On the Limitations of Ordered Representations of Functions (Revised Version)

Jayram S. Thathachar

Technical Report UW-CSE-97-02-01
February 1997

Department of Computer Science and Engineering
University of Washington
Box 352350
Seattle, WA 98195

On the Limitations of Ordered Representations of Functions (Revised Version)

Jayram S. Thathachar*
jayram@cs.washington.edu
Department of Computer Science and Engineering
University of Washington
Box 352350
Seattle, Washington 98195–2350

Abstract

We introduce a lower bound technique that applies to a broad spectrum of functional representations including Binary Decision Diagrams (BDDs), Binary Moment Diagrams (*-BMDs), Hybrid Decision Diagrams (HDDs), and their variants. These representations have been widely used for formal verification of hardware systems, particularly symbolic model checking and digital-system design, testing and verification. We define a representation called the Binary Linear Diagram (BLD) that generalizes all these representations and then apply our lower bound technique to show exponential size bounds for a wide range of functions. We also give the first examples of integer functions including integer division, remainder, high/low-order words of multiplication, square root and reciprocal that require exponential size in all these representations. Finally, we show that there is a simple regular language that requires exponential size to be represented by any *-BMD, even though BDDs can represent any regular language in linear size.

1. Introduction

In recent years, symbolic methods have become one of the most important techniques for formal verification of hardware systems. Central to these symbolic methods is an underlying representation for various boolean and integer functions. Ideally, these representations have to be concise, canonical and easy to manipulate. The Ordered Binary Decision Diagram (OBDD) representation due to Bryant [Bry86] has been successfully used as the underlying representation in a variety of symbolic techniques for verification. The main drawback of OBDDs is in concisely representing some important functions, particularly integer functions such as multiplication which requires exponential size. Therefore, a large number of other representations, such as Multi-Terminal Binary Decision Diagrams (EVBDDs) [CMZ+93], Edge-Valued Binary Decision Diagrams (EVBDDs) [McM93], Binary Moment Diagrams (*-BMDs)[1] [BC95] and Hybrid Decision Diagrams (HDDs) [CFZ95], have been proposed and applied to overcome some of the limitations of OBDDs. This research aims at understanding the power of these representations.

*This work was supported by the National Science Foundation under Grant CCR-9303017.

[1] *-BMDs have generated considerable excitement in the formal verification community because of their ability to efficiently represent many integer functions. They have been used to verify and identify errors in a SRT division circuit similar to the one used in Pentium.

We demonstrate the *weakness* of all of the representations above by showing that *none* of them can represent many important boolean and integer functions concisely. We define a general representation, called the *Binary Linear Diagram*(BLD), that encompasses all of these representations. We then show that a variety of integer and boolean functions from arithmetic, formal languages, and graph theory have *exponential* complexity in the BLD representation. Our bounds also apply to other ordered representations such as Functional Decision Diagrams (FDDs) [KSR92] and *-BDDs [End95].

Bryant [Bry86] introduced the OBDD representation for boolean functions and showed that OBDDs can be manipulated efficiently and can compactly represent many useful functions, thereby enabling many tasks in digital-system design, verification and testing to be performed efficiently. Subsequently, OBDDs were also used as the underlying representation in symbolic model checking to alleviate the state explosion problem (see, for example, [BCM$^+$90], [BCL$^+$94] and [McM93]). Despite its success, the OBDD representation has proved to be unsatisfactory for many important functions. To overcome some of its limitations, there have been several efforts to extend the OBDD concept to various *ordered representations* that, like OBDDs, still preserve the notion of an implicit order on the variables, but represent functions with boolean, integer or real ranges. Thus, representations such as MTBDDs [CMZ$^+$93] and EVBDDs [LS92] were defined that have been effective for some additional functions but still have exponential size complexity for other functions such as multiplication and exponentiation. Progress in the direction of concisely representing the multiplication function was made by Bryant and Chen [BC95] who proposed the *-BMD representation for efficiently representing multiplication and other integer functions. An immediate and important question that arose was whether *-BMDs are more powerful than MTBDDs or EVBDDs or, at the least, OBDDs. This question was answered in the negative by Enders [End95], who exhibited functions with exponential complexity in the *-BMD representation but only need polynomial size OBDDs. Recently, Clarke *et al.* (see [CFZ95] and [CKZ96]) defined a generalization of MTBDDs and BMDs, called HDD, that combines the advantages of both representations. It is essential to understand the power and usefulness of all these representations by characterizing the complexity of various important functions in these representations.

We derive our results by showing that for any function f, the size of any BLD for f is bounded by the *rank* of a certain matrix associated with f. This matrix is the standard communication complexity matrix applied to the *best-case* partition of the inputs. Note that this matrix is usually considered for a fixed partition of the inputs and handling the best-case partition is considerably harder. (An excellent source for results and references pertaining to the two kinds of partition is [KN95].) Our technique is analogous to that used in characterizing the minimum size of a *multiplicity automaton* [Fli74, CP71] computing some function in terms of the rank of the Hankel matrix associated with that function.

Our technique provides insight into the contrast between boolean and integer representations. For example, consider the multiplication function. For the boolean function which computes the middle bit of the product, one of our results shows that the associated matrix has exponential rank, but it can be easily verified that the matrix of the integer function has constant rank! This gives insight as to why the integer function has linear-sized *-BMDs but the middle-bit version requires exponential size in all of the ordered representations.

We show that in contrast to multiplication, other integer functions such as integer division (*Div*), remainder (*Mod*), high-order word (*HiMult*) and low-order word (*LoMult*) of multiplication, integer square root (*Sqrt*), and reciprocal (*Inv*) require exponential-sized BLDs by directly bounding the rank

of the associated matrix.[2] These are the first theoretical results that show the limitations of the ordered representations, particularly *-BMDs and HDDs, for representing integer functions.

We also get exponential bounds for many boolean predicates including factor verification, pattern matching, selection/equality, membership in a deterministic context free language, shifted equality, and finally graph predicates such as connectivity, s-t connectivity and bipartiteness. These results are a byproduct of two main approaches that have been traditionally used for bounding the best-case communication complexity of boolean functions. For the graph predicates stated above, Hajnal *et al.* [HMT88] bounded the rank directly by using deep algebraic and combinatorial arguments and showed that the matrix associated with each predicate has exponential rank. In general, directly bounding the rank seems to be a hard problem, so there are fewer results of this kind. The other method that we use for bounding the rank is to construct large *fooling sets.* Dietzfelbinger *et al.* [DHS94] showed that for any boolean function, the rank is at least the square-root of the size of any fooling set; applying this, we can recast all the fooling set bounds for boolean functions as size bounds for the BLD representation. Examples of boolean functions that have exponentially large fooling sets — including those that have been stated above — can be found in Lipton and Sedgewick [LS81], Papadimitriou and Sipser [PS84] and Bryant [Bry91].

Our final result concerns the separation between *-BMDs and OBDDs. We know that *-BMDs can efficiently represent many arithmetic functions that have exponential complexity in the OBDD representation. However, Enders [End95] showed that the graph-predicate that checks whether a graph is a triangle has polynomial-sized OBDDs but has exponential complexity in the *-BMD representation. An interesting problem is to contrast these representations for natural classes of languages. For *regular* languages, we know that OBDDs can represent any regular language in *linear* size. Therefore, a natural question to ask is whether *-BMDs can also represent regular languages efficiently. We answer this in the *negative*, by exhibiting a simple regular language requiring exponential size in the *-BMD representation.

The paper is organized as follows. In Section 2, we define the BLD representation and illustrate how it generalizes all the ordered representations. Section 3 describes the connection between multiplicity automata and BLDs and how standard results on multiplicity automata size can be applied to get size bounds for BLDs; this section can be skipped, if necessary. In Section 4, we describe our approach of directly bounding the minimum size of a BLD computing some function in terms of the ranks of certain matrices associated with that function. Applying this technique, we prove in Section 5.1 that the integer functions *Div*, *Mod*, *HiMult*, *LoMult*, *Sqrt* and *Inv* require exponential-sized BLDs. In Section 5.2 and Section 5.3, we describe the fooling set approach for boolean functions and give exponential lower bounds for many boolean functions by either using fooling sets or directly bound the rank. Finally, in Section 6, we demonstrate for a simple regular language that the *-BMD complexity is exponential.

2. Binary Linear Diagrams

Let X be any finite *variable* set. Informally, each function f that we deal with in this paper is defined by associating f with a variable set and defining the inputs to f to be the various 0-1 assignments to the variables. Formally, a *boolean input assignment* $\sigma : X \to \{0,1\}$, or input for short, is an assignment of 0-1 values to X. For technical reasons, we will also allow X to be the empty set in which case σ is

[2] For n-bit integers x and y, $HiMult(x,y) \stackrel{\text{def}}{=} \lfloor xy/2^n \rfloor$, $LoMult(x,y) \stackrel{\text{def}}{=} xy \bmod 2^n$, $Sqrt(x) \stackrel{\text{def}}{=} \lfloor \sqrt{x} \rfloor$, and $Inv(x) \stackrel{\text{def}}{=} \lfloor 2^{2n}/x \rfloor$.

the (unique) empty input. We will use monomials to denote inputs; for example, $\overline{x}yz$ denotes an input $\sigma : \{x, y, z\} \to \{0, 1\}$, where $\sigma(x) = 0$ and $\sigma(y) = \sigma(z) = 1$. Define f to be a *pseudo-boolean function on* X if its domain is the set of inputs that assign 0-1 values to the variables of X and its range is some fixed ground field $\mathcal{K}$.[3] To simplify things, whenever it is clear from the context, we will refer to a pseudo-boolean function plainly as a function and we will not mention the variable set on which it is defined.

