strictly larger than that of well–structured graph–driven ⊕BP1s. This result is generalized in Section 3.6. Furthermore, the separating function used there is very similar to the function considered in this section. Nevertheless, there are important differences that are examined at the beginning of Section 3.6. graph–driven ⊕BP1s.

We consider the following functions on matrices of n^2 Boolean variables. $\mathbb{1}_{\mathrm{C}}^n$ takes value 1 if each column contains exactly one entry 1. $\mathbb{1}_{\mathrm{R}}^{n-2}$ takes value 1 if $n-2$ rows contain exactly one entry 1 and the remaining two rows consist entirely of 0s.

$$\mathbb{1}_{\mathrm{C}}^n = \begin{cases} 1 & \text{if each column of } X \text{ contains exactly one 1;} \\ 0 & \text{otherwise.} \end{cases}$$

$$\mathbb{1}_{\mathrm{R}}^{n-2} = \begin{cases} 1 & \text{if } n-2 \text{ rows of } X \text{ contain exactly one 1} \\ & \text{and two rows are all–0'ones;} \\ 0 & \text{otherwise.} \end{cases}$$

In the following we consider the disjunction of these two functions, $f = \mathbb{1}_{\mathrm{C}}^n \vee \mathbb{1}_{\mathrm{R}}^{n-2}$. First we construct a ⊕BP1 that represents $f = \mathbb{1}_{\mathrm{C}}^n \vee \mathbb{1}_{\mathrm{R}}^{n-2}$ succinctly. It is easy to construct an OBDD with size bounded by $\mathcal{O}(n^2)$ testing the variables in a columnwise manner and representing $\mathbb{1}_{\mathrm{C}}^n$. In the same way we get a linear sized OBDD that tests the variables in a rowwise manner and represents $\mathbb{1}_{\mathrm{R}}^{n-2}$. Joining the sources of these two OBDDs we get a ⊕BP1 of linear size that represents $\mathbb{1}_{\mathrm{C}}^n \vee \mathbb{1}_{\mathrm{R}}^{n-2}$. Next we prove a superpolynomial lower bound for $f = \mathbb{1}_{\mathrm{C}}^n \vee \mathbb{1}_{\mathrm{R}}^{n-2}$. With the same method we can achieve a lower bound for the more natural function $\mathbb{1}_{\mathrm{C}}^n \vee \mathbb{1}_{\mathrm{R}}^n$ with

$$\mathbb{1}_{\mathrm{R}}^n = \begin{cases} 1 & \text{if each row of } X \text{ contains exactly one 1;} \\ 0 & \text{otherwise.} \end{cases}$$

We consider $\mathbb{1}_{\mathrm{C}}^n \vee \mathbb{1}_{\mathrm{R}}^{n-2}$, since for this function it is easy to derive the upper bound for ⊕BP1s presented above.

Theorem 2.11 *Each well-structured graph–driven ⊕BP1 representing* $\mathbb{1}_{C}^n \vee \mathbb{1}_{R}^{n-2}$ *has size bounded below by* $\Omega\left(n^{-9/4}2^{n/2}\right)$.

Proof. Let $\mathcal{B}$ be a well-structured graph–driven ⊕BP1 with graph ordering G that represents $f = \mathbb{1}_{\mathrm{C}}^n \vee \mathbb{1}_{\mathrm{R}}^{n-2}$. The definition of the sets of indices C and R is exactly the same as in the proof of Theorem 2.8. Let the variable $x_{i,j}$ correspond to the entry (i,j). Throughout this proof we denote by z the variable tested in the node v under consideration. For each node v of G we let the set A_v consist of partial assignments that lead in the graph ordering to v such that the following holds. In each element α of A_v

- exactly $n/2$ variables are tested 1,

- for $\alpha \neq \beta$ in A_v it holds that $C(\alpha) \neq C(\beta)$ or $|R(\alpha) \cap R(\beta)| \leq (n/2) - 3$ and
- there is an assignment α' to the variables not set by α, such that z is set to 1 and (α, α') is a permutation matrix.

We prove that we can choose these sets such that

$$\sum |A_v| \geq \frac{n!}{(n^2 \cdot \frac{n}{2}!)^2}.$$

We consider the $n!$ assignments according to permutation matrices and the corresponding paths in the ordering. We truncate these paths after $n/2$ variables have been tested 1 and the next variable to test would be tested 1, too (in other words, we truncate the paths just before the $(n/2+1)$th variable is tested 1). For $R, C \subseteq \{1, \ldots, n\}$ with $|R| = n/2$ there are less than n^4 possiblities to choose a set $R' \subseteq \{1, \ldots, n\}$ with $|R \cap R'| \geq (n/2) - 2$. Combining for these sets R' the $(n/2)!$ bijections from R' to C and the $(n/2)!$ bijections from $\{1, \ldots, n\} \setminus R'$ to $\{1, \ldots, n\} \setminus C$ we get all inputs a with $C(a) = C$ and $|R(a) \cap R| \leq (n/2) - 2$ and see that its number is less than $(n^2 \cdot \frac{n}{2}!)^2$. So the lower bound on $\sum |A_v|$ follows.

Without loss of generality we suppose that n is even and not less than 8. We now prove that for each v in the ordering the set $\{f|_\alpha \,;\, \alpha \in A_v\}$ consists of at least $\lceil |A_v|^{1/2} \rceil$ elements that are linearly independent with respect to z. Then by Lemma 2.7 it follows that the number of nodes of $\mathcal{B}$ is bounded below by $\sum \lceil |A_v|^{1/2} \rceil \geq \frac{(n!)^{1/2}}{n^2 \cdot (n/2)!}$ and the claim follows with Stirling's formula.

Let v be a node of the ordering. We partition A_v into disjoint sets $S_1, \ldots, S_\nu$ consisting of all $\alpha \in A_v$ that are mapped to the same column set $C(\alpha)$. It follows that either (1) $\nu \geq \lceil |A_v|^{1/2} \rceil$ or (2) there is a set S_i such that $|S_i| \geq \lceil |A_v|^{1/2} \rceil$.

In case (1) we choose one element from every set and show that the associated subfunctions are linearly independent with respect to z. For a linear combination $f|_{\alpha_1} + \ldots + f|_{\alpha_m}$ we get by choice of A_v that there is an assignment α_1' such that (α_1, α_1') is a permutation matrix and thus $f|_{\alpha_1}(\alpha_1') = 1$. We get that $f|_{\alpha_i}(\alpha_1') = 0$ for $i > 1$, since by choice $C(\alpha_1) \neq C(\alpha_i)$. In addition, $f|_{\alpha_i}(\alpha_j^*) = 0$ for all $i, j \leq m$ if α_j^* results from α_j' by switching z, since the number of ones in (α_i, α_j^*) equals $n - 1$ and neither $\mathbb{1}_{\mathrm{C}}^n$ nor $\mathbb{1}_{\mathrm{R}}^{n-2}$ can take value 1.

In case (2) we choose the elements of the set S_i with size greater than or equal to $\lceil |A_v|^{1/2} \rceil$. From the assignment α_j' used in case (1) we construct an assignment α_j'' such that $\mathbb{1}_{\mathrm{R}}^{n-2}(\alpha_j, \alpha_j'') = 1$. To do so we take two variables that are tested 1 and are not equal to z and switch them to 0. Now $f(\alpha_j, \alpha_j'') = 1$, but for $i \neq j$ the matrix (α_i, α_j'') contains

three rows without a single entry 1, since by choice $R(\alpha_i)$ differs from $R(\alpha_j)$ in at least three indices. As in the first case we get that each linear combination essentially depends on z, since switching z reduces the number of 1s of an arbitrary assignment (α_i, α_j'') to $n-3$. □

2.9 Summary and further results

Figure 2.4 illustrates the impact of the lower bounds presented in this chapter. For the notation $\mathcal{P}(X)$ we refer to Section 1.3. An arrow denotes containment, a crossed one proper containment. ψ denotes the function $\mathrm{ite}_{n_1+n_2+1}\big(z, \text{INDEX-EQ}^{\vee}_{n_1}, \mathrm{MSA}_{n_2}\big)$ defined in Section 2.2. The relations (1) and (2) are well–known. Consider, for instance, the *hidden weighted bit function* $\mathrm{HWB}_n(x_1, \ldots, x_n) = x_{sum}$ for $sum = x_1 + \ldots + x_n$, where $+$ denotes integer addition, and $x_0 = 0$. In [Bry91], Bryant has observed that the OBDD size of HWB_n is $\Omega\left(2^{n/5}\right)$. It is not difficult to find succinct representations of HWB_n by BP1s and ⊕OBDDs.

To see that PERM_n is contained in $\mathcal{P}(\oplus\,\mathrm{BP})$ we observe that Wigdersons result $\mathrm{NL/\,Poly} \subseteq \oplus\mathrm{L/\,Poly}$ (see Section 1.6 and [Wig94]) implies that each function representable by polynomial size nondeterministic BP1s is in $\mathcal{P}(\oplus\,\mathrm{BP})$. Since $\overline{\mathrm{PERM}_n}$ can be represented by polynomial size nondeterministic OBDDs, see Section 1.5, and $\mathcal{P}(\oplus\,\mathrm{BP})$ is closed under complement, we conclude that $\mathrm{PERM}_n \in \mathcal{P}(\oplus\,\mathrm{BP})$.

The results presented in [BHW01] inspired further research on well–structured graph–driven ⊕BP1s. In [BWW02] the first strong exponential lower bound for integer multiplication is proved, using a criterion that follows from the algebraic characterization in [BHW01]. Moreover, in [BW02] it is proved that the computational power of general parity BPs is strictly stronger than that of well–structured graph–driven ⊕BP1s. Our approach to prove the same result in Section 2.8 is a little less technically involved. Some authors consider well–structured graph–driven nondeterministic BP1s. But lower bounds for nondeterministic BP1s without the restriction being well–structured graph–driven are known for a long time, see Section 1.5.

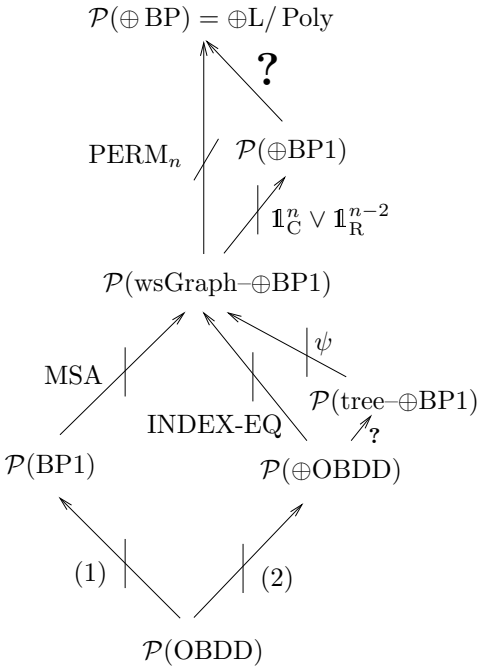

Figure 2.4: The results of this chapter.

Chapter 3

General Graph–Driven $\oplus$BP1s

Outline of this chapter. In this chapter we consider graph–driven $\oplus$BP1s without the well–structuredness restriction (see Definition 2.1). In Section 3.1 we characterize all $\oplus$BP1s for that we can construct a graph–ordering guiding them. In Section 3.2 we briefly consider the algorithmic properties of graph–driven $\oplus$BP1s and compare them to well–structured ones. In Section 3.3 we prove a lower bound criterion for graph–driven $\oplus$BP1s that we apply to linear codes in Section 3.4, in Section 3.10 on permutation matrices and in Section 3.6 for proving that the computational power of unrestricted $\oplus$BP1s is strictly higher than that of graph–driven $\oplus$BP1s. In Section 3.7 we derive further lower bounds, namely for the determinant, for the Directed Hamiltonian cycle problem and for integer multiplication.

The main results of this chapter are published in [BHW03].

3.1 Characterizing general graph–driven $\oplus$–BP1s

At the first glance it seems that the property being graph–driven is quite restricting. The next Proposition shows that this is not true. Informally spoken, graph–driven $\oplus$BP1s represent the same class of functions succinctly as $\oplus$BP1s where for each input a the variable are tested in the same order on all computation paths associated with a.

Proposition 3.1 *Let $\mathcal{B}$ be a $\oplus$–BP1 on the set of variables $\{x_1, x_2, \ldots, x_n\}$. Then $\mathcal{B}$ is graph–driven (i.e. there exists a graph–ordering G such that $\mathcal{B}$ is guided by G) if and only if the following condition is satisfied. For each input a there is an ordering $\sigma(a)$ of $\{x_1, x_2, \ldots, x_n\}$ such that on each computation path for a the bits of a are queried according to $\sigma(a)$.*

Proof. It is clear that the condition is necessary.

Assume now that the condition is fulfilled for a ⊕–BP1 $\mathcal{B}$. We show that we can choose a variable x_i such that for each input a the variables can be tested according to an ordering that starts with x_i. Then the graph–ordering G can be constructed as follows. At the very beginning we create the unlabeled source s. The unique successor u of s is labeled with x_i. Then we calculate the subdiagrams $\mathcal{B}_{x_i=0}$ ($\mathcal{B}_{x_i=1}$, resp.) by setting in $\mathcal{B}$ the variable x_i to 0 (to 1, resp.). Now for $\mathcal{B}_{x_i=b}$ $(b = 0, 1)$ there is another variable x_{j_b} that can be tested first in $\mathcal{B}_{x_i=b}$ for all inputs a with $a_i = b$. So we label the $b-$successor of u by x_{j_b} and then the procedure iterates.

We assume that for each variable x_i there is an input a_i such that all orderings compatible with a_i cannot start with x_i. Then for each x_i there is an input a_i and a computation path p_i for a_i such that a variable x_j $(j \neq i)$ is tested on p_i before x_i.

After having renamed the indices we get inputs $a_1, a_2, \ldots, a_\nu$ and computation paths $p_1, p_2, \ldots, p_\nu$ such that

- the variable x_ν is tested before x_1 on p_1;
- for $i = 2, \ldots, \nu$ the variable x_{i-1} is the first variable tested on p_i, and x_i occurs on p_i, too.

Clearly, the number ν is always greater than or equal to 2 and less than or equal to n. We call the sequence $x_\nu, x_1, \ldots, x_{\nu-1}, x_\nu$ a *cycle* with respect to the inputs $a_1, a_2, \ldots, a_\nu$ and the corresponding computation paths $p_1, p_2, \ldots, p_\nu$.

For $i = 1, \ldots, \nu$, let S_i be the set of variables tested on p_i before x_i with x_i being excluded. The number $\sum_{i=1}^{\nu} |S_i|$ is called the *weight* of the cycle.

