Available online at www.sciencedirect.com

SCIENCE DIRECT®

Journal of Computer and System Sciences 71 (2005) 520 – 534

JOURNAL OF COMPUTER AND SYSTEM SCIENCES

www.elsevier.com/locate/jcss

Bounds on the OBDD-size of integer multiplication via universal hashing

Philipp Woelfel*

Department of Computer Science, University Dortmund, D-44221 Dortmund, Germany

Received 25 January 2005; received in revised form 23 May 2005

Available online 3 August 2005

Abstract

Bryant [On the complexity of VLSI implementations and graph representations of boolean functions with applications to integer multiplication, IEEE Trans. Comput. 40 (1991) 205–213] has shown that any OBDD for the function $\mathrm{MUL}_{n-1,n}$, i.e. the middle bit of the n-bit multiplication, requires at least $2^{n/8}$ nodes. In this paper a stronger lower bound of essentially $2^{n/2}/61$ is proven by a new technique, using a universal family of hash functions. As a consequence, one cannot hope anymore to verify e.g. 128-bit multiplication circuits using OBDD-techniques because the representation of the middle bit of such a multiplier requires more than $3 \cdot 10^{17}$ OBDD-nodes. Further, a first non-trivial upper bound of $7/3 \cdot 2^{4n/3}$ for the OBDD-size of $\mathrm{MUL}_{n-1,n}$ is provided.

Keywords: OBDDs; Integer multiplication; Binary decision diagrams; Branching programs

1. Introduction and results

Ordered binary decision diagrams (OBDDs) belong to the most important data structures for representing Boolean functions. Efficient algorithms on OBDDs are known for all important operations, as e.g. synthesis operation, equivalence test, satisfiability test or minimization. Therefore, OBDDs have found a wide variety of applications, especially in the areas of model checking and circuit verification.

* Fax: +49 231 755 2047.

E-mail address: philipp.woelfel@cs.uni-dortmund.de.

0022-0000/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcss.2005.05.004

Definition 1. Let $X_n = \{x_1, \dots, x_n\}$ be a set of Boolean variables.

(1) A *variable ordering* on X_n is a bijection $\pi : \{1, \dots, n\} \rightarrow X_n$, leading to the ordered list $\pi(1), \dots, \pi(n)$ of the variables.
(2) A π-OBDD on X_n for a variable ordering π is a directed acyclic graph with one root and two sinks satisfying the following properties: one sink is labeled with 0, the other with 1. Each inner node is labeled by a variable in X_n and has two outgoing edges, one labeled by 0, the other by 1. If an edge leads from a node labeled by x_i to a node labeled by x_j, then $\pi^{-1}(x_i) < \pi^{-1}(x_j)$. This means that any path on the graph passes the nodes in an order respecting the variable ordering π.
(3) The *computation path* of an input $a = (a_1, \dots, a_n) \in \{0, 1\}^n$ is the path starting at the root and leaving any x_i node over the edge labeled by the value of a_i. The OBDD represents a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ if for any $a \in \{0, 1\}^n$ the sink reached by the computation path of a is labeled with $f(a)$.
(4) The *size* of a π-OBDD is the number of its nodes. The π-OBDD-*size* of a Boolean function f (π-OBDD(f)) is the size of the minimum π-OBDD computing f. Finally, the OBDD-*size* of f (OBDD(f)) is the minimum of π-OBDD(f) for all variable orderings π.

For an in-depth discussion of OBDDs and their operations we refer to [14].

OBDDs can be used, e.g. for circuit verification as follows. In order to test a circuit for a function *f* against its specification (function *g*), one can use synthesis operations to obtain π-OBDDs for *f* and *g* and then check whether *f* and *g* are equal by using the equivalence test. The minimization algorithm is used to ensure that the OBDDs obtained during such a procedure are as small as possible.

Although each single operation used in a such a verification procedure is possible in polynomial (i.e. quadratic or even better) time with respect to the sizes of the corresponding input OBDDs, the total procedure may be infeasible because the sizes of the involved OBDDs may grow almost quadratically with each synthesis operation.

It is not surprising that a lot of research effort has been spent in trying to verify multiplier circuits using OBDDs. However, it was not before 1998 that an OBDD for the 16-bit multiplication circuit $c6288$ could be computed [18] (the $c6288$ is one of the most important ISCAS—International Symposium on Circuits and Systems—benchmark circuits). The resulting OBDD consisted of more than 40 million nodes, the largest OBDD obtained during the synthesis operations even had 110 million nodes and the maximum memory requirement was 3803 megabyte. According to the author's knowledge, nobody could successfully compute an OBDD for a larger multiplier circuit, yet.

These experiences do not necessarily mean that there are no small OBDDs for e.g. 16- or 32-bit multiplier circuits, because the size of a π-OBDD can be quite sensitive to the chosen variable ordering π, and finding a variable ordering leading to small or even minimal π-OBDDs is a hard problem (see [1,4,12]). Therefore, it is necessary to prove large lower bounds for the OBDD-size of integer multiplication in order to be sure that verification of multipliers using OBDDs is infeasible.

There is also a more theoretical motivation for the investigation of the OBDD-size of multiplication. It can be easily seen that almost all functions require an exponential number of elements in any realization by networks using only primitive elements. But this would not be disturbing as long as for all practical relevant families of functions small representations of a certain kind exist. Therefore, one is interested in finding exponential lower bounds for the size of OBDDs (and other representation types) computing important and natural families of functions.

Definition 2. The Boolean function $\mathrm{MUL}_{k,n} : \{0,1\}^{2n} \rightarrow \{0,1\}$ computes the bit z_k of the product $(z_{2n-1}\dots z_0)$ of two integers $(y_{n-1}\dots y_0)$ and $(x_{n-1}\dots x_0)$.

The first step towards bounding the size of OBDDs for integer multiplication was done by Bryant [4]. He showed that for any variable ordering π, there exists an index k, such that the π-OBDD-size for $\mathrm{MUL}_{k,n}$ is at least $2^{n/8}$. This result, though, would still allow the possibility that one might obtain polynomial size OBDDs for all functions $\mathrm{MUL}_{k,n}$ by choosing different variable orderings for different output bits. In 1991, Bryant proved that computing the middle bit of multiplication (that is $\mathrm{MUL}_{n-1,n}$) requires a π-OBDD containing $2^{n/8}$ nodes for any variable ordering π [5].

Although this proves the exponential size of OBDDs for multiplication, the bound is not satisfactory. Despite Bryant's bound there is still the possibility that 64-bit multipliers can be represented by OBDDs containing only 256 nodes, while it is widely conjectured that OBDDs computing $\mathrm{MUL}_{n-1,n}$ have a size of at least 2^n. This would mean that such a multiplier could not even be realized with millions of nodes. Since one would like to use OBDDs for realistic applications one is interested in either finding small constructions or a better lower bound. The following result, which will be proven in the next section, provides the second alternative:

Theorem 3. *The* OBDD-*size of* $\mathrm{MUL}_{n-1,n}$ *is at least* $2^{\lfloor n/2 \rfloor}/61 - 4$.

This bound shows that any OBDD for 64-bit multiplication must be constructed of more than 70 million nodes and therefore the representation of 64-bit multipliers using OBDDs requires a huge amount of resources. The verification of 128-bit multipliers is infeasible because more than $3 \cdot 10^{17}$ OBDD-nodes would be necessary.