An important concept that will be used throughout the paper is the notion of restricting some of the variables of X to fixed values and considering the resulting function. Given two inputs $\sigma : Y \to \{0, 1\}$ and $\pi : Z \to \{0, 1\}$, where Y and Z are *disjoint*, let $\sigma \cdot \pi : Y \cup Z \to \{0, 1\}$ denote their union, that is, $(\sigma \cdot \pi)(x) = \sigma(x)$ when $x \in Y$, and $(\sigma \cdot \pi)(x) = \pi(x)$ when $x \in Z$. Fix an input $\sigma : Y \to \{0, 1\}$, and let σ' be the restriction of σ to $Y \cap X$. The *subfunction* f_σ of f denotes a function that depends on the variable set $X \backslash Y$ such that for each input $\pi : X \backslash Y \to \{0, 1\}$, $f_\sigma(\pi) = f(\sigma' \cdot \pi)$.

We now define our abstraction of the ordered representations, the *(Ordered) Binary Linear Diagram (BLD)*. Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of variables and let $x_{p_1}, x_{p_2}, \ldots, x_{p_n}$ be an order imposed on the variables of X. The basic structure of a BLD is a labeled, directed acyclic graph. The nodes that have out-degree 0 are called the sinks; each sink is labeled with an element of $\mathcal{K}$. Every other node v has out-degree two and the two edges that are directed from v are distinguished as the 0-edge and 1-edge, respectively. The node that the 0-edge (respectively, 1-edge) is incident to is called the 0-child (respectively, 1-child). The node v is labeled with a variable from X and a 2×2 matrix $\begin{bmatrix} v_{00} & v_{01} \\ v_{10} & v_{11} \end{bmatrix}$, with entries in $\mathcal{K}$. The BLD is required to satisfy the constraint in every directed path, the sequence of variables appearing in order along that path must be a subsequence of $x_{p_1}, x_{p_2}, \ldots, x_{p_n}$. In other words, if u is a node labeled with the variable x_{p_i}, for some i, and v is either a 0-child or a 1-child of u labeled with the variable x_{p_j}, for some j, then $j > i$.

The 2×2 matrix associated with a node describes the linear relationship between the function computed at the node and the two functions computed at its children. Formally, we define the semantics of computation in a BLD by associating a *node* function g_v with each node v.[4] For a sink node v, g_v is a constant function (on the empty variable set) as given by its label. For a non-sink node v labeled with the variable x_{p_k} for some k, $1 \leq k \leq n$, g_v is defined on the variable set $\{x_{p_k}, x_{p_{k+1}}, \ldots, x_{p_n}\}$ in terms of its 0-child u and 1-child w as follows:

$$\begin{bmatrix} (g_v)_{\overline{x_{p_k}}} \\ (g_v)_{x_{p_k}} \end{bmatrix} = \begin{bmatrix} v_{00} & v_{01} \\ v_{10} & v_{11} \end{bmatrix} \cdot \begin{bmatrix} g_u \\ g_w \end{bmatrix}.$$

Finally, there is a designated node with in-degree 0 called the *source*. We say that the BLD *computes* the node function associated with the source node and the *size* of the representation is the number of nodes that it contains. It is important to note that unlike many of the ordered representations that have canonical representations of functions, it is possible to have different BLDs computing the same function. However, this is not a drawback since our technique for proving lower bounds for functions does not require their BLD representations to be unique.

Given the description of a function in any of the known ordered representations, there is a natural

[3] For boolean functions, this field is $GF[2]$.

[4] The easiest way to compute these node functions is to evaluate them in a bottom-up fashion, starting from the sinks and then proceeding towards the root.

and easy transformation to a BLD of the same size computing the same function. For example, given an OBDD or an MTBDD for a function f, the BLD for f has the same underlying acyclic graph with the same variable and sink labelings and with the associated matrix for each node being the 2×2 identity matrix. The transformation of *-BMDs to BLDs is best illustrated in Example 1 that we give below. HDDs are *oblivious* forms of BLDs. Here the BLD is a leveled acyclic graph with all its edges going between adjacent levels. All the nodes in any level (except the level corresponding to the sinks) are labeled with the same variable and the same matrix that the HDD associates with that variable.

Example 1: To illustrate BLDs and the transformation of *-BMDs to BLDs, consider the integer multiplication function for a pair of two-bit numbers. Figure 1 shows both the *-BMD representation and the corresponding BLD representation of that function. Using our definition, we can compute and show that the node function at node d is y_0 and at node c is $(1-y_1)\cdot(1\cdot y_0+0\cdot 2)+y_1\cdot(1\cdot y_0+1\cdot 2) = y_0+2\cdot y_1$. Note that in the BLD, the matrix associated with node a abstracts both the Galois expansion and the weight associated with the 1-edge of the corresponding node in the *-BMD.

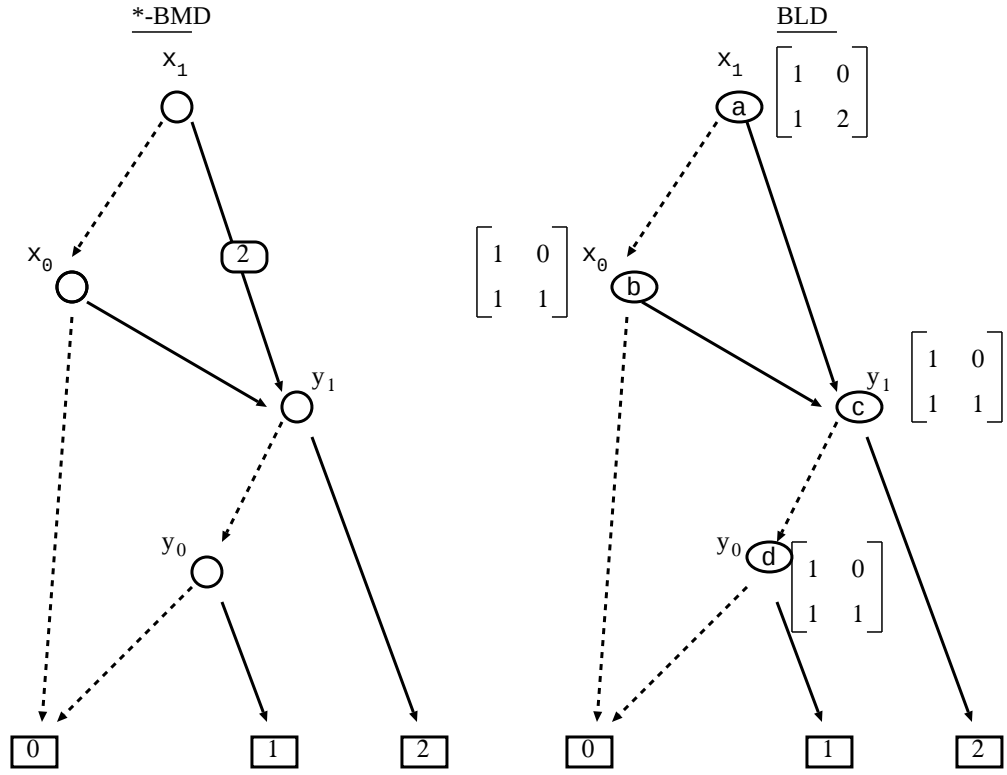

Figure 1: *The *-BMD (left) and BLD(right) for the multiplication function with the order of variables being x_1, x_0, y_1, y_0. The dashed lines denote the* 0*-edges and the solid lines denote the* 1*-edges.*

In Section 4, we describe our main result for getting BLD size lower bounds for a function that holds independent of the order of the variables. A variation of this result can be proved by transforming BLDs to *multiplicity automata*, and applying a fundamental theorem of multiplicity automata. We describe this approach in the next section and can be skipped if so desired. Our approach, which we describe in

Section 4, is to avoid the transformation to multiplicity automata and directly analyze the structure of BLDs. The bounds we obtain here can be applied to get size bounds for multiplicity automata as well. Moreover, our approach leads naturally to considering partitions rather than orders of the variable set, which is more helpful in getting bounds for BLD size that hold for *all* orders of the variables.

3. Transformation to Multiplicity Automata

A *multiplicity automaton* of size r consists of r states $Q = \{1, 2, \ldots, r\}$, where 1 is the start state, two $r \times r$ matrices μ_0 and μ_1 and a vector $\gamma = (\gamma_1, \gamma_2, \ldots, \gamma_r)$ with entries in some field $\mathcal{K}$. We interpret $[\mu_0]_{i,j}$ and $[\mu_i]_{i,j}$ to be the weights of the edge (i, j) corresponding to the symbols 0 and 1 respectively, and γ_i to be the weight of state i. Define $\mu(\epsilon)$ to be the identity matrix and $\mu(x)$, for any binary string $x = x_1, x_1, \ldots x_n$ to be the product of the matrices $\mu_{x_1} \cdot \mu_{x_2} \cdots \mu_{x_n}$. We define the function $g : \{0,1\}^* \to \mathcal{K}$ computed by this automaton as $g(x) \stackrel{\text{def}}{=} [\mu(x)]_1 \cdot \gamma$, where $[\mu(x)]_1$ is the first row of $\mu(x)$. In other words, for each sequence of $n+1$ states $1 = q_1, q_2, \ldots q_{n+1}$, we take the product $[\mu_{x_1}]_{q_1,q_2} \cdot [\mu_{x_2}]_{q_2,q_3} \cdots [\mu_{x_n}]_{q_n,q_{n+1}} \cdot \gamma_{q_{n+1}}$, and sum over all possible sequences.

