Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 107 (2004)

New Results on the Complexity of the Middle Bit of Multiplication

Ingo Wegener* Philipp Woelfel†

Abstract

It is well known that the hardest bit of integer multiplication is the middle bit, i.e. $\mathrm{MUL}_{n-1,n}$. This paper contains several new results on its complexity. First, the size s of randomized read-k branching programs, or, equivalently, its space ($\log s$) is investigated. A randomized algorithm for $\mathrm{MUL}_{n-1,n}$ with $k = \mathcal{O}(\log n)$ (implying time $\mathcal{O}(n \log n)$), space $\mathcal{O}(\log n)$ and error probability n^{-c} for arbitrarily chosen constants c is presented.

Second, the size of general branching programs and formulas is investigated. Applying Nechiporuk's technique, lower bounds of $\Omega\big(n^{3/2}/\log n\big)$ and $\Omega\big(n^{3/2}\big)$, respectively, are obtained. Moreover, by bounding the number of subfunctions of $\mathrm{MUL}_{n-1,n}$, it is proven that Nechiporuk's technique cannot provide larger lower bounds than $\mathcal{O}(n^{7/4}/\log n)$ and $\mathcal{O}(n^{7/4})$, respectively.

1 Introduction

Integer multiplication is certainly one of the most important functions for computer science. Therefore, a lot of effort been spent in designing good algorithms and small and shallow circuits and in determining its complexity. Most algorithms such as the Schönhage-Strassen method require at least linear space in order to compute the product of two n-bit integers. On the other hand, the school method can easily be implemented with $\mathcal{O}(\log n)$ space, but for the price of a higher, almost quadratic $\Omega(n^2/\log n)$ running time. This is not surprising, because even in general nonuniform computation models such as branching programs a time-space product of $\Omega(n^2)$ is necessary (see Dietzfelbinger, 1996, and Mansour, Nisan and Tiwari, 1993). On the other hand, in nonuniform computation models, any boolean function $f : \{0,1\}^n \to \{0,1\}^m$ can be implemented in $\mathcal{O}(n)$ time and $\mathcal{O}(m)$ space by simply using table-lookups. It is interesting, though, that regarding time-space tradeoffs in nonuniform models, the school method is the most general algorithm because for any $\log n \le k \le n$ it can be implemented in $\mathcal{O}(k)$ space and $\mathcal{O}(n^2/k)$ time.

*Department of Computer Science, University Dortmund, D-44221 Dortmund. ingo.wegener@cs.uni-dortmund.de

†Department of Computer Science, University Dortmund, D-44221 Dortmund. philipp.woelfel@cs.uni-dortmund.de

ISSN 1433-8092

However, the complexity of computing single output bits of integer multiplication is far from being as well bounded as that of computing all output bits. The main reason is probably, that for single output bit boolean functions $f : \{0,1\}^n \to \{0,1\}$ the known lower bound techniques are yet too weak to obtain better than only slightly super-linear time-space products and even such lower bounds are rare (see Ajtai, 1999, and Beame, Saks, Sun and Vee, 2003, for the best lower bounds). Nevertheless, good space lower bounds can been shown in more restricted models.

Definition 1 A *branching program* for a function $f : \{0,1\}^n \to \{0,1\}^m$ is a directed acyclic graph with one source and one sink and the following properties. The internal nodes are marked with the n input variables $x_1, \ldots, x_n$ and have two outgoing edges, a 0-edge and a 1-edge. Some edges may additionally be marked with a pair (y_i, c) where y_i, $1 \leq i \leq m$, is an output variable and c is either 0 or 1. Then any assignment $(a_1, \ldots, a_n)$ defines a *computation path*, which starts at the source and leaves any internal node marked with x_i via the a_i-edge. Finally, each computation path for an input $(a_1, \ldots, a_n)$ for all $1 \leq i \leq m$ passes over exactly one edge marked with (y_i, b_i), $b_i \in \{0,1\}$, in such a way that $f(x_1, \ldots, x_n) = (b_1, \ldots, b_m)$. A *randomized* branching program may use additional *randomized nodes* with arbitrary outdegree at which the outgoing edge a computation path uses is chosen randomly. In this case, the computation of f may err with a certain probability.

The *size* of a branching program is the number of its nodes, its *space* is the logarithm of its size, and its *time* or *length* is the length of the longest computation path. The *branching program size* of f is the size of a branching program computing f with minimal size. A branching program is called *read-k*, if any source-to-sink path contains each variable at most k times.

If a branching program computes only one output bit (i.e. $m = 1$), then it is more convenient to define it in such a way that it has two sinks instead of marked output edges — a 0-sink and a 1-sink, corresponding to the two possible outputs. In the following discussion we restrict ourselves to branching programs computing only one output bit.

Branching programs are such a general computation model that Turing machines or register machines can be simulated by them with essentially the same time and space resources. On the other hand, it is easy to see that branching programs can be simulated by *nonuniform* Turing machines or nonuniform register machines using asymptotically as much time and space as the branching program (at least as long as the space is $\Omega(\log n)$).

While using counting arguments one can show that almost all boolean function have an exponential branching program size, such lower bounds cannot be proven for explicitly defined functions (i.e. functions in NP). Since 40 years (more precisely, since Nechiporuk, 1966) the best lower bounds for explicitly defined functions are of the order $n^2/\log^2 n$ and all lower bounds close to n^2 have been proven with the same arguments, namely with Nechiporuk's technique.

However, restrictions on e.g. the number of queries of each variable have led

to good lower bound results. It should be pointed out that the read-k restriction is syntactic in the sense that it applies to graph-theoretical paths in a branching program and not only to computation paths. While —as we have mentioned earlier— lower bounds for time-restricted branching programs with one output bit are hard to prove (due to the fact that a time restriction is semantical) and therefore rarely exist, good lower bound techniques for read-k branching programs have been known for quite some time (the first superpolynomial size lower bounds were proven by Borodin, Razborov and Smolensky, 1993, and Okol'nishnikova, 1993, and an overview of several other results can be found in the monograph of Wegener, 2000). There are also good lower bound results for the computation of single output bits of natural functions as e.g. integer multiplication.

Let $\mathrm{MUL}_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^{2n}$ be the function mapping two n-bit integers to their $2n$-bit product and let $\mathrm{MUL}_{i,n} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ be the boolean function which computes the ith least significant bit of the product of two n-bit integers, i.e. $(x_{n-1} \dots x_0, y_{n-1} \dots y_0) \mapsto z_i$, where $z_{2n-1} \dots z_0 = \mathrm{MUL}_n(x_{n-1} \dots x_0, y_{n-1} \dots y_0)$. It is well known that the "middle bit", that is the function $\mathrm{MUL}_{n-1,n}$, is the most difficult to compute. More precisely, $\mathrm{MUL}_{i,n}$ can be computed by any branching program for $\mathrm{MUL}_{j-1,j}$, where $j = \min\{i+1,\, 2n-i-1\}$, if one reorders the input bits and replaces some inputs either by their negations or by the constants 0 or 1 (see Bollig and Wegener, 1996).

For read-once branching programs computing $\mathrm{MUL}_{n-1,n}$, Ponzio (1995) was the first to prove a superlogarithmic space lower bound of $\Omega(\sqrt{n})$, and later, even a linear lower bound was obtained by Bollig and Woelfel (2001). Only recently Sauerhoff and Woelfel (2003) proved a space lower bound of $\Omega\big(\log\big(1/\epsilon(n)\big) \cdot k(n)^{-2} \cdot 3^{-2k(n)}\big)$ for randomized read-$k(n)$ branching programs, where $\epsilon(n)$ is the maximal error probability. Thus, if we restrict ourselves to (weakly) exponentially small error, then a log-space computation of $\mathrm{MUL}_{n-1,n}$ requires $k(n) = \Omega(\log n)$.

These results as well as our inability to design fast deterministic log-space algorithms for computing single product bits indicate that $\mathrm{MUL}_{n-1,n}$ is a "hard" function. Therefore, it is quite surprising that one can approximate it even with read-once branching programs in log-space and with polynomially small error, as shown by Sauerhoff and Woelfel (2003). (Approximating f with error ϵ means computing a function f' which equals f on all but an ϵ-fraction of the inputs.) In addition, it is easy to see that using arithmetics modulo a randomly chosen prime, one can verify in logarithmic space and with a small error probability whether the product of two integers equals some given output (Ablayev and Karpinski, 2003). However, computing the middle output bit of integer multiplication is harder than approximating it and apparently also harder than verifying all output bits. Therefore, the following upper bound which we prove in this paper is also rather surprising.

Theorem 1 *For any constant c and for $k(n) = \mathcal{O}(\log n)$ there is a randomized read-$k(n)$ branching program for $\mathrm{MUL}_{n-1,n}$ of size $n^{\mathcal{O}(1)}$ with a two-sided error probability of at most n^{-c}.*

Hence, querying each input bit only logarithmically many times allows a log-space computation of any product bit with a polynomially small error probability. This is the first upper bound on the time-space product of randomized computations of product bits which is better than $\Omega(n^2)$ and it shows that using randomization and a very moderate error probability, we can at least reach the theoretical lower bound for deterministic computations.

