On the complexity of multivariate blockwise polynomial multiplication
Joris Van Der Hoeven, Grégoire Lecerf

To cite this version:
Joris Van Der Hoeven, Grégoire Lecerf. On the complexity of multivariate blockwise polynomial multiplication. 2012. <hal-00660454>

HAL Id: hal-00660454
https://hal.archives-ouvertes.fr/hal-00660454
Submitted on 16 Jan 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
On the complexity of multivariate blockwise polynomial multiplication*

by Joris van der Hoeven and Grégoire Lecerf
Laboratoire d’Informatique, UMR 7161 CNRS
École polytechnique
91128 Palaiseau Cedex, France
{vdhoeven, lecerf}@lix.polytechnique.fr

Preliminary version of January 16, 2012

ABSTRACT
In this article, we study the problem of multiplying two multivariate polynomials which are somewhat but not too sparse, typically like polynomials with convex supports. We design and analyze an algorithm which is based on block-wise decomposition of the input polynomials, and which performs the actual multiplication in an FFT model or some other more general so called “evaluated model”. If the input polynomials have total degrees at most $d$, then, under mild assumptions on the coefficient ring, we show that their product can be computed with $O(s^{1.5337})$ ring operations, where $s$ denotes the number of all the monomials of total degree at most $2d$.

Keywords
sparse polynomial multiplication, multivariate power series, evaluation-interpolation, algorithm

A.M.S. subject classification
68W30, 12-04, 30B10, 42-04

1. INTRODUCTION

Let $A$ be an effective ring, which means that we have a data structure for representing the elements of $A$ and algorithms for performing the ring operations in $A$, including the equality test. The complexity for multiplying two univariate polynomials with coefficients in $A$ and of degree $d$ is rather well understood [11, Chapter 8]. Let us recall that for small $d$, one uses the naive multiplication, as learned at high school, of complexity $O