Multiplicity automata are an important generalization of classic automata and have been used in a variety of areas. In learning theory, they have attracted a lot of attention because of their implications in the learnability of several classes of DNF-formulae (see [BBB+96] for references to work in this area). In conjunction with the theory of formal series, they have been used to solve some old problems in automata theory (see, for example, [HK91]). They have also been used to model certain Markov-like stochastic processes with external inputs [CP71]. In this case, μ_0 and μ_1 are the stochastic matrices containing the transition probabilities corresponding to the external inputs 0 and 1 respectively, and γ is the characteristic vector of the desirable final states.

Given any BLD P computing a function f on the variable set X, and using the order $x_{p_1}, x_{p_2}, \ldots, x_{p_k}$, we can transform it to a multiplicity automaton N that computes a related function f' with at most a linear blow-up in size. For this we define a correspondence between input assignments to the variables X and strings in $\{0,1\}^n$ as follows: Given any string $b = b_1 b_2 \ldots b_n \in \{0,1\}^n$, we define an input assignment σ_b that assigns b_i to x_{p_i}. The transformation should satisfy the property that for any string in $b \in \{0,1\}^n$, $f'(b) = f(\sigma_b)$. (We don't care about the values that f' takes for other strings in $\{0,1\}^*$.) Before we describe the construction, we note that for a family of BLDs, we are constructing a family of multiplicity automata, one for each BLD in the family.

First, we transform P to an oblivious BLD P' of size at most $n \cdot \text{size}(P)$. A brief sketch of this transformation is as follows: We divide the nodes of P into $n+1$ levels, where the i^{th} level consists of nodes that have the label x_i, for $1 \leq i \leq n$, and the $n+1^{th}$ level contains the sinks. For each node v in level i, we add $i-1$ dummy nodes, one in each of the levels 1 through $i-1$ (call them v as well). A dummy node v in level j is labeled by the variable x_j, the identity matrix $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, and both its 0-child and 1-child are the node v in level $j+1$. For any edge in P from a node u in level i to a node v in level $j > i+1$ (that skips levels), we delete that edge in P' and add an edge from u to the dummy node v in level $i+1$. It is easy to see that P' computes f, is oblivious, and has size at most $n \cdot \text{size}(P)$.

Now, we view P' as the multiplicity automaton N as follows: The nodes of P' become the states of N, and the source node of P' becomes being the start state of N. For a node v in P' with the associated

matrix $\begin{bmatrix} v_{00} & v_{01} \\ v_{10} & v_{11} \end{bmatrix}$, and whose 0-child and 1-child are u_0 and u_1 respectively, we define the weight of the edge (v, u_b) in N corresponding to the symbol b', where $b, b' \in \{0,1\}$, to be $[\mu_{b'}]_{v,u_b} \stackrel{\text{def}}{=} v_{b'b}$. Edges which are absent have zero weights on them corresponding to any symbol. The vector γ in N has an entry equal to the label of v for each sink node v and zero entries elsewhere. We can now show that N' computes the same value for strings in $\{0,1\}^n$ as P for the corresponding input assignments.

An important property of a multiplicity automaton is that the rank of a certain matrix bounds its size. For a function $f : \{0,1\}^* \to \mathcal{K}$, define its *Hankel matrix* F as an infinite matrix consisting of rows and columns labeled by strings in $\{0,1\}^*$. The $(x,y)^{th}$ entry of F for binary strings x and y is $f(x \circ y)$. It has been shown ([Fli74, CP71]) that the size of a minimal automaton that computes f is equal to the rank of its Hankel matrix F over $\mathcal{K}$. Thus, we can get lower bounds for BLD size as well but with a loss of the $O(1/n)$ factor due to making the BLD oblivious.

4. The Rank Bound for BLDs

In this section, we show that we can prove size lower bounds for BLDs computing a function f by computing the ranks of various special matrices associated f.

Let $X = \{x_1, x_2, \ldots, x_n\}$ be the variable set of f and fix an order of variables $x_{p_1}, x_{p_2}, \ldots, x_{p_n}$. Fix a k, $0 \leq k \leq n$, let $L = \{x_{p_1}, x_{p_1}, \ldots, x_{p_k}\}$ be the first k variables in this order and R be the remaining variables. For each input $\sigma : L \to \{0,1\}$, we can associate a unique node in the BLD that can be reached from the source by tracing the path of 0-edges and 1-edges as defined by σ and stopping as soon as either a sink or a node labeled with a variable of R is reached. For example, in the BLD of Figure 1, if we choose $k = 2$ so that $L = \{x_1, x_0\}$ and $R = \{y_1, y_0\}$, then the nodes corresponding to the inputs $\overline{x_1 x_0}$ and $x_1 \overline{x_0}$ are the sink labeled with 0 and the node labeled with y_1 respectively. Let V_k denote the set of nodes associated, in the manner described above, with all the inputs that assign 0-1 values to the variables of L. The following lemma shows that the subfunction f_σ, for any input $\sigma : L \to \{0,1\}$, is linearly related to the node functions associated with the nodes in V_k.

Lemma 1: Let X, f, P, k, L and V_k be as defined above. Then, for any input $\sigma : L \to \{0,1\}$, there exist scalars $t_{\sigma,w}$, $w \in V_k$, in the ground field such that

$$f_\sigma = \sum_{w \in V_k} t_{\sigma,w} \cdot g_w.$$

Proof: The proof is by induction on k.

Basis ($k = 0$:) Here $L = \emptyset$ and σ is the empty input so $f = f_\sigma = g_s$, where $s \in V_0$ is the source node.

Induction Let $k > 0$ and suppose the statement is true for $k-1$. Let $L' = L \backslash \{x_k\}$ and $R' = X \backslash L'$. Fix any input $\sigma : L \to \{0,1\}$, and note that we can express it either as $\sigma' \cdot x_{p_k}$ or as $\sigma' \cdot \overline{x_{p_k}}$, for some input $\sigma' : L' \to \{0,1\}$. Assume that $\sigma = \sigma' \cdot x_{p_k}$; the proof for the other case is similar. By the induction hypothesis, there exist scalars $t_{\sigma',w'}$, $w' \in V_{k-1}$, such that $f_{\sigma'} = \sum_{w' \in V_{k-1}} t_{\sigma',w'} \cdot g_{w'}$. Therefore,

$$f_\sigma = (f_{\sigma'})_{x_{p_k}} = \sum_{w' \in V_{k-1}} t_{\sigma',w'} \cdot (g_{w'})_{x_{p_k}}. \tag{1}$$

Note that each w' in the sum above is either a sink or labeled with the variable x_{p_j}, for some $j \geq k$. For any $w' \in V_{k-1}$ labeled by x_{p_k} in the sum above, there exist the 0-child $u \in V_k$ and 1-child $v \in V_k$ in P such that $(g_{w'})_{x_{p_k}} = w'_{10} \cdot g_u + w'_{11} \cdot g_v$. Substitute this expression into Equation 1 for each such w'.

On the other hand, for any $w' \in V_{k-1}$ labeled by x_{p_j}, for some $j > k$, $(g_{w'})_{x_{p_k}} = g_{w'}$. Moreover, w' is the (unique) node in P associated with the input σ so w' also belongs to V_k.

Combining the two observations above, we can see that f_σ is a linear combination of the node functions associated with the nodes in V_k, proving the lemma. □

We will describe the linear relationship of Lemma 1 by a matrix equation. Again, fix a $0 \leq k \leq n$, and let L, R and V_k be as in the statement of Lemma 1 above. Define a matrix M_f associated with f of 2^k rows, one for each input $\sigma : L \to \{0,1\}$, and 2^{n-k} columns, one for each input $\pi : R \to \{0,1\}$. The (σ, π)-th entry of M_f is $f(\sigma \cdot \pi)$. Similarly, define a $|V_k| \times 2^{n-k}$ matrix M_g associated with the node functions in V_k. In this matrix, the (w, π)-th entry, for each $w \in V_k$ and each input $\pi : R \to \{0,1\}$, is $g_w(\pi')$, where π' is the input π restricted to the variable set of g_w. Finally, let T denote the $2^k \times |V_k|$ matrix that expresses the linear relationship between M_f and M_g. In other words, the (σ, w)-th entry of T, for each input $\sigma : L \to \{0,1\}$ and each node $w \in V_k$ is the $t_{\sigma,w}$ of Lemma 1.

The relationship between M_f and M_g, as given by Lemma 1 is $M_f = T \cdot M_g$. Therefore, we can infer from elementary linear algebra that

$$\text{rank}(M_f) \leq \text{rank}(M_g) \leq |V_k|.$$

Thus, we can prove lower bounds on the size of P by bounding the rank of M_f. The power of this approach is that since M_f depends only on f and not on P, we can ignore the structure of P or any other BLD that computes f and can get bounds that hold uniformly in all the ordered representations by solely concentrating on f.

Example 2: Consider the multiplication function f of Example 1. If we use the order x_1, x_0, y_1, y_0, as in the BLD in Figure 1 and fix $k = 2$, we get the following matrix:

	$\overline{y_1}\,\overline{y_0}$	$\overline{y_1}\,y_0$	$y_1\,\overline{y_0}$	$y_1 y_0$
$\overline{x_1}\,\overline{x_0}$	0	0	0	0
$\overline{x_1}\,x_0$	0	1	2	3
$x_1\,\overline{x_0}$	0	2	4	6
$x_1 x_0$	0	3	6	9

Notice that this matrix has rank 1 (over reals or rationals). It is easy to verify that for larger input sizes, the ranks of the associated matrices remain constant at 1.