We remark that due to the fact that the algorithm we present merely uses arithmetics over $\mathcal{O}(\log n)$-bit registers, the same time and space bounds hold e.g. for nonuniform word-RAMs with word size $\Theta(\log n)$. With constant word size, the time increases by an $\mathcal{O}(\log n)$-factor. Furthermore, the only reason why the algorithm is nonuniform is that it requires to choose a prime number randomly.

After proving the upper bound in the following section, we consider the size of unrestricted branching programs and formulas for $\mathrm{MUL}_{n-1,n}$. A *boolean formula* is a circuit with fan-out 1. Here, we consider formulas over the basis B_2 of all binary operations (in short: B_2-formulas). Note that for formulas with restricted basis, e.g. $\{\wedge, \vee\}$, better results than those presented below are known.

As we have mentioned before, today, the best lower bounds one can prove for explicitly defined functions are of the order $n^2/\log^2 n$ for the branching program size and of the order $n^2/\log n$ for the B_2-formula size. All lower bounds in this order of magnitude were proven with the method of Nechiporuk (1966), and it is known that this method cannot yield better lower bounds. Moreover, most lower bounds were proven for functions which are nor really interesting for implementations or hardware realization. This is obviously different for integer multiplication, and therefore the branching program and formula size of $\mathrm{MUL}_{n-1,n}$ should be investigated.

Theorem 2

1. *Any branching program for $\mathrm{MUL}_{n-1,n}$ has at least $\Omega\big(n^{3/2}/\log n\big)$ nodes and any B_2-formula for $\mathrm{MUL}_{n-1,n}$ has size at least $\Omega(n^{3/2})$.*
2. *Using Nechiporuk's technique, it is not possible to prove a better lower bound than $\mathcal{O}\big(n^{7/4}/\log n\big)$ and $\mathcal{O}\big(n^{7/4}\big)$ for the branching program and B_2-formula size of $\mathrm{MUL}_{n-1,n}$, respectively.*

The second part of the theorem tells us that we should not attempt to prove better lower bounds than $\mathcal{O}(n^{7/4})$ until we are aware of a new lower bound technique.

In the following section, we first introduce some notation and then present the randomized space-bounded algorithm for $\mathrm{MUL}_{n-1,n}$ and prove Theorem 1. In Section 3 we count the number of subfunctions of $\mathrm{MUL}_{n-1,n}$ in order to obtain Theorem 2.

2 A Randomized Algorithm for Space-Bounded Computation of Product Bits

In order to compute the product of two n-bit integers, we will consider the corresponding problem of adding n n-bit integers, as it is done in the school method for multiplication. We use the operations mod and div, where mod is the usual modulo operation and $x \operatorname{div} y := \lfloor x/y \rfloor$ is defined for rational numbers x and y. Let x be a rational number represented by the finite bit string $x_n \dots x_0.x_{-1} \dots x_{-m}$, i.e. $x = \sum_{i=-m}^{n} x_i \cdot 2^i$. Then we say that x_i is the ith bit of x. In order to be able to address an integer y represented by the bitstring $y_{j-i} \dots y_0 := x_j \dots x_i$, $j \geq i$, we use the following notion:

$$\begin{aligned} \langle x \rangle_i^j &= x \operatorname{div} 2^i \bmod 2^{j-i+1} \\ &\left(= x \bmod 2^{j+1} \operatorname{div} 2^i \text{ for } j \geq 0\right). \end{aligned}$$

Note that for $j \geq 0$ the operation $x \bmod 2^{j+1}$ can be viewed as zeroing out all bits in the binary representation of x which are farther left than the jth bit. The division $x \operatorname{div} 2^i$ is best imagined as a right shift of the binary representation of x by i bit positions (note that in the case $i \leq 0$, the operation $x \operatorname{div} 2^i$ is in fact a multiplication of x with the integer 2^{-i} and thus can be viewed as a left shift of the binary representation by $|i|$ bit positions).

We abbreviate $\langle x \rangle_i^i$ by $\langle x \rangle_i$. Finally, we denote by $\mathbb{Z}_r$ the set of integers $\{0, \dots, r-1\}$.

Assume that we want to compute k consecutive bits of the sum of m integers $w_1, \dots, w_m \in \mathbb{Z}_{2^n}$. Let for $a \geq 0$, $k \geq 1$

$$\begin{aligned} S_{a,k}(w_1, \dots, w_m) &= \langle w_1 + \dots + w_m \rangle_a^{a+k-1} \\ &= \big((w_1 + \dots + w_m) \bmod 2^{a+k}\big) \operatorname{div} 2^a. \end{aligned}$$

For $x \in \mathbb{Z}_{\geq 0}$ let $H(x) = x \operatorname{div} 2^a$ be the "high part" of x, $M(x) = \langle x \rangle_{a-\ell}^{a-1} = x \bmod 2^a \operatorname{div} 2^{a-\ell}$ be the "middle part" and $L(x) = x \bmod 2^{a-\ell}$ be the "low part". Then $x = 2^a \cdot H(x) + 2^{a-\ell} \cdot M(x) + L(x)$. In the following we try to compute an approximation of the sum $S_{a,k}$. Recall that an *approximation* of a function f is a function g such that for a large fraction of the inputs the function values of f and g are equal. We can get an approximation of $S_{a,k}$ by summing only the high and middle part of the integers and ignoring the low part. Thus, let $H = \sum_{i=1}^{m} H(w_i)$, $M = \sum_{i=1}^{m} M(w_i)$ and $L = \sum_{i=1}^{m} L(w_i)$. Note that

$$S_{a,k}(w_1, \dots, w_m) = \left\langle 2^a H + 2^{a-\ell} M + L \right\rangle_a^{a+k-1}.$$

(This value does not depend on ℓ). In order to approximate $S_{a,k}(w_1, \dots, w_m)$, we may compute

$$\begin{aligned} S_{a,k,\ell}^{**}(w_1, \dots, w_m) &= \big(2^\ell \cdot H + M\big) \bmod 2^{k+\ell} \\ &= \left(\sum_{i=1}^{m} w_i \operatorname{div} 2^{a-l} \right) \bmod 2^{k+\ell} \end{aligned}$$

and let the approximation be

$$\begin{aligned} S^*_{a,k,\ell}(w_1,\ldots,w_m) &= S^{**}_{a,k,\ell}(w_1,\ldots,w_m) \operatorname{div} 2^\ell \\ &= \left(2^\ell H + M\right) \bmod 2^{\ell+k} \operatorname{div} 2^\ell = \left\langle 2^\ell \cdot H + M \right\rangle_\ell^{\ell+k-1} \\ &= \left\langle \sum_{i=1}^{m} w_i \operatorname{div} 2^{a-\ell} \right\rangle_\ell^{\ell+k-1}. \end{aligned}$$

In order to simplify the notation, we abbreviate $S_{a,k}(w_1,\ldots,w_m)$ by $S_{a,k}$ and analogously use the abbreviations $S^*_{a,k,\ell}$ and $S^{**}_{a,k,\ell}$, if it is clear from the context which integers we use for the sums. To compare $S_{a,k}$ and $S^*_{a,k,\ell}$ it may also be useful to note that

$$S^*_{a,k,\ell}(w_1,\ldots,w_m) = \left\langle 2^a \cdot H + 2^{a-\ell} \cdot M \right\rangle_a^{a+k-1}.$$

2.1 The Approximation Lemma

The idea is that ignoring the low part of the sum, we can compute the approximation $S^*_{a,k,\ell}$ of $S_{a,k}$ with less space if $k+\ell$ is small. On the other hand, in most cases the carry induced by the low part should not influence the more significant bits of the sum if ℓ is not too small. If ℓ is somewhat larger than $\log n$, then the carry induced by L (which has a value of at most n) can only carry over all the following ℓ bits, if they have a value of at least $2^\ell - n$. We remark that this was already observed by Sauerhoff and Woelfel (2003). However, the following lemma is more general. E.g., it tells us how we can detect the situations in which the approximation is correct and will later help us to obtain the true value even if the algorithm errs.

Lemma 1 *Let $m \geq 2$, $\ell \geq \lceil \log m \rceil + 1$, $a, k \geq 1$, and $z = S^*_{a,k,\ell}(w_1,\ldots,w_m)$.*

(a) $S_{a,k}(w_1,\ldots,w_m) \in \left\{z,\ (z+1) \bmod 2^k\right\}$ *and*

(b) $S_{a,k}(w_1,\ldots,w_m) \neq z$ *if and only if the following two inequalities are both true:*

(i) $S^{**}_{a,k,\ell}(w_1,\ldots,w_m) \bmod 2^\ell > 2^\ell - m$

(ii) $S_{a-\ell,\ell}(w_1,\ldots,w_m) < m$.