The technique leading to this result is new. It relies on a universal family of hash functions [16] and makes use of a new lemma showing how such functions distribute two arbitrary sets over the range. Universal hashing is introduced in the next section. We remark, that following the conference version [17] of this paper, quite some progress has been made in proving lower bounds for the BDD-size of $\mathrm{MUL}_{n-1,n}$. For example, in [2] a lower bound of $\Omega(2^{n/4})$ was proven for read-once branching programs (improving the earlier $2^{\Omega(\sqrt{n})}$ bound of Ponzio [10]) and in [11] super-linear time-space tradeoffs were shown for even less restricted BDD-models. All these results extend the main proof-idea of this paper, which builds on universal hashing. Furthermore, the result for OBDDs presented here is the only lower bound achieving such large constant factors (in the exponent and the coefficient) that it has relevance for the verification of multiplier circuits of realistic bit-length.

By a straightforward reduction, one can easily obtain lower bounds on the OBDD-size of the other output bits of integer multiplication, as observed by Bryant [5, Corollary 1]. For $0 \leqslant j < n$, clearly, $\mathrm{MUL}_{j,n} = \mathrm{MUL}_{j,j+1}$. Moreover, if we embed two $(n-j)$ bit integers $(x_{n-j-1}\dots x_0)$ and $(y_{n-j-1}\dots y_0)$ within n bit integers $x' = (x_{n-j-1}\dots x_0 0\dots 0)$ and $y' = (y_{n-j-1}\dots y_0 0 \dots 0)$, then $\mathrm{MUL}_{n+j-1,n}(x',y') = \mathrm{MUL}_{n-j-1,n-j}(x,y)$. Hence, any OBDD for $\mathrm{MUL}_{j,n}$ can be converted into a smaller OBDD for $\mathrm{MUL}_{j,j+1}$ and any OBDD computing $\mathrm{MUL}_{n+j-1,n}$ can be converted into a smaller OBDD for $\mathrm{MUL}_{n-j-1,n-j}$. Substituting $j+1$ by i in the first case and $n+j-1$ by i in the second case, yields the following corollary.

Corollary 4. *Let* $0 \leqslant i \leqslant 2n-1$. *The* OBDD-*size of* $\mathrm{MUL}_{i,n}$ *is at least*

$$\frac{\min\left\{2^{\lfloor (i+1)/2\rfloor}, 2^{\lfloor 2n-i-1\rfloor/2}\right\}}{61} - 4.$$

Since it is generally believed that the true bound on the OBDD-size for $\mathrm{MUL}_{n-1,n}$ is still larger than $2^{n/2}$, it is of interest to have an upper bound, too. Note that for any Boolean function on m variables, there exists an OBDD of size $(2+o(1))2^m/m$ [3], so a trivial upper bound for $\mathrm{OBDD}(\mathrm{MUL}_{n-1,n})$ is roughly $2^{2n}/n$. The following upper bound, proved in Section 3, is the first non-trivial one.

Theorem 5. *The* OBDD-*size for* $\mathrm{MUL}_{n-1,n}$ *is at most* $7/3 \cdot 2^{4n/3}$.

The bound shows that the middle bit of a 16-bit multiplication can be represented by an OBDD containing less than 6.2 million nodes. Constructions of OBDDs satisfying this bound can be derived from the proof.

2. The lower bound

We first describe a general technique to prove lower bounds for the OBDD-size of Boolean functions. The technique is principally well known (see e.g. [13]) but we state it here in a way which suits our needs best. For $a_1, \ldots, a_i \in \{0,1\}$, $1 \leqslant i \leqslant n$, denote by $f_{|a_1,\ldots,a_i}$ the subfunction of f that computes $f(x_1,\ldots,x_n)$, where for $1 \leqslant j \leqslant i$ the jth input-variable according to π (that is $\pi(j)$) is fixed by the constant a_j.

Lemma 6. *Let* $f : \{0,1\}^n \to \{0,1\}$ *and* π *be a variable ordering on* $X_n = \{x_1,\ldots,x_n\}$ *and* $k \in \{1,\ldots,n\}$. *Further let* $T \subseteq X_n$ *contain the first k variables w.r.t.* π, *i.e.* $T = \{\pi(i) \mid 1 \leqslant i \leqslant k\}$ *and let* s_k *be the number of different subfunctions* $f_{|a_1,\ldots,a_k}$ *where* $a_1,\ldots,a_k \in \{0,1\}$. *Then* π-$\mathrm{OBDD}(f) \geqslant 2 \cdot s_k - 1$.

Proof. Let G be an arbitrary π-OBDD for f and assume w.l.o.g. that $\pi(i) = x_i$ for $1 \leqslant i \leqslant n$. Hence, the variables appear on any computation path in the order $x_1,\ldots,x_n$ and $T = \{x_1,\ldots,x_k\}$. For any $a = (a_1,\ldots,a_k) \in \{0,1\}^k$ let v_a be the unique vertex reached by the computation paths starting with a right after the variable x_k has been tested. Denote by V the set of all vertices v_a with $a \in \{0,1\}^k$. Obviously, any directed path leading from a vertex $v_a \in V$ to a sink contains only x_i-nodes with $i > k$. Therefore, any assignment $b = (b_{k+1},\ldots,b_n) \in \{0,1\}^{n-k}$ to these variables defines a unique path from v_a to a sink.

Consider now two different assignments $a, a' \in \{0,1\}^k$. If $v_a = v_{a'}$ then $f_{|a} = f_{|a'}$ because for all $b \in \{0,1\}^{n-k}$ the computation paths of ab and $a'b$ reach the same sink. Therefore $|V| \geqslant s_k$. Moreover, no vertex $v_a \in V$ lies on the path from the source to a vertex $v_{a'} \in V - \{v_a\}$ because then its label would be a variable x_i with $i \leqslant k$. Hence, we can embed a tree in the subgraph of G consisting only of the paths from the root to the vertices in V such that each vertex of the tree has at most two children. Since this tree has $|V|$ leaves it consists of at least $2|V|-1$ vertices. Therefore, G has at least $2|V|-1 \geqslant 2s_k - 1$ nodes. □

Before we start proving the lower bound for the OBDD-size of $\mathrm{MUL}_{n-1,n}$ we shall sketch the main idea. Let for $a \in \{0,1\}^n$ the function $\mathrm{MUL}^a_{k,n} : \{0,1\}^n \to \{0,1\}$ be defined by $\mathrm{MUL}^a_{k,n}(x) = \mathrm{MUL}_{k,n}(a,x)$. Similar as in Bryant's proof we will show that for any given variable ordering π there exists an integer a for which the π-OBDD-size of $\mathrm{MUL}^a_{n-1,n}$ is large. However, Bryant chose this integer a in such a way that only two bits, a_i and a_j, were 1 and all other bits were 0. This way, the product $a \cdot x$ simplified to the sum $a_i \cdot 2^i \cdot x + a_j \cdot 2^j \cdot x$, and lower bounds for the size of π-OBDDs computing such sums can be obtained easily.