In contrast, consider the bit-level multiplication function f' representing the middle (second least significant) bit of the product:

	$\overline{y_1}\,\overline{y_0}$	$\overline{y_1}\,y_0$	$y_1\,\overline{y_0}$	$y_1 y_0$
$\overline{x_1}\,\overline{x_0}$	0	0	0	0
$\overline{x_1}\,x_0$	0	0	1	1
$x_1\,\overline{x_0}$	0	1	0	1
$x_1 x_0$	0	1	1	0

In this case, the associated matrix has rank 2 over $GF[2]$. We will see shortly that this rank increases exponentially for larger input sizes (for this and any other order of the variables).

In determining the complexity of a function in the BLD representation, we must consider all possible orders of variables. Therefore, to prove lower bounds using the rank method above, we must show that for any order, there is always a partition of X into L and R such that the associated matrix has high rank. We state these observations as a theorem below:

Theorem 2: For any k, $0 \leq k \leq n$, and for any order of the variables $x_{p_1}, x_{p_2}, \ldots, x_{p_n}$, let $M_{f,k}^{p_1,p_2,\ldots,p_n}$ denote the matrix where the (σ,π)-th entry is $f(\sigma \cdot \pi)$, for each σ and π that assign 0-1 values to $\{x_{p_1}, x_{p_2}, \ldots, x_{p_k}\}$ and $\{x_{p_{k+1}}, x_{p_{k+2}}, \ldots, x_{p_n}\}$, respectively. Then, any BLD that computes f must have size at least

$$\min_{p_1,p_2,\ldots,p_n} \max_{k} \text{ rank}(M_{f,k}^{p_1,p_2,\ldots,p_n}).$$

Corollary 3: The statement in Theorem 2 holds when we substitute any of the ordered representations like MTBDDs, EVBDDS, *-BMDs, HDDs in place of BLDs.

5. BLD Lower Bounds for Integer and Boolean Functions

We now consider the various methods to obtain lower bounds on the rank of many important functions that holds independent of the order imposed on the variables. We will be primarily interested in those functions that have exponential rank.

We will use a weaker form of Theorem 2 that is simpler to deal with and still allows us to prove exponential bounds. Notice that once we fix an order of the variables and k in that theorem, the rank of the matrix $M_{f,k}^{p_1,p_2,\ldots,p_n}$ depends only on L and R and not on the order of the variables in L or R. Therefore, let us denote this matrix by $M_f^{L,R}$. Call a family of partitions $\mathcal{P} \subseteq \{(L,R) \mid R = X \backslash L\}$ as *balanced* if for every order $x_{p_1}, x_{p_2}, \ldots, x_{p_n}$ of the variables, there is at least one k such that $(\{x_{p_1}, x_{p_2}, \ldots, x_{p_k}\}, \{x_{p_{k+1}}, x_{p_{k+2}}, \ldots, x_{p_n}\})$ belongs to $\mathcal{P}$. Now, any bound that shows that the rank of $M_f^{L,R}$ is large for all partitions in a balanced family also implies a lower bound on the BLD size. Thus, we are interested in choosing a balanced family of partitions and then proving lower bounds for the *best-partition rank* which is defined as the minimum rank of $M_f^{L,R}$ over all partitions in that family.

5.1. Integer Functions with Exponential BLD Size

In this section, we show that many natural integer functions that compute integer division, remainder, high/low-order words of multiplication, square root, and reciprocal require exponential-sized BLDs. Formally, the input for all these functions consists of possibly many integers x, y, etc., each of which is represented as an unsigned integer of n bits. We define division function $Div(x,y)$ as $\lfloor x/y \rfloor$ and the mod function $Mod(x,y)$ as $x \bmod y$. The functions $HiMult(x,y)$ and $LoMult(x,y)$ represent the high-order word and low-order word, respectively of the product xy, that is $HiMult(x,y) = \lfloor \frac{xy}{2^n} \rfloor$ and $LoMult(x,y) = xy \bmod (2^n)$. Finally, let the square root function $Sqrt(x)$ denote $\lfloor \sqrt{x} \rfloor$ and the reciprocal function $Inv(x)$ denote $\left\lfloor \frac{2^{2n}}{x} \right\rfloor$.

The following theorem shows that the BLD complexity of each of these integer functions is exponential. The exact size bounds that we prove is listed in Figure 2.

f	Minimum Size
Div	$2^{n/16} - 1$
Mod	$2^{n/16} - 3$
HiMult	$2^{n/16} - 1$
LoMult	$2^{n/16} - 3$
Sqrt	$2^{\Omega(n)}$
Inv	$2^{\Omega(n)}$

Figure 2: *The BLD size bounds that we prove for each of the integer functions*

Theorem 4: The functions *Div*, *Mod*, *HiMult*, *LoMult*, *Sqrt* and *Inv* require exponential-sized BLDs over any field that includes the integers.

Proof: We will first describe the general paradigm for proving exponential lower bounds for an integer function $f(x, y, \ldots)$. We will then use this paradigm to prove lower bounds for the specific functions mentioned above. Let the variable sets $X = x_{n-1}x_{n-2}\ldots x_0$, $Y = y_{n-1}y_{n-2}\ldots y_0$, etc., each represent n-bit integer inputs of f (corresponding to x, y, etc.). In all of the functions that we consider, we will choose a consecutive set of $2m$ variables in the integer input X, for some m. Call this set Z. We define the balanced family of partitions as $\mathcal{P} = \{(L, R) \mid R = X \backslash L, |L \cap Z| = |Z|/2\}$, We will refer to assignments to the variables of L (respectively, R) as *row* (respectively, *column*) assignments.

Suppose that we fixed $Z = U \cup V \subseteq X$, where $U = \{x_{\ell+2m-1}, x_{\ell+2m-2}, \ldots, x_{\ell+m}\}$ and $V = \{x_{\ell+m-1}, x_{\ell+m-2}, \ldots, x_\ell\}$, for some fixed ℓ. Let (L, R) be any partition in $\mathcal{P}$. The following proposition which is proved in [Bry91] will be used to construct a submatrix of $M_f = M_f^{L,R}$ having exponential rank.

Proposition 5 ([Bry91, Lemma 3]): There exists an index set $I \subseteq \{1, 2, \ldots, m\}$, with $|I| \geq m/8$, and integers p, $\ell + m \geq p \geq \ell + \max(I)$, and q, $\ell + 2m \geq q \geq \ell + m + \max(I)$ such that the two sets

$$A = \{x_{q-i} \mid i \in I\} \subseteq U \text{ and } B = \{x_{p-i} \mid i \in I\} \subseteq V,$$

satisfy the property that either $A \subseteq L$ and $B \subseteq R$ or $A \subseteq R$ and $B \subseteq L$.

Thus, the words U and V can be aligned in a such a way that the variables in A and B can be "matched" (see Figure 3). Without loss of generality, assume from the proposition above that $A \subseteq L$ and $B \subseteq R$.

Once we obtain A and B, the lower bound argument proceeds by restricting all the variables that are *not* in $A \cup B$ (and only those variables) to certain *fixed* values that depends on the function f. We will then show that the submatrix $N_f = N_f^{A,B}$ that consists of only those row and column assignments in M_f that conform to these restrictions has (almost) full rank. Since there is no restriction on the values that the variables in A and B can be set to, N_f has $2^{|I|}$ rows and columns; therefore if $m = \Omega(n)$, then N_f and consequently M_f has exponential rank.

We will adopt the following conventions in referring to the various assignments (fixed or varying). Each set of boolean values b_i, $i \in I$, can be thought of as assigning values to the variables of A or B and thus can be associated with a unique row or column of N_f. We will identify the set b_k, $k \in I$, with

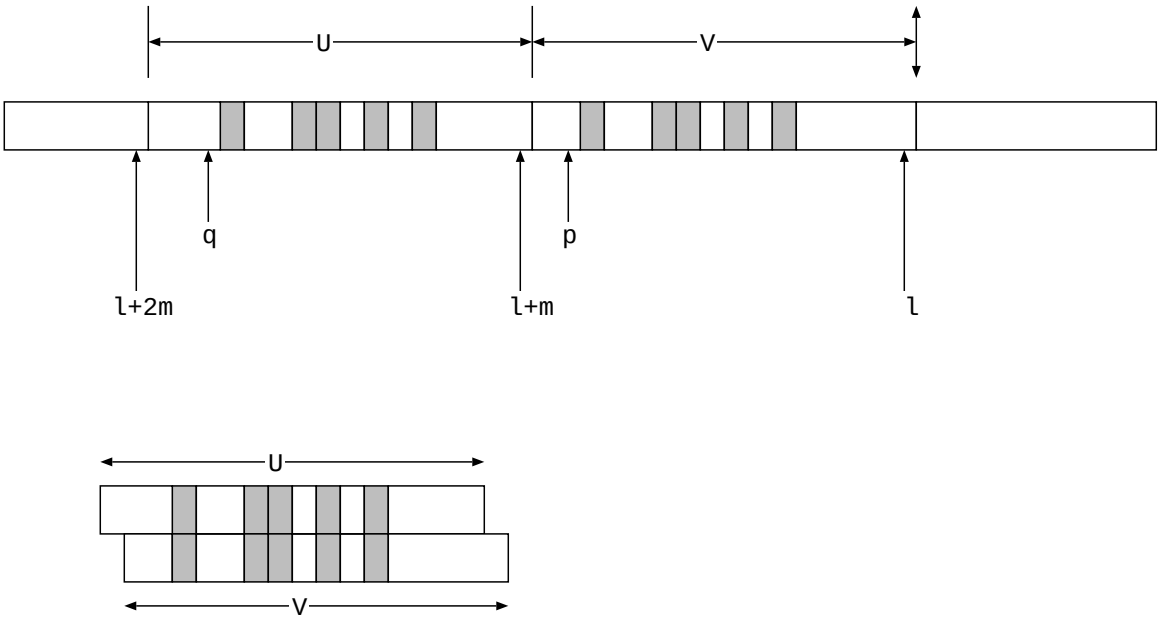

Figure 3: *This figure illustrates Proposition 5 for a particular U and V. The sets $A \subseteq U$ and $B \subseteq V$ (shown shaded in the figure) which are obtained via the proposition can be matched by suitably aligning the words U and V.*