Let us consider from now on a cycle $x_\nu, x_1, \ldots, x_{\nu-1}, x_\nu$ with respect to the inputs $a_1, a_2, \ldots, a_\nu$ and the corresponding computation paths $p_1, p_2, \ldots, p_\nu$ of minimal weight.

We observe that the minimality entails that the sets S_i $(i = 1, \ldots, \nu)$ are pairwise disjoint.

Since the sets $S_1, S_2, \ldots, S_\nu$ are pairwise disjoint, there is an input a such that for all $i = 1, \ldots, \nu$, $a\,|_{S_i} = a_i\,|_{S_i}$. Contradiction. □

3.2 Comparing general graph–driven ⊕BP1s to well–structured ones

Well–structured graph–driven ⊕BP1s differ from general ones by the additional level function, introduced in Definition 2.1. This restriction may look quite artificial at the first

glance. But without this further restriction it is not clear how to characterize size–minimal instances. Therewith, we do not know how to minimize a given instance.

Nevertheless, using the well–known *product construction* we can easily perform the Boolean synthesis operations. Note that the following proposition only holds for graph–driven ⊕BP1s guided by a common ordering, since from the lower bound presented in Section 3.6 we get Corollary 3.12 stating that synthesizing graph–driven ⊕BP1s guided by different orderings may cause an exponential blow–up of the result's size.

Proposition 3.2 *Let $\mathcal{B}'$ and $\mathcal{B}''$ be two graph–driven ⊕BP1s on $\{x_1, \ldots, x_n\}$ guided by G. One can perform the Boolean synthesis operations such that the results size is less or equal to* $\mathrm{SIZE}(G) \cdot (\mathrm{SIZE}(\mathcal{B}') \cdot \mathrm{SIZE}(\mathcal{B}''))$.

Let $\mathcal{B}'$ and $\mathcal{B}''$ be two graph–driven ⊕BP1s that are guided by a common ordering G. It is unknown if there is an efficient deterministic algorithm testing whether $\mathcal{B}'$ and $\mathcal{B}''$ represent the same function. But even for general ⊕BP1s it is possible to adopt a well–known coRP algorithm due to [BCW80]. This fact, stated in the following proposition, implies that the equivalence test for general ⊕BP1s is not coNP-complete unless RP≠NP.

Proposition 3.3 *Let $\mathcal{B}'$ and $\mathcal{B}''$ be two ⊕BP1s. There is a randomized polynomial–time algorithm that outputs "equivalent" if $\mathcal{B}'$ and $\mathcal{B}''$ represent the same function and that that outputs "not equivalent" with probability at least* $1/2$ *else.*

But how is the computational power of well–structured graph–driven ⊕BP1s compared to the computational power of general graph–driven ⊕BP1s? Deciding whether general graph–driven ⊕BP1s are stronger than well–structured ones is still an open problem. The following proposition characterizes the connection between the two models.

Proposition 3.4 *Let f be a Boolean function. Then f is representable by a polynomial size well–structured graph–driven ⊕BP1 $\mathcal{B}$ if and only if f is representable by a polynomial size graph–driven ⊕BP1 $\mathcal{B}'$ guided by a polynomial–sized ordering G.*

Proof. First we have to show that the condition is necessary. Let $\mathcal{B}'$ be guided by G given. To transform $\mathcal{B}'$ into a well–structured graph–driven ⊕BP1 one has to rebuild $\mathcal{B}'$ such that a level function as claimed in Definition 2.1 can be chosen. This is tractable by multiplying some nodes and so assigning them to different levels. Since the number of levels is less or equal to $\mathrm{SIZE}(G)$ there is a well–structured graph–driven ⊕BP1 guided by G with $\mathrm{SIZE}(\mathcal{B}) \leq \mathrm{SIZE}(\mathcal{B}') \cdot \mathrm{SIZE}(G)$.

Now assume that the condition is fulfilled. In [BWW02] the following is shown. Given a polynomial–sized well–structured graph–driven ⊕BP1 guided by an ordering. Then

there is a ordering G such that $\mathcal{B}$ is guided by G with $\mathrm{SIZE}(G) \leq 2 \cdot n \cdot \mathrm{SIZE}(\mathcal{B})$ and the condition being well–structured is fulfilled. The claim follows. □

Now we can restate the open problem deciding whether general graph–driven ⊕BP1s are stronger than well–structured ones.

Is there a function f such that

- *f is representable by a polynomial–sized graph–driven ⊕BP1 $\mathcal{B}$ with an ordering G of unrestricted size and*
- *all graph–driven ⊕BP1s $\mathcal{B}'$ representing f that are guided by a polynomial–sized ordering G' have super–polynomial size?*

3.3 A lower bound criterion for graph–driven ⊕BP1s

Let $\mathcal{B}$ be a graph–driven ⊕BP1 on the set of variables $\{x_1, x_2, \ldots, x_n\}$ guided by a graph ordering G representing the Boolean function f. We define two vector spaces over $\mathbb{F}_2$. The space $\mathbb{B}(\mathcal{B})$ is spanned by the functions Res_v, where v is a node of $\mathcal{B}$ and Res_v denotes the function represented by the subdiagram of $\mathcal{B}$ rooted at v. The second space, denoted by $\mathbb{B}_G(f)$, is the span of all subfunctions $f|_\pi$, where π is a path from the source to a node w in G and $f|_\pi$ results from f by setting the variable according to the labels of the nodes and edges on π.

We are now in the position to state a lower bound criterion, whose proof is very easy. It estimates the size of a graph–driven ⊕BP1 by an invariant of the graph ordering and the function represented.

Theorem 3.5 *Let $\mathcal{B}$ be a graph–driven ⊕BP1 guided by a graph ordering G representing the Boolean function f.*

Then

$$\mathrm{SIZE}(\mathcal{B}) \geq \dim_{\mathbb{F}_2} \mathbb{B}_G(f).$$

Proof. First we observe that $\mathrm{SIZE}(\mathcal{B}) \geq \dim_{\mathbb{F}_2} \mathbb{B}(\mathcal{B})$.

Let $f|_\pi$ be any generating element of the vector space $\mathbb{B}_G(f)$, and let α be the partial assignment to the set of variables $\{x_1, x_2, \ldots, x_n\}$ associated with the path π. Since the branching program $\mathcal{B}$ is guided by the graph ordering G, we are led to nodes v_1, v_2, …, v_ν when traversing $\mathcal{B}$ starting at the source according to the partial assignment α. Consequently, $f|_\pi = \sum_{j=1}^{\nu} \mathrm{Res}_{v_j}$. This entails that $\mathbb{B}(\mathcal{B})$ contains $\mathbb{B}_G(f)$ as a subspace. The

claim follows. □

How to apply Theorem 3.5? As already defined, for a partial assignment α to the set of variables $\{x_1, x_2, \ldots, x_n\}$, the subfunction $f|_\alpha$ results from f by setting the variables to constants according to α.

Corollary 3.6 *Let $\pi_1, \pi_2, \ldots, \pi_\nu$ be paths in G starting at the source. Let $\alpha_1, \ldots, \alpha_\nu$ be the partial assignments associated with these paths.*

If the subfunctions $f|_{\alpha_1}, \ldots, f|_{\alpha_\nu}$ are linearly independent, then

$$\mathrm{SIZE}\,(\mathcal{B}) \geq \nu.$$

Proof. The subspace of $\mathbb{B}_G(f)$ spanned by $\{f|_{\alpha_1}, \ldots, f|_{\alpha_\nu}\}$ is of dimension ν. □

3.4 A lower bound for linear codes

A *linear code* C is a linear subspace of $\mathbb{F}_2^n$. Our first explicit lower bound is for the *characteristic function* of such a linear code C, that is $f_C : \mathbb{F}_2^n \to \{0,1\}$ defined by $f_C(a) = 1 \iff a \in C$. To this end we will give some basic definitions and facts on linear codes.

The *Hamming distance* of two code words $a, b \in C$ is defined to be the number of 1's of $a \oplus b$. The *minimal distance* of a code C is the minimal Hamming distance of two distinct elements of C. The *dual* $C^\perp$ is the set of all vectors b such that $a_1 b_1 \oplus \ldots \oplus a_n b_n = 0$, for all elements $a \in C$. A set $D \subseteq \mathbb{F}_2^n$ is defined to be *k-universal*, if for any subset of k indices $I \subseteq \{1, \ldots, n\}$ the projection onto these coordinates restricted to the set D gives the whole space $\mathbb{F}_2^k$.

The next lemma is well–known. See [Juk99b] for a proof.

Lemma 3.7 *If C is a code of minimal distance $k+1$, then its dual $C^\perp$ is k–universal.*

For the proof of the next theorem we apply results of Jukna ([Juk99b]).

Theorem 3.8 *Let $C \subseteq \mathbb{F}_2^n$ be a linear code of minimal distance d whose dual $C^\perp$ has minimal distance $d^\perp$.*

Then each graph–driven $\oplus$BP1 representing its characteristic function f_C has size bounded below by $2^{(min\{d,d^\perp\}-1)}$.

Proof. Let $\mathcal{B}$ be a graph–driven ⊕BP1 guided by G representing $f = f_C$. Consider the set of all nodes of the graph ordering G at depth k from the source, where $k := \min\{d, d^\perp\} - 1$. Thus for each such node v and each path π leading from the source to v exactly k variables are tested on π. For $m := 2^k$, let $\alpha_1, \ldots, \alpha_m$ be the partial assignments of the variables $x_1, x_2, \ldots, x_n$ resulting from these paths.

Observe, that the code C is both of distance $k+1$ and k–universal.

We consider the subfunctions $f|_{\alpha_1}, f|_{\alpha_2}, \ldots, f|_{\alpha_m}$ and take notice of the fact that these functions formally depend on all variables $x_1, x_2, \ldots, x_n$. According to Corollary 3.6 it suffices to prove that $f|_{\alpha_1}, f|_{\alpha_2}, \ldots, f|_{\alpha_m}$ are linearly independent.

Let $\{f|_{\alpha_{i_1}}, f|_{\alpha_{i_2}}, \ldots, f|_{\alpha_{i_\mu}}\}$ be any nonempty subset of $\{f|_{\alpha_1}, f|_{\alpha_2}, \ldots, f|_{\alpha_m}\}$. Having assumed without loss of generality that $(\alpha_{i_1}, \alpha_{i_2}, \ldots, \alpha_{i_\mu})$ is equal to $(\alpha_1, \alpha_2, \ldots, \alpha_\mu)$, we have to show that $\bigoplus_{i=1}^{\mu} f|_{\alpha_i} \neq 0$.

For each partial assignment α_1' to the variables $\{x_1, x_2, \ldots, x_n\}$ whose domain is complementary to the domain of α_1, we can define a vector $(\alpha_1, \alpha_1') := (a_1, a_2, \ldots, a_n) \in \{0,1\}^n$ as follows.

$$a_j := \begin{cases} \alpha_1(x_j) & \text{if } \alpha_1(x_j) \text{ is defined;} \\ \alpha_1'(x_j) & \text{if } \alpha_1'(x_j) \text{ is defined} \quad (j = 1, 2, \ldots n). \end{cases}$$

Since C is k–universal, there is an α_1' such that $a := (\alpha_1, \alpha_1')$ is a member of C. Consequently $f|_{\alpha_1}(a) = 1$. For each $1 < i \leq \mu$, we show that $f|_{\alpha_i}(a) = 0$.

Obviously, $f|_{\alpha_i}(a) = f(a^{(i)})$, where

$$a_j^{(i)} := \begin{cases} \alpha_i(x_j) & \text{if } \alpha_i(x_j) \text{ is defined;} \\ a_j & \text{otherwise.} \end{cases}$$

Since the distance between a and $a^{(i)}$ is less than or equal to k, the claim follows. □

Now we are able to formulate the following corollary, that states our first super–polynomial lower bound for unrestricted graph–driven ⊕BP1s. Recall that the r–th order binary Reed–Muller code $R(r,l)$ of length $n = 2^l$ is the set of graphs of all polynomials in l variables over $\mathbb{F}_2$ of degree not more than r.

Corollary 3.9 *Let $n = 2^l$ and $r = \lfloor l/2 \rfloor$.*

Then every graph–driven ⊕BP1 representing the characteristic function of $R(r,l)$ has size bounded below by $2^{\Omega(\sqrt{n})}$.

Proof. We apply that the code $R(r,l)$ is linear and has minimal distance 2^{l-r}. It is known that the dual of $R(r,l)$ is $R(l-r-1,l)$ (see [MS77]). □

3.5 A lower bound for permutation matrices

The function PERM_n depending on n^2 Boolean variables and accepting exactly those inputs corresponding to permutation matrices has already been considered in section 2.6.

We adapt ideas used there in order to apply them to Corollary 3.6.

Theorem 3.10 *Each graph–driven $\oplus$BP1 representing $PERM_n$ has size bounded below by $\Omega\left(n^{-1/2}2^n\right)$.*

Proof. Let $\mathcal{B}$ be a graph–driven $\oplus$BP1 guided by G representing the function $f := \mathrm{PERM}_n$ depending on the variables x_{ij} $(i, j = 1, 2, \ldots, n)$.

We consider the $n!$ inputs $a = (a_{ij})_{1\leq i,j\leq n}$ that correspond to permutation matrices and the corresponding paths in the graph ordering G. Having tested exactly $n/2$ variables 1, we truncate these paths. Let

$$A_1 := \{\alpha_1, \alpha_2, \ldots, \alpha_\nu\}$$

be the partial assignments to the set of variables $\{x_{ij}\,;\, i, j = 1, 2, \ldots, n\}$ associated with these truncated paths.

For such a partial assignment α let $R(\alpha)$ be the set of row indices i such that $\alpha(x_{ij}) = 1$, for a column index j. Analogously, let $C(\alpha)$ be the set of column indices j such that $\alpha(x_{ij}) = 1$, for a row index i. Then $|C(\alpha)| = |R(\alpha)| = n/2$ by construction.

We consider sets A of the above defined partial assignments such that for distinct $\alpha, \beta \in A$ it holds that $R(\alpha) \neq R(\beta)$ or $C(\alpha) \neq C(\beta)$. We recapitulate the proof given in [Kra88] for the fact, that there is one of these subsets A such that

$$|A| \geq \frac{n!}{\left(\frac{n}{2}!\right)^2}.$$

Indeed, if we fix two subsets $C, R \subseteq \{1, 2, \ldots, n\}$ of columns and rows, where $|C| = |R|$, then there are exactly $\left(\frac{n}{2}!\right)^2$ inputs a that lead to partial assignments α such that $C(\alpha) = C$ and $R(\alpha) = R$. They result from combining the $(n/2)!$ bijections from R to C with the $(n/2)!$ bijections from $\{1, 2, \ldots, n\} \setminus R$ to $\{1, 2, \ldots, n\} \setminus C$. Since there are exactly $n!$ accepted inputs, the claim follows.