Our goal is to allow more choices for the integer a and we need another way to write $a \cdot x$ as the sum of two integers. Consider the set T of the first $n/2$ variables with respect to π and let B be the remaining $n/2$ variables. (Usually, OBDDs are drawn with the root at the top and each child below its parent. This motivates the letters T and B, since the *top* half of the OBDD contains the variables in T and the *bottom* half contains the variables in B.) We construct two sets M and N of integers in $\{0,\dots,2^n-1\}$ in such a way that M contains the integers p for which all variables in T are fixed to 0 and N contains the integers q for which all variables in B are fixed to 0. Clearly, any integer x in $\{0,\dots,2^n-1\}$ can be uniquely expressed as $q+p$ for some $p \in M$ and some $q \in N$. Furthermore, q is uniquely determined by the variables in T and thus determines a unique subfunction $g_{|q}$, where $g = \mathrm{MUL}^a_{n-1,n}$ for an appropriately chosen integer a. According to Lemma 6 it suffices to show that for many different $q \in N$ the subfunctions $g_{|q}$ are different. We do this by determining a constant a and two subsets $M' \subseteq M$ and $N' \subseteq N$ with the following property: for any distinct q, q' in N', there exists $p \in M'$ such that $a(p+q)$ and $a(p+q')$ differ in the nth bit. Since q and q' are determined only by the top variables and p is determined by the bottom variables, it follows that the $2^{n/2}$ assignments of constants to the top variables yield at least $\left|N'\right|$ different subfunctions.

The difficult part is to find the constant a and the sets M' and N' with the above-described property. We consider the function $f_a : \{0,\dots,2^n-1\} \to \{0,1\}^k$ for some appropriately chosen k, which maps an n-bit integer to the k bits $(y_{n-1}\dots y_{n-k})$, where $(y_{2n-1}\dots y_0)$ is the binary representation of the product $y = a \cdot x$. We then use the fact that the middle bit of the product $a(p+q)$ equals in most cases the kth bit of $f_a(p) + f_a(q)$. Hence, if $f_a(q)$ and $f_a(q')$ are different k-bit integers and if we have a lot of different k-bit values $f_a(p)$ with $p \in M'$, then we should be able to find such a $p \in N'$ such that $f_a(p) + f_a(q)$ and $f_a(p) + f_a(q')$ differ in the kth bit. Then it follows that the subfunctions of $\mathrm{MUL}^a_{n-1,n}$ induced by q and q' are different, too.

In order to find many different subfunctions we have to prove that for some integer a there are many different function values $f_a(p)$ with $p \in M$ and many function values $f_a(q)$ with $q \in N$. Here universal hashing comes into play.

2.1. Universal hashing

The concept of universal hashing was introduced by Carter and Wegman in 1979 [6]. While one of its original purposes was to use randomization in hashing schemes instead of relying on the distribution of the inputs, it has found over the years a large variety of applications in areas of all different kinds. Universal hash families are usually defined by the following notation: let $\mathcal{H}$ be a family of hash functions $U \to R$. U and R are called *universe* and *range*, respectively. For arbitrary $x, x' \in U$ and $h \in \mathcal{H}$, we define

$$\delta_h(x,x') = \begin{cases} 1 & \text{if } x \neq x' \text{ and } h(x) = h(x') \\ 0 & \text{otherwise.} \end{cases}$$

If one or more of h, x and x' are replaced in $\delta_h(x, x')$ by sets, then the sum is taken over the elements from these sets. For example, for $H \subseteq \mathcal{H}$, $V \subseteq U$ and $x \in U$, $\delta_H(x, V)$ means

$$\sum_{h\in H} \sum_{x'\in V} \delta_h(x, x').$$

Definition 7. A family $\mathcal{H}$ of hash functions $U \to R$ is *universal*, if for any two distinct $x, x' \in U$

$$\delta_{\mathcal{H}}(x, x') \leqslant \frac{|\mathcal{H}|}{|R|}.$$

A stronger definition of so-called *strongly universal hash families* was given in [15]. Among the many applications, there were also interesting results concerning general branching programs. Mansour et al. [9] investigated the computational complexity of strongly universal hashing, and gave a lower bound for the time-space tradeoff of branching programs computing the functions of such families. For OBDDs computing only single output bits of hash functions from a universal hash family, it is generally not possible to show exponential lower bounds. For example, the convolution of two bit strings can be viewed as a strongly universal hash family [9], but it can be easily seen that for any output bit of the convolution, there exists a variable ordering π yielding a linear π-OBDD-size.

The property of universal hash families we will use here can be described in the following way: if there are two large enough subsets of the universe given, then there exists a hash function under which the function values of the elements from each set cover a large fraction of the range.

For a function $h : U \to R$ and a subset $M \subseteq U$, define $h(M)$ to be the image of M under h, i.e., $h(M) := \{y \in R \mid \exists x \in M : h(x) = y\}$.

Lemma 8. *Let $\mathcal{H}$ be a universal family of hash functions $U \to R$ and $0 \leqslant \varepsilon < 1$. Then for arbitrary $M, N \subseteq U$ with*

$$|M|, |N| > 2(|R| - 1)\frac{\varepsilon}{1 - \varepsilon},$$

there exists a hash function $h \in \mathcal{H}$ such that $h(M)$ and $h(N)$ contain more than $\varepsilon|R|$ elements each.

Proof. Let $r = |R|$ and assume w.l.o.g. that M and N both have the same cardinality m, where $m > 2 \cdot (r - 1) \cdot \varepsilon/(1 - \varepsilon)$. For $h \in \mathcal{H}$ let the random variable X_h be the sum of $\delta_h(M, M)$ and $\delta_h(N, N)$. Using the assumption that $\mathcal{H}$ is universal, we obtain for a randomly chosen function $h \in \mathcal{H}$ an upper bound for the expectation of X_h:

$$\begin{aligned} E_{h\in\mathcal{H}}[X_h] &= \sum_{\substack{x,x'\in M\\ x\neq x'}} E_{h\in\mathcal{H}}\left[\delta_h(x, x')\right] + \sum_{\substack{y,y'\in N\\ y\neq y'}} E_{h\in\mathcal{H}}\left[\delta_h(y, y')\right] \\ &\leqslant |M|(|M| - 1)\frac{1}{r} + |N|(|N| - 1)\frac{1}{r} = \frac{2}{r}\, m(m - 1). \end{aligned}$$

This means by the probabilistic method that there exists $h_0 \in \mathcal{H}$ with

$$X_{h_0} \leqslant \frac{2}{r}\, m(m - 1). \tag{1}$$

In order to prove that this h_0 fulfills the claim, we assume that $h_0(M)$ contains at most εr elements. By summing over the ordered pairs of elements in ${h_0}^{-1}(y) \cap M$ for each $y \in h_0(M)$, we get

$$\begin{aligned}\delta_{h_0}(M, M) &= \sum_{y \in h_0(M)} \left|{h_0}^{-1}(y) \cap M\right| \left(\left|{h_0}^{-1}(y) \cap M\right| - 1\right) \\ &= \sum_{y \in h_0(M)} \left(\left|{h_0}^{-1}(y) \cap M\right|^2 - \left|{h_0}^{-1}(y) \cap M\right|\right) \\ &= -m + \sum_{y \in h_0(M)} \left|{h_0}^{-1}(y) \cap M\right|^2. \qquad (2)\end{aligned}$$