the (unique) number $\sum_{k \in I} b_k 2^{-k}$. Let $0 \leq s_1 < s_2 < \ldots < s_{2^{|I|}} < 1$ be all the numbers arising this way; note that for each i, $s_i = 2^{\max(I)} a_i$ for some integer a_i. Permute the rows and columns of N_f so that the j^{th} row and and j^{th} column are associated with s_j, for $1 \leq j \leq 2^{|I|}$. Let t be the integer which corresponds to the fixed assignment of values to the variables of $X \backslash (AUB)$.[5] Note that the input X which corresponds to the i^{th} row and j^{th} column is $X_{ij} = 2^q s_i + 2^p s_j + t$.

Intuitively, for each function $f(X, \ldots)$, t will be chosen in such a way that when computing $f(X_{ij}, \ldots)$, the integers $2^q s_i$ and $2^p s_j$ will be multiplied by suitable factors so as to obtain a term which aligns s_i and s_j. Since these numbers affect the same bit positions, we will use the alignment to affect the value of f for the various X_{ij}'s in a way that N_f (and consequently M_f) has almost full rank.

The following elementary properties of rank will be used (implicitly) in these proofs:

1. $\text{rank}(M + N) \geq \text{rank}(M) - \text{rank}(N)$, and for any non-zero c, $\text{rank}(cM) = \text{rank}(M)$.

2. Let c, $a_1, a_2, \ldots, a_k$, $b_1, b_2, \ldots, b_k$ be some numbers and let M denote a $k \times k$ matrix where the $(i, j)^{th}$ entry is $a_i + b_j + c$. Then M has rank at most 2.

We now describe the details specific to each function for proving exponential bounds. For each function f, we will choose U and V, and apply Proposition 5 to obtain I, p, q, A, and B. Then, we will set all the variables except for those in $A \cup B$ to a particular set of fixed values and finally prove that the resulting matrix N_f as defined above has almost full rank.

[5] By this, we mean that if each $x_j \in X \backslash (A \cup B)$ is set to $c_j \in \{0, 1\}$, then the integer $\sum_{x_j \in X \backslash (A \cup B)} c_j 2^j$ evaluates to t.

(a) *Div*

Let $n = 2m$. Choose $U = \{x_{2m-1}, x_{2m-2}, \ldots, x_m\}$, and $V = \{x_{m-1}, x_{2m-2}, \ldots, x_0\}$ and apply Proposition 5 to obtain I, p, q, A, and B. Set each of the variables in $X \backslash (A \cup B)$ to 0 so that $X_{ij} = 2^q s_i + 2^p s_j$. Fix Y to be the integer $2^{q-\max(I)} + 2^{p-\max(I)}$ by setting both $y_{p-\max(I)}$ and $y_{q-\max(I)}$ to 1 and each remaining variable in Y to 0.

To analyze the $(i,j)^{th}$ entry of N_{Div}, $Div(X_{ij}, Y)$, we first observe that

$$X_{ij} = (2^{\max(I)} s_i)Y + 2^p(s_j - s_i).$$

Moreover, $|2^p(s_j - s_i)| < 2^p < Y$. From these observations, we can deduce that

- $Div(X_{ii}, Y) = 2^{\max(I)} s_i$, and
- for all $j < i$, $Div(X_{ij}, Y) = 2^{\max(I)} s_i - 1$.

It is now an easy exercise in linear algebra to show that this matrix has rank at least $2^{|I|} - 1 \geq 2^{n/16} - 1$.

(b) *Mod*

For the same parameters considered in part (a) above, note that $N_{Mod} = M - Y \cdot N_{Div}$, where the $(i,j)^{th}$ entry of M is $2^q s_i + 2^p s_j (= X_{ij})$. Therefore,

$$\text{rank}(N_{Mod}) \geq \text{rank}(N_{Div}) - \text{rank}(M) \geq \text{rank}(N_{Div}) - 2 \geq 2^{|I|} - 3 = 2^{n/16} - 3.$$

(c) *HiMult*

Let $n = 2m$. Choose $U = \{x_{2m-1}, x_{2m-2}, \ldots, x_m\}$, and $V = \{x_{m-1}, x_{2m-2}, \ldots, x_0\}$ and apply Proposition 5 to obtain I, p, q, A, and B. For each $k \in I' = \{1, 2, \ldots, \max(I)\} \backslash I$, we set the variable $x_{q-k} \in X \backslash (A \cup B)$ to 1, and all the remaining variables in $X \backslash (A \cup B)$ to 0; these variables form the integer $2^q r$, where $r = \sum_{k \in I'} 2^{-k}$. Therefore, $X_{ij} = 2^q(s_i + r) + 2^p s_j$. We also set both y_{2m-q} and y_{2m-p} to 1 and the remaining variables in Y to 0 so that the input Y corresponds to the integer $2^{2m-q} + 2^{2m-p}$.

First, we compute $HiMult(X_{ij}, Y) = \left\lfloor \frac{X_{ij} \cdot Y}{2^{2m}} \right\rfloor$:

$$\begin{aligned} HiMult(X_{ij}, Y) &= \left\lfloor \frac{2^{2m-p+q}(s_i + r) + 2^{2m-q+p} s_j + 2^{2m}(s_i + s_j + r)}{2^{2m}} \right\rfloor \\ &= 2^{q-p}(s_i + r) + \lfloor (s_i + s_j + r) + 2^{p-q} s_j \rfloor \qquad (2) \end{aligned}$$

We analyze the expression in Line 2 as follows:

- $2^{p-q} s_j < 2^{-\max(I)}$.
- $s_i + s_j + r = a_{ij} 2^{-\max(I)}$, for some *integer* a_{ij}. Therefore, $\lfloor (s_i + s_j + r) + 2^{p-q} s_j \rfloor = \lfloor s_i + s_j + r \rfloor$.

- Let $i^* = 2^{|I|} - i + 1$. If $s_i = \sum_{k \in I} b_k 2^{-k}$, for some b_k, $k \in I$, then $s_{i^*} = \sum_{k \in I} \overline{b_k} 2^{-k}$, that is, s_{i^*} is the one's complement of s_i with respect to the bit positions in I. Therefore, $s_i + s_{i^*} = \sum_{k \in I} 2^{-k}$, so
$$s_i + s_{i^*} + r = \sum_{i=1}^{\max(I)} 2^{-i} = 1 - 2^{-\max(I)}. \tag{3}$$

- Using Equation 3, we have for $j \leq i^*$ that $\lfloor s_i + s_j + r \rfloor = 0$, but on the other hand, $s_i + s_{i^*+1} + r \geq s_i + s_{i^*} + 2^{-\max(I)} + r = 1$, implying that $\lfloor s_i + s_{i^*+1} + r \rfloor \geq 1$.

Therefore, when $j \leq i^*$, $\mathit{HiMult}(X_{ij}, Y) = 2^{q-p}(s_i + r)$, whereas $\mathit{HiMult}(X_{i^*+1,j}, Y) \geq 2^{q-p}(s_i + r) + 1$. It follows that N_{HiMult} has rank at least $2^{|I|} - 1 \geq 2^{n/16} - 1$.

(d) *LoMult*

Observe that with the same parameters as in part (c) above, $N_{\mathit{LoMult}} = M - 2^m N_{\mathit{HiMult}}$, where the $(i,j)^{th}$ entry of M is $X_{ij} \cdot Y = (2^{2m-q} + 2^{2m-p}) X_{ij}$. Therefore,

$$\operatorname{rank}(N_{\mathit{LoMult}}) \geq \operatorname{rank}(N_{\mathit{HiMult}}) - \operatorname{rank}(M) \geq \operatorname{rank}(N_{\mathit{HiMult}}) - 2 \geq 2^{|I|} - 3 = 2^{n/16} - 3.$$

(e) *Sqrt*

Let $n = 10m$, for large enough m. Choose $U = \{x_{5m-1}, x_{5m-2}, \ldots, x_{4m}\}$, $V = \{x_{4m-1}, x_{4m-2}, \ldots, x_{3m}\}$, and apply Proposition 5 to obtain I, p, q, A, and B. Fix each of the variables $x_{2q-2\max(I)-2}$, $x_{2p-2\max(I)-2}$, $x_{p+q-2\max(I)-1}$ (which are in $X \backslash (A \cup B)$ because $2p - 2\max(I) - 2 > 5m$) to 1 and all the remaining variables in $X \backslash (A \cup B)$ to 0. Therefore, $X_{ij} = r^2 + 2^q s_i + 2^p s_j$, where $r = 2^{q-\max(I)-1} + 2^{p-\max(I)-1}$.