Let $A = \{\alpha_1, \alpha_2, \ldots, \alpha_m\}, m \geq n!/\left(\frac{n}{2}!\right)^2$, be a set of partial assignments the existence of which we have just proved. To prove that A fulfils the prerequisites of Corollary 3.6, we choose an arbitrary subset A' of A. We may assume without loss of generality that $A' = \{\alpha_1, \alpha_2, \ldots, \alpha_\mu\}$. We show that $\bigoplus_{i=1}^{\mu} f|_{\alpha_i} \neq 0$.

By the choice of A there is an partial assignment α_1' such that (α_1, α_1') is a permutation matrix. Thus $f|_{\alpha_1}(0,\ldots,0,\alpha_1') = 1$, where $(0,\ldots,0,\alpha_1')$ is the following matrix.

$$(0,\ldots,0,\alpha_1')_{k,\ell} := \begin{cases} \alpha_1'(x_{k,\ell}) & \text{if } \alpha_1'(x_{k,\ell}) \text{ is defined;} \\ 0 & \text{otherwise.} \end{cases}$$

We show now that $f|_{\alpha_i}(0,\ldots,0,\alpha_1') = 0$ for $i > 1$.

For the sake of deriving a contradiction, let us assume that there is an index $i > 1$ such that $f|_{\alpha_i}(0,\ldots,0,\alpha_1') = 1$. Then there is a permutation matrix consisting of only those ones set by α_i and α_1'. But this implies $R(\alpha_i) = R(\alpha_1)$ and $C(\alpha_i) = C(\alpha_1)$, because otherwise there is a row or a column of the permutation matrix a defined by

$$a_{k,\ell} := \begin{cases} \alpha_i(x_{k,\ell}) & \text{if } \alpha_i(x_{k,\ell}) \text{ is defined;} \\ (0,\ldots,0,\alpha_1')_{k,\ell} & \text{if } \alpha_i(x_{k,\ell}) \text{ is not defined.} \end{cases}$$

without any one. Contradiction.

Now the claim follows from Stirling's formula. □

3.6 Unrestricted ⊕BP1s are stronger than graph–driven ones

To get a function that has small unrestricted but requires exponential sized graph–driven ⊕BP1s we consider the following function that as PERM_n depends on a matrix of n^2 Boolean variables. It is defined by $\mathbb{1}_{\mathrm{C}}^n \vee \mathbb{1}_{\mathrm{R}}^{n-1,1}$ where

$$\mathbb{1}_{\mathrm{C}}^n = \begin{cases} 1 & \text{if each column of } X \text{ contains exactly one 1;} \\ 0 & \text{otherwise.} \end{cases}$$

$$\mathbb{1}_{\mathrm{R}}^{n-1,1} = \begin{cases} 1 & \text{if } n-1 \text{ rows of } X \text{ contain exactly one 1} \\ & \text{and one row contains exactly two 1's;} \\ 0 & \text{otherwise.} \end{cases}$$

The function $\mathbb{1}_{\mathrm{C}}^n \vee \mathbb{1}_{\mathrm{R}}^{n-1,1}$ looks very similar to the function $\mathbb{1}_{\mathrm{C}}^n \vee \mathbb{1}_{\mathrm{R}}^{n-2}$ considered in Section 2.8. Nevertheless, if one compares the proofs of Theorems 2.11 and 3.11 very carefully, the following turns out.

- For well–structured graph–driven ⊕BP1s a lower bound for $\mathbb{1}_{\mathrm{C}}^{n} \vee \mathbb{1}_{\mathrm{R}}^{n-2}$ is easier to derive, than one for $\mathbb{1}_{\mathrm{C}}^{n} \vee \mathbb{1}_{\mathrm{R}}^{n-1,1}$, since the existence of certain accepting assignments is clear.

- It is not clear how to apply Corollary 3.6 to $\mathbb{1}_{\mathrm{C}}^{n} \vee \mathbb{1}_{\mathrm{R}}^{n-2}$ and get a lower bound for general graph–driven ⊕BP1s for this function. This is why in this case we consider $\mathbb{1}_{\mathrm{C}}^{n} \vee \mathbb{1}_{\mathrm{R}}^{n-1,1}$.

We start with an upper bound for this function. It is easy to construct an OBDD of size bounded above by $\mathcal{O}(n^2)$ testing the variables in a columnwise manner and taking the value one if each column contains a single 1. In the same way we get a linear sized OBDD that tests the variables in a rowwise manner and accepts if $n-1$ rows contain a single 1 and one row contains exactly two. Joining the sources of these two OBDDs together, we get a ⊕BP1 of linear size that represents $\mathbb{1}_{\mathrm{C}}^{n} \vee \mathbb{1}_{\mathrm{R}}^{n-1,1}$.

Before proving a superpolynomial lower bound for this function we like to mention that by the same method a lower bound for the function $\mathbb{1}_{\mathrm{C}}^{n} \vee \mathbb{1}_{\mathrm{R}}^{n}$ follows, see Section 2.8. But it is not clear how to prove an upper bound for this function.

Theorem 3.11 *Each graph–driven ⊕BP1 representing* $\mathbb{1}_{C}^{n} \vee \mathbb{1}_{R}^{n-1,1}$ *has size bounded below by* $\Omega\left(n^{-1/4} \cdot 2^{n/2}\right)$.

Proof. Let $\mathcal{B}$ be a graph–driven ⊕BP1 guided by a graph ordering G that represents $f := \mathbb{1}_{\mathrm{C}}^{n} \vee \mathbb{1}_{\mathrm{R}}^{n-1,1}$ on the variables x_{ij} $(i,j = 1,2,\ldots,n)$. Without loss of generality we suppose that n is even.

As in the case of Theorem 3.10, we consider the $n!$ inputs $a = (a_{ij})_{1 \le i,j \le n}$ that correspond to permutation matrices and the corresponding paths in the graph ordering G. Having tested exactly $n/2$ variables 1, we truncate these paths. We consider the partial assignments $A_1 := \{\alpha_1, \alpha_2, \ldots, \alpha_\nu\}$ to the set of variables $\{x_{ij}\,;\, i,j = 1,2,\ldots,n\}$ associated with these truncated paths.

We observe that

$$|A_1| = \nu \le n^2 \cdot \binom{n^2 - n/2}{n/2} \le n^2 \cdot (2en)^{n/2}. \tag{3.1}$$

Indeed, there are $\binom{i+n/2}{n/2}$ paths in G starting from the source along which i variables are tested 0, and $n/2$ variables are tested 1. Permutation matrices can only follow those paths, where the number of variables tested 0 is less than or equal to $n^2 - n$. Equation 3.1 follows.

Second, without loss of generality let $A_2 := \{\alpha_1, \alpha_2, \ldots, \alpha_\mu\}$ be those elements of A_1 that can be extended to at least two permutation matrices. By the pigeon hole principle, we get

$$\nu - \mu + \mu \cdot \left(\frac{n}{2}\right)! \geq n!$$

Consequently,

$$|A_2| = \mu \geq \frac{n! - \nu}{(n/2)! - 1}.$$

Without loss of generality, let $A = \{\alpha_1, \alpha_2, \ldots, \alpha_\kappa\}$ for $\kappa \leq \nu$ be those elements of A_2 such that $C(\alpha) \neq C(\beta)$ or $R(\alpha) \neq R(\beta)$ (see the proof of Theorem 3.10 for the definitions of R and C). Since not more than $(n/2)!$ elements $\alpha \in A_2$ may have the same pair (R, C), we obtain

$$|A| = \kappa \geq \frac{n! - \nu}{(n/2)! \cdot (n/2)!}.$$

Since

$$\lim_{n\to\infty} \frac{\nu}{(n/2)! \cdot (n/2)!} = 0,$$

we get

$$|A| = \kappa \geq \frac{n!}{(n/2)! \cdot (n/2)!} - o(1).$$

Now let A_C (A_R) be a subset of A of maximal size satisfying the following property. If $\alpha, \beta \in A_C$ ($\alpha, \beta \in A_R$) are two distinct elements, then $C(\alpha) \neq C(\beta)$ ($R(\alpha) \neq R(\beta)$). It follows from an easy counting argument, that $|A_C| < \sqrt{|A|}$ implies $|A_R| \geq \sqrt{|A|}$. Thus we have the following two cases to distinguish.

Case $|A_C| \geq \sqrt{|A|}$.

Let $A'_C = \{\beta_1, \beta_2, \ldots, \beta_\nu\}$ be any subset of the set A_C. We have to show, that

$$\bigoplus_{i=1}^{\nu} f|_{\beta_i} \neq 0. \tag{3.2}$$

By the choice of A there is a partial assignment β'_1 such that (β_1, β'_1) is a permutation matrix. Thus $f|_{\beta_1}(0, \ldots, 0, \beta'_1) = 1$, where the matrix $(0, \ldots, 0, \beta'_1)$ is defined as in the proof of theorem 3.10. Moreover, we get that $f|_{\beta_i}(0, \ldots, 0, \beta'_1) = 0$ for $i > 0$. By the

definition of A_C we have that $C(\beta_i) \neq C(\beta_1)$ for $i > 0$. Thus there is a column of the matrix a defined by

$$a_{k,\ell} := \begin{cases} \beta_i(x_{k,\ell}) & \text{if } \beta_i(x_{k,\ell}) \text{ is defined;} \\ (0,\ldots,0,\beta'_1)_{k,\ell} & \text{if } \beta_i(x_{k,\ell}) \text{ is not defined,} \end{cases} \tag{3.3}$$

without any one. So we get that $f|_{\beta_i}(0,\ldots,0,\beta'_1) = 0$ for $i > 0$ and (3.2) follows.

Case $|A_R| \geq \sqrt{|A|}$.

Let $A'_R = \{\beta_1, \beta_2, \ldots, \beta_\nu\}$ be any subset of the set A_R. Again we have to show, that $\bigoplus_{i=1}^{\nu} f|_{\beta_i} \neq 0$. Each element β_i of A_R can be extended to at least two permutation matrices (β_i, β'_i) and (β_i, β''). So we can construct an assignment β_i^* such that (β_i, β_i^*) is a matrix that contains $n-1$ rows with exactly one entry 1 and one row that contains exactly two ones. Thus $f|_{\beta_1}(0,\ldots,0,\beta_1^*) = 1$. For $i > 0$ we get that $f|_{\beta_1}(0,\ldots,0,\beta_i^*) = 0$ since similar to the first case there is a row of the matrix a as defined in (3.3) without any one. So the claim follows. □

In Proposition 3.2 we have already observed that one can efficiently synthesize two graph–driven ⊕BP1s, provided they are guided by the same ordering. Now we get an corollary that this has not to be the case if there is no common ordering.

Corollary 3.12 *Given two graph–driven ⊕BP1s $\mathcal{B}$ and $\mathcal{B}'$, applying Boolean synthesis may cause an exponential blow–up of the result's size.*

Proof. It is plain how to represent $\mathbb{1}_{\mathrm{C}}^n$ and $\mathbb{1}_{\mathrm{R}}^{n-1,1}$ by two linear size graph–driven ⊕BP1s testing the variables in a columnwise and a rowwise manner. Theorem 3.11 states that each graph–driven ⊕BP1 representing the disjunction is of exponential size. □

Corollary 3.13 *Given a graph–driven ⊕BP1 $\mathcal{B}$ guided by a fixed ordering G, setting a variable to Boolean constants may cause an exponential blow–up of the result's size.*

Proof. We can argue in the same way as Sieling and Wegener in [SW95]. Consider a graph–driven ⊕BP1 $\mathcal{B}$ representing $f = s \wedge \mathbb{1}_{\mathrm{C}}^n \vee (s \oplus 1) \wedge \mathbb{1}_{\mathrm{R}}^{n-1,1}$ (see Section 3.6) guided by the following ordering G. The successor of G's source is labeled by the variable s. For $s = 1$, a columnwise ordering is used and for $s = 0$, a rowwise. It is plain that $\mathcal{B}$ can be constructed with linear size. After replacing s by 1, $\mathcal{B}$ represents $\mathbb{1}_{\mathrm{C}}^n$, both for inputs with $s = 1$ and $s = 0$. Furthermore, for inputs with $s = 0$, according to the unchanged ordering G, the variables have to be tested in a rowwise manner. It is easy to see that the

result's size is not polynomial bounded, since in the other case there would be polynomial size ⊕OBDDs representing $\mathbb{1}_{\mathrm{C}}^{n}$ and $\mathbb{1}_{\mathrm{R}}^{n-1,1}$, both of them testing the variables in a rowwise manner. By the Boolean synthesis operation we would get a polynomial size ⊕OBDD representing $\mathbb{1}_{\mathrm{C}}^{n} \vee \mathbb{1}_{\mathrm{R}}^{n-1,1}$. Contradiction to Theorem 3.11. □

3.7 Further lower bounds

The lower bounds proved so far, entail lower bounds for several other very interesting functions. We present these lower bounds in the following, all of them straight–forward consequences of Corollary 3.9 or Theorem 3.10.

3.7.1 The determinant and the Hamiltonian cycles

Testing whether a matrix represents a permutation is linked to some other important functions. If we consider a matrix of n^2 constants one may be interested in the permanent and the determinant of this matrix. In the case of Boolean constants the permanent equals the determinant that is defined as

$$\mathrm{DET}_n(X) = \bigoplus_{\sigma \in \Sigma_n} \prod_{i=1}^{n} x_{i,\sigma(i)},$$

where Σ_n denotes the group of permutations on a set with cardinality n. Using this notation we can define the characteristic function of permutation matrices as

$$\mathrm{PERM}_n(X) = \bigvee_{\sigma \in \Sigma_n} \Big(\prod_{i=1}^{n} x_{i,\sigma(i)} \wedge \prod_{j=1, j \neq \sigma(i)}^{n} \overline{x}_{i,j}\Big).$$

Another related computational problem is deciding whether an directed graph contains a directed Hamiltonian cycle. If we take an adjacency matrix as input, the function DHC_n outputs 1 if and only if the corresponding graph contains a Hamiltonian cycle, i.e.

$$\mathrm{DHC}_n(X) = \bigvee_{\substack{\sigma \in \Sigma_n, \\ \sigma \text{ has 1 cycle}}} \prod_{i=1}^{n} x_{i,\sigma(i)}.$$

First we show how to obtain lower bounds for DET_n from our lower bounds for PERM_n. This function deserves interest, since amongst other reasons it is complete

for the logarithmic space counting class #L. This was observed by several authors, for instance by Damm in [Dam91],

Corollary 3.14 *Each graph–driven $\oplus$BP1 representing* DET_n *has size bounded below by* $\Omega\left(n^{-1/2}2^n\right)$.