Note that the function $f : \mathbb{R}^t \to \mathbb{R}, (x_1, \dots, x_t) \mapsto \sum_{i=1}^t x_i^2$ is convex. Under the constraint $\sum_{i=1}^t x_i = S$, $f(x_1, \dots, x_t)$ has its minimum for $x_i = S/t, 1 \leqslant i \leqslant t$. Hence, $f(x_1, \dots, x_t) \geqslant t \cdot (S/t)^2 = S^2/t$. We use this in order to bound the last sum in Eq. (2) from below. Since $|h_0(M)| \leqslant \varepsilon r$ and $\sum_{y \in h_0(M)} \left|{h_0}^{-1}(y) \cap M\right| = m$, we obtain

$$\delta_{h_0}(M, M) \geqslant -m + \frac{m^2}{\varepsilon r} = m\left(\frac{m}{\varepsilon r} - 1\right).$$

For N we can use analogous arguments, but instead of using the assumption $h_0(M) \leqslant \varepsilon r$ we use the obvious fact that $h_0(N) \leqslant r$. Then, replacing each appearance of εr by r, we obtain $\delta_{h_0}(N, N) \geqslant m(m/r - 1)$. Therefore, we have the following lower bound on X_{h_0}:

$$X_{h_0} \geqslant m\left(\frac{m}{\varepsilon r} + \frac{m}{r} - 2\right).$$

Plugging in the upper bound of (1) yields

$$2 - \frac{2}{r} \geqslant \frac{m}{\varepsilon r} + \frac{m}{r} - \frac{2}{r} \cdot m = m \cdot \left(\frac{1}{\varepsilon r} - \frac{1}{r}\right) = m \cdot \frac{1 - \varepsilon}{\varepsilon r}$$

and thus

$$m \leqslant \left(2 - \frac{2}{r}\right) \cdot \frac{\varepsilon r}{1 - \varepsilon} = 2(r - 1) \cdot \frac{\varepsilon}{1 - \varepsilon}.$$

But this contradicts the assumption on m. □

We now consider hash functions, which map the n-bit universe $U := \{0, \dots, 2^n - 1\}$ to the k-bit range $R_k := \left\{0, \dots, 2^k - 1\right\}$. For $a, b \in U$ let

$$h^k_{a,b} : U \to R_k, \quad x \mapsto \left((ax + b) \bmod 2^n\right) \operatorname{div} 2^{n-k},$$

where “div” is the integer division without remainder (i.e. $x \operatorname{div} y = \lfloor x/y \rfloor$). In a bitwise view, the result of the modulo operation $x \bmod 2^n$ is represented by the n least significant bits of x. On the other hand, the division $x \operatorname{div} 2^{n-k}$ can be viewed as shifting x by $n - k$ bit-positions to the right. In other words, if the value of the linear function $ax + b$ is represented by $(y_{2n-1} \dots y_0)$, then $h^k_{a,b}$ is the integer, which is represented by the k bits $(y_{n-1} \dots y_{n-k})$. The following result has been established in [16] following the investigation of similar hash classes in [7,8].

Theorem 9 (*Woelfel [16]*). *Let $1 \leqslant k \leqslant n$. Then there exist sets $A \subseteq U$ and $B \subseteq \{0, \ldots, 2^{n-k}-1\}$ such that the family of hash functions $h^k_{a,b}$ with $a \in A$ and $b \in B$ is universal.*

In order to make the paper self-contained, we prove this theorem. We first investigate the division more closely. Let g_b be the mapping $U \mapsto R_k$, $x \mapsto ((x+b) \bmod 2^n) \operatorname{div} 2^{n-k}$.

Claim 10. *Let x_1, x_2 be arbitrary elements from U and $d = (x_2 - x_1) \bmod 2^n$. Then for a randomly chosen $b \in \{0, \ldots, 2^{n-k} - 1\}$ the probability that $g_b(x_1)$ equals $g_b(x_2)$ is exactly*

$$\begin{cases} 1 - d/2^{n-k} & \textit{if } d \in \{0, \ldots, 2^{n-k} - 1\}, \\ 0 & \textit{if } d \in \{2^{n-k}, \ldots, 2^n - 2^{n-k} - 1\} \textit{ and} \\ 1 - (2^n - d)/2^{n-k} & \textit{if } d \in \{2^n - 2^{n-k}, \ldots, 2^n - 1\}. \end{cases}$$

Proof. Assume first that $2^{n-k} \leqslant d < 2^n - 2^{n-k}$. Then either $x_2 \leqslant x_1 - 2^{n-k}$ or $x_2 \geqslant x_1 + 2^{n-k}$. In any case, $g_b(x_1)$ and $g_b(x_2)$ obviously have different function values.

Consider now the case in which $0 \leqslant d < 2^{n-k}$, and let $\lambda = x_1 \operatorname{div} 2^{n-k}$ and $\tau = x_1 \bmod 2^{n-k}$. If $b \in \{0, \ldots, 2^{n-k} - 1\}$ then $g_b(x_1)$ equals λ if $b + \tau < 2^{n-k}$ and otherwise equals $(\lambda + 1) \bmod 2^k$. Since x_2 may be written as $\left(2^{n-k}\lambda + \tau + d\right) \bmod 2^n$, $g_b(x_2)$ equals λ if $b + \tau + d < 2^{n-k}$ and equals $(\lambda+1) \bmod 2^k$ if $2^{n-k} \leqslant b + \tau + d < 2^{n-k+1}$. Otherwise $g_b(x_2)$ equals $(\lambda+2) \bmod 2^k$. (Just for the case $k = 1$ note that $b + \tau + d \geqslant 2^{n-k+1}$ implies $b + \tau \geqslant 2^{n-k}$ and thus $g_b(x_1) = (\lambda + 1) \bmod 2^k \neq g_b(x_2)$.)

Therefore, $g_b(x_1) = g_b(x_2)$ if and only if

$$0 \leqslant b < 2^{n-k} - \tau - d \quad \text{or} \quad 2^{n-k} - \tau \leqslant b < 2^{n-k+1} - \tau - d.$$

It can be easily verified that there are exactly $2^{n-k} - d$ values $0 \leqslant b < 2^{n-k}$ which satisfy this condition.

For the last case in which $2^n - 2^{n-k} \leqslant d < 2^n$, we consider $d' = (x_1 - x_2) \bmod 2^n$ instead of d. By observing that d' equals $2^n - d$ which is in $\{0, \ldots, 2^{n-k} - 1\}$, the claim easily follows from the former case. □

Proof of Theorem 9. Let A be the set of odd numbers in U and $B = \{0, \ldots, 2^{n-k} - 1\}$. We show that the family $\mathcal{H}_{A,B}$ consisting of the functions $h^k_{a,b}$ with $a \in A$ and $b \in B$ is a universal family of hash functions. (Note that as stated in [16] a significant smaller subset B suffices. However, for this a more involved proof is required.)