For the proof of this lemma, the following obvious statement is helpful.

Proposition 1 *Let $a, b \in \mathbb{Z}_{\geq 0}$ and $k \in \mathbb{Z}_{>0}$. Then*

(a) $(a+b) \operatorname{div} k \in \{z,\ z+1\}$, *where* $z = a \operatorname{div} k + b \operatorname{div} k$.

(b) The following three statements are equivalent:

1. $(a+b) \operatorname{div} k = a \operatorname{div} k + b \operatorname{div} k$,
2. $a \bmod k + b \bmod k < k$,
3. $(a+b) \bmod k \geq a \bmod k$.

Proof of Lemma 1: Recall the definitions of H, M and L from above. First of all note that L is the sum of m $(a-\ell)$-bit integers and thus bounded by $m(2^{a-\ell}-1)$. Hence, $L \operatorname{div} 2^a < m \cdot 2^{-\ell}$ and thus $L \operatorname{div} 2^a = 0$ by the assumption on ℓ. Let

$$r := 2^a H + 2^{a-\ell} M + L \quad \text{and} \quad z' := 2^a H + 2^{a-\ell} M.$$

Using $L \operatorname{div} 2^a = 0$ and Proposition 1(a) we obtain

$$r \operatorname{div} 2^a \ \in \ \{z' \operatorname{div} 2^a,\, z' \operatorname{div} 2^a + 1\}.$$

Now part (a) follows because $S_{a,k} = r \operatorname{div} 2^a \bmod 2^k$ and

$$z' \operatorname{div} 2^a \bmod 2^k \ = \ (2^\ell H + M) \operatorname{div} 2^\ell \bmod 2^k \ = \ S^*_{a,k,\ell}.$$

For part (b) first note that $r \operatorname{div} 2^a \neq z' \operatorname{div} 2^a$ is equivalent to $S_{a,k} \neq S^*_{a,k,\ell}$ because $z' \operatorname{div} 2^a \bmod 2^k \ \neq \ (z' \operatorname{div} 2^a + 1) \bmod 2^k$. Furthermore, $r \bmod 2^a = (2^{a-\ell} M + L) \bmod 2^a$ and $z' \bmod 2^a = (2^{a-\ell} M) \bmod 2^a$. Hence, by Proposition 1(b) each of the following two statements is equivalent to $S_{a,k} \neq S^*_{a,k,\ell}$ (and to $r \operatorname{div} 2^a \neq z' \operatorname{div} 2^a$):

$$\big(2^{a-\ell} M\big) \bmod 2^a + L \bmod 2^a \geq \ 2^a \quad \text{and} \qquad (*)$$

$$\big(2^{a-\ell} M + L\big) \bmod 2^a < L \bmod 2^a. \qquad (**)$$

Recalling that $L < m \cdot 2^{a-\ell} < 2^a$ and dividing both sides of inequality $(*)$ by $2^{a-\ell}$, we obtain

$$M \bmod 2^\ell \ \geq \ 2^\ell - L/2^{a-\ell} \ > \ 2^\ell - m.$$

This shows that inequality (i) in part (b) of the lemma is fulfilled because $S^{**}_{a,k,\ell} \equiv M \pmod{2^\ell}$. Similarly, dividing both sides of inequality $(**)$ by $2^{a-\ell}$ yields

$$(2^{a-\ell} M + L) \bmod 2^a \operatorname{div} 2^{a-\ell} \ < \ m.$$

By definition, the left part of this inequality equals $S_{a-\ell,\ell}$. Hence, inequality (ii) is true.

Finally, assume that inequalities (i) and (ii) are both fulfilled. Then $M \bmod 2^\ell > 2^\ell - m$ and $(2^{a-\ell} M + L) \bmod 2^a \operatorname{div} 2^{a-\ell} \leq m - 1$. Multiplying with $2^{a-\ell}$ this implies

$$(2^{a-\ell} M) \bmod 2^a \ > \ 2^a - m \cdot 2^{a-\ell}$$

and

$$(2^{a-\ell} M + L) \bmod 2^a \ < \ m \cdot 2^{a-\ell}.$$

Due to the assumption $\ell \geq \lceil \log m \rceil + 1$ we know that $m \cdot 2^{a-\ell} \leq 2^{a-1} \leq 2^a - m \cdot 2^{a-\ell}$. Hence, we conclude

$$(2^{a-\ell} M) \bmod 2^a \ > \ (2^{a-\ell} M + L) \bmod 2^a.$$

But then $(*)$ follows right from Proposition 1. As we have shown above, this inequality is equivalent to the inequality $S_{a,k} \neq S^*_{a,k,\ell}$. Since $z = S^*_{a,k,\ell}$, we have shown that the inequalities (i) and (ii) imply $S_{a,k} \neq z$. This completes the proof of part (b). ∎

2.2 Computing and Testing the Approximation Value

Now knowing that the approximation $S^*_{a,k,\ell}$ cannot differ much from the true value $S_{a,k}$ we shall verify that it can in fact be efficiently computed by a space-bounded algorithm if the sum consists of the addends obtained from the school method for multiplication. Unfortunately, this is only true for small values of k (and ℓ). However, if k is large we can at least test randomly whether $S^*_{a,k,\ell}$ equals some arbitrary k-bit value.

Lemma 2 *Let $x, y \in \mathbb{Z}_{2^n}$ and $z \in \mathbb{Z}_{2^k}$ be given by their binary representations and let $w_i = x \cdot 2^i$ and $y_i = \langle y \rangle_i$ for $0 \le i < n$.*

(a) For any $c > 0$ and $\ell > \lceil \log n \rceil$ there is a nonuniform randomized algorithm which queries each input bit at most 4 times, uses space $\max\{4\ell, 3c(\log n + \log\log n)\} + \mathcal{O}(1)$, accepts if $S^*_{a,k,\ell}(w_0 \cdot y_0, \ldots, w_{n-1} \cdot y_{n-1}) = z$ and otherwise rejects with a probability of at least $1 - \mathcal{O}(k \log n / n^c)$.

(b) There is a deterministic algorithm which queries each input bit at most once and computes $S^*_{a,k,\ell}(w_0 \cdot y_0, \ldots, w_{n-1} \cdot y_{n-1})$ in space $2(k+\ell)+\mathcal{O}(1)$.

The following statement is well known and has been widely used in randomized algorithms such as pattern matching or fingerprinting.

Fact 1 *Let Q_t be the set of all primes smaller than t and let $d \in \mathbb{Z} - \{0\}$. Then the probability that $d \equiv 0 \pmod p$ for a randomly chosen prime $p \in Q_t - \{2\}$ is bounded by $\mathcal{O}\big((\log |d|) \cdot (\log t)/t\big)$.*

Proof: It is well known that there are $\Theta(i)$ primes in $Q_{i\lceil \log i \rceil}$ and thus Q_t consists of $\Omega(t/\log t)$ different primes. Since $|d|$ is a multiple of less than $\log |d|$ different primes, the probability that one of them is in $Q_t - \{2\}$ is bounded by $\mathcal{O}\big((\log |d| \cdot (\log t)/t\big)$. ∎

Proof of Lemma 2:

Proof of part (a): Let

$$w'_i \;:=\; w'_i(a,k,\ell) \;:=\; \langle w_i \rangle_{a-\ell}^{a+k-1} \;=\; \langle x \rangle_{a-\ell-i}^{a+k-1-i}$$

for $1 \le i \le n$ (recall that according to our definition the integer $\langle x \rangle_t^j$ has for $t < 0$ the binary representation $x_j \ldots x_0 0 \ldots 0$, where the number of 0s following x_0 is $-t$). Further, let $S = \sum_{i=0}^{n-1} w'_i \cdot y_i$. Then

$$\begin{aligned} &S^*_{a,k,\ell}(w_0 y_0, \ldots, w_{n-1} y_{n-1}) \\ &= \left(\sum_{i=0}^{n-1} \langle w_i \rangle_{a-\ell}^{a+k-1} \cdot y_i \right) \bmod 2^{k+\ell} \operatorname{div} 2^\ell \;=\; \left(\sum_{i=0}^{n-1} w'_i y_i \right) \bmod 2^{k+\ell} \operatorname{div} 2^\ell \\ &= \langle S \rangle_\ell^{k+\ell-1} \;=\; S_{\ell,k}(w'_0 y_0, \ldots, w'_{n-1} y_{n-1}). \end{aligned}$$

In the following, we use the abbreviation $S_{\dots}$ for $S_{\dots}(w'_0 y_0, \dots, w'_{n-1} y_{n-1})$ and the analogously defined abbreviations $S^*_{\dots}$ and $S^{**}_{\dots}$.