Proof. We describe how to transform the proof of Theorem 3.10 to get our claim. In that proof we proceeded as follows. For each graph–ordering G we can choose a set

$$A = \{\alpha_1, \ldots, \alpha_\nu\}$$

of partial assignments such that

- each α_i is associated with a path in G starting at the source, and so we get subfunctions $f|_{\alpha_1}, \ldots, f|_{\alpha_\nu}$ of $f = \mathrm{PERM}_n$ in line with Corollary 3.6,
- for each α_i there is an assignment α_i' such that (α_i, α_i') forms a permutation matrix,
- for $i \neq j$, the assignment $a^{(i,j)} = (a_{k,\ell}^{(i,j)})_{1\leq k,\ell\leq n}$ defined by

$$a_{k,\ell}^{(i,j)} := \begin{cases} \alpha_i(x_{k,\ell}) & \text{if } \alpha_i(x_{k,\ell}) \text{ is defined;} \\ (0,\ldots,0,\alpha_j')_{k,\ell} & \text{if } \alpha_i(x_{k,\ell}) \text{ is not defined,} \end{cases}$$

  is no permutation matrix (note that $a^{(i,i)} = (\alpha_i, \alpha_i')$), and
- $\nu = \Omega\left(n^{-1/2}2^n\right)$.

This entails that $f|_{\alpha_1}(0,\ldots,0,\alpha_1') = 1$ and $f|_{\alpha_i}(0,\ldots,0,\alpha_1') = 0$ for $i > 0$. Thus, the subfunctions $f|_{\alpha_1}, \ldots, f|_{\alpha_\nu}$ are linearly independent and the claim of that theorem follows with Corollary 3.6.

Now we consider $g = \mathrm{DET}_n$. Since each permutation matrix is accepted by g, in the notation introduced above we get that for each $i \in \{1,\ldots,\mu\}$ the assignment $a^{(i,i)} = (\alpha_i, \alpha_i')$ is accepted by DET_n. We observe that for $i \neq j, a^{(i,j)}$ is no permutation matrix and possesses at most n entries 1. So this assignment composed from α_i and $(0,\ldots,0,\alpha_j')$ is not accepted by DET_n. With the same argument as above the subfunctions $g|_{\alpha_1}, \ldots, g|_{\alpha_\nu}$ are linearly independent and our claim follows with Corollary 3.6. □

Next we consider DHC_n. A permutation matrix is mapped to 1 by DHC_n if it consists of exactly one cycle.

Corollary 3.15 *Each graph–driven ⊕BP1 representing* DHC_n *has size bounded below by* $\Omega\left(n^{-3/2}2^n\right)$.

Proof. There are $(n-1)!$ permuation matrices consisting of exactly one cycle. Using these matrices rather than all permutation matrices, we can proceed in the same way as in the proof of Corollary 3.14. □

3.7.2 Integer multiplication

For some basics on linear codes, see Section 3.4. In [Juk99b], Jukna proved that one specific property of linear codes - their universality - makes them difficult for ⊕OBDDs. Moreover, these functions may be quite appropriate for proving lower bounds for ⊕BP1s.

An interesting fact is that linear codes give some information about the hardness of integer multiplication. For an integer X we denote the i-th bit of its binary representation by X_i. For a subset of bits $S \subseteq \{1,\ldots,n\}$, we denote by Mult_n^S the following Boolean function on $2n$ variables. For n–bit integers X and Y, $\mathrm{Mult}_n^S(X,Y) = 1$ if and only if, $(X \cdot Y)_i = 1$ for all $i \in S$. In [Juk95a], Jukna proved the following.

Theorem 3.16 *For every linear code* $C \subseteq \{0,1\}^n$ *there is an integer* $A \in \{1,\ldots,2^\nu\}, \nu = (n+1)\cdot n \cdot \log n + n + 1$, *an injection* $\phi : \{0,1\}^n \to \{0,1\}^\nu$ *and a subset of bits* S, $|S| \leq n - \dim C$, *such that for every* $x \in \{0,1\}^n$, $x \in C$ *if and only if* $\mathrm{Mult}_\nu^S(A,\phi(x)) = 1$. *Furthermore,* x *can be got from* $\phi(x)$ *by setting some variables to constants.*

In [Juk99a], Jukna applied this Theorem to get a lower bound for ⊕OBDDs and Mult_n^S. Adopting his method we are able to prove Corollary 3.18. We only have to examine the impact of setting variables to constants on the size of a graph–driven ⊕BP1.

Proposition 3.17 *Given a graph–driven ⊕BP1* $\mathcal{B}$ *representing* f, *a variable* x_i *and a Boolean constant* e. *Then there is a graph–driven ⊕BP1* $\mathcal{B}'$ *representing* $f|_{x_i=e}$ *such that* $\mathrm{SIZE}(\mathcal{B}') \leq \mathrm{SIZE}(\mathcal{B})$.

Proof. We observe that setting a variable to a constant does not affect the existence of a variable ordering $\sigma(a)$ according to that the bits are queried on all computation paths associated with an input a. Now the claim follows with Proposition 3.1. □

Note that changing the graph–ordering is crucial as we stated in Corollary 3.13.

Corollary 3.18 *For any sufficiently large n there exists an n-bit integer A and a subset of its bits such that each graph–driven $\oplus$BP1 representing* $\mathrm{Mult}_n^S(A,X)$ *has size exponential in $n^{1/4-\epsilon}$.*

Proof. Let $\mathcal{B}$ be a graph–driven $\oplus$BP1 representing Mult_n^S. Let $C \subseteq \{0,1\}^m$ denote the Reed–Muller code $R(r,l)$, $m = 2^l$ and $r = \lfloor l/2 \rfloor$. By Corollary 3.9 we get that each graph–driven $\oplus$BP1 representing C has size $2^{\Omega\left(\sqrt{m}\right)}$. Applying Theorem 3.16 we get an n–bit integer A, $n = \mathcal{O}\left(m^2 \log m\right) = \mathcal{O}\left(m^{2+\epsilon'}\right)$, for each $\epsilon' > 0$, and a subset of its bits S such that the characteristic function of C can be obtained from $\mathrm{Mult}_n^S(A,X)$ by setting a variables to constants.

Thus with Proposition 3.17 we conclude that any graph–driven $\oplus$BP1 representing $\mathrm{Mult}_n^S(A,X)$ is of size exponential in $(n^{1/(2+\epsilon')})^{1/4}$ and the claim follows. □

3.8 Summary

Figure 3.1 complements Figure 2.4 by results due to the lower bounds proved in this section. In Section 2.9 we argued why $\mathrm{PERM}_n \in \mathcal{P}(\oplus\,\mathrm{BP})$. Observing that the negations of linear codes are computable by polynomial size nondeterministic OBDDs, we analogously get that $\mathrm{LinearCodes} \in \mathcal{P}(\oplus\,\mathrm{BP})$.

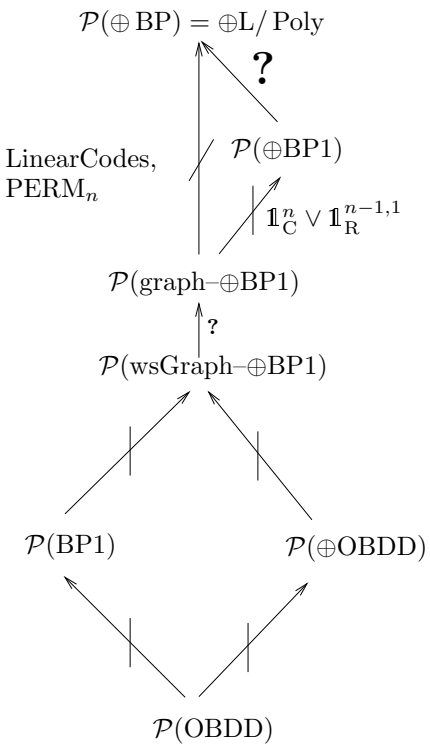

Figure 3.1: The results of this chapter.

Chapter 4

On sums of graph–driven $\oplus$BP1s

Outline of this chapter. In the last chapter we proved lower bounds for graph–driven $\oplus$BP1s. It turned out in Section 3.1 that a $\oplus$BP1 is graph–driven if and only if for each input there is a variable ordering that is compatible with each computation path for the input. In Section 4.1 we generalize this model in such a way that for each input there may be several variable orderings; each computation path must be compatible with some of these orderings. We introduce this model in Definition 4.1 and later on examine its connection to graph–driven $\oplus$BP1s. After that we consider a more restricted variant, namely sums of graph–driven $\oplus$BP1s. We show that sums of graph–driven $\oplus$BP1s guided by polynomial size graph–orderings have a strictly larger computational power than well–structured graph–driven $\oplus$BP1s as well as $(\oplus, k)$-BPs, examined by Savický and Sieling in [SS00].

In Section 4.2 we prove a lower bound criterion for sums of graph–driven $\oplus$BP1s. This criterion is applied in Section 4.3 in order to derive lower bounds for linear codes.

The results of this chapter are published in the Technical Report [Hom03a].

4.1 Motivating sums of graph–driven $\oplus$BP1s

In this chapter we consider a variant of $\oplus$BP1s that strictly generalizes graph–driven $\oplus$BP1s. One can hope that lower bounds for this model mean an important step towards lower bounds for read–once $\oplus$BP1s.

Definition 4.1 *Let k be any positive integer. A k–$\oplus$BP1 is a $\oplus$BP1 with the following additional restriction. For each input a there are not more than k variable orderings*

$\sigma_1(a), \ldots, \sigma_k(a)$ such that on each computation path for a the bits of a are queried according to $\sigma_i(a)$ for an i, $1 \leq i \leq k$.

This restriction appears to be quite natural. In fact, we have already considered the case $k = 1$, since Proposition 3.1 entails that for each 1–⊕BP1 a graph–ordering exists. So we can identify 1–⊕BP1s with graph–driven ⊕BP1s.

Proposition 4.2 *It holds that $\mathcal{P}$(graph–⊕BP1)= $\mathcal{P}$(1–⊕BP1).*

Next we observe that in terms of computational power 2–⊕BP1s strictly generalize graph–driven ⊕BP1s. In Section 3.6 it has been proved that each graph–driven ⊕BP1 representing the function $\mathbb{1}_{\mathrm{C}}^{n} \vee \mathbb{1}_{\mathrm{R}}^{n-1,1}$ has exponential size, where

$$\mathbb{1}_{\mathrm{C}}^{n} = \begin{cases} 1 & \text{if each column of } X \text{ contains exactly one 1;} \\ 0 & \text{otherwise.} \end{cases}$$

$$\mathbb{1}_{\mathrm{R}}^{n-1,1} = \begin{cases} 1 & \text{if } n-1 \text{ rows of } X \text{ contain exactly one 1} \\ & \text{and one row contains two 1s;} \\ 0 & \text{otherwise.} \end{cases}$$

Moreover, a ⊕BP1 is constructed which represents this function succinctly. That ⊕BP1 is in fact a 2–⊕BP1, since it is constructed by joining two OBDDs to a source of fanout 2. One OBDD tests the variables in a rowwise and the other one in a columnwise manner. We state this observation as

Corollary 4.3 *$\mathcal{P}$(1–⊕BP1) is a proper subset of $\mathcal{P}$(2–⊕BP1).*

Is it possible to prove an analogon to Proposition 3.1 - where for each $\mathcal{P}$(1–⊕BP1) a graph–ordering is constructed - for $\mathcal{P}(k$–⊕BP1) with $k \geq 2$? A direct generalization of that proposition could be formulated as follows. *Let $\mathcal{B}$ be a k–⊕BP1. Then there are k (deterministic) graph–orderings $G_1, \ldots, G_k$ with the following property. Let $a \in \{0,1\}^n$ be an input and let π be an arbitrary computation path for a. Then there is an index $i, 1 \leq i \leq k$, such that on π the variables are tested in the same ordering as in G_i (corresponding to a).*

This statement does not hold. To see this, we consider the example given in Figure 4.1. It is due to Brosenne, [Bro03]. This ⊕BP1, representing the function $(x_1 + 1) \cdot (x_2 + 1) + x_1 \cdot x_2 + (x_3 + 1) \cdot (x_1 + 1)$ is a 2–⊕BP1. But it is not possible to choose two variables y_1, y_2 such that for each input a on each corresponding computation path the first variable tested *may be* y_1 or y_2, resp. Note that this would be a necessary condition of the statement above.

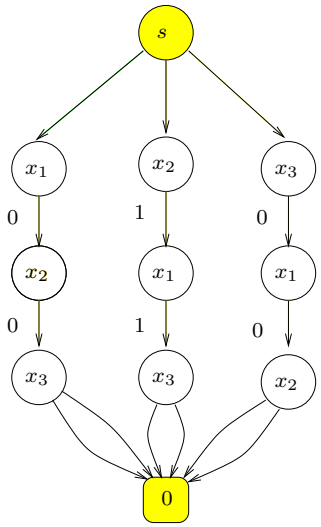

Figure 4.1: A 2–⊕BP1.

Definition 4.4 *A ⊕BP1 $\mathcal{B}$ is a* sum of k graph–driven ⊕BP1s *driven by a sequence of graph–orderings $G = (G_1, \ldots, G_k)$, if $\mathcal{B}$ consists of k disjoint ⊕BP1s $\mathcal{B}_1, \ldots, \mathcal{B}_k$ joined to a common source such that for each i, $\mathcal{B}_i$ is guided by G_i.*

*We call such a $\mathcal{B}$ a G–*driven sum of graph–driven ⊕BP1s*.*

In the next section we present a lower bound method for sums of graph–driven ⊕BP1s in order to prove lower bounds for this model with the additional restriction that the orderings have polynomial size, i.e. $|G| = |G_1| + \ldots + |G_k| = n^{\mathcal{O}(1)}$. The following propositions state the connection between sums of graph–driven ⊕BP1s driven by an ordering G of polynomial size and well–structured graph–driven ⊕BP1s. Both of them are direct consequences of Proposition 3.4. By $\mathcal{P}(k^*\text{–}\oplus\text{BP1})$ we denote all functions representable by polynomial size sums of graph–driven ⊕BP1s guided by a sequence of graph–orderings $G = (G_1, \ldots, G_k)$ with $|G| = n^{\mathcal{O}(1)}$.

Proposition 4.5 *It holds that $\mathcal{P}(wsGraph\text{–}\oplus BP1) = \mathcal{P}(1^*\text{–}\oplus BP1)$.*

For the next proposition we have to observe the following. The 2–⊕BP1 for $\mathbb{1}_{\mathrm{C}}^{n} \vee \mathbb{1}_{\mathrm{R}}^{n-1,1}$ constructed in the context of Corollary 4.3, is guided by two graph–orderings of polynomial size, since it is constructed by joining two OBDDs.

Proposition 4.6 *$\mathcal{P}(wsGraph\text{–}\oplus BP1)$ is a proper subset of $\mathcal{P}(2^*\text{–}\oplus BP1)$.*

We conclude that the notion of a $(G_1, \ldots, G_k)$–driven sum of graph–driven ⊕BP1s with polynomial size graph-orderings is a natural restriction.