For each $i \geq 2$, $(r + 2^{\max(I)} s_i)^2 = r^2 + 2^q s_i + 2^p s_i + (2^{\max(I)} s_i)^2$. Since $3\max(I) \leq 3m \leq p$, it follows that $(2^{\max(I)} s_i)^2 < 2^{2\max(I)} \leq 2^{p-\max(I)} \leq 2^p (s_{i+1} - s_i)$, implying that

$$X_{ii} < (r + 2^{\max(I)} s_i)^2 < X_{i,i+1}.$$

Moreover, since $p \geq 3\max(I)$ and $q - p \geq \max(I)$,

$$(r + 2^{\max(I)} s_i - 1)^2 = r^2 + 2^q s_i - (2^{q-\max(I)} - 2^p s_i) - (2^{p-\max(I)} - 2^{\max(I)} s_i - 1)^2) \leq r^2 + 2^q s_i \leq X_{ij},$$

for each j. Therefore, for each $j \leq i$, $\mathit{Sqrt}(X_{ij}) = r + 2^{\max(I)} s_i - 1$ whereas $\mathit{Sqrt}(X_{i,i+1}) \geq r + 2^{\max(I)} s_i$, from which we can verify that the rank of N_{Sqrt} is at least $2^{|I|} - 2 = 2^{\Omega(n)}$.

(f) *Inv*

Let $n = 18m$, $U = \{x_{2m-1}, x_{2m-2}, \ldots, x_m\}$, and $V = \{x_{m-1}, x_{2m-2}, \ldots, x_0\}$ and apply Proposition 5 to obtain I, p, q, A, and B. Fix each of the variables x_{14m}, x_{6m-p-1}, x_{6m-q-1} and x_{q-k}, for all $k \in I' = \{1, 2, \ldots, \max(I)\} \backslash I$ to 1 and all the remaining variables in $X \backslash (A \cup B)$ to 0. If r denotes the value $\sum_{k \in I'} 2^{-k}$, then we can check that $X_{ij} = 2^{14m} + \alpha_{ij}$, where $\alpha_{ij} = 2^{6m-p-1} + 2^{6m-q-1} + 2^q (s_i + r) + 2^p s_j$.

Now, we compute $Inv(X_{ij})$, for $j \geq 2$:

$$\begin{aligned}\left\lfloor \frac{2^{2n}}{X_{ij}} \right\rfloor &= \left\lfloor \frac{2^{36m}}{2^{14m}+\alpha_{ij}} \right\rfloor = \left\lfloor \frac{2^{22m}}{1+\frac{\alpha_{ij}}{2^{14m}}} \right\rfloor \\ &= 2^{22m} - 2^{8m}\alpha_{ij} + \left\lfloor \frac{\alpha_{ij}^2}{2^{6m}} - \frac{\alpha_{ij}^3}{2^{20m}(1+\frac{\alpha_{ij}}{2^{14m}})} \right\rfloor \qquad (4)\end{aligned}$$

(Using the elementary equation $\frac{1}{1+z} = 1 - z + z^2 - \frac{z^3}{1+z}$)

To simplify the expression in Line 4, we first compute

$$\begin{aligned}\alpha_{ij}^2 &= (2^{6m-p-1} + 2^{6m-q-1})^2 + 2^{6m-p+q}(s_i + r) \\ &+ 2^{6m}(s_i + s_j + r) + 2^{6m-q+p}s_j + (2^q(s_i + r) + 2^p s_j)^2.\end{aligned}$$

Therefore,

$$\begin{aligned}\frac{\alpha_{ij}^2}{2^{6m}} &= (2^{3m-p-1} + 2^{3m-q-1})^2 + 2^{q-p}(s_i + r) \qquad (5) \\ &+ (s_i + s_j + r) + 2^{p-q}s_j + \frac{(2^q(s_i + r) + 2^p s_j)^2}{2^{6m}}\end{aligned}$$

Observing that the expression in Line 5 is an integer, we can simplify

$$\begin{aligned}Inv(X_{ij}) &= 2^{22m} - 2^{8m}\alpha_{ij} + (2^{3m-p-1} + 2^{3m-q-1})^2 + 2^{q-p}(s_i + r) \qquad (6) \\ &+ \left\lfloor (s_i + s_j + r) + 2^{p-q}s_j + \frac{(2^q(s_i + r) + 2^p s_j)^2}{2^{6m}} - \frac{\alpha_{ij}^3}{2^{20m}(1+\frac{\alpha_{ij}}{2^{14m}})} \right\rfloor \qquad (7)\end{aligned}$$

To simplify the expression in Line 7, observe the following:

- $\frac{(2^q(s_i+r)+2^p s_j)^2}{2^{6m}} < 2^{2q-6m} \leq 2^{-2m} \leq 2^{-q}$.
- Since $\alpha_{ij} < 2^{6m}$,
$$\frac{\alpha_{ij}^3}{2^{20m}(1+\frac{\alpha_{ij}}{2^{14m}})} \leq \frac{\alpha_{ij}^3}{2^{20m}} < \frac{2^{18m}}{2^{20m}} \leq 2^{-q}.$$
- Combining the last two inequalities,
$$\left| \frac{(2^q(s_i + r) + 2^p s_j)^2}{2^{6m}} - \frac{\alpha_{ij}^3}{2^{20m}(1+\frac{\alpha_{ij}}{2^{14m}})} \right| < 2^{-q} \leq 2^{p-q-\max(I)}.$$
- $2^{p-q}s_j = 2^{p-q-\max(I)}a_j$, for some *positive* integer a_j, and $2^{p-q}s_j < 2^{-\max(I)}$.
- $s_i + s_j + r = 2^{-\max(I)}a_{ij}$, for some integer a_{ij}.

Therefore, the expression in Line 7 equals $\lfloor s_i + s_j + r \rfloor$.

Let $i^* = 2^{|I|} - i + 1$. Similar to the case of *HiMult*, we can show that for all $j \leq i^*$, $\lfloor s_i + s_j + r \rfloor = 0$, but on the other hand, $\lfloor s_i + s_{i^*+1} + r \rfloor \geq 1$. Since the expression in Line 6 can be expressed as $us_i + vs_j + w$, for some constants u, v and w, it follows that for all $j \leq i^*$, $Inv(X_{ij}) = us_i + vs_j + w$, but $Inv(X_{i,i^*+1}) = us_i + vs_j + w + 1$. Thus, N_{Inv} has rank $2^{\Omega(n)}$.

□

5.2. Rank versus Fooling Sets

For this and the next section, we will focus on *boolean* functions and obtain exponential bounds on the rank for many boolean functions that hold over *any* field.[6] These results imply exponential size bounds for BLDs computing such functions over any field. We will first describe an approach that computes this measure indirectly and is easier to apply; we will then describe the results that compute this measure directly.

For a fixed partition of X into L and R, the matrix $M_f^{L,R}$ is also the matrix of the two party communication complexity for f ([Yao79]). One method for getting good lower bounds on this measure is to construct large *boolean fooling sets*. A fooling set $\mathcal{A}$ consists of pairs of inputs that assign values to the variables of L and R, such that for some $\delta \in \{0, 1\}$, the following hold:

(a) for each pair (σ, π) in $\mathcal{A}$, $f(\sigma \cdot \pi) = \delta$, and

(b) for any two distinct pairs (σ, π) and (σ', π') in $\mathcal{A}$, either $f(\sigma \cdot \pi') \neq \delta$ or $f(\sigma' \cdot \pi) \neq \delta$.

For our application, we are interested in knowing how the fooling set size relates to the rank. The following proposition due to Dietzfelbinger *et al.* [DHS94] shows that if the fooling set size is exponential, then so is the rank. Although they considered equipartitions, the proof can be extended to handle unequal-sized partitions as well.

Proposition 6 ([DHS94]): For any boolean function f, and any partition of its variable set into L and R, let $M_f^{L,R}$ be the associated matrix of f with respect to this partition. Suppose s is the size of any fooling set and suppose r is the rank of $M_f^{L,R}$ over any field. Then, $r \geq \sqrt{s} - 1$.

Since we are interested in the best-partition rank, the more relevant measure is the best-partition communication complexity ([PS84]) in which one computes the communication cost for the best choice of a partition into L and R in some fixed balanced family of partitions. Proposition 6 implies that any scheme that gives lower bounds on the best-partition communication complexity by constructing exponential size fooling sets for all partitions in a balanced family also gives bounds on the best-partition rank and consequently gives exponential size bounds for the BLD representation.

In general, constructing fooling sets is easier than computing the rank directly. However, there are functions for which the rank is exponentially larger than the size of any fooling set. In fact, Dietzfelbinger *et al.* [DHS94], in the same paper referred to above, showed that *almost all* boolean functions satisfy the property that the rank is exponential but no fooling set is larger than linear in

[6] We are dealing with 0-1 matrices here, so the rank is well-defined over any field.

size. Therefore, for some functions we have to resort to computing the rank directly. This again is a classic problem that has been extensively studied in communication complexity. As shown by Mehlhorn and Schmidt [MS82], the fixed-partition communication complexity of any function is bounded below by the logarithm of the rank of the associated matrix. In terms of the best-partition model, this means that the logarithm of the best-partition rank is a lower bound on the best-partition communication complexity. Since there a few results that take this approach, they directly give bounds for the BLD size.