Let $t > 2$ be an integer to be determined later. Our algorithm executes the following three main steps:

1. Compute $S^*_{k+\ell,\ell,\ell}$.
2. Compute $S \bmod 2^\ell$.
3. Randomly choose a prime number $p \in Q := Q_t - \{2\}$. For
$$z^* = S - 2^{k+\ell} \cdot S^*_{k+\ell,\ell,\ell} - (S \bmod 2^\ell)$$
accept if $(z^* - 2^\ell \cdot z) \bmod p \in \left\{0,\ 2^{k+\ell} \bmod p\right\}$ and otherwise reject.

We first discuss the implementation details and resource bounds and later prove the claimed bound on the error probability. Since our computation model is nonuniform, we can assume that in Step 3 the algorithm randomly samples a prime $p \in Q$ by simply choosing a random value $r \in \{1, \dots, |Q|\}$. All computations involving p then are uniquely determined by r.

Let
$$w^*_i \ := \ w^*_i(k,\ell) \ := \ \langle w_i \rangle^{a+k+\ell-1}_{a+k-\ell} \ = \ \langle x \rangle^{a+k+\ell-i-1}_{a+k-\ell-i} .$$
Then
$$\begin{aligned} S^*_{k+\ell,\ell,\ell} \ &= \ \Big(\sum_{i=0}^{n-1} (w'_i \cdot y_i) \operatorname{div} 2^k\Big) \bmod 2^{2\ell} \operatorname{div} 2^\ell \\ &= \ \Big(\sum_{i=0}^{n-1} w^*_i \cdot y_i\Big) \bmod 2^{2\ell} \operatorname{div} 2^\ell. \end{aligned}$$
(Note that the right hand side of the latter equation also depends on k, because $w^*_i = w^*_i(k,\ell)$ depends on k.) It is easy to see that knowing w^*_i it suffices to query one x-bit (namely the bit $\langle x \rangle_{a+k-\ell-i-1}$) in order to obtain w^*_{i+1}. Hence, $S^*_{k+\ell,\ell,\ell}$ can be computed by querying each x- and y-bit at most once and using two registers — one storing the subtotals modulo $2^{2\ell}$ and the other the value w^*_i. Since w^*_i is a 2ℓ-bit value, the total amount of space required for the first step is $4\ell + \mathcal{O}(1)$.

In the second step $S \bmod 2^\ell$ can be computed by summing up the n ℓ-bit integers $(w'_i y_i) \bmod 2^\ell$ in $\mathbb{Z}/2^\ell\mathbb{Z}$. Analogously to w^*_i in the first step, $w'_{i+1} \bmod 2^\ell$ can be computed from $w'_i \bmod 2^\ell$ by reading one x-bit. Therefore, querying each x- and each y-bit once suffices in order to compute $S \bmod 2^\ell$ if we use one ℓ-bit register for storing the subtotals and one for storing $w'_i \bmod 2^\ell$. Hence, for the Steps 1 and 2 the time bounds total to $\mathcal{O}(n)$ and the space bounds total to $4\ell + \mathcal{O}(1)$.

Now we investigate the computation of the sum S modulo p in the third step. The binary value of w'_{i+1} is obtained from w'_i by first removing its most significant bit (i.e. by subtracting $2^{k+\ell-1} \cdot \langle w'_i \rangle_{k+\ell-1} = 2^{k+\ell-1} \langle x \rangle_{a+k-i-1}$), then

shifting the result by one bit position to the left (i.e. multiplying it with 2), and finally adding the least significant bit of w'_{i+1}, which is $\langle x \rangle_{a-\ell-i-1}$. Hence,

$$w'_{i+1} \;=\; 2 * w'_i - 2^{k+\ell-1} \langle x \rangle_{a+k-i-1} + \langle x \rangle_{a-\ell-i-1} \,.$$

Therefore, knowing $w'_i \bmod p$ we can easily compute $w'_{i+1} \bmod p$ by simply querying two x-bits and doing the above computation modulo p. Hence, once the results of the Steps 1 and 2 are known and their sum is taken modulo p, the algorithm can compute $z^* \bmod p$ by querying each x-variable twice and each y-variable once. Besides storing p, it suffices to store the subtotals and in the ith step the addend w'_i in one $\lceil \log p \rceil$-bit register, each. Finally, $(z^* - 2^\ell \cdot z) \bmod p$ can be obtained by querying each z-variable, using again one $\lceil \log p \rceil$-bit register for the subtotals. Altogether, Step 3 of the algorithm is possible with $3n + k$ variable queries and with $3 \log p + \mathcal{O}(1)$ space. Totaling over all three steps we obtain that each variable needs to be queried at most 4 times and that $\max\{4\ell,\, 3 \log p\} + \mathcal{O}(1)$ space suffices.

We shall now bound the error probability of the algorithm depending on the choice of t (recall that the prime p is chosen from $Q_t - \{2\}$). Since S is the sum of n $(k+\ell)$-bit integers and thus bounded by $2^{\log n + k + \ell}$, we have $S \bmod 2^{k+2\ell} = S$ (recall that $\ell \geq \lceil \log n \rceil + 1$). Therefore, $S_{k+\ell,\ell} = S \operatorname{div} 2^{k+\ell}$, which implies

$$\begin{aligned} S - 2^{k+\ell} \cdot S_{k+\ell,\ell} - S \bmod 2^\ell \;&=\; S - 2^{k+\ell} \cdot \big(S \operatorname{div} 2^{k+\ell}\big) - S \bmod 2^\ell \\ &=\; 2^\ell \cdot \langle S \rangle_\ell^{k+\ell-1} \;=\; 2^\ell \cdot S_{\ell,k}. \end{aligned}$$

By first plugging this into the definition of z^* and then applying Lemma 1(a) we obtain

$$\begin{aligned} &z^* - 2^\ell \cdot S_{\ell,k} \\ &\qquad =\; 2^{k+\ell} \cdot \big(S_{k+\ell,\ell} - S^*_{k+\ell,\ell,\ell}\big) \;\in\; \big\{0, 2^{k+\ell}\big\}. \quad (1) \end{aligned}$$

Hence, if $S_{\ell,k} = z$ then $(z^* - 2^\ell \cdot z) \bmod p \in \big\{0, 2^{k+\ell} \bmod p\big\}$ and the algorithm accepts correctly.

Now assume that this is not the case, i.e. $S_{\ell,k} \neq z$, and let $d = z - S_{\ell,k}$. Then $d \in \big\{\pm 1, \ldots, \pm(2^k - 1)\big\}$ and using (1) yields

$$r \;:=\; z^* - 2^\ell \cdot z \;=\; z^* - 2^\ell \cdot S_{\ell,k} - 2^\ell \cdot d \;\in\; \big\{-2^\ell \cdot d,\, 2^\ell (2^k - d)\big\}.$$

The algorithm only falsely accepts if $r \equiv 0$ or $(2^{k+\ell} - r) \equiv 0 \pmod p$. But since $p \neq 2$ and r is a multiple of 2^ℓ, this is equivalent to $r' \equiv 0$ or $(r' - 2^k) \equiv 0 \pmod p$, where $r' = r \operatorname{div} 2^\ell$. According to Fact 1 the probability that this is the case is bounded by $\mathcal{O}(k(\log t)/t)$, because $|r'| < 2^{k+1}$ and $|r'| \notin \big\{0, 2^k\big\}$. Hence, for $c > 1$ and $t = \lceil n^c \rceil$ we obtain an error probability of $\mathcal{O}(k \log n / n^c)$. Using the fact that any prime $p \in Q_t$ is bounded by $\mathcal{O}(t \log t)$ the space of our algorithm is bounded by $\max\{3c(\log n + \log\log n),\, 4\ell\} + \mathcal{O}(1)$.

Proof of part (b): The proof of this part is already implicitly contained in the proof of part (a). Recall the computation of $S^*_{k+\ell,\ell,\ell}$ in the first step of the

randomized algorithm. Exactly the same arguments show that we can compute $S^*_{a,k,\ell}$ by the summation of $(k+\ell)$-bit integers $w'_0 \cdot y_0, \ldots, w'_{n-1} \cdot y_{n-1}$ modulo $2^{k+\ell}$, where w'_{i+1} can be computed by querying only one x-bit if we know w'_i. It suffices to query each x-variable and each y-variable once and to store w'_i and the partial sums in two $(k+\ell)$-bit registers. Hence, the sum can be computed in $2(k+\ell)+\mathcal{O}(1)$ space. ∎

2.3 The Algorithm

We now state an algorithm which computes $\langle w_1 + \cdots + w_m \rangle_{n-1} = S_{n-1,1}(w_1, \ldots, w_m)$ for m integers $w_1, \ldots, w_m \in \mathbb{Z}_{2^n}$. As subroutines we use the algorithms for computing and testing $S^*_{a,k,\ell}$ from the former section. The time bounds of these subroutines are meaningful only if the addends w_i are the values $x \cdot 2^i \cdot \langle y \rangle_i$ from the school method for integer multiplication. Hence, the following algorithm is only useful for multiplication and not for adding multiple integers. However, its correctness can be proven for arbitrary sums.