In [SS00], Savický and Sieling proved exponential lower bounds for pointer functions on the size of $(\oplus, k)$–BP1s. A $(\oplus, k)$–BP1 is a read–once BP with the source being the only nondeterministic node, where k denotes the fan–out of the source.

We prove that our model strictly generalizes $(\oplus, k)$–BPs. By $\mathcal{P}((\oplus, k)\text{–BP1})$ we denote the set of functions representable by polynomial size $(\oplus, k)$–BPs. First we observe that each $(\oplus, k)$–BP $\mathcal{B}$ can be considered as a sum of k graph–driven ⊕BP1s guided by itself. So we can construct a sequence of k graph–orderings driving $\mathcal{B}$ of the same size as $\mathcal{B}$, and conclude $\mathcal{P}((\oplus, k)\text{–BP1}) \subseteq \mathcal{P}(k^*\text{–}\oplus\text{BP1})$. To see that this containment is proper consider the following functions f_n^k that are examined in [SS00]. These functions are defined on the variables $X = \{x_0, \ldots, x_{n-1}\}$. The set X is partitioned in $k(k+1)$ blocks $B_{i,j}, 1 \leq i \leq k+1, 1 \leq j \leq k$, and if necessary, some remaining variables. Each block $B_{i,j}$ consists of $\log n$ subblocks of size

$$s = \lfloor \frac{n}{k(k+1)\log m} \rfloor.$$

For our purposes we consider only the blocks $B_{1,1}, \ldots, B_{1,k}$. Each $B_{1,j}$ computes a binary representation of a pointer $p(j)$. Each of the $\log n$ bits of $p(j)$ is determined by the majority of the s bits in one of the $\log n$ subblocks of the block $B_{i,j}$. $f_n^k(x)$ outputs 1 if and only if all bits addressed by the pointers equal 1, i.e.

$$x_{p(1)} = x_{p(2)} = \ldots = x_{p(k)} = 1.$$

In [SS00] it is proved that f_n^k has no representation by polynomial size $(\oplus, k)$–BPs for $k \leq (1/2 - \gamma)\log n$ for any $\gamma > 0$. In the following we show that f_n^k can be represented by ⊕OBDDs of size $\mathcal{O}\left(n^{k+2}\right)$. The following algorithm computes f_n^k.

1. Guess the binary representation of the pointers $p_1, \ldots, p_k$.
2. Verify this choice and check, whether $x_{p(1)} = x_{p(2)} = \ldots = x_{p(k)} = 1$.

We illustrate step 2 for a certain guess. We test the variables according to an ordering, such that for each subblock of some $B_{1,j}$ all s variables are tested successively. If we read a bit x_i that is adressed by one of the guessed pointers the computation stops, or 0 is the output. Since each majority vote can be accomplished by $\mathcal{O}(s^2)$ nodes, step 2 describes an OBDD of size $\mathcal{O}(n^2)$. All n^k OBDDs of this kind can be constructed with a common ordering and in fact the algorithm stated as steps 1 and 2 describes a ⊕OBDD for f_n^k, since for each input step 2 accepts it if and only if the pointers are correct and all adressed bits equal 1.

Proposition 4.7 *For $k \leq (2/3)\log^{1/2} n$ it holds that $\mathcal{P}((\oplus, k)$–BP1$)$ is a proper subset of $\mathcal{P}(k^*$–$\oplus$BP1$)$.*

Proof. For k constant the claim follows immediately by the construction of the ⊕OBDD of size $\mathcal{O}\left(n^{k+2}\right)$ presented above. For nonconstant k we are able to apply the same padding arguments that are used in [SS00] to prove Theorem 15 of that paper. □

4.2 A lower bound criterion for sums of graph–driven ⊕BP1s

In more restricted models like deterministic BP1s, ⊕OBDDs or graph–driven ⊕BP1s (1–⊕BP1s, resp.) the nodes or sets of nodes reached by certain partial assignments represent subfunctions of the function represented by the whole diagram. This is not the case for sums of graph–driven ⊕BP1s, but certainly there is *some* connection between the functions represented by the nodes and the function represented by the whole diagram. We recall the following definition from Section 3.3.

Let $\mathcal{B}$ be a ⊕BP1 driven by a graph–ordering G. By $\mathbb{B}_G(f)$ we denote the span of all subfunctions $f|_\pi$, where π is a path from the source to a node w in G and $f|_\pi$ results from f by setting the variable according to the labels of the nodes and edges on π. Let $\mathcal{B}$ be a sum of k graph–driven ⊕BP1s $\mathcal{B}_1, \ldots, \mathcal{B}_k$. Then $\mathrm{Res}(\mathcal{B}) = \mathrm{Res}(\mathcal{B}_1) + \ldots + \mathrm{Res}(\mathcal{B}_k)$. We consider the direct sum of spaces $\mathbb{B}_{G_1}(g^1) + \ldots + \mathbb{B}_{G_k}(g^k)$ for functions $g^1, \ldots, g^k$ with $g^1 + \ldots + g^k = f$.

Lemma 4.8 *Let $\mathcal{B} = (\mathcal{B}_1, \ldots, \mathcal{B}_k)$ be a $(G_1, \ldots, G_k)$–driven sum of ⊕BP1s representing f. Then there are functions $g^1, \ldots, g^k$ with $f = g^1 + \ldots + g^k$ such that*

$$\mathrm{SIZE}(\mathcal{B}) \geq \dim_{\mathbb{F}_2}\left(\mathbb{B}_{G_1}(g^1) + \ldots + \mathbb{B}_{G_k}(g^k)\right).$$

Proof. We define $\mathbb{B}(\mathcal{B}) = \mathrm{span}_{\mathbb{F}_2}\{\mathrm{Res}_v\,;\, v \in \mathcal{B}\}$. Observe that $\mathrm{SIZE}(\mathcal{B}) \geq \dim_{\mathbb{F}_2}\mathbb{B}(\mathcal{B})$. For $\mathcal{B} = (\mathcal{B}_1, \ldots, \mathcal{B}_k)$ we set $g^1 = \mathrm{Res}(\mathcal{B}_1), \ldots, g^k = \mathrm{Res}(\mathcal{B}_k)$ and prove that $\mathbb{B}_{G_1}(g^1) + \ldots + \mathbb{B}_{G_k}(g^k) \subseteq \mathbb{B}(\mathcal{B})$. Then the claim follows, since $g^1 + \ldots + g^k = \mathrm{Res}(\mathcal{B}) = f$.

Let $g^i|_\pi$ be any generating element of the vector space $\mathbb{B}_{G_i}(g^i)$ for some $i = 1, \ldots, k$, and let α be the partial assignment to the set of variables $\{x_1, x_2, \ldots, x_n\}$ associated with the path π in G_i. Since the branching program $\mathcal{B}_i$ is guided by the graph ordering G_i, we are led to nodes v_1, v_2, …, v_ν when traversing $\mathcal{B}_i$ starting at the source according to the partial assignment α. Consequently, $g^i|_\pi = \sum_{j=1}^{\nu} \mathrm{Res}_{v_j}$, and so every generating element

of $\mathbb{B}_{G_1}(g^1) + \ldots + \mathbb{B}_{G_k}(g^k)$ is contained in $\mathbb{B}(\mathcal{B})$. The claim follows. □

In order to apply this lemma as a lower bound criterion, we have to examine the spaces $\mathbb{B}_{G_1}(g^1)+\ldots+\mathbb{B}_{G_k}(g^k)$ for all decompositions $f = g^1+\ldots+g^k$ of f. For a special case this is done in Lemma 4.10. To describe the setting of that lemma, we need further notation.

We examine how to combine several partial assignments. For partial assignments $\alpha_1, \ldots, \alpha_\nu$ with pairwise disjoint domains $\mathrm{V}(\alpha_i), i = 1, \ldots, n$, we denote by $(\alpha_1, \ldots, \alpha_\nu)$ the assignment α defined on $\mathrm{V}(\alpha_1) \cup \ldots \cup \mathrm{V}(\alpha_\nu)$ as

$$\alpha(x_j) := \begin{cases} \alpha_1(x_j) & \text{if } \alpha_1(x_j) \text{ is defined;} \\ \quad\vdots & \quad\vdots \\ \alpha_\nu(x_j) & \text{if } \alpha_\nu(x_j) \text{ is defined.} \end{cases}$$

If the domains $\mathrm{V}(\alpha_i), i = 1, \ldots, n$ are not pairwise disjoint, we require that for all $1 \leq i, j \leq \nu$ and for all $x_k \in \mathrm{V}(\alpha_i) \cap \mathrm{V}(\alpha_j)$, the assignments to x_k are equal for α_i and for α_j, i.e. $\alpha_i(x_k) = \alpha_j(x_k)$. In this case the notion $\alpha = (\alpha_1, \ldots, \alpha_\nu)$ as defined above is well–defined. Clearly, $\mathrm{V}(\alpha_1, \ldots, \alpha_\nu) = \bigcup_{i=1}^{\nu} \mathrm{V}(\alpha_i)$. By $\overline{\mathrm{V}(\alpha)}$ we denote the complement $\{x_1, \ldots, x_n\} \setminus \mathrm{V}(\alpha)$.

Let $v = (v_1, \ldots, v_k)$ be in $G_1 \times \ldots \times G_k$. We denote by $V(v_i)$ the variables that are tested in G_i on a path from the source to v_i, excluding the variable tested in v_i. Let $\alpha_1, \ldots, \alpha_k$ be partial assignments such that α_i corresponds to a path from the source of G_i to v_i.

Definition 4.9 *Given a sequence of graph–orderings $G_1, \ldots, G_k$ and $v = (v_1, \ldots, v_k) \in G_1 \times \ldots \times G_k$, we call a tuple $(\alpha_1, \ldots, \alpha_k)$ of partial assignments a* v-assignment, *if*

- *for $1 \leq i \leq k$, α_i corresponds to the path from the source of G_i to v_i, and*
- *for $1 \leq i, j \leq k$, α_i coincides with α_j on $V(v_i) \cap V(v_j)$, i.e., $\alpha_i(x) = \alpha_j(x)$ for all x in $V(v_i) \cap V(v_j)$.*

We consider a v-assignment $\alpha = (\alpha_1, \ldots, \alpha_k)$ as an assignment defined on $\mathrm{V}(\alpha_1) \cup \ldots \cup \mathrm{V}(\alpha_k)$. An easy way of getting v-assignments is truncating the k paths in $G_1, \ldots, G_k$ for an input $a \in \{0,1\}^n$ simultanously.

Lemma 4.10 *Let $\mathcal{B}$ be a $(G_1, \ldots, G_k)$–driven sum of graph–driven ⊕BP1s representing f and let v be in $G_1 \times \ldots \times G_k$. For $i = 1, \ldots, k$ let A_i be some set of assignments to $\mathrm{V}(v_i)$,*

such that each α in $A_1 \times \ldots \times A_k$ is a v-assignment. Moreover, for all $\alpha \in A_1 \times \ldots \times A_k$ let there be an assignment δ defined on the variables not set by α with

$$f(\alpha, \delta) = 1, \text{ and, } f(\alpha', \delta) = 0,$$

for each $\alpha' \in A_1 \times \ldots \times A_k \setminus \{\alpha\}$.

Then $\mathrm{SIZE}(\mathcal{B}) \geq \min\{|A_i|\,;\, i = 1, \ldots, k\}$.

Proof. Since the proof for an arbitrary k is a straightforward but technically rather involved generalization of the case $k = 2$, we begin with the latter. We wish to apply Lemma 4.8 and, to this end, we prove that for each pair of functions g^1, g^2 with $g^1 + g^2 = f$ the dimension of the space $\mathbb{B}_+ = \mathbb{B}_{G_1}(g^1) + \mathbb{B}_{G_2}(g^2)$ has a dimension greater than or equal to $\min\{|A_1|, |A_2|\}$. To derive a contradiction we assume the opposite. Since $\{g^1|_\alpha\,;\, \alpha \in A_1\} \subseteq \mathbb{B}_+$ and $\{g^2|_\beta\,;\, \beta \in A_2\} \subseteq \mathbb{B}_+$, the assumption $\dim_{\mathbb{F}_2}\mathbb{B}_+ < \min\{|A_1|, |A_2|\}$ implies for assignments $\alpha \in A_1, \beta \in A_2$ linear dependencies that we can state (after renumbering the indices) as

$$\begin{aligned} g^1|_\alpha &= g^1|_{\alpha_1} + \ldots + g^1|_{\alpha_\mu}, \text{ and} \\ g^2|_\beta &= g^2|_{\beta_1} + \ldots + g^2|_{\beta_\nu}, \end{aligned} \tag{4.1}$$

with $\mu, \nu \geq 0, \alpha_i \in A_1 \setminus \{\alpha\}$ for $1 \leq i \leq \nu$ and $\beta_j \in A_2 \setminus \{\beta\}$ for $1 \leq j \leq \mu$. Since the setting of this lemma postulates some δ such that $f(\alpha, \beta, \delta) = 1$, we get that

$$g^1|_\alpha(\alpha, \beta, \delta) + g^2|_\beta(\alpha, \beta, \delta) = f(\alpha, \beta, \delta) = 1. \tag{4.2}$$

Note that in (4.2) the function $g^1|_\alpha + g^2|_\beta$ may essentially depend on all variables on those the function f is defined. Thus, for convenience we consider such a subfunction $g^1|_\alpha$ as formally depending on all those variables.

From (4.2) we derive a contradiction in four steps. First we apply the linear dependencies (4.1) and get

$$1 = \bigoplus_{i=1}^{\mu} g^1|_{\alpha_i}(\alpha_i, \beta, \delta) + \bigoplus_{j=1}^{\nu} g^2|_{\beta_j}(\alpha, \beta_j, \delta). \tag{4.3}$$

Since $g^1|_{\alpha_i}(\alpha_i, \beta, \delta) + g^2|_\beta(\alpha_i, \beta, \delta) = f(\alpha_i, \beta, \delta) = 0$ and $g^1|_\alpha(\alpha, \beta_j, \delta) + g^2|_{\beta_j}(\alpha, \beta_j, \delta) = f(\alpha, \beta_j, \delta) = 0$, we conclude that

$$1 = \bigoplus_{i=1}^{\mu} g^2|_\beta(\alpha_i, \beta, \delta) + \bigoplus_{j=1}^{\nu} g^1|_\alpha(\alpha, \beta_j, \delta). \tag{4.4}$$

Again, we apply the linear dependencies (4.1). Consequently,

$$1 = \bigoplus_{i=1}^{\mu}\bigoplus_{j=1}^{\nu} g^2|_{\beta_j}(\alpha_i,\beta_j,\delta) + \bigoplus_{i=1}^{\mu}\bigoplus_{j=1}^{\nu} g^1|_{\alpha_i}(\alpha_i,\beta_j,\delta) \tag{4.5}$$

$$= \bigoplus_{i=1}^{\mu}\bigoplus_{j=1}^{\nu} f(\alpha_i,\beta_j,\delta) = 0. \tag{4.6}$$

Contradiction.