5.3. Boolean Functions with Exponential BLD Size

In this section, we list some important boolean functions/predicates that we can show to require exponential size BLDs over any field; for each function, we will indicate the approach that was taken to show that the rank is exponentially large.

Pattern Matching: Verify if the binary pattern string of αn bits occurs in the binary text string of $(1-\alpha)n$ bits, where $0 < \alpha < 1$. (Fooling Set) [LS81]

Factor Verification: Verify if the product of two n-bit numbers equals a $2n$-bit number. (Fooling Set) [LS81]

Middle bit of Product: In contrast with the previous function, here we compute the middle bit of the product of two n bit numbers. (Fooling Set) [Bry91]

Selection/Equality Testing: Given two n bit numbers, x and y such that x has exactly $n/2$ bits set to 1, check if the $n/2$-bit number obtained by selecting those bits in y at the positions corresponding to the 0s in x equals the remaining $n/2$-bit number in y. (Fooling Set) [LS81]

A Deterministic Context-Free Language: The input is an encoding of a string $u \in \{0, 1, c, *\}^*$ and we have to verify that the string with the $*$'s removed from u is of the form wcw^R, for some $w \in \{0, 1\}^*$. (Fooling Set) [LS81]

Shifted Equality: Given two input strings and a number i, the function evaluates to 1 if and only if the first string equals the second shifted circularly to the right i times. (Fooling Set) [Len90]

The proof that this function has fooling sets of exponential size under all partitions was generalized by Lam and Ruzzo [LR92]. Using this result, one can transform a function f that has a large fooling set under some *fixed* partition to a shifting version of f which can be shown to have large fooling sets under *all* partitions. The drawback is that these shifted versions may not be natural.

Graph Properties: Verifying any of the following predicates on undirected graphs: Connectivity, Bipartiteness, and s-t-Connectivity. (Rank) [HMT88]

For more details on these and other functions, see [LS81], [PS84], [Bry91] and [HMT88].

6. *-BMDs and Regular Languages

In the earlier sections, we saw that the rank method is a useful tool for proving bounds that hold uniformly in all the ordered representations. A related and important problem is to contrast specific representations in order to understand what representations are best suited for a class of functions or languages. For *regular* languages, we know that OBDDs can represent any regular language in *linear* size by keeping track of the state in the automaton that represents it. In this section, we show that there is a simple regular language that has exponential complexity in the *-BMD representation. In order to prove this, we use Enders' [End95] approach in bounding the number of distinct *path* functions.

We now state our main result of this section.

Theorem 7: Let the sets A_i, for $i = 0, 2, 3, 4$, be defined as $A_i = \{w \in \{0,1\}^7 : w \text{ has } i \text{ 1s }\}$. Then, any *-BMD representing the regular language

$$S = A_0^* A_3 (A_0 \cup A_2)^* \cup A_0^* A_4 A_0^*$$

requires size $2^{\Omega(n)}$.

We will first describe the technique that Enders introduced to derive bounds for the *-BMD representation and then apply it to prove our result.

Fix an order $x_{p_1}, x_{p_2}, \ldots, x_{p_n}$ on the variables and a *-BMD that computes $f = f_n^S$. For any input σ that assigns values to the first k variables in this order, for some $k \leq n$, let v_σ be the node reached in the *-BMD by taking the path corresponding to σ and let E_σ denote the product of the edge weights from the source to v_σ. As before, we will denote the node function corresponding to a node v by g_v.

Let the *path*[7] function $h_{(\sigma)}$, corresponding to σ, be defined as $h_{(\sigma)} = E_\sigma \cdot g_{v_\sigma}$. Enders showed that the path function can also be expressed in terms of the subfunctions using Mobius inversion. To describe this equation, we will need the following two notations: for any input σ, $|\sigma|$ denotes the number of variables set to 1 by σ and for any two inputs $\sigma, \tau : Y \to \{0,1\}$, we denote $\tau \leq \sigma$ to mean that $\tau(y) \leq \sigma(y)$ for each variable $y \in Y$. As before, partition X into L and R, where $L = \{x_{p_1}, x_{p_1}, \ldots, x_{p_k}\}$. For any input $\sigma : L \to \{0,1\}$, we have the following equations:

Proposition 8 (Enders [End95]):

$$\begin{aligned} f_\sigma &= \sum_{\tau \leq \sigma} h_{(\tau)}. \\ h_{(\sigma)} &= \sum_{\tau \leq \sigma} (-1)^{|\sigma|-|\tau|} f_\tau. \end{aligned} \tag{8}$$

To bound the number of nodes in any particular level, we define a variant of the fooling set that we used earlier. This variant is similar in spirit to the one used by Enders [End95] and Bryant [Bry91]. Here, the fooling set $\mathcal{A}$ consists of inputs that assign values to the variables of L only and satisfies the property that for any two distinct inputs $\sigma, \tau : L \to \{0,1\}$ in $\mathcal{A}$, there exist $\pi, \rho : R \to \{0,1\}$ such that

$$h_{(\sigma)}(\pi) h_{(\tau)}(\rho) \neq h_{(\sigma)}(\rho) h_{(\tau)}(\pi). \tag{9}$$

[7]We use $h_{(\sigma)}$ rather than h_σ to emphasize the fact that this is not a subfunction.

We claim that the *-BMD must have at least $|\mathcal{A}|$ nodes. For otherwise, by the pigeon-hole principle, there are two inputs σ and τ in $\mathcal{A}$ such that $v_\sigma = v_\tau$. But this implies that for all $\pi, \rho : R \to \{0,1\}$

$$\begin{aligned} h_{(\sigma)}(\pi)h_{(\tau)}(\rho) &= E_\sigma E_\tau g_{v_\sigma}(\pi) g_{v_\tau}(\rho) \\ &= E_\sigma E_\tau g_{v_\tau}(\pi) g_{v_\sigma}(\rho) \\ &= h_{(\sigma)}(\rho)h_{(\tau)}(\pi), \end{aligned}$$

which contradicts Equation 9.

With this machinery, we are now ready to prove the result.

Proof:[Of Theorem 7] Assume that $n = 14m$, for some large enough m. Divide the variables $X = \{x_1, x_2, \ldots, x_{14m}\}$ into $2m$ blocks of seven consecutive variables each. In other words, the block B_i, for $1 \leq i \leq 2m$, will contain the variables x_j, for $7(i-1) < j \leq 7i$. Any input σ that assigns values to the variables of a block B_i, for some i, induces a binary string $\sigma(7(i-1)+1)\sigma(7(i-1)+2)\ldots\sigma(7i)$. We will be interested in the various binary strings that are induced by the inputs in the appropriate blocks.

Fix an order $x_{p_1}, x_{p_2}, \ldots, x_{p_n}$ on X and also fix a *-BMD that uses this order to compute the characteristic function $f = f_n^S$ of S. Let $L = \{x_{p_1}, x_{p_1}, \ldots, x_{p_{n/2}}\}$ and $R = X \backslash L$ be an equipartition of X.

A simple counting argument shows that we can always find $2s$ blocks, for some *even* $s \geq m/8$, indexed by the set I ($|I| = 2s$), such that for $\ell \in I$, $|B_\ell \cap L| \geq 3$. For each $\ell \in I$, we will arbitrarily choose three variables in $B_\ell \cap L$ and call them *special*. Similarly, we can find a single block B_r, $r \notin I$, such that $|B_r \cap R| \geq 4$ and arbitrarily choose four special variables in $B_r \cap R$. We will call the blocks B_i, for $i \in I \cup \{r\}$, *special* as well.

The set of inputs that we will deal with will be obtained by carefully assigning values to the special variables. By default, the non-special variables in any input are always assigned to 0 and we will not mention that henceforth. Note that this means that each non-special block induces the binary string 0000000, which is the unique element of A_0.

The fooling set $\mathcal{A}$ consists of inputs that correspond to the various ways of choosing a set $I' \subset I$ of cardinality s; for each such choice of an I', the corresponding input is defined as follows: For each $\ell \in I$,

(a) if $\ell \in I'$, then the three special variables in $B_\ell \cap L$ are each set to 1.

(b) Otherwise, if $\ell \in I \backslash I'$, then set the special variable with the smallest index in $B_\ell \cap L$ to 0 and the other two to 1.

The fooling set has the right size of

$$\binom{2s}{s} = 2^{\Omega(n)}.$$

Note that if (a) (respectively, (b)) above was used to assign values to the special variables of a block, then assigning a 0 to each of the other variables (which are non-special and hence get a 0 value eventually anyway) induces a string in A_3 (respectively, A_2).

We will now show that the set we have constructed is indeed a fooling set by exhibiting two inputs $\pi, \rho : R \to \{0,1\}$ such that

1. for all σ in $\mathcal{A}$, $h_{(\sigma)}(\pi) = 1$, and
2. $h_{(\sigma)}(\rho)$ is *distinct* for each $\sigma \in \mathcal{A}$.

Then, by Equation 9, we will have proved the theorem.

Define the input $\pi : R \to \{0,1\}$ by setting all the four special variables in B_r to 1. To see that $h_{(\sigma)}(\pi) = 1$, for any $\sigma \in \mathcal{A}$, via Equation 8, note that the induced string in block B_r is an element of A_4. The only way to ensure that $\tau \leq \sigma$ and $f_\tau(\pi) \neq 0$ is by having τ assign 0s to all the variables of L so that $\tau \cdot \pi$ induces an element of $A_0^* A_4 A_0^*$ in all the blocks put together. Then,

$$h_{(\sigma)}(\pi) = (-1)^{|\sigma|-|\tau|} f_\tau(\pi) = 1,$$

since s is even and σ assigns $5s$ variables to 1 whereas τ assigns no variable to 1.