Let $\ell = \lceil \log m \rceil + 1$, $a = \lfloor n/2 \rfloor$ and $k = n - a$.

Algorithm for computing $S_{n-1,1}(w_1, \ldots, w_m)$:

1. If $n \leq 3\ell$, then compute $s = \sum_{i=1}^{m} w_i$ and return $\langle s \rangle_{n-1}$.
2. Otherwise, compute $z^* := S^*_{n-1,1,\ell}(w_1, \ldots, w_m)$.
3. Test whether the equation $S^*_{a,k-1,\ell}(w_1, \ldots, w_m) = 2^{k-1} - 1$ is true.
   (a) If **true**, then compute recursively $s' = S_{a,1}(w_1, \ldots, w_m)$ and return $(z^* + s' + 1) \bmod 2$.
   (b) If **false**, then let $w'_i = w_i \operatorname{div} 2^{a-\ell}$ (for $1 \leq i \leq m$) and compute recursively $S_{k+\ell-1,1}(w'_1, \ldots, w'_m)$. Return the result.

Before formally proving the algorithm's correctness, let us sketch the idea behind it. Let $s = w_1 + \cdots + w_m$ and $s_N \ldots s_0$ be the binary representation of s. The algorithm first computes z^* as an approximation for s_{n-1}. Consider the sum $S = S_{a,k-1}(w_1, \ldots, w_m)$, which has the binary representation $s_{n-2} \ldots s_a$. Let $s^*_{n-2} \ldots s^*_a$ be the binary representation of the approximation $S^*_{a,k-1,\ell}(w_1, \ldots, w_m)$ of S. Note that this approximation is obtained by summing the integers $w_1, \ldots, w_m$ but ignoring the carry value C induced at the bit position $a - \ell + 1$. In Step 3 we test whether $s^*_{n-2} = \cdots = s^*_a = 1$.

Assume first that this is not the case, i.e. there is a bit $s^*_i = 0$, $a \leq i \leq n-1$. Then adding the carry value C may lead to a flip of this bit but it can be seen that the more significant bits will remain unchanged. Hence, the bits $s_{n-2} \ldots s_{i+1}$ equal their approximation $s^*_{n-2} \ldots s^*_{i+1}$. With the same argument it follows that if we sum $w_1, \ldots, w_m$ but ignore the carry C, the resulting bit at position $n-1$ is in fact s_{n-1}. This is realized in Step 3(b) by summing up the integers $w'_1, \ldots, w'_m$, where w'_i is in fact the integer w_i shifted by $a - \ell$ bit positions to the right.

On the other hand, if $s^*_{n-2} = \cdots = s^*_a = 1$, then it turns out that $z^* s^*_{n-2} \ldots s^*_a$ is the binary representation of $S^*_{a,k,\ell}$, the approximation of $S_{a,k}$. Now adding the carry value C to the sum used for determining the approximation will either lead to a flip of all the bits $z^* s^*_{n-2} \ldots s^*_a$ or none of these bits flip. Hence, if $s_a = 1 = s^*_a$, then the carry value has no influence and z^* is already the true value s_{n-1}. On the other hand, if $s_a = 0 \neq s^*_a$, then the true value s_{n-1} is the negation of z^*. Therefore, it suffices to compute s_a in order to be able to conclude on s_{n-1} by means of z^*. This is realized in Step 3(a).

In order to give formal proof of the correctness, we need the following statement, which can be verified easily by plugging in the definitions of $S^*_{\ldots}$ and $S_{\ldots}$.

Proposition 2 *Let $w_1, \ldots, w_m \in \mathbb{Z}_{\geq 0}$. Then*

$$S^*_{a,b,c}(w_1, \ldots, w_m) = S_{c,b}(w_1 \operatorname{div} 2^{a-c}, \ldots, w_m \operatorname{div} 2^{a-c}).$$

Proof of Correctness: Similar as before, we use the abbreviation $S_{\ldots}$ for $S_{\ldots}(w_1, \ldots, w_m)$ and do the same for $S^*_{\ldots}$ and $S^{**}_{\ldots}$. If $n \leq 3\ell$, the correctness is obvious by Step 1 of the algorithm. Hence, assume in the following $n > 3\ell$.

Assume first that the algorithm executes Step 3(b). We have to prove that in this case $S_{k+\ell-1,1}(w'_1, \ldots, w'_m) = S_{n-1,1}$. Using Proposition 2 we know that $S_{k+\ell-1,1}(w'_1, \ldots, w'_m) = S^*_{n-1,1,k+\ell-1}$. Assume that Step 3(b) does not return the correct result, i.e. $S^*_{n-1,1,k+\ell-1} \neq S_{n-1,1}$. Then by Lemma 1 we get

$$S^{**}_{n-1,1,k+\ell-1} \bmod 2^{k+\ell-1} \;>\; 2^{k+\ell-1} - m$$

and thus (using $\ell > \log m$)

$$S^{**}_{n-1,1,k+\ell-1} \bmod 2^{k+\ell-1} \operatorname{div} 2^{\ell} \quad \geq \quad \left\lfloor 2^{k-1} - m/2^{\ell} \right\rfloor \quad \geq \quad 2^{k-1} \;-\; 1.$$

We investigate the left side of this inequality:

$$S^{**}_{n-1,1,k+\ell-1} \bmod 2^{k+\ell-1} \operatorname{div} 2^{\ell} \;=\; \left(\sum_{i=1}^{m} w_i \operatorname{div} 2^{n-k-\ell} \right) \bmod 2^{k+\ell-1} \operatorname{div} 2^{\ell}$$
$$= \; S^*_{a,k-1,\ell}.$$

As derived above, this term is at least $2^{k-1} - 1$ and since $S^*_{a,k-1,\ell}$ is a $(k-1)$-bit integer we even have equality, i.e. $S^*_{a,k-1,\ell} = 2^{k-1} - 1$. But if this is the case, then the test in Step 3 is positive which contradicts our assumption that Step 3(b) is being executed.

Now assume that the algorithm executes Step 3(a) and thus $S^*_{a,k-1,\ell} = 2^{k-1} - 1$. Consider first the case that this approximation of $S_{a,k-1}$ is correct, i.e. $S_{a,k-1} = 2^{k-1} - 1$. Let $R = \sum_{i=1}^{m} w_i$. Clearly, the binary representation of $S_{a,k-1} = \langle R \rangle_a^{a+k-2} = \langle R \rangle_a^{n-2}$ consists only of ones. Hence, $s' = S_{a,1} = \langle R \rangle_a = 1$. Since the algorithm returns $(z^* + s' + 1) \bmod 2$, it suffices

to show that $S_{n-1,1} = z^*$. If this is not the case, i.e. $S^*_{n-1,1,\ell} \neq S_{n-1,1}$, then by Lemma 1

$$m \;>\; S_{n-1-\ell,\ell} \;=\; \langle R \rangle^{n-2}_{n-1-\ell}\,.$$

Since $\ell < n/2$ (ensured by the first step of the algorithm), $n-1-\ell \geq a$. Hence, the binary representation of $\langle R \rangle^{n-2}_{n-1-\ell}$ consists of ℓ ones and thus $R^{n-2}_{n-1-\ell} = 2^\ell - 1$. But then the above inequality simplifies to $m > 2^\ell - 1$ which contradicts the choice of ℓ.

The last case we have to consider is that the algorithm executes Step 3(a) but $S_{a,k-1} \neq S^*_{a,k-1,\ell} = 2^{k-1}-1$. Then Lemma 1(a) tells us that $S_{a,k-1} \bmod 2^{k-1} = 0$ and now the binary representation of $S_{a,k-1} = \langle R \rangle^{a+k-2}_{a}$ is a 0-string. Using the same arguments as in the former case, this implies $s' = 0$ and that it suffices to show $S_{n-1,1} \neq z^*$. Assume that this is note the case, i.e. $S_{n-1,1} = S^*_{n-1,1,\ell}$. We already know that $S_{n-\ell,\ell} = \langle R \rangle^{n-1}_{n-\ell} = 0 < m$. Hence, inequality (ii) of Lemma 1 (b) is true and thus inequality (i) cannot be fulfilled. This means that

$$S^{**}_{n-1,1,\ell} \bmod 2^\ell \;\leq\; 2^\ell - m. \tag{2}$$

Due to the assumption $S_{a,k-1} \neq S^*_{a,k-1,\ell} = 2^{k-1} - 1$ we have by Lemma 1 (b)

$$S^{**}_{a,k-1,\ell} \bmod 2^\ell \;>\; 2^\ell - m.$$

This way we obtain

$$\begin{aligned} S^{**}_{a,k-1,\ell} \;=\; S^*_{a,k-1,\ell} \cdot 2^\ell + S^{**}_{a,k-1,\ell} \bmod 2^\ell \;&>\; (2^{k-1}-1)\cdot 2^\ell + 2\ell - m \\ &=\; 2^{k+\ell-1} - m. \end{aligned} \tag{3}$$