Now we consider the case $k > 2$. For applying Lemma 4.8, we have to prove that for each choice of k functions $g^1,\ldots,g^k$ with $g^1+\ldots+g^k = f$ the dimension of the space $\mathbb{B}_+ = \mathbb{B}_{G_1}(g^1)+\ldots+\mathbb{B}_{G_k}(g^k)$ has a dimension greater or equal to $\min\{|A_i|\,;\, i=1,\ldots,k\}$. To derive a contradiction we assume the opposite. For all $i=1,\ldots,k$, $\{g^i|_{\alpha^i}\,;\,\alpha^i\in A_i\}\subseteq \mathbb{B}_+$. So, $\dim_{\mathbb{F}_2}\mathbb{B}_+ < \min\{|A_i|\,;\,i=1,\ldots,k\}$ implies for some $\alpha_0^i\in A_i, i=1,\ldots,k$ (after renumbering) the validity of the following linear equations.

$$g^i|_{\alpha_0^i} = g^i|_{\alpha_1^i}+\ldots+g^i|_{\alpha_{\mu(i)}^i}, \tag{4.7}$$

with $\alpha_j^i\in A_i\setminus\{\alpha_0^i\}$ for $j>0$ and $i=1,\ldots,k$. Furthermore, by the setting of this lemma there is an assignment $\delta=\delta(\alpha_0^1,\ldots,\alpha_0^k)$ such that for $\alpha\in A_1\times\ldots\times A_k$

$$f(\alpha,\delta)=1, \text{if and only if,} \alpha=(\alpha_0^1,\ldots,\alpha_0^k).$$

Consequently, for $\alpha=(\alpha_{j(1)}^1,\ldots,\alpha_{j(k)}^k)$

$$g^1|_{\alpha_{j(1)}^1}(\alpha,\delta)+\ldots+g^k|_{\alpha_{j(k)}^k}(\alpha,\delta)=1, \tag{4.8}$$

if and only if $j(1)=\ldots=j(k)=0$.

Since during the proof we have to deal with a huge number of summands, we express them by sets Σ of elements in $\{1,\ldots,k\}\times\{0,1\}^k$. The significance of this definition is described by the following interpretation $\phi:\{1,\ldots,k\}\times\{0,1\}^k\to\mathbb{B}_n$.

For convenience we identify $(i,b_1,\ldots,b_k)$ and $(i,(b_1,\ldots,b_k))$. We consider a $\sigma=(i,b)$ with $i\in\{1,\ldots,k\}$ and $b=(b_1,\ldots,b_k)\in\{0,1\}^k$. From b we derive k sets of indices $I_1(b),\ldots,I_k(b),I_j(b)\subseteq\{0,\ldots,\mu(j)\}$ according to (4.7), by defining

$$I_j(b) := \begin{cases}\{0\} & \text{if } b_j=0;\\ \{1,\ldots,\mu(j)\} & \text{if } b_j=1,\end{cases}$$

for $j = 1 \dots k$. Informally, $b_j = 0$ corresponds to the left side of equation (4.7) and $b_j = 1$ to the right side. Now we set

$$\phi(i,b) = \bigoplus_{(j(1),\dots,j(k)) \in I_1(b) \times \dots \times I_k(b)} g^i|_{\alpha^i_{j(i)}}(\alpha^1_{j(1)}, \dots, \alpha^k_{j(k)}, \delta). \tag{4.9}$$

So, informally, the i in $\sigma = (i,b)$ determines the index of the function g^i. Making use of this notation, for some set Σ of such elements, we define

$$\phi(\Sigma) = \bigoplus_{\sigma \in \Sigma} \phi(\sigma).$$

In the end of this proof, we have restated the case $k = 2$ in terms of this notation. The reader may now already refer to that. Now we consider two rules (R1) and (R2), associated with the identities (4.7) and (4.8).

(R1) While Σ contains an element (i,b) with $b_i = 0$,

- remove (i,b) from Σ,
- add (i,b') to Σ, where b' results from b by skipping bit i.

(R2) For each $b \in \{0,1\}^k$ consider $S(b) = \Sigma \cap \{(1,b), \dots, (k,b)\}$. For $b \neq (0,\dots,0)$ and $S(b) \neq \emptyset$, remove all elements in $S(b)$ from Σ and add all elements in $\overline{S(b)} = \{(i,b)\,;\, i = 1, \dots, k\} \setminus S(b)$.

Informally, (R1) expresses an application of the linear dependencies (4.7). (R2) expresses an application of (4.8) with $j(\nu) \neq 0$ for some $\nu \in \{1, \dots, k\}$.

Correctness of (R1). We show that, if Σ' is derived from Σ by applying rule (R1), then $\phi(\Sigma') = \phi(\Sigma)$. We just observe that in the notation of (R1)'s description, some $\phi(i,b)$ with $b_i = 0$ consists of a sum of terms of the form $g^i|_{\alpha^i_0}(a,\delta)$, with $a \in A_1 \times \dots \times A_k$ and $a(x) = \alpha^i_0(x)$ for α^i_0 is defined on x. This is the case, since in the setting of (4.9) we have $I_i(b) = \{0\}$. Applying (4.7) on each of these summands we get $\phi(i,b) = \phi(i,b')$.

Correctness of (R2). We observe that

$$\begin{aligned}
&\phi(S(b) \cup \overline{S(b)}) \\
&= \sum_{i=1,\dots,k} \phi(i,b) \\
&= \bigoplus_{(j(1),\dots,j(k)) \in I_1(b) \times \dots \times I_k(b)} f(\alpha^1_{j(1)}, \dots, \alpha^k_{j(k)}, \delta) \\
&= 0,
\end{aligned}$$

for $b \neq 0$ by (4.8). So $\phi(S(b)) = \phi(\overline{S(b)})$, for $b \neq (0, \ldots, 0)$, and the correctness of (R2) follows.

The contradiction. Next we show that one obtains by alternating applications of (R1) and (R2) for

$$\Sigma_0 = \{(1, 0, \ldots, 0), \ldots, (k, 0, \ldots, 0)\}$$

via

$$\Sigma_0 \overset{R1}{\to} \Sigma_1 \overset{R2}{\to} \Sigma_2 \overset{R1}{\to} \Sigma_3 \overset{R2}{\to} \Sigma_4 \overset{R1}{\to} \ldots \overset{R1}{\to} \Sigma_{2k-1} \overset{R2}{\to} \Sigma_{2k},$$

the set $\Sigma_{2k} = \emptyset$. Then we get the desired contradiction

$$1 = f(\alpha_0^1, \ldots, \alpha_0^k, \delta) = \phi(\Sigma_0) = \phi(\Sigma_{2k}) = \phi(\emptyset) = 0.$$

Let for any Boolean vector b, $|b|$ denote the number of bits b_i being 1. We show that Σ_{2i} consists of all elements (j, b) such that

- $b \in \{0,1\}^k$ with $|b| = i$ and $b_j = 0$.

Note that then Σ_{2k} is indeed empty. For Σ_0 the claim holds by definition. Let us assume that for Σ_{2i} the claim holds. Then we get by rule (R1) that Σ_{2i+1} consists of all (j, b) such that

- $|b| = i + 1$ and $b_j = 1$.

By applying rule (R2) the stated situation is achieved immediately. Note that in neither of the two cases an element is produced twice, since otherwise the conclusion $\phi(\Sigma_i) = \phi(\Sigma_{i+1})$ would not be true.

Now putting all parts of this proof together the claim of this lemma follows. To illustrate this proof we finally restate the case $k = 2$ in its terminology. For $\Sigma_0 = \{(1,0,0),(2,0,0)\}$ we get $\phi(\Sigma_0) = 1$ in line with (4.2). We get $\Sigma_1 = \{(1,1,0),(2,0,1)\}$ corresponding to (4.3) and $\Sigma_2 = \{(2,1,0),(1,0,1)\}$ corresponding to (4.4). Applying rule (R1) we get $\Sigma_3 = \{(2,1,1),(1,1,1)\}$ in line with (4.5) and by rule (R2) we get $\Sigma_4 = \emptyset$, corresponding to (4.6) □

The next lemma deals with the situation that in the setting of Lemma 4.10 for two nodes v_i and v_j with $i \neq j$ the same sets of variables are tested, i.e. $\mathrm{V}(v_i) = \mathrm{V}(v_j)$. Then the condition that each α in $A_1 \times \ldots \times A_k$ is a v-assignment implies that $|A_i| = |A_j| = 1$. But in that situation for every $\tilde{\alpha}$ defined on $\mathrm{V}(v_i) = \mathrm{V}(v_j)$ we can combine those nodes reached according to G_1 and those reached according to G_2 to one set. This is possible, since the assignments in A_i and A_j can not be combined independently. In each v-assignment$(\alpha_1, \ldots, \alpha_k)$ we have $\alpha_i = \alpha_j$.

Lemma 4.11 *Let $\mathcal{B}$ be a $(G_1, \ldots, G_k)$–driven sum of ⊕BP1s representing f and let v be in $G_1 \times \ldots \times G_k$. For $i = 1, \ldots, k$ let A_i be a set of assignments to $\mathrm{V}(v_i)$, such that $A_i = A_j$ for $\mathrm{V}(v_i) = \mathrm{V}(v_j), 1 \leq i \leq j \leq k$. Let $\mathcal{A}$ be a subset of $A_1 \times \ldots \times A_k$ such that for $\alpha \in \mathcal{A}$ it holds that $\alpha_i = \alpha_j$ if $\mathrm{V}(v_i) = \mathrm{V}(v_j)$, $1 \leq i \leq j \leq k$.*

We assume that each $\alpha \in \mathcal{A}$ is a v-assignment and that for all $\alpha \in \mathcal{A}$ there is an assignment δ, defined on the variables not set by α with

$$f(\alpha, \delta) = 1, \text{ and, } f(\alpha', \delta) = 0,$$

for each $\alpha' \in \mathcal{A} \setminus \{\alpha\}$.

Then $\mathrm{SIZE}(\mathcal{B}) \geq \min\{|A_i| \,;\, i = 1, \ldots, k\}$.

Proof. First we recapitulate the preceding arguments in a slightly modified way. Let $\mathcal{B}$ be a $(G_1, \ldots, G_k)$–driven sum of ⊕BP1s representing f and let v be in $G_1 \times \ldots \times G_k$. For $i = 1, \ldots, k$ let A_i be a set of assignments to $\mathrm{V}(v_i)$, such that each $\alpha \in (\alpha_1, \ldots, \alpha_k)$ is a v-assignment. Then we get with the proof of Lemma 4.10 that $\dim_{\mathbb{F}_2}\big(\mathrm{span}_{\mathbb{F}_2}\{g^1|_{\alpha^1} \,;\, \alpha^1 \in A_1\} + \ldots + \mathrm{span}_{\mathbb{F}_2}\{g^k|_{\alpha^k} \,;\, \alpha^k \in A_k\}\big) \geq \min\{|A_i| \,;\, i = 1, \ldots, k\}$. Together with the proof of Lemma 4.8 we get the following. There are functions $g^1, \ldots, g^k$ with $g^1 + \ldots + g^k = f$ such that $\mathrm{SIZE}(\mathcal{B}) \geq \dim_{\mathbb{F}_2}\big(\mathrm{span}_{\mathbb{F}_2}\{g^1|_{\alpha^1} \,;\, \alpha^1 \in A_1\} + \ldots + \mathrm{span}_{\mathbb{F}_2}\{g^k|_{\alpha^k} \,;\, \alpha^k \in A_k\}\big)$.

Now we turn to the proof of this lemma and assume that $A_1 = A_2$ and that for each $\tilde{\alpha} \in A_1$ and each $(\alpha_3, \ldots, \alpha_k) \in A_3 \times \ldots \times A_k$ the sequence $(\tilde{\alpha}, \tilde{\alpha}, \alpha_3, \ldots, \alpha_k)$ is a v-assignment. We show that in this situation the claim of the lemma holds and the claim on the general setting follows by repeating this argument.

Considering the proof of Lemma 4.8 it is easy to see that there are functions $g^2, \ldots, g^k$ with $g^2 + \ldots + g^k = f$ such that

$$\mathrm{SIZE}(\mathcal{B}) \geq \dim_{\mathbb{F}_2}\big(\mathrm{span}_{\mathbb{F}_2}\{g^2|_{\alpha^2} \,;\, \alpha^2 \in A_2\} + \ldots + \mathrm{span}_{\mathbb{F}_2}\{g^k|_{\alpha^k} \,;\, \alpha^k \in A_k\}\big).$$

Now we can apply Lemma 4.10 and get that $\dim_{\mathbb{F}_2}\big(\mathrm{span}_{\mathbb{F}_2}\{g^2|_{\alpha^2} \,;\, \alpha^2 \in A_2\} + \ldots + \mathrm{span}_{\mathbb{F}_2}\{g^k|_{\alpha^k} \,;\, \alpha^k \in A_k\}\big) \geq \min\{|A_i| \,;\, i = 2, \ldots, k\}$. The claim follows. □

In the next proposition we state the observation that we are able to set some of the variables on that a sum of graph–driven ⊕BP1s is defined to constants without a blow–up of the size. This may be considered to be plain, but it can be necessary to change the ordering, since this is the case for graph–driven ⊕BP1s, see Corollary 3.13.

Lemma 4.12 *Let $\mathcal{B}$ be a $(G_1, \ldots, G_k)$–driven sum of ⊕BP1s in the variables $\{x_1, \ldots, x_n\}$ representing f. Then for a variable x_i and a Boolean constant e there is a sum of*

graph–driven ⊕BP1s $\mathcal{B}'$ in the variables $\{x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n\}$ that is guided by orderings $(G'_1, \ldots, G'_k)$ representing $f_{x_i=e}$ with $\mathrm{SIZE}(\mathcal{B}') \leq \mathrm{SIZE}(\mathcal{B})$.

Furthermore, for every v-assignment α with $\alpha(x_i) = e$ provided α is defined on x_i, there is a $v' \in G'_1 \times \ldots \times G'_k$ such that α is a v'-assignment.

Proof. The standard method to set x_i to e is the following. For all x_i-nodes v redirect all edges reaching v to the e-successor of v. Observe that applying this method results in a sum of graph–driven ⊕BP1s representing $f_{x_i=e}$. In the same way we get from $G = (G_1, \ldots, G_k)$ a sequence of read–once BPs $G' = (G'_1, \ldots, G'_k)$ on the variables $\{x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n\}$. To see that $\mathcal{B}'$ is driven by G' consider an assignment a to $\{x_1, \ldots, x_n\}$ and observe that if in G_λ, $\lambda \in \{1, \ldots, k\}$, the variable x_ν is tested before x_μ, $\nu, \mu \neq i$, then the same holds in G'_λ.