Consider two distinct elements $x_1 < x_2$ in U and let $\delta = x_2 - x_1$. We can write δ as $r2^s$ for some odd $r < 2^{n-s}$. Then

$$(ax_2 - ax_1) \bmod 2^n = (ar2^s) \bmod 2^n = \left((ar) \bmod 2^{n-s}\right) \cdot 2^s. \tag{3}$$

Now let a be chosen randomly from A. Then $a \bmod 2^{n-s}$ is uniformly distributed over the odd numbers $\{1, 3, \ldots, 2^{n-s} - 1\}$. Since these numbers form a group with respect to the multiplication modulo 2^{n-s}, $(ar) \bmod 2^{n-s}$ is uniformly distributed over this group. This means by Eq. (3) that $(ax_2 - ax_1) \bmod 2^n$ is uniformly distributed over

$$M = \left\{1 \cdot 2^s, 3 \cdot 2^s, \ldots, \left(2^{n-s} - 1\right) \cdot 2^s\right\}.$$

Note that $h_{a,b}(x) = g_b((ax) \bmod 2^n)$. For randomly chosen $a \in A$ and $b \in B$ it follows from Claim 10, that

$$\begin{aligned} p :=& \operatorname{Prob}\left(h_{a,b}(x_1) = h_{a,b}(x_2)\right) \\ =& \frac{1}{|M|} \cdot \left(\sum_{d \in M_1} \left(1 - \frac{d}{2^{n-k}}\right) + \sum_{d \in M_2} \left(1 - \frac{2^n - d}{2^{n-k}}\right)\right), \end{aligned}$$

where $M_1 = M \cap \{0, \ldots, 2^{n-k} - 1\}$ and $M_2 = M \cap \{2^n - 2^{n-k}, \ldots, 2^n - 1\}$. Since the set M_2 consists of the numbers $2^n - d$ with $d \in M_1$, we obtain

$$p = \frac{2}{|M|} \cdot \sum_{d \in M_1} \left(1 - \frac{d}{2^{n-k}}\right).$$

M_1 is the set $\{1 \cdot 2^s, 3 \cdot 2^s, \ldots, (2^{n-k-s} - 1) \cdot 2^s\}$, which is the empty set, if $s \geqslant n - k$. In this case, p equals 0. If $s < n - k$, then

$$\begin{aligned} p =& \frac{2}{|M|}\left(|M_1| - \frac{2^s}{2^{n-k}}\left(1 + 3 + \cdots + (2^{n-k-s} - 1)\right)\right) \\ =& \frac{2}{2^{n-s-1}}\left(2^{n-k-s-1} - 2^{s-n+k} \cdot \left(2^{n-k-s}\right)^2/4\right) \\ =& \frac{2}{2^{n-s-1}}\left(2^{n-k-s-1} - 2^{n-k-s-2}\right) = 2^{-k} = \frac{1}{|R_k|}. \end{aligned}$$

This shows that $\delta_{\mathcal{H}}(x_1, x_2)$ (which equals $|\mathcal{H}| \cdot p$ by definition) is bounded above by $|\mathcal{H}|/|R_k|$. □

2.2. Proof of the main theorem

Since the functions $h^k_{a,b}$ are evaluated not only by a multiplication, but also by an addition, we cannot use Lemma 8 for the proof of the lower bound and the OBDD-size of $\mathrm{MUL}^a_{n-1,n}$ directly. Let $f^k_a := h^k_{a,0}$ be the functions that can be evaluated without addition. The following lemma gives a result similar to that of Lemma 8. Note that as stated in [8], the hash functions f^k_a form an *almost* universal hash class (which means that in Definition 7 $|\mathcal{H}|/|R|$ is replaced by $c|\mathcal{H}|/|R|$ for some constant $c > 1$). But this property seems not to be sufficient to prove a result as strong as the one stated below.

Lemma 11. *Let $\frac{1}{2} \leqslant \varepsilon < 1$ and let M and N be subsets of $\{0, \ldots, 2^n - 1\}$, each of them containing more than $2 \cdot (2^{k+1} - 1) \cdot \varepsilon/(1 - \varepsilon)$ elements. Then there exists an integer $a \in \{0, \ldots, 2^n - 1\}$, such that $f^k_a(M)$ and $f^k_a(N)$ contain at least $(2\varepsilon - 1)2^k$ elements each.*

Proof. By Lemma 8 and Theorem 9, there exist integers $a \in \{0, \ldots, 2^n - 1\}$ and $b \in \{0, \ldots, 2^{n-k-1} - 1\}$ such that $h^{k+1}_{a,b}(M)$ and $h^{k+1}_{a,b}(N)$ contain more than $\varepsilon|R_{k+1}| = \varepsilon 2^{k+1}$ elements each. Let these a, b be fixed and let $f = f^k_a$. We show that $f(M)$ contains at least $(2\varepsilon - 1)2^k$ elements; the claim then follows for N with the same argument.

Let $M' \subseteq M$ with $|M'| = \varepsilon 2^{k+1}$, such that all $x \in M'$ have distinct function values under $h^{k+1}_{a,b}$. Since R_{k+1} contains exactly 2^k even elements, there are at least $|M'| - 2^k$ elements in M', which have an odd function value under $h^{k+1}_{a,b}$. Let M'' be a subset of M' containing exactly $\varepsilon 2^{k+1} - 2^k = 2^k(2\varepsilon - 1)$ elements with an odd function value. To prove the claim, it suffices to show that for any two distinct $x, x' \in M''$ we have $f(x) \neq f(x')$. Let $h^{k+1}_{a,b}(x) = z$ and $h^{k+1}_{a,b}(x') = z'$. Then

$$z \cdot 2^{n-k-1} \leqslant (a \cdot x + b) \bmod 2^n < (z+1) \cdot 2^{n-k-1}.$$

Note that $z \neq 0$ because it is odd. Since by definition $0 \leqslant b < 2^{n-k-1}$, it follows that

$$(z-1) \cdot 2^{n-k-1} \leqslant (a \cdot x) \bmod 2^n < (z+1) \cdot 2^{n-k-1}.$$

Further, by z being odd, $(z-1)/2$ equals $\lfloor z/2 \rfloor$ and $(z+1)/2$ equals $\lfloor z/2 \rfloor + 1$. Therefore, the above inequalities imply

$$\lfloor z/2 \rfloor \cdot 2^{n-k} \leqslant (a \cdot x) \bmod 2^n < (\lfloor z/2 \rfloor + 1) \cdot 2^{n-k}.$$

This means that $f(x) = \lfloor z/2 \rfloor$, and with the same argument also $f(x') = \lfloor z'/2 \rfloor$. But because z and z' are both odd and different, $\lfloor z/2 \rfloor$ and $\lfloor z'/2 \rfloor$ are different, too. So, we obtain the desired result $f(x) \neq f(x')$. □

We are now ready to prove that for any variable ordering π there is an integer a such that the OBDD-size of $\mathrm{MUL}^a_{n-1,n}$ is large (recall that $\mathrm{MUL}^a_{n-1,n}(x) = \mathrm{MUL}_{n-1,n}(a, x)$). In order to do so, we need some more notation. Let x be an integer represented in a bitwise notation as $(x_{n-1} \ldots x_0)$. Then we write $[x]_k$ for the $(k+1)$th bit x_k.

Theorem 12. *Let π be an arbitrary variable ordering on X_n. Then there exists an integer $a \in \{0, \ldots, 2^n - 1\}$ such that any π-OBDD for $\mathrm{MUL}^a_{n-1,n}$ consists of at least $2^{\lfloor n/2 \rfloor}/121 - 1$ nodes.*

Proof. Let w.l.o.g. n be even and let the input variables for the π-OBDD be $x_{n-1}, \ldots, x_0$. Consider the set T of the first $n/2$ variables with respect to π and let B be the remaining $n/2$ variables. We now construct two sets M and N of integers in $\{0, \ldots, 2^n - 1\}$ as follows: M contains all integers $(x_{n-1} \ldots x_0)$ for which the T-components are 0 (i.e., $x_i = 0$ if $x_i \in T$), and N contains all integers $(x_{n-1} \ldots x_0)$ for which the B-components are 0. Note that any integer in $\{0, \ldots, 2^n - 1\}$ can be uniquely expressed as $p + q$ for $p \in M$ and $q \in N$.