Now let $A = \sum_{i=0}^{m} \langle w_i \rangle^{n-1}_{n-\ell-1}$ and $B = \sum_{i=0}^{m} \langle w_i \rangle^{n-\ell-2}_{n-k-\ell}$. Using the definition of $S^{**}_{n-1,1,\ell}$ and inequality (2) yields

$$A \bmod 2^{\ell+1} \;=\; S^{**}_{n-1,1,\ell} \;\leq\; 2^\ell - m.$$

Note that B is the sum of m $(k-1)$-bit numbers, and thus bounded above by $m \cdot (2^{k-1}-1)$. Therefore, we obtain

$$\left(A \cdot 2^{k-1} + B\right) \bmod 2^{k+\ell-1} \;\leq\; (2^\ell - m)\cdot 2^{k-1} + m\cdot(2^{k-1}-1) \;=\; 2^{k+\ell-1} - m.$$

But this contradicts (3), because

$$\left(A \cdot 2^{k-1} + B\right) \bmod 2^{k+\ell-1} \;=\; \left(\sum_{i=1}^{m} w_i \operatorname{div} 2^{a-\ell}\right) \bmod 2^{k+\ell-1} \;=\; S^{**}_{a,k-1,\ell}.$$

■

Now that we know that the algorithm is correct, we finally discuss its resource requirements and show that it can in fact be implemented by read-$k(n)$ branching programs, $k(n) = \mathcal{O}(\log n)$, as claimed in Theorem 1.

Let $x, y \in \mathbb{Z}_{2^n}$ and let $w_i = 2^i \cdot x \cdot \langle y \rangle_i$, $1 \leq i \leq n$, be the ith addend used in the school method for integer multiplication. It is obvious

that $\mathrm{MUL}_{n-1,n}(x,y) = S_{n-1,1}(w_1,\ldots,w_n) =: S_{n-1,1}$. If we use the algorithm in order to compute $S_{n-1,1}$ then it recursively computes $S_{n',1}$, where $n' \leq \max\{a,\, k+\ell-1\} \leq n/2+\ell$. Hence, as long as $n' > 3\ell$, the value n'' used in the next recursion call is at most $n'/2 + n'/3 = (5/6)n'$. Therefore, after $\mathcal{O}(\log n)$ recursive calls we have $n' \leq 3\ell$ and the algorithm terminates after the execution of Step 1.

The algorithm may only err in the test of Step 3. But by Lemma 2 we may obtain an error probability of n^{-d} for this step, where d is an arbitrarily large constant (increasing the space requirements only by a constant factor). Since Step 3 is executed at most $\mathcal{O}(\log n)$ times, the overall error probability can be bounded by n^{-c} for any constant c.

In order to bound the space requirements, consider first the terminal case ($n' \leq 3\ell$), in which the algorithm executes Step 1. According to Proposition 2 $\langle s\rangle_{n'-1} = S_{n'-1,1}(w_1,\ldots,w_n)$ is the same as $S^*_{n'-1,1,n'-1}$. By Lemma 2, this can be computed by querying each input bit at most once and in space $\mathcal{O}(\log n') = \mathcal{O}(\ell) = \mathcal{O}(\log n)$. If the algorithm is not in the terminal case, then it is easy to see by Lemma 2 that the algorithm can execute each single step (without the recursive calls) in $\mathcal{O}(\log n)$ space and with a constant number of queries of each variable. Note also that the algorithm can be easily implemented in such a way that it needs no recursive calls but instead loops until the terminal case is reached. In order to achieve this, we keep track of the positions p_1 and p_2 such that during an iteration the sum of the integers $\langle w_i\rangle_{p_1}^{p_2}$, $1 \leq i \leq m$, is relevant. Furthermore, we have to store a bit b which tells us whether the result has to be negated or not, if we reach the terminal case. The value of this bit changes in the case that 3 (a) is executed and $z^* \bmod 2 = 0$. Since $\mathcal{O}(\log n)$ bits suffice for keeping track of $p_1, p_2 \in \{0,\ldots,n-1\}$ and $b \in \{0,1\}$ and all computations during one iteration can be done in $\mathcal{O}(\log n)$ space, the total space requirement is $\mathcal{O}(\log n)$. Furthermore, the number of iterations (recursive calls) is bounded by $\mathcal{O}(\log n)$ and thus the algorithm queries each variable $\mathcal{O}(\log n)$ times, overall.

Now it is obvious that there is a branching program computing $\mathrm{MUL}_{n-1,n}$ in $\mathcal{O}(\log n)$ space such that on each computation path each variable appears at most $k(n) = \mathcal{O}(\log n)$ times. However, due to the syntactic restriction of read-$k(n)$ branching programs we have to ensure that there is no graph theoretical path such that some variable appears on it more than $k(n)$ times. But any computation corresponding to an arbitrary graph theoretical path of the branching program can be simulated by "faking" the input variables. That is, if the algorithm queries a variable, an arbitrary value in $\{0,1\}$ is returned instead of a value corresponding to some assignment. It is obvious that if the algorithm queries each variable at most $k(n)$ times even for all faked inputs, then it corresponds to a read-$k(n)$ branching program.

But if one looks at the subroutines as described in the proof of Lemma 2 and used to determine $S^*_{n-1,1,\ell}$ (in Step 2) and to test $S^*_{a,k-1,\ell}$ (in Step 3), then it is easy to see that the order in which the variables are queried during the execution of such a subroutine is even oblivious (i.e. it only depends on n, a, k and ℓ but not on the outcome of the variable queries). The same is true in the

terminal case (Step 1). Hence, if we fake the input, we can influence the results of the subroutine calls, and thus the decision of the algorithm whether to execute Step 3(a) or 3(b), but in both cases the recursion is continued properly. Hence, a branching program exists that satisfyies the syntactic read-$k(n)$ property and the claimed space bounds. This completes the proof of Theorem 1.

3 Formulas and General Branching Programs

We now present Nechiporuk's technique for proving lower bounds for the branching program and B_2-formula size of boolean functions.

Theorem 3 (Method of Nechiporuk, 1966) *Let the boolean function $f : \{0,1\}^n \to \{0,1\}$ essentially depend on the n variables in $X = \{x_1, \ldots, x_n\}$. Further, let $V_1, \ldots, V_\ell \subseteq X$ be disjoint sets of variables, and let v_i be the number of subfunctions of f on V_i. Then the number of nodes of any branching program for f is bounded below by $\Omega\big(\sum_{i=1}^{\ell} \log v_i / \log\log v_i\big)$ and any B_2-formula for f has size at least $\Omega\big(\sum_{i=1}^{\ell} \log v_i\big)$.*

In order to obtain a lower bound for the branching program size of $\mathrm{MUL}_{n-1,n}$, it suffices to find many disjoint sets V_i such that almost all subfunctions of $\mathrm{MUL}_{n-1,n}$ on each set are different.

Lemma 3 *Let $n = k^2$ and $X = \{x_0, \ldots, x_{n-1}\}$ and $Y = \{y_0, \ldots, y_{n-1}\}$. For any $0 \le i \le k-2$ let $V_i = \{x_{i+kj} \mid 0 \le j \le k-1\} \cup \big\{y_{n-k(i+1)-i+j} \;\big|\; 0 \le j < k\big\}$. Then $\mathrm{MUL}_{n-1,n}$ has at least $2^{n-2k-i+1} = 2^{\Omega(k^2)}$ subfunctions on V_i.*

Plugging this lemma in Nechiporuk's method, part (a) of Theorem 2 follows easily: We can assume w.l.o.g. that $n = k^2$. It is obvious that $\mathrm{MUL}_{n-1,n}$ essentially depends on all x- and y-variables. We choose the sets V_i, $0 \le i \le k-2$, as in the proof of Lemma 3. One can easily verify that these sets are in fact disjoint. Since $\mathrm{MUL}_{n-1,n}$ has $2^{\Omega(k^2)}$ subfunctions on V_i and since there are $k-1$ disjoint sets V_i, the branching program size of $\mathrm{MUL}_{n-1,n}$ is $\Omega\big(k \cdot k^2 / \log(k^2)\big) = \Omega\big(n^{3/2} / \log n\big)$ and the B_2-formula size is $\Omega(k^3) = \Omega(n^{3/2})$. Hence, part (a) of Theorem 2 follows.

Before we prove Lemma 3, we state an obvious proposition.