The latter claim follows immediately by the construction of G'. □

4.3 Lower bounds for linear codes

For basics on linear codes see Section 3.4. By f_C we denote the characteristic function of a code C, i.e. $f_C(x) = 1$ if and only if, $x \in C$.

The following theorem shows how to apply our lower bound criterion to linear codes.

Theorem 4.13 *Let $C \subseteq \mathbb{F}_2^n$ be a linear code of minimal distance d whose dual $C^\perp$ has minimal distance $d^\perp$.*

Then each sum of k ⊕BP1s guided by a sequence of graph–orderings $G = (G_1, \ldots, G_k)$ representing its characteristic function f_C has size bounded below by $2^{\Omega\left(\min\{d, d^\perp\}/2^k\right)}/(|G_1| \cdot \ldots \cdot |G_k|)$.

Proof. Let $\mathcal{B}$ be a sum of graph–driven ⊕BP1s guided by $G = (G_1, \ldots, G_k)$ representing $f = f_C$. We set $l := \min\{d, d^\perp\} - 1$. Observe, that the code C is both of distance $l+1$ and l–universal. We wish to find a tuple v and sets of partial assignments $A_1, \ldots, A_k$ such that we can apply Lemma 4.11. We use an inductive approach and in order to make the proof readable we define the following predicate P.

We define $P(i)$ to hold if and only if

- there is a tuple $v = (v_1, \ldots, v_i) \in G_1 \times \ldots \times G_i$,
- there are sets of variables $V_1^{(i)}, \ldots, V_i^{(i)}$ with $V_j^{(i)} \subseteq \mathrm{V}(v_j)$ and $|V_j^{(i)}| \geq l/2^i$ such that for $j, k \leq i$, either $V_j^{(i)} = V_k^{(i)}$ or $V_j^{(i)} \cap V_k^{(i)} = \emptyset$,

- there is a set $\mathcal{A}_i$ of assignments with $|\mathcal{A}_i| \geq 2^n/(|G_1| \cdot \ldots \cdot |G_i|)$ such that for $j = 1, \ldots, k$ each $a \in \mathcal{A}_i$ passes in G_j the node v_j, and
- $|\bigcup_{j \leq i} \mathrm{V}(v_j)| \leq l/2 + l/4 + \ldots + l/2^i$.

Before we inductively show that $P(k)$ holds, *we argue how $P(k)$ implies the claim.* Since our aim is to find coherent assignments defined on $V_1^{(k)}, \ldots, V_k^{(k)}$, first according to $\mathcal{A}_k$ we set all variables in

$$V' = \bigcup_{j<i} (\mathrm{V}(v_j) \setminus V_j^{(k)}),$$

to constants. Since there are at most $2^{|V'|}$ assignments defined on $|V'|$, we can fix an assignment γ with $\mathrm{V}(\gamma) = V'$ such that for

$$\mathcal{A}_\gamma = \{\alpha \in \mathcal{A}_k \,;\, \alpha(x) = \gamma(x) \text{ for } x \in V'\},$$

we have

$$|\mathcal{A}_\gamma| \geq 2^{n-|\mathrm{V}(\gamma)|}/(|G_1| \cdot \ldots \cdot |G_k|).$$

Now by a similar argument we choose sets $A_1, \ldots, A_k$ by decomposing $\mathcal{A}_\gamma$ according to $V_1^{(k)}, \ldots, V_k^{(k)}$.

Let $\mathcal{V}$ be a set of variables. For each subset $M \subseteq \mathcal{V}$, there are at most $2^{|\mathcal{V}|-|M|}$ assignments defined on $\mathcal{V} \setminus M$. For $j = 1, \ldots, k$ we apply this to $M = V_j^{(k)}$ and $\mathcal{V} = \{x_1, \ldots, x_n\} \setminus \mathrm{V}(\gamma)$ and define A_j as the projection of $\mathcal{A}_\gamma$ onto $V_j^{(k)}$. Since the elements of A_γ differ only on variables contained in $\mathcal{V}$, projecting $\mathcal{A}_\gamma$ to $M = V_i^{(k)}$ results in at least

$$|\mathcal{A}_\gamma|/2^{|\mathcal{V}\setminus M|} = \left(2^{n-|\mathrm{V}(\gamma)|}/(|G_1| \cdot \ldots \cdot |G_k|)\right)/2^{n-|\mathrm{V}(\gamma)|-|V_i^{(k)}|}$$

different partial assignments. Thus we can choose sets $A_1, \ldots, A_k$ such that A_i consists of partial assignments defined on $V_i^{(k)}$ with size

$$|A_i| \geq 2^{|V_i^{(k)}|}/(|G_1| \cdot \ldots \cdot |G_k|) \geq 2^{l/2^k}/(|G_1| \cdot \ldots \cdot |G_k|).$$

Next we apply Lemma 4.12 for transforming $\mathcal{B}$ into a sum of graph–driven $\oplus$BP1s $\mathcal{B}'$ representing $f|_\gamma$, i.e. we set all variables in $\mathrm{V}(\gamma)$ according to γ. Moreover, there is a sequence of graph–orderings $G' = (G'_1, \ldots, G'_k)$ and a node $v' \in G'_1 \times \ldots \times G'_k$ such that each v-assignment a becomes a v'-assignment a' with

$$a'(x) = \begin{cases} a(x) & \text{if } x \notin \mathrm{V}(\gamma); \\ \text{undefined} & \text{if } x \in \mathrm{V}(\gamma). \end{cases}$$

In line with Lemma 4.11 we let $\mathcal{A}$ contain each α in $A_1 \times \ldots \times A_k$ with $\alpha_i = \alpha_j$ for $A_i = A_j$. It is plain that each element of $\mathcal{A}$ is a v'-assignment. Thus, to apply Lemma 4.11 we only have to find for each $\alpha \in \mathcal{A}$ some partial assignment δ defined on the variables not tested up to v with $f(\alpha, \gamma, \delta) = 1$ and $f(\alpha', \gamma, \delta) = 0$ for each $\alpha' \in \mathcal{A}$ with $\alpha' \neq \alpha$. We do this with the help of the following standard arguments on linear codes that are due to Jukna ([Juk99b]) and that we have already used in the proof of Theorem 3.8.

Since $|\bigcup_{j \leq i} \mathrm{V}(v_j)| \leq l/2 + l/4 + \ldots + l/2^i < l$ we get by the l-universality the existence of an assignment δ as claimed. $f(\alpha', \gamma, \delta) = 0$ for $\alpha' \neq \alpha$ follows since the hamming distance of two accepting assignments has to be greater or equal to l. Now we get with Lemma 4.11, that $\mathrm{SIZE}(\mathcal{B}') \geq \min\{|A_j| \,;\, j = 1, \ldots, k\} \geq 2^{l/2^k}/(|G_1| \cdot \ldots \cdot |G_k|)$ and the claim follows.

In the setting of this theorem $P(1)$ holds. We consider all nodes of G_1 at depth $l/2$ from the source. Thus for each such node v and each path π leading from the source to v exactly $l/2$ variables are tested on π. One of these nodes is passed by $2^n/|G_1|$ of these paths. We denote this node by v_1 and define $\mathcal{A}_1$ to contain all the assignments associated with these paths. We set $V_1^{(1)} = \mathrm{V}(v_1)$ and see that $P(1)$ holds.

$P(i-1)$ implies $P(i)$. For each node w of G_i we denote by

$$old(w) = \mathrm{V}(w) \cap \bigcup_{j<i} \mathrm{V}(v_j),$$

all variables tested on the path from the source of G_i to w that are already tested on the path from the source to the node $v_j, j < i$. By

$$new(w) = \mathrm{V}(w) \setminus \bigcup_{j<i} \mathrm{V}(v_j),$$

we denote those variables in $\mathrm{V}(w)$ not tested on a path to the node $v_j, j < i$. Let C be the set of all nodes w of G_i such that

- $|new(w)| = l/2^i$ and $|old(w) \cap V_j^{(i-1)}| < l/2^i$ for all $j = 1, \ldots, i-1$,

or,

- $|new(w)| < l/2^i$, $|old(w) \cap V_j^{(i-1)}| = l/2^i$ for some $j \in \{1, \ldots, i-1\}$, and $|old(w) \cap V_m^{(i-1)}| < l/2^i$, for all m with $V_m^{(i-1)} \neq V_j^{(i-1)}$.

Since each path in G_i passes exactly one node of C, there is a node v_i such that $|\mathcal{A}_{i-1}|/|G_i|$ paths associated with elements of $\mathcal{A}$ pass it. We determine sets $V_1^{(i)}, \ldots, V_i^{(i)}$ in line with $P(i)$. To this end we have to distinguish two cases, dependent on the choice of v_i.

(1) Case $|new(v_i)| = l/2^i$. After definition of C we additionally get $|old(v_i) \cap V_j^{(i-1)}| < l/2^i$ for all $j = 1, \ldots, i-1$.

First we define

$$V_i^{(i)} = new(v_i),$$

and

$$V_j^{(i)} = V_j^{(i-1)} \setminus old(v_i),$$

for $j = 1, \ldots, i-1$. Then $|V_i^{(i)}| = l/2^i$ and $|V_j^{(i)}| \geq l/2^{i-1} - l/2^i = l/2^i$ for $j = 1, \ldots, i-1$.

(2) Case $|new(v_i)| < l/2^i$. In addition it holds that $|old(v_i) \cap V_j^{(i-1)}| = l/2^i$ for a $j \in \{1, \ldots, i-1\}$ and for all $V_m^{(i-1)} \neq V_j^{(i-1)}$ it holds that $|old(v_i) \cap V_m^{(i-1)}| < l/2^i$. Let $j(1), \ldots, j(\lambda)$ be all indices such that

$$|old(v_i) \cap V_{j(1)}^{(i-1)}| = \ldots = |old(v_i) \cap V_{j(\lambda)}^{(i-1)}| = l/2^i.$$

Recall that by the choice of the sets $V_j^{(i-1)}$, $V_{j(1)}^{(i-1)} = \ldots = V_{j(\lambda)}^{(i-1)}$ and for $j \in \{j(1), \ldots, j(\lambda)\}$ and $m \notin \{j(1), \ldots, j(\lambda)\}$, $V_j^{(i-1)}$ and $V_m^{(i-1)}$ are disjoint. We define

$$V_j^{(i)} := \begin{cases} old(v_i) \cap V_j^{(i-1)} & \text{for } j \in \{j(1), \ldots, j(\lambda)\}; \\ V_j^{(i-1)} \setminus old(v_i) & \text{for } j \notin \{j(1), \ldots, j(\lambda)\}. \end{cases}$$

Note that $|V_j^{(i)}| \geq l/2^i$ for $j = 1, \ldots, i$. So $P(i)$ holds and the claim follows. □

Now we are able to formulate the following corollary, that states our first lower bound for an explicitly defined function. For the notion of Reed–Muller codes see Section 3.4.

Corollary 4.14 *Let* $n = 2^l$ *and* $r = \lfloor l/2 \rfloor$.

Then every sum of graph–driven $\oplus$*BP1s guided by a sequence of graph–orderings* $G = (G_1, \ldots, G_k)$ *representing the characteristic function of* $R(r, l)$ *has size bounded below by* $2^{\Omega\left(n^{1/2}/2^k\right)}/(|G_1| \cdot \ldots \cdot |G_k|)$.

Proof. We apply that the code $R(r, l)$ is linear and has minimal distance 2^{l-r}. It is known that the dual of $R(r, l)$ is $R(l-r-1, l)$, see [MS77]. □

An easy calculation shows that this bound is superpolynomial for

$$k = o\left(\frac{\log n}{\log\log n \cdot \log\log |G|}\right),$$

with $|G| = |G_1| + |G_2| + \ldots + |G_k|$. So we can conclude that for $k = o(\log n/(\log\log n)^2)$, the considered linear code is not contained in $\mathcal{P}(k^*\text{–}\oplus\text{BP1})$. We get the same result even if we allow G to have quasipolynomial size, $|G| = 2^{\log^{\mathcal{O}(1)} n}$.

4.4 Summary

Figure 4.2 complements Figure 3.1 by results due to the lower bounds proved in this section. Additionally, $\mathcal{P}(\text{wsGraph–}\oplus\text{BP1}) = \mathcal{P}(1^*\text{–}\oplus\text{BP1})$ and $\mathcal{P}(\text{graph–}\oplus\text{BP1}) = \mathcal{P}(1\text{–}\oplus\text{BP1})$. k must not exceed the borders stated in Proposition 4.7 and subsequently to Theorem 4.13.

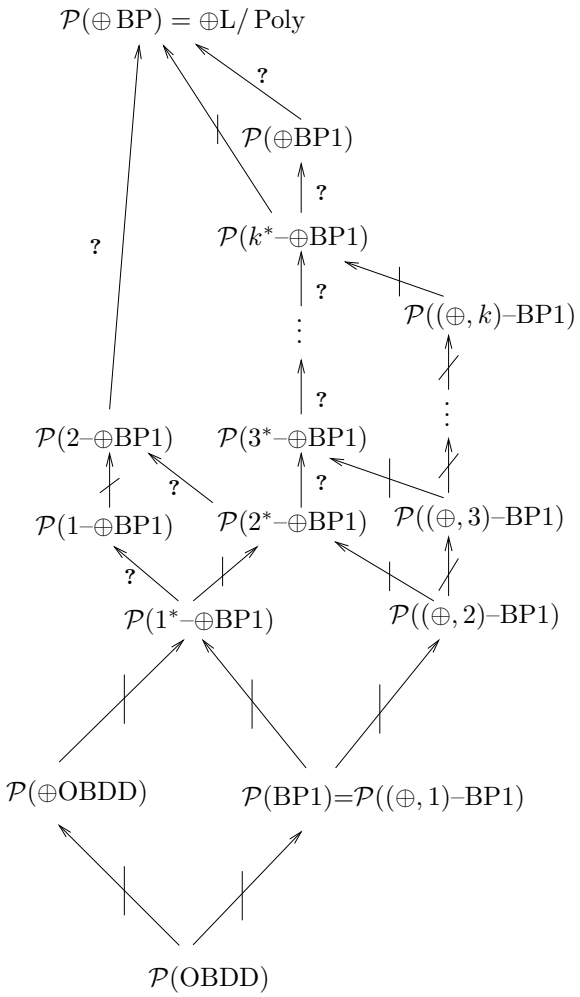

Figure 4.2: The results of this chapter.

Appendix A

Important notation

In this Section we collect our notation on Boolean functions and assignments to Boolean variables. Notation is always introduced in those sections where it is used first. Thus the definitions are spread over the whole work and this list may be of some use.