Our goal is to find an appropriate constant a and two subsets $M' \subseteq M$ and $N' \subseteq N$ with the following property: for any distinct q, q' in N', there exists $p \in M'$ such that $a(p+q)$ and $a(p+q')$ differ in the nth bit. More formally,

$$\forall q, q' \in N', q \neq q' \exists p \in M' : \ [a(p+q)]_{n-1} \neq \left[a(p+q')\right]_{n-1}. \tag{4}$$

Since q and q' are determined only by the top variables and p is determined by the bottom variables, it follows that the $2^{n/2}$ assignments of constants to the top variables yield at least $\left|N'\right|$ different subfunctions. Therefore, we conclude from Lemma 6 that any π-OBDD for $\mathrm{MUL}^a_{n-1,n}$ has size at least $2N' - 1$.

It remains to bound N'. Let $\varepsilon = \frac{16}{17}$ and $k = n/2 - 6$. Then

$$|M| = |N| = 2^{n/2} = 2 \cdot 2^{k+1} \cdot 16 > 2 \cdot \left(2^{k+1} - 1\right) \cdot \frac{\varepsilon}{1 - \varepsilon}.$$

By Lemma 11, there exists an integer $a \in \{0, \ldots, 2^n - 1\}$ such that $f_a^k(M)$ and $f_a^k(N)$ contain at least $(2\varepsilon - 1)2^k = 15/17 \cdot 2^k$ elements each. We fix this a, define $f = f_a^k$ and continue to determine appropriate M' and N'.

As an intermediate step, we choose M^* and N^* to be minimal subsets of M and N, respectively, such that $f(M^*)$ and $f(N^*)$ contain exactly $13/17 \cdot 2^{k-1}$ even elements. Such sets exist, since at most 2^{k-1} of the 2^k possible function values are odd, and thus at least $15/17 \cdot 2^k - 2^{k-1} = 13/17 \cdot 2^{k-1}$ elements in M and N, respectively, have distinct and even function values under f. Note that because we required M^* and N^* to be minimal, f is injective on M^* and N^*.

The following observation is crucial for the remaining proof: for any $p \in M^*$ and any $q \in N^*$, the kth bit of $f(p) + f(q)$ has the same value as the nth bit of $a(p+q)$. Or formally

$$[f(p) + f(q)]_{k-1} = [a(p+q)]_{n-1}. \tag{5}$$

The reason for this is that the least significant bits of $f(p)$ and $f(q)$ are both zero (since these values are even). Recalling that the division executed by f is in fact a right-shift by $n-k$ bits, we obtain $[a \cdot p]_{n-k} = [a \cdot q]_{n-k} = 0$. Therefore, the bits of $ap + aq$ with higher index than $n-k$ are not influenced by a carry bit resulting from the addition of the less significant bits $([a \cdot p]_{n-k} \ldots [a \cdot p]_0) + ([a \cdot q]_{n-k} \ldots [a \cdot q]_0)$. This means that $f(p) + f(q)$ has in all bits (except possibly the least significant one) the same value as $a(p+q)$ in the bits with indices $n-1, \ldots, n-k$, and Eq. (5) is true.

In order to satisfy property (4) it is sufficient by the above arguments that the sets M' and N' are subsets of M^* and N^* and that the following is true:

$$\forall q, q' \in N', q \neq q' \exists p \in M' : \; [f(p) + f(q)]_{k-1} \neq [f(p) + f(q')]_{k-1}. \tag{6}$$

We let $M' = M^*$ and

$$N' = \left\{ q \in N^* \;\middle|\; \exists p \in M' : f(q) = 2^k - f(p) \right\}. \tag{7}$$

In order to prove claim (6), let q and q' be arbitrary distinct elements from N'. Since q and q' are in N^* and therefore have distinct function values under f, we may assume w.l.o.g. that

$$0 < \big(f(q') - f(q)\big) \bmod 2^k \leqslant 2^{k-1} \tag{8}$$

(otherwise we achieve this by exchanging q and q'). By construction, there exists a $p \in M'$ with $f(p) + f(q) = 2^k$. For this p, obviously the kth bit of $f(p) + f(q)$, that is $[f(p) + f(q)]_{k-1}$, equals 0. But on the other hand, by inequations (8), the value of $\big(f(p) + f(q')\big) \bmod 2^k$ is in $\{2^{k-1}, \ldots, 2^k - 1\}$. This means that the kth bit of $f(p) + f(q')$ equals 1, and thus claim (6) is proven.

So far, we have constructed subsets $M' \subseteq M$ and $N' \subseteq N$, which satisfy claim (4), implying by our arguments a lower bound on the π-OBDD-size of $2|N'| + 1$. All that is left to do, is to give an appropriate lower bound on $|N'|$. Recall the definition of N' in (7), and that $f(M') = f(M^*)$ and $f(N^*)$ contain $\frac{13}{17} \cdot 2^{k-1}$ even elements each. Because for any even $f(p)$ also $2^k - f(p)$ is even, the set $L := \{2^k - f(p) \mid p \in M'\}$ contains $\frac{13}{17} \cdot 2^{k-1}$ even elements, too. We now let K be the set of 2^{k-1} even elements in $\{0, \ldots, 2^k - 1\}$. Since $f(N^*) \subseteq K$, $L \subseteq K$, and f is injective on N', we have

$$\begin{aligned} |N'| &= |f(N')| = |f(N^*) \cap L| \geqslant |f(N^*)| + |L| - |K| \\ &= \tfrac{26}{17} \cdot 2^{k-1} - 2^{k-1} = \tfrac{9}{17} \cdot 2^{k-1}. \end{aligned}$$

By the choice of k we obtain that $2\left|N'\right| - 1$ (and thus also the size of the π-OBDD) is bounded below by

$$2 \cdot \frac{9}{17} \cdot 2^{k-1} - 1 = \frac{9}{17} \cdot 2^{n/2-6} - 1 > \frac{2^{n/2}}{121} - 1 \qquad \square$$

This theorem shows the general result for $\mathrm{MUL}_{n-1,n}$ of Theorem 3 by the following straightforward observation: if for some constant B and some variable ordering π there exists an integer a for which the π-OBDD-size of $\mathrm{MUL}^a_{n-1,n}$ is at least $B + 1$, then the π-OBDD-size of $\mathrm{MUL}_{n-1,n}$ is at least $2B$. This is because in any OBDD computing $\mathrm{MUL}_{n-1,n}(x, y)$ either the input x or the input y may be fixed to the constant a. In both cases the resulting OBDD contains at least $B - 1$ inner nodes, not counting those for variables fixed to constants (since they may be deleted without changing the function). According to the last theorem the OBDD for $\mathrm{MUL}_{n-1,n}$ has a size of at least $2 \cdot (2^{\lfloor n/2 \rfloor}/121 - 2)$, which proves the main result (Theorem 3).