Proposition 3 *Let $a, b \in \mathbb{Z}$, $\ell \ge 0$ and $k \ge 1$. If $\langle a \rangle_\ell = \langle b \rangle_\ell = 0$, then*

$$\langle a+b \rangle_{\ell+1}^{\ell+k} = \Big(\langle a \rangle_{\ell+1}^{\ell+k} + \langle b \rangle_{\ell+1}^{\ell+k} \Big) \bmod 2^k.$$

Proof of Lemma 3: Fix $0 \le i \le k-2$ arbitrarily. Throughout this proof we assign all variables $x_j \in X - V_i$ the value 0. Clearly, the subfunction $\mathrm{MUL}'_{n-1,n}$ obtained by this assignment has at most as many subfunctions on V_i as $\mathrm{MUL}_{n-1,n}$. Further, for $0 \le j \le k-1$ we let $\sigma(j) = n - i - k(j+1)$ (for $j = k-1$ the value of $\sigma(j)$ is negative). Hence, $V_i \cap Y = \big\{y_{\sigma(i)}, \ldots, y_{\sigma(i)+k-1}\big\}$.

Consider now all assignments a to the y-variables not in V_i. Each such assignment defines a unique integer $b(a)$ in $\mathbb{Z}_{2^n}$ if we assume the other y-variables to be 0 (although not all integers in $\mathbb{Z}_{2^n}$ are possible, of course). If we assign all variables in $Y - V_i$ the value 0 and choose an assignment $\hat{y}$ to the y-variables in V_i, we obtain another integer $b(\hat{y}) \in M_i := \{r \cdot 2^{\sigma(i)} \mid 0 \leq r < 2^k\}$. In fact, by choosing an appropriate assignment $\hat{y}$ all the values in M_i are possible. The bit string obtained by the assignment a together with the assignment $\hat{y}$ corresponds to the integer $b(a, \hat{y}) = b(a) + b(\hat{y}) \in \mathbb{Z}_{2^n}$. Note also that for $b(\hat{y}) = r \cdot 2^{\sigma(i)}$ and $0 \leq j \leq k-1$

$$\langle b(a,\hat{y})\rangle_{\sigma(j)}^{\sigma(j)+k-1} = \begin{cases} \langle b(\hat{y})\rangle_{\sigma(i)}^{\sigma(i)+k-1} = r & \text{if } j = i \\ \langle b(a)\rangle_{\sigma(j)}^{\sigma(j)+k-1} & \text{otherwise.} \end{cases}$$

We consider now all assignments a to the y-variables not in V_i where $b(a) < 2^{n-i}$ and $\langle b(a)\rangle_{\sigma(j)} = 0$ for all $0 \leq j \leq k-1$, $j \neq i$. (Recall that $\sigma(j)$ is negative for $j = k-1$ and thus $\langle b(a)\rangle_{\sigma(j)}$ is 0, anyway). Similarly, we will consider only choices of $\hat{y}$ where $\langle b(\hat{y})\rangle_{\sigma(i)} = 0$, thus restricting the choice of $b(\hat{y})$ to the set of integers $r \cdot 2^{\sigma(i)}$ where $r \in \{0, 2, \ldots, 2^k - 2\}$ is even. This way we achieve that for all assignments a and all choices of $\hat{y}$ we have $\langle b(a,\hat{y})\rangle_{\sigma(j)} = 0$ for $0 \leq j \leq k-1$.

Consider two arbitrary such assignments $a \neq a'$. We prove that the two subfunctions of $\mathrm{MUL}'_{n-1,n}$ obtained by the assignments a and a' differ. Let t be an arbitrary value in $\{0, \ldots, k-2\}$ such that $\langle b(a)\rangle_{\sigma(t)}^{\sigma(t)+k-1} \neq \langle b(a')\rangle_{\sigma(t)}^{\sigma(t)+k-1}$. Such a t exists due to the assumption $b(a), b(a') < 2^{n-i}$. Since $\langle b(a)\rangle_{\sigma(t)} = \langle b(a')\rangle_{\sigma(t)} = 0$, we even have

$$b := \langle b(a)\rangle_{\sigma(t)+1}^{\sigma(t)+k-1} \neq \langle b(a')\rangle_{\sigma(t)+1}^{\sigma(t)+k-1} =: b'. \tag{4}$$

Now we choose $x_{kt+i} = x_{ki+i} = 1$ and let all other x-variables in V_i be 0 (recall that all x-variables not in V_i have been assigned the value 0, too). We can still choose an arbitrary $\hat{y}$ such that $b(\hat{y}) \in M_i$. For each choice of $\hat{y}$ the product of $b(a, \hat{y})$ and the integer represented by the x-assignment equals the sum

$$\begin{aligned} S(a,\hat{y}) &:= \sum_{j=0}^{n-1} x_j \cdot 2^j \cdot b(a,\hat{y}) \\ &= \left(2^{kt+i} + 2^{ki+i}\right) \cdot b(a,\hat{y}) \\ &= \left(2^{kt+i} + 2^{ki+i}\right) \cdot (b(a) + b(\hat{y})). \end{aligned}$$

Hence, in order to show that the subfunctions of $\mathrm{MUL}'_{n-1,n}$ obtained by the assignments a and a' differ, it suffices to prove that there is an even value $0 \leq r < 2^k$ such that for $b(\hat{y}) = r \cdot 2^{\sigma(i)}$

$$\langle S(a,\hat{y})\rangle_{n-1} \neq \langle S(a',\hat{y})\rangle_{n-1}.$$

Recall that for r being even we have $\langle b(a,\hat{y})\rangle_{\sigma(j)} = 0$ for all $0 \le j < k$, and thus $\langle 2^{kj+i} b(a,\hat{y})\rangle_{n-k} = \langle b(a,\hat{y})\rangle_{\sigma(j)} = 0$. Hence, using Proposition 3 we obtain

$$\begin{aligned}
\langle S(a,\hat{y})\rangle_{n-k+1}^{n-1} &= \left\langle 2^{kt+i} b(a,\hat{y}) + 2^{ki+i} b(a,\hat{y})\right\rangle_{n-k+1}^{n-1} \\
&= \left(\left\langle 2^{kt+i} b(a,\hat{y})\right\rangle_{n-k+1}^{n-1} + \left\langle 2^{ki+i} b(a,\hat{y})\right\rangle_{n-k+1}^{n-1}\right) \bmod 2^{k-1} \\
&= \left(\left\langle b(a,\hat{y})\right\rangle_{\sigma(t)+1}^{\sigma(t)+k-1} + \left\langle b(a,\hat{y})\right\rangle_{\sigma(i)+1}^{\sigma(i)+k-1}\right) \bmod 2^{k-1} \\
&= \left(\left\langle b(a)\right\rangle_{\sigma(t)+1}^{\sigma(t)+k-1} + \left\langle b(\hat{y})\right\rangle_{\sigma(i)+1}^{\sigma(i)+k-1}\right) \bmod 2^{k-1} \\
&= (b + r/2) \bmod 2^{k-1}.
\end{aligned}$$

Replacing a with a' in the above computation yields analogously $\langle S(a',\hat{y})\rangle_{n-k+1}^{n-1} = (b' + r/2) \bmod 2^{k-1}$. Since according to (4) $b \neq b'$ and thus $b \not\equiv b' \pmod{2^{k-1}}$, it is easy to choose an even value $r \in \{0,\ldots,2^k-2\}$ such that $\langle S(a,\hat{y})\rangle_{n-k+1}^{n-1} \ge 2^{k-2}$ and $\langle S(a',\hat{y})\rangle_{n-k+1}^{n-1} < 2^{k-2}$ or vice versa. In any case, $S(a,\hat{y})_{n-1} \neq S(a',\hat{y})_{n-1}$. Hence, the two subfunctions of MUL' obtained by the assignments a and a' are different.

It remains to count the number of valid assignments a to the y-variables not in V_i. There are $n-k$ variables in $Y - V_i$. Out of these, at most $k-1$ variables, namely the variables $y_{\sigma(j)}$ with $0 \le j < k$, $j \neq i$, have the fixed assignment 0. Finally, we require that $b(a) < 2^{n-i}$ which is achieved by assigning the i variables $y_{n-i},\ldots,y_{n-1}$ the constant 0. Hence, there remain $n-2k-i+1$ free variables. This yields $2^{n-2k-i+1}$ possibilities to choose an assignment a, each of which defines another subfunction of MUL on V_i. ∎

Now we state an upper bound on the number of subfunctions of $\mathrm{MUL}_{n-1,n}$ on any arbitrary set V of variables. Since Nechiporuk's method requires many such subfunctions, this demonstrates that one cannot prove lower bounds arbitrary close to $n^2/\log^2 n$ for the branching program size of $\mathrm{MUL}_{n-1,n}$.

Lemma 4 *Let $X = \{x_0,\ldots,x_{n-1}\}$, $Y = \{y_0,\ldots,y_{n-1}\}$, $V \subseteq X \cup Y$ and $k = |V|$. The number of subfunctions of $\mathrm{MUL}_{n-1,1}$ on V is bounded above by $2^{\mathcal{O}(k^4)}$.*

In order to prove this lemma, we need the notion of quadratic threshold functions.