Let $X = \{x_1, \ldots, x_n\}$ be a set of Boolean variables, i.e. each variable may be assigned by a Boolean constant in $\{0, 1\}$. A *Boolean function* in n variables is a function $f : \{0,1\}^n \to \{0,1\}$. It is convenient to regard $\mathbb{B}_n$, the set of all Boolean functions in n variables, as an $\mathbb{F}_2$–algebra, where $\mathbb{F}_2$ is the prime field of characteristic 2. For $f, g \in \mathbb{B}_n$ the product $f \wedge g$ is defined as componentwise conjunction and the sum $f \oplus g$ as the componentwise exclusive–or. For a set of Boolean functions $A \subset \mathbb{B}_n$ we denote by $\operatorname{span}_{\mathbb{F}_2} A$ the linear space spanned by these functions.

A (Boolean) *assignment* a in $X = \{x_1, \ldots, x_n\}$ is a function $X \to \{0, 1\}$. The i-th component is denoted by $a(x_i)$. A partial assignment α is an assignment defined on a subset of X. By $\mathrm{V}(\alpha)$ we denote the domain $\alpha^{-1}(\{0, 1\})$, i.e. those variables x_i for that $\alpha(x_i)$ is defined.

If α is a (partial) assignment and S is a subset of X, we define

$$\alpha_{|S}(x_j) := \begin{cases} \alpha(x_j) & \text{if } x_j \in S \text{ and } \alpha(x_j) \text{ is defined;} \\ \text{undefined} & \text{else.} \end{cases}$$

as the *projection* of α to S.

For a partial assignment α to some variables, the *subfunction* f_α, or $f|_\alpha$, results by setting all variables in $\mathrm{V}(\alpha)$ to the constants according to α. Sometimes it is more convenient to express f_α, or $f|_\alpha$, as $f(\alpha)$. A function f is called *essentially dependent* on the variable x_i, if different settings to this variable result in different subfunctions, i.e.

$f_{x_i=0} \neq f_{x_i=1}$. For a partial assignment α the subfunction $f(\alpha)$ formally depends on all variables in X, but indeed is not essentially dependent on the variables set by α.

Next we examine how to combine several partial assignments. For partial assignments $\alpha_1, \ldots, \alpha_\nu$ with pairwise disjoint domains $\mathrm{V}(\alpha_i), i = 1, \ldots, n$, we denote by $(\alpha_1, \ldots, \alpha_\nu)$ the assignment α defined on $\mathrm{V}(\alpha_1) \cup \ldots \cup \mathrm{V}(\alpha_\nu)$ as

$$\alpha(x_j) := \begin{cases} \alpha_1(x_j) & \text{if } \alpha_1(x_j) \text{ is defined;} \\ \quad\vdots & \quad\vdots \\ \alpha_\nu(x_j) & \text{if } \alpha_\nu(x_j) \text{ is defined.} \end{cases}$$

If the domains $\mathrm{V}(\alpha_i), i = 1, \ldots, n$ are not pairwise disjoint, it is required that for all $1 \leq i, j \leq \nu$ and for all $x_k \in \mathrm{V}(\alpha_i) \cap \mathrm{V}(\alpha_j)$, the assignments to x_k are equal for α_i and for α_j, i.e. $\alpha_i(x_k) = \alpha_j(x_k)$. Then the notion $\alpha = (\alpha_1, \ldots, \alpha_\nu)$ as defined above is well–defined.

Now it is clear, that $\mathrm{V}(\alpha_1, \ldots, \alpha_\nu) = \bigcup_{i=1}^{\nu} \mathrm{V}(\alpha_i)$. By $\overline{\mathrm{V}(\alpha)}$ we denote the complement $\{x_1, \ldots, x_n\} \setminus \mathrm{V}(\alpha)$.

Appendix B

Mentioned variants of branching programs

In the literature one can find an enormous number of different variants of branching programs and even in this thesis there are mentioned enough variants to possibly confuse the reader. In the introduction it is described how interest in branching programs or binary decision diagrams arises from two different points of view. Those mostly interested in data structures use the term *Binary Decision Diagram (BDD)* whenever a complexity theorist uses the term *branching program (BP)*. The following table may help keeping track of all the variants.

The terms printed in **boldface** are those used in this thesis.

Computation Model	**Data Structure**	**Definition**
branching program, BP	binary decision diagram, BDD	Definition 1.1
read–once BP, BP1	free BDD, FBDD	Definition 1.5
oblivious BP1	**ordered BDD**, OBDD	Definition 1.7
read–once $\oplus$–BP, $\oplus$BP1	free parity BDD, parity–FBDD	Section 1.5
oblivious $\oplus$BP1	**ordered parity BDD**, $\oplus$OBDD	Section 1.5
graph–driven $\oplus$BP1	graph–driven parity–FBDD	Definition 1.6
well–structured graph–driven $\oplus$BP1	well–structured graph–driven parity–FBDD	Definition 2.1
$(\oplus, k)$**–BP**	–	Section 1.7, Section 4.1
k–$\oplus$**BP1**	–	Definition 4.1

Bibliography

[ABH+86] M Ajtai, L Babai, P Hajnal, J Komlos, and P Pudlak. Two lower bounds for branching programs. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 30–38. ACM Press, 1986.

[Ajt99] M. Ajtai. A non-linear time lower bound for Boolean branching programs. In *Proceedings, 40th FOCS*, pages 60–70, 1999.

[BCW80] M. Blum, A.K. Chandra, and M.N. Wegman. Equivalence of free boolean graphs can be decided probabilistically in polynomial time. *Information Processing Letters*, 10:80–82, 1980.

[BHW01] H. Brosenne, M. Homeister, and St. Waack. Graph–driven free parity BDDs: Algorithms and lower bounds. In *Proceedings of the 26th symposion on Mathematical Foundations of Computer Science (MFCS)*, volume 2136 of *Lecture Notes in Computer Science*, pages 212–223. Springer Verlag, 2001.

[BHW02] H. Brosenne, M. Homeister, and St. Waack. Characterizing the complexity of boolean functions represented by well–structured graph–driven parity–FBDDs. *RAIRO Theoretical Informatics and Applications*, 36:229–247, 2002.

[BHW03] H. Brosenne, M. Homeister, and St. Waack. Lower bounds for general graph-driven read-once parity branching programs. In *Proceedings of the 28th symposion on Mathematical Foundations of Computer Science (MFCS)*, volume 2747 of *Lecture Notes in Computer Science*, pages 290–299. Springer Verlag, 2003.

[Bol00] B. Bollig. Restricted nondeterministic read–once branching programs and an exponential lower bound for integer multiplication. In *Proceedings, 25th MFCS*, volume 1893 of *Lecture Notes in Computer Science*, pages 222–231. Springer Verlag, 2000.

[Bro03] H. Brosenne. Personal communication, 2003.

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

[Bry86] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, 35:677–691, 1986.

[Bry91] R. E. Bryant. On the complexity of VLSI implementations of Boolean functions with applications to integer multiplication. *IEEE Transactions on Computers*, 40:205–213, 1991.

[Bry92] R. E. Bryant. Symbolic boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[BSSV00] P. Beame, M. Saks, X. Sun, and E. Vee. Super–linear time-space tradeoff lower bounds for randomized computations. In *Proceedings, 41st FOCS*, pages 169–179, 2000.

[BST98] P. Beame, M. Saks, and J. S. Thathachar. Time-space tradeoffs for branching programs. In *Proceedings, 39th FOCS*, pages 254–263, 1998.

[BV02] P. Beame and E. Vee. Time-space trade-offs, multiparty communication complexity, and nearest neighbour problems. In *Proceedings, 34th STOC*, pages 688–697, 2002.

[BW97] B. Bollig and I. Wegener. Complexity theoretical results on partitioned (nondeterministic) binary decision diagrams. In *Proceedings, 22th MFCS*, volume 1295 of *Lecture Notes in Computer Science*, pages 159–168. Springer Verlag, 1997.

[BW98] J. Behrens and St. Waack. Equivalence test and ordering transformation for parity-OBDDs of different variable ordering. In *Symposium on Theoretical Aspects of Computer Science*, pages 227–237, 1998.

[BW02] B. Bollig and P. Woelfel. A lower bound technique for nondeterministic graph-driven read-once branching programs and its applications. In *Proceedings, 27th MFCS*, Lecture Notes in Computer Science. Springer, 2002.

[BWW02] B. Bollig, St. Waack, and P. Woelfel. Parity graph-driven read-once branching programs and an exponential lower bound for integer multiplication. In *Proceedings 2nd IFIP International Conference on Theoretical ComputerScience*, 2002.

[CMI00] The Clay Mathematics Institute. `www.claymath.org/Millennium_Prize_Problems/`, 2000.

[Cob66] A. Cobham. The recognition problem for the set of perfect squares. In *7th Symp. on Switching and Automata Theory, IEEE*, pages 78–87, 1966.

[CW90] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In *J. Symbolic Computation*, volume 9, pages 251–280, 1990.

[Dam91] C. Damm. DET=$L^{\#L}$. In *Informatik-Preprint 8, FB Informatik der HU Berlin*, 1991.

[Ger94] J. Gergov. Time-space tradeoffs for integer multiplication on various types of input oblivious sequential machines. *Information Processing Letters*, 51:265–269, 1994.

[GM93] J. Gergov and Ch. Meinel. Frontiers of feasible and probabilistic feasible Boolean manipulation with branching programs. In *Proceedings, 10th STACS*, volume 665 of *Lecture Notes in Computer Science*, pages 576–585. Springer Verlag, 1993.

[GM96] J. Gergov and Ch. Meinel. Mod-2-OBDDs – a data structure that generalizes exor-sum-of-products and ordered binary decision diagrams. *Formal Methods in System Design*, 8:273–282, 1996.

[HO98] L.A. Hemaspaandra and M. Ogihara. *The Complexity Theory Companion.* EATCS Series. Springer Verlag, 1998.

[Hom03a] M. Homeister. Lower bounds for the sum of graph–driven read–once parity branching programs. In *Electronic Colloquium on Computational Complexity, ECCC Report TR03-068.* `www.eccc.uni-trier.de`, 2003.

[Hom03b] M. Homeister. On well–structured parity–FBDDs. In *Proceedings of the 6th International Symposium on Representations and Methodology of Future Computing Technology*, 2003.

[Juk89] S. Jukna. On the effect of null–chains on the complexity of contact schemes. In *FCT 1989*, volume 380 of *LNCS*, pages 246–256. Springer Verlag, 1989.

[Juk95a] S. Jukna. The graph of integer multiplication is hard for read–k–times networks. *Tech. Rep. 95-10, University of Trier*, 1995.

[Juk95b] S. Jukna. A note on read–k–times branching programs. *RAIRO Theorectical Informatics and Applications*, 29:75–83, 1995.

[Juk99a] S. Jukna. *Combinatorics of Finite Computations - The Lower Bounds Problem.* Habilitationsschrift, University of Trier, 1999.

[Juk99b] S. Jukna. Linear codes are hard for oblivious read-once parity branching programs. *Information Processing Letters*, 69:267–269, 1999.

[KMW91] M. Krause, Ch. Meinel, and St. Waack. Separating the eraser Turing machine classes L_e, NL_e, co-NL_e, and P_e. *Theoretical Computer Science*, 86:267–275, 1991.

[Kra88] M. Krause. Exponential lower bounds on the complexity of local and real-time branching programs. *Journal of Information Processing and Cybernetics (EIK)*, 24:99–110, 1988.

[Kra02] M. Krause. BDD-based cryptanalysis of keystream generators. In *Theory and Application of Cryptographic Techniques*, pages 222–237, 2002.

[MS77] E. J. MacWilliams and N. J. A. Sloane. *The Theory of Error–Correcting Codes.* Elsevier, 1977.

[MS01a] Ch. Meinel and H. Sack. Heuristics for $\oplus$-OBDDs. In *Proceedings, IEEE/ACM International Workshop of Logic and Synthesis*, pages 304–309, 2001.

[MS01b] Ch. Meinel and H. Sack. Improving XOR-node placements for $\oplus$-OBDDs. In *Proceedings, 5th International Workshop of Reed-Muller Expansion in Circuit Design*, pages 51–55, 2001.

[Nec66] È. I. Nechiporuk. A Boolean function. *Sov. Math. Doklady*, 7:999–1000, 1966.

[Oko97a] E. A. Okol'nishnikova. On comparison between the sizes of read–k–times branching programs. In *Operations Research and Discrete Analysis*, pages 205–225. Kluwer Academic Publishers, Norwell MA, 1997.

[Oko97b] E. A. Okol'nishnikova. On the hierarchy of nondeterministic branching k–programs. In *FCT'97*, volume 1279 of *Lecture Notes in Computer Science*, pages 376–387. Springer Verlag, 1997.

[Pon95] S. Ponzio. A lower bound for integer multiplication with read–once branching programs. In *27th STOC*, pages 130–139, 1995.

[Sac01] H. Sack. *Improving the Power of OBDDs by Integrating Parity Nodes.* PhD thesis, Univ. Trier, 2001.

[Sie99] D. Sieling. Lower bounds for linear transformed OBDDs and FBDDs. In *Proceedings, FSTTCS*, number 1738 in Lecture Notes in Computer Science, pages 356–368. Springer Verlag, 1999.

[SS93] J. Simon and M. Szegedy. A new lower bound theorem for read–only–once branching programs and its applications. In *DIMACS Series in Discrete Mathematics and Theoretical CS*, number 13 in Advances in Computational Complexity Theory, pages 183–193, 1993.

[SS00] P. Savický and D. Sieling. A hierarchy result for read–once branching programs with restricted parity nondeterminism. In *Proceedings, 25th MFCS*, volume 1893 of *LNCS*, pages 650–659. Springer Verlag, 2000.

[SW95] D. Sieling and I. Wegener. Graph driven BDDs – a new data structure for Boolean functions. *Theoretical Computer Science*, 141:238–310, 1995.

[Tha98] J. Thathachar. On separating the read-k-times hierarchy. In *Proceedings, 30th STOC*, pages 653–662, 1998.

[Waa01] St. Waack. On the descriptive and algorithmic power of parity ordered binary decision diagrams. *Information and Computation*, 166:61–70, 2001.

[Weg88] I. Wegener. On the complexity of branching programs and decision trees for clique functions. *Journal of the ACM*, 35(2):461–471, 1988.

[Weg00] I. Wegener. *Branching Programs and Binary Decision Diagrams – Theory and Applications.* SIAM Monographs on Discrete Mathematics and Applications. SIAM, Philadelphia, 2000.

[Wig94] A. Wigderson. NL/Poly $\subseteq \oplus$L/Poly. In *Proceedings of the 9th IEEE Structure in Complexity Theory*, pages 59–62, 1994.

[Žák84] S. Žák. An exponential lower bound for one-time-only branching programs. In *Proceedings of the 11th MFCS*, number 176 in Lecture Notes in Computer Science, pages 562–566. Springer Verlag, 1984.