On the other hand, the input $\rho : R \to \{0,1\}$ is defined by assigning a 0 to all the special variables in B_r (and hence to all variables in R). For each $\sigma \in \mathcal{A}$, we will obtain an exact expression for $h_{(\sigma)}(\rho)$ via Equation 8 and then show that it is distinct for each σ.

Notice that $\sigma \cdot \rho$ induces an element of A_3 in each of some s special blocks, an element of A_2 in each of some other s special blocks and the element of A_0 in each of the rest. To obtain a $\tau \leq \sigma$ such that $f_\tau(\pi) \neq 0$, the only choice is in assigning values to those blocks in which $\sigma \cdot \rho$ does *not* induce the element of A_0. Moreover, the string that $\tau.\rho$ induces contains no block that can be a member of A_4, so this string must be a member of $A_0^* A_3 (A_0 \cup A_2)^*$. This implies that $|\sigma| - |\tau|$ is odd so that $h_{(\sigma)}(\rho)$ is negative and its magnitude is the number of $\tau \leq \sigma$ such that $\tau \cdot \rho$ induces a string in $A_0^* A_3 (A_0 \cup A_2)^*$.

In the string induced by $\sigma \cdot \rho$, delete all the blocks corresponding to A_0 and call the resulting string of length $14s$ as w. Let $w = w_{2s} w_{2s-1} \ldots w_1$, where $w_i \in A_2 \cup A_3$, and let $n_s > n_{s-1} > \cdots > n_1$ be the positions such that for $1 \leq j \leq s$, $w_{n_j} \in A_3$. From the discussion above, it is clear that the magnitude of $h_{(\sigma)}(\rho)$ is exactly the number of strings u in $A_0^* A_3 (A_0 \cup A_2)^*$ such that u has the same length as w and the bit value for each position in u is no more than the bit value for the corresponding position in w. Let $u = u_{2s} u_{2s-1} \ldots u_1$, where $u_i \in A_0 \cup A_2 \cup A_3$, for $1 \leq i \leq 2s$. For each j, $1 \leq j \leq s$, observe that the number of u's in which $u_{n_j} \in A_3$ is exactly $2^{n_j - j} 4^{j-1} = 2^{n_j + j - 2}$. Summing up over all j, we have,

$$h_{(\sigma)}(\rho) = -\sum_{1 \leq j \leq s} 2^{n_j + j - 2}.$$

By a similar argument one can show that for an input $\sigma' \in \mathcal{A}$ different from σ, by deleting the blocks corresponding to A_0 in the string induced by $\sigma' \cdot \rho$, if $m_s > m_{s-1} > \cdots > m_1$ are the positions where the elements of A_3 appear, then,

$$h_{(\sigma')}(\rho) = -\sum_{1 \leq j \leq s} 2^{m_j + j - 2}.$$

By our construction of the fooling set, the vectors $(n_s, n_{s-1}, \ldots, n_1)$ and $(m_s, m_{s-1}, \ldots, m_1)$ are *different*. Let q be the largest integer such that $n_q \neq m_q$ and assume, without loss of generality, that $n_q > m_q$. Then,

$$\begin{aligned} & h_{(\sigma')}(\rho) - h_{(\sigma)}(\rho) \\ = & \sum_{1 \leq j \leq q} 2^{n_j + j - 2} - \sum_{1 \leq j \leq q} 2^{m_j + j - 2} \end{aligned}$$

$$= \quad 2^{n_q+q-2} - (2^{m_q+q-1} - 1)$$

$$> \quad 0.$$

This proves our claim that $h_{(\sigma)}(\rho)$ is distinct for each $\sigma \in \mathcal{A}$ and completes the proof of the theorem. □

7. Conclusions

We have shown that a variety of integer functions such as integer division, remainder, high/low-order words of multiplication, square root, and reciprocal require exponential-sized BLDs. We then showed that a variety of boolean functions have exponential complexity in all the ordered representations by relating its complexity to two measures, the fooling set size and the rank. The general definition of the BLD representation implies that minor variations in the known ordered representations will not suffice to handle a broader class of functions and we have to go beyond the "linear nature" implicit in their definitions to be able to handle these hard functions. Another direction that can be taken is to study the power of read-once representations that relax the notion of an implicit order on the variables. An important example is the Free Binary Decision Diagram [GM92] representation but we could also consider generalizations of this representation similar to the BLD representation.

Acknowledgments

I am indebted to Paul Beame for his invaluable guidance and support during the course of this work and for his comments and clarifications in the paper. I also thank Richard Anderson for his guidance and for directing me towards studying the *-BMD-complexity of regular languages. Finally, I thank Martin Tompa and William Chan for their comments and suggestions.

References

[BBB+96] Amos Beimel, Francesco Bergadano, Nader H. Bshouty, Eyal Kushilevitz, and Stefano Varricchio. On the applications of multiplicity automata in learning. In *37th Annual Symposium on Foundations of Computer Science*, Burlington, Vermont, 14–16 October 1996. IEEE. To appear.

[BC95] R.E. Bryant and Y.-A. Chen. Verification of arithmetic circuits with binary moment diagrams. In *32nd ACM/IEEE Design Automation Conference*, Pittsburgh, June 1995. Carnegie Mellon University.

[BCL+94] J.R. Burch, E.M. Clarke, D.E. Long, K.L. MacMillan, and D.L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(4):401–424, April 1994.

[BCM+90] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 1–33, Washington, D.C., June 1990. IEEE Computer Society Press.

[Bry86] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[Bry91] R. E. Bryant. On the complexity of VLSI implementations and graph representations of boolean functions with application to integer multiplication. *IEEE Transactions on Computers*, 40(2):205–213, February 1991.

[CFZ95] E. M. Clarke, M. Fujita, and X. Zhao. Hybrid decision diagrams — overcoming limitations of MTBDDs and BMDs. In *International Conference on Computer Aided Design*, pages 159–163, Los Alamitos, Ca., USA, November 1995. IEEE Computer Society Press.

[CKZ96] E.M. Clarke, Manpreet Khaira, and Xudong Zhao. Word level symbolic model checking — avoiding the Pentium FDIV error. In *33rd ACM/IEEE Design Automation Conference*, pages 645–648, 1996.

[CMZ+93] E. Clarke, K.L. McMillian, X. Zhao, M. Fujita, and J.C.-Y. Yang. Spectral transforms for large boolean functions with application to technologie mapping. In *30th ACM/IEEE Design Automation Conference*, pages 54–60, Dallas, TX, June 1993.

[CP71] J. W. Carlyle and A. Paz. Realizations by stochastic finite automata. *Journal of Computer and System Sciences*, 5(1):26–40, February 1971.

[DHS94] M. Dietzfelbinger, J. Hromkovic, and G. Schnitger. A comparison of two lower bound methods for communication complexity. In *Symposium on Mathematical Foundations of Computer Science*, pages 326–335, 1994.

[End95] R. Enders. Note on the complexity of binary moment diagram representations. Manuscript, 1995.

[Fli74] M. Fliess. Matrices de Hankel. *J. Math. Pures et Appl.*, 53:197–224, 1974.

[GM92] J. Gergov and Ch. Meinel. Efficient boolean manipulation with OBDD's can be extended to read-once only branching programs. Technical report, Univ. Trier, 1992.

[HK91] Harju and Karhumaki. The equivalence problem of multitape finite automata. *Theoretical Computer Science*, 78, 1991.

[HMT88] András Hajnal, Wolfgang Maass, and György Turán. On the communication complexity of graph properties. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 186–191, Chicago, Illinois, 2–4 May 1988.

[KN95] E. Kushilevitz and N. Nisan. Manuscript, 1995.

[KSR92] U. Kebschull, E. Schubert, and W. Rosenstiel. Multilevel logic synthesis based on functional decision diagrams. In *29th ACM/IEEE Design Automation Conference*, pages 43–47, 1992.

[Len90] Thomas Lengauer. VLSI theory. In *Handbook of Theoretical Computer Science, Ed. Jan van Leeuwen, Elsevier and MIT Press (Volume A (= "1"): Algorithms and Complexity)*, volume 1. The MIT Press/Elsevier, 1990.

[LR92] Tak Wah Lam and Larry Ruzzo. Results on communication complexity classes. *Journal of Computer and System Sciences*, 44, 1992.

[LS81] Richard J. Lipton and Robert Sedgewick. Lower bounds for VLSI. In *Conference Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computation*, pages 300–307, Milwaukee, Wisconsin, 11–13 May 1981.

[LS92] Y.-T. Lai and S. Sastry. Edge-valued binary decision diagrams for multi-level hierarchical verification. In *29th ACM/IEEE Design Automation Conference*, pages 608–613, 1992.

[McM93] K.L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, Norwell Massachusetts, 1993.

[MS82] Kurt Mehlhorn and Erik M. Schmidt. Las Vegas is better than determinism in VLSI and distributed computing (extended abstract). In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, pages 330–337, San Francisco, California, 5–7 May 1982.

[PS84] C. Papadimitriou and M. Sipser. Communication complexity. *Journal of Computer and System Sciences*, 28, 1984.

[Yao79] Andrew Chi-Chih Yao. Some complexity questions related to distributive computing (preliminary report). In *Conference Record of the Eleventh Annual ACM Symposium on Theory of Computing*, pages 209–213, Atlanta, Georgia, 30 April–2 May 1979.