3. Upper bounds

Bryant's proof as well as ours on the lower bounds for the OBDD-size of $\mathrm{MUL}_{n-1,n}$ have both in common that they rely only on the existence of a constant factor a for each variable ordering π, for which $\mathrm{MUL}^a_{n-1,n}$ leads to a large π-OBDD representation. If one would want to improve our lower bound, then there might be two possibilities. One could either try to consider multiple values for a or improve the lower bound for the π-OBDD-size of $\mathrm{MUL}^a_{n-1,n}$ for an appropriately chosen constant a. We now show that at least the latter approach cannot yield significant better lower bounds because Theorem 12 is optimal up to a small constant factor:

Theorem 13. *There is a variable ordering π such that for any integer $a \in \{0, \dots, 2^n - 1\}$ the π-OBDD-size of $\mathrm{MUL}^a_{n-1,n}$ is less than $3 \cdot 2^{n/2}$.*

In the remainder of this section we prove the upper bounds of the Theorems 5 and 13. Generally, an upper bound on the π-OBDD-size of a Boolean function f can be proved as follows: one describes an algorithm which queries the variables in the order determined by the variable ordering π. In the ith step the variable $\pi(i)$ is queried and after the query the algorithm stores a state-value q_i which depends only on the previous stored value and the result of the variable query. Each possible stored state-value q_i of the algorithm corresponds to a node labeled with the variable $\pi(i + 1)$ and thus the sum of the number of possible state-values q_i over all $0 \leqslant i \leqslant n$ is the number of OBDD-nodes (q_0 is the unique starting state corresponding to the root of the OBDD and the two possible final state values $q_{n+1} \in \{0, 1\}$ correspond to the sinks of the OBDD). It is obvious how to construct the OBDD corresponding to such an algorithm.

Proof of Theorem 13. Let π be the variable ordering with $\pi(i) = x_{i-1}$, $1 \leqslant i \leqslant n$, and let $m = \lfloor n/2 \rfloor$. In the first half of the steps the algorithm queries the variables $x_0, \dots, x_{m-1}$ and stores in its state the value of all previously queried variables. Therefore, 2^i state-values are necessary after the ith step and the first $m + 1$ levels of the π-OBDD form a complete binary tree whose leaves are 2^m nodes labeled with x_m.

Let for $0 \leqslant k \leqslant n$

$$s_k = \left(a \cdot \sum_{i=0}^{k-1} 2^i \cdot x_i \right) \operatorname{div} 2^k \quad \text{and} \quad s'_k = s_k \bmod 2^{n-k}.$$

Obviously, s'_m is uniquely determined by $x_0, \dots, x_{m-1}$ and each of the 2^m OBDD-nodes marked with x_m can be uniquely associated with the corresponding s'_m-value. In other words, after the mth step the algorithm stores the value s'_m. We now show that for $k = m, \dots, n-2$ the value s'_{k+1} is uniquely determined by s'_k and the value of x_k. Hence, our algorithm can successively compute $s'_{m+1}, \dots, s'_{n-1}$ by querying the variables $x_m, \dots, x_{n-2}$. We will also see that $\mathrm{MUL}^a_{n-1,n}(x)$ is uniquely determined by s'_{n-1} and the value of x_{n-1}. Hence, once s'_{n-1} is computed, the algorithm can determine the correct result by querying x_{n-1}. Note that $(r + q \cdot 2^k) \operatorname{div} 2^k = r \operatorname{div} 2^k + q$ for any two integers q, r. Hence,

$$\begin{aligned} s_{k+1} &= \left(\left(a \cdot \sum_{i=0}^{k} 2^i \cdot x_i \right) \operatorname{div} 2^{k+1} \right) \\ &= \left(\left(a \cdot 2^k \cdot x_k + a \cdot \sum_{i=0}^{k-1} 2^i \cdot x_i \right) \operatorname{div} 2^k \right) \operatorname{div} 2 = (a \cdot x_k + s_k) \operatorname{div} 2. \end{aligned}$$

Thus,

$$\begin{aligned} s'_{k+1} &= s_{k+1} \bmod 2^{n-k-1} = \Big((a \cdot x_k + s_k \bmod 2^{n-k}) \operatorname{div} 2 \Big) \bmod 2^{n-k-1} \\ &= \big((a \cdot x_k + s'_k) \operatorname{div} 2 \big) \bmod 2^{n-k-1}. \end{aligned} \tag{9}$$

This shows that s'_{k+1} is uniquely determined by s'_k and x_k. Analogously it follows that

$$\mathrm{MUL}^a_{n-1,n}(x) = \left(\left(a \sum_{i=0}^{n-1} 2^i x_i \right) \operatorname{div} 2^{n-1} \right) \bmod 2 = (a \cdot x_{n-1} + s'_{n-1}) \bmod 2 \tag{10}$$

is uniquely determined by s'_{n-1} and x_{n-1}.

It remains to bound the size of the OBDD defined by this algorithm. The first $m+1$ levels (i.e. the nodes labeled with $x_0, \dots, x_m$) form a complete binary tree and thus consist of $2^{m+1} - 1$ nodes. The number of x_k-nodes with $k > m$ is the number of possible values for s'_k. Using $s'_k \in \{0, \dots, 2^{n-k} - 1\}$ for $k = m+1, \dots, n-1$ and counting also the two sinks we obtain the following upper bound on the π-OBDD-size of $\mathrm{MUL}^a_{n-1,n}$:

$$2^{m+1} + 1 + \sum_{k=m+1}^{n-1} 2^{n-k} = 2^{m+1} + 1 + \sum_{i=1}^{n-m-1} 2^i = 2^{m+1} + 2^{n-m} - 1.$$

Since $m = \lfloor n/2 \rfloor$, this simplifies for even n to $2^{n/2+1} + 2^{n/2} - 1 = 3 \cdot 2^{n/2} - 1$ and for odd n to

$$2^{\lfloor n/2 \rfloor + 1} + 2^{\lceil n/2 \rceil} - 1 = 2 \cdot 2^{(n+1)/2} - 1 = 2 \cdot \sqrt{2} \cdot 2^{n/2} - 1 < 3 \cdot 2^{n/2} - 1. \qquad \square$$

We can use the upper bound of Theorem 13 directly in order to obtain an $O(2^{3n/2})$ upper bound on the OBDD-size of $\mathrm{MUL}_{n-1,n}$: The corresponding algorithm just queries one factor y completely and then computes $\mathrm{MUL}^{y}_{n-1,n}(x)$ for the other factor x. The naive approach requires that the values of all y-variables are kept in memory once they are known. However, it is easy to see that after querying the k least significant bits of x, the algorithm can "forget" the value of the k most significant bits of y. This yields a more space-efficient algorithm and an upper bound of $O(2^{4n/3})$ on the OBDD-size of $\mathrm{MUL}_{n-1,n}$.

Proof of Theorem 5. Let $X = \{x_0, \ldots, x_{n-1}\}$, $Y = \{y_0, \ldots, y_{n-1}\}$ and let π be the variable ordering of $X \cup Y$ with

$$(\pi(1), \ldots, \pi(2n)) = (y_0, \ldots, y_{n-1}, x_0, \ldots, x_{n-1}).$$

Similar as in the proof of Theorem 13 we describe an algorithm which queries all bits in the order defined by π and which stores after each query a value which corresponds to a node of an OBDD.

Let $m = \lceil n/3 \rceil - 1$. First, the algorithm queries the variables $y_0, \ldots, y_{n-1}, x_0, \ldots, x_{m-1}$ and stores the values of all queried variables. That is, the upper part of the OBDD is a complete binary tree whose 2^{n+m} leaves are the OBDD-nodes labeled with x_m. Let now s_k and s'_k be defined as in the proof of Theorem 13, i.e.

$$s_k = \left(y \cdot \sum_{i=0}^{k-1} 2^i \cdot x_i \right) \operatorname{div} 2^k \quad \text{and} \quad s'_k = s_k \bmod 2^{n-k}.$$

In the proof of Theorem 13 we have already shown that s'_{k+1} is uniquely determined by s'_k, y and x_k. This followed from (9) which states that

$$s'_{k+1} = \big((y \cdot x_k + s'_k) \operatorname{div} 2\big) \bmod 2^{n-k-1}.$$

However, since the term on the right-hand side is taken modulo 2^{n-k-1}, we could have taken y modulo 2^{n-k} beforehand. This shows that s'_{k+1} is independent from the bits $y_{n-k}, \ldots, y_{n-1}$. Hence, s'_{k+1} is in fact uniquely determined by s'_k and the values of the variables $y_0, \ldots, y_{n-k-1}$ and x_k. Similarly, it can be seen from (10), that $\mathrm{MUL}_{n-1,n}(x, y)$ is uniquely determined by s'_{n-1}, y_0 and x_{n-1}.

Therefore, before querying x_k, $0 \leqslant k \leqslant n-2$, it suffices for our algorithm to store the value of s'_k as well as the values of $y_0, \ldots, y_{n-k-1}$ in order to compute s'_{k+1} by the next query x_k. Then, once x_k is queried, the algorithm can "forget" y_{n-k-1}. Before the last query (i.e. the x_{n-1}-query), s'_{n-1} as well as y_0 are stored. Then $\mathrm{MUL}_{n-1,n}(x, y)$ is uniquely determined by the outcome of the last variable query.

It remains to bound the size of the OBDD defined by this algorithm by bounding the number of possible states after each step of the algorithm. For s'_k there are 2^{n-k} possible values and $(y_0, \ldots, y_{n-k-1})$ can take 2^{n-k} values, too. Hence, for $k \geqslant m+1$, 2^{2n-2k} x_k-vertices are sufficient for the OBDD. Adding the two sinks as well as the complete binary tree of size $2^{n+m+1} - 1$ for the y-vertices and the x_k-vertices with $k \leqslant m$, we obtain an OBDD-size of

$$\begin{aligned} 2^{n+m+1} + 1 + \sum_{k=m+1}^{n-1} 2^{2n-2k} &= 2^{n+m+1} + 1 + 4 \cdot \sum_{i=0}^{n-m-2} 2^{2i} \\ &= 2^{n+m+1} + 1 + 4 \cdot \frac{4^{n-m-1} - 1}{3} < 2^{n+m+1} + \frac{4^{n-m}}{3}. \end{aligned}$$

Since $m = \lceil n/3 \rceil - 1$, we have $m = (n + \tau)/3 - 1$ for some $\tau \in \{0, 1, 2\}$. This yields an upper bound of

$$2^{(4/3)n+\tau/3} + \frac{4^{(2/3)n-\tau/3+1}}{3} = 2^{(4/3)n} \cdot \left(2^{\tau/3} + \frac{4^{1-\tau/3}}{3}\right).$$

A simple case distinction shows that the term in parentheses is maximal for $\tau = 0$ and thus is bounded by $\frac{7}{3}$. Therefore, the OBDD constructed here has at most $(7/3) \cdot 2^{(4/3)n}$ vertices. □

Acknowledgments

The author thanks Ingo Wegener for valuable comments on early drafts of the proofs.

References

[1] B. Bollig, I. Wegener, Improving the variable ordering of OBDDs is NP-complete, IEEE Trans. Comput. 45 (1996) 993–1002.

[2] B. Bollig, I. Wegener, Asymptotically optimal bounds for OBDDs and the solution of some basic OBDD problems, in: Proceedings of the 25th International Colloquium on Automata, Languages and Programming (ICALP), 2000, pp. 187–198.

[3] Y. Breitbart, H.B. Hunt III, D.J. Rosenkrantz, On the size of binary decision diagrams representing boolean functions, Theoret. Comput. Sci. 145 (1995) 45–69.

[4] R.E. Bryant, Graph-based algorithms for boolean function manipulation, IEEE Trans. Comput. C-35 (1986) 677–691.

[5] R.E. Bryant, On the complexity of VLSI implementations and graph representations of boolean functions with applications to integer multiplication, IEEE Trans. Comput. 40 (1991) 205–213.

[6] J.L. Carter, M.N. Wegman, Universal classes of hash functions, J. Comput. System Sci. 18 (1979) 143–154.

[7] M. Dietzfelbinger, Universal hashing and *k*-wise independent random variables via integer arithmetic without primes, in: Proceedings of the 13the Annual Symposium on Theoretical Aspects of Computer Science (STACS), vol. 1046, Lecture Notes in Computer Science, 1996, pp. 569–580.

[8] M. Dietzfelbinger, T. Hagerup, J. Katajainen, M. Penttonen, A reliable randomized algorithm for the closest-pair problem, J. Algorithms 25 (1997) 19–51.

[9] Y. Mansour, N. Nisan, P. Tiwari, The computational complexity of universal hashing, Theoret. Comput. Sci. 107 (1993) 121–133.

[10] S. Ponzio, A lower bound for integer multiplication with read-once branching programs, in: Proceedings of the 27th Annual ACM Symposium on Theory of Computing (STOC), 1995, pp. 130–139.

[11] M. Sauerhoff, P. Woelfel, Time-space tradeoff lower bounds for integer multiplication and graphs of arithmetic functions, in: Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC), 2003, pp. 186–195.

[12] D. Sieling, On the existence of polynomial time approximation schemes for OBDD minimization, in: Proceedings of the 15the Annual Symposium on Theoretical Aspects of Computer Science (STACS), 1998, pp. 205–215.

[13] D. Sieling, I. Wegener, NC-algorithms for operations on binary decision diagrams, Parallel Processing Lett. 48 (1993) 139–144.

[14] I. Wegener, Branching Programs and Binary Decision Diagrams—Theory and Applications, SIAM, Philadelphia, PA, 2000.

[15] M.N. Wegman, J.L. Carter, New classes and applications of hash functions, in: Proceedings of the 20th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 1979, pp. 175–182.

[16] P. Woelfel, Efficient strongly universal and optimally universal hashing, in: Mathematical Foundations of Computer Science: 24th International Symposium (MFCS), vol. 1672, Lecture Notes in Computer Science, 1999, pp. 262–272.

[17] P. Woelfel, New bounds on the OBDD-size of integer multiplication via universal hashing, in: Proceedings of the 18th Annual Symposium on Theoretical Aspects of Computer Science (STACS), vol. 2010, Lecture Notes in Computer Science, 2001, pp. 563–574.

[18] B. Yang, Y.-A. Chen, R.E. Bryant, D.R. O'Hallaron, Space- and time-efficient BDD construction via working set control, in: Proceedings of the Asia South-Pacific Design Automation Conference (ASPDAC), 1998, pp. 423–432.