Definition 2 A boolean function $f : \{0,1\}^n \to \{0,1\}$ is called *quadratic threshold function* if there are a threshold $T \in \mathbb{Z}$ and weights $w_{i,j} \in \mathbb{Z}$, $1 \le i \le j \le n$, such that $f(x_1,\ldots,x_n) = 1$ if and only if $\sum_{1\le i\le j\le n} w_{i,j} \cdot x_i \cdot x_j > T$.

The idea is that we can represent any subfunction of $\mathrm{MUL}_{n-1,n}$ by multiple quadratic threshold functions. In fact, it turns out that each subfunction of $\mathrm{MUL}_{n-1,n}$ on k variables is the parity of $\mathcal{O}(k)$ quadratic threshold functions.

Our result follows easily due to the well-known fact that the number of quadratic threshold functions on k is bounded by $2^{\mathcal{O}(k^3)}$ (see Hassoun, 1995).

Proof of Lemma 4: Let $k = |V|$ and let I_x be the set of indices $i \in \{0, \ldots, n-1\}$ for which V contains the variable x_i. For y-variables we analogously define the set I_y. The product of two integers $x, y \in \mathbb{Z}_{2^n}$ given by assignments to the variables in $X \cup Y$ is defined as

$$S(x,y) \;:=\; \sum_{0 \le i,j < n} x_i \cdot y_j \cdot 2^{i+j}.$$

Hence, $\mathrm{MUL}_{n-1,n}(x,y) = \langle S(x,y) \rangle_{n-1}$.

Consider a fixed subfunction S' of S on V obtained by assigning arbitrary bit values to all variables not in V. Then by factoring out first all $x_i \in X \cap V$ and then all $y_j \in Y \cap V$ we obtain nonnegative integer weights w, w_i, $i \in I_x$, and w'_j, $j \in I_y$, such that

$$S'(x_{|V}, y_{|V}) \;=\; w + \sum_{i \in I_x} w_i \cdot x_i + \sum_{j \in I_y} w'_j \cdot y_j + \sum_{(i,j) \in I_x \times I_y} 2^{i+j} \cdot x_i \cdot y_j.$$

Note that the weights do only depend on the assignments to the variables not in V. Since we are only interested in the value $S'(x_{|V}, y_{|V}) \bmod 2^n$, we replace all weights w, w_i and w'_j by their $\bmod 2^n$-values and consider the sum

$$\begin{aligned} S^*(x_{|V}, y_{|V}) \;:=\; & w \bmod 2^n + \sum_{i \in I_x} (w_i \bmod 2^n) \cdot x_i + \\ & \sum_{j \in I_y} (w'_j \bmod 2^n) \cdot y_j + \sum_{\substack{(i,j) \in I_x \times I_y \\ i+j < n}} 2^{i+j} \cdot x_i \cdot y_j. \end{aligned}$$

Since I_x and I_y contain together k variables there are for any $0 \le \ell < n$ at most $\lfloor k/2 \rfloor$ pairs $(i,j) \in I_x \times I_y$ with $i + j = \ell$. Hence, all powers 2^{i+j} with $(i,j) \in I_x \times I_y$ and $i+j < n$ sum up to a value of at most $(2^{n-1} + \cdots + 2^0) \cdot k/2 = (2^n - 1) \cdot k/2$. In addition, all weights $w \bmod 2^n$, $w_i \bmod 2^n$ and $w'_j \bmod 2^n$ are bounded by $2^n - 1$. Therefore, we obtain

$$S^*(x_{|V}, y_{|V}) \;\le\; (2^n - 1) \cdot (k/2 + 1 + |I_x| + |I_y|) \;<\; (3k+2) \cdot 2^{n-1}.$$

Note that $S^*(x_{|V}, y_{|V}) \operatorname{div} 2^{n-1}$ is even if and only if an even number of values $T \in \{1, \ldots, 3k+1\}$ satisfies $T \le S^*(x_{|V}, y_{|V}) \operatorname{div} 2^{n-1}$ or equivalently $T \cdot 2^{n-1} \le S^*(x_{|V}, y_{|V})$. We let $f_T(x_{|V}, y_{|V})$ for $1 \le T \le 3k+1$ be the quadratic threshold function with a function value of 1 if and only if $S^*(x_{|V}, y_{|V}) \ge 2^{n-1} \cdot T$. Then by the discussion above

$$\begin{aligned} \mathrm{MUL}_{n-1,n}(x,y) \;&=\; \langle S(x,y) \rangle_{n-1} \;=\; S^*(x_{|V}, y_{|V}) \operatorname{div} 2^{n-1} \bmod 2 \\ &=\; f_1(x_{|V}, y_{|V}) \oplus f_2(x_{|V}, y_{|V}) \oplus \cdots \oplus f_{3k+1}(x_{|V}, y_{|V}). \end{aligned}$$

(Here $\oplus$ denotes parity.) Each of these functions f_T, $1 \leq T \leq 3k+1$, is a quadratic threshold function on k variables. As already mentioned, the number of such threshold functions is bounded by $2^{\mathcal{O}(k^3)}$. Since —as we have shown— for each assignment to the variables not in V we obtain a subfunction of $\mathrm{MUL}_{n-1,n}$ which is uniquely defined by $3k+1$ quadratic threshold functions, the number of subfunctions of $\mathrm{MUL}_{n-1,n}$ is bounded above by $\left(2^{\mathcal{O}(k^3)}\right)^{3k+1} = 2^{\mathcal{O}(k^4)}$. ∎

We can now show how the last lemma yields part (b) of Theorem 2. Let $V_1, \ldots, V_\ell \subseteq X \cup Y$ be the disjoint sets yielding the largest value of Nechiporuk's bound for the branching program size of $\mathrm{MUL}_{n-1,n}$. Further, let $k_i = |V_i|$ and v_i be the number of subfunctions of $\mathrm{MUL}_{n-1,n}$ on V_i. Clearly, there are only 2^{2n-k_i} possibilities to choose an assignment to the variables not in V_i. Hence, $v_i = 2^{\mathcal{O}(n)}$. On the other hand, according to Lemma 4 $v_i = 2^{\mathcal{O}({k_i}^4)}$. Therefore, the lower bound obtained by Nechiporuk's method is $\mathcal{O}(B)$, where $B = \sum_{i=1}^{\ell} \min\left\{n/\log n,\, {k_i}^4/\log k_i\right\}$. Since $\sum_{i=1}^{\ell} k_i \leq 2n$, B is maximal if each k_i, $1 \leq i \leq \ell$, equals the largest integer K satisfying $K^4/\log K \leq n/\log n$ and if $\ell = 2n/K$. Hence, we can bound B by assuming $k_i = \Theta(n^{1/4})$ for all $1 \leq i \leq \ell$ and $\ell = \mathcal{O}(n^{3/4})$. This yields $B = \mathcal{O}(n^{7/4}/\log n)$.

Exactly the same arguments (but without the logarithmic terms in the denominators of the above computations) show that a lower bound of $c \cdot n^{7/4}$ for the B_2-formula size cannot be proven for all constants c with Nechiporuk's technique. This completes the proof of Theorem 2.

References

F. Ablayev and M. Karpinski (2003). A lower bound for integer multiplication on randomized ordered read-once branching programs. *Information and Computation*, volume 186, pp. 78–89.

M. Ajtai (1999). A non-linear time lower bound for boolean branching programs. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 60–70.

P. Beame, M. Saks, X. Sun and E. Vee (2003). Time-space tradeoff lower bounds for randomized computation of decision problems. *Journal of the ACM*, volume 50, pp. 154–195.

B. Bollig and I. Wegener (1996). Read-once projections and formal circuit verification with binary decision diagrams. In *Proceedings of the 13th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 1046 of *Lecture Notes in Computer Science*, pp. 491–502.

B. Bollig and P. Woelfel (2001). A read-once branching program lower bound of $\Omega(2^{n/4})$ for integer multiplication using universal hashing. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC)*, pp. 419–424.

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

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

M. H. Hassoun (1995). *Fundamentals of Artificial Neural Networks*. MIT Press, first edition.

Y. Mansour, N. Nisan and P. Tiwari (1993). The computational complexity of universal hashing. *Theoretical Computer Science*, volume 107, pp. 121–133.

E. I. Nechiporuk (1966). A Boolean function. *Soviet Mathematics Doklady*, volume 7, pp. 999–1000.

E. A. Okol'nishnikova (1993). On lower bounds for branching programs. *Siberian Advances in Mathematics*, volume 3, pp. 152–166.

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

M. Sauerhoff and P. Woelfel (2003). 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)*, pp. 186–195.

I. Wegener (2000). *Branching Programs and Binary Decision Diagrams - Theory and Applications*. SIAM.

ECCC ISSN 1433-8092
http://www.eccc.uni-trier.de/eccc
ftp://ftp.eccc.uni-trier.de/pub/eccc
ftpmail@ftp.eccc.uni-trier.de, subject 'help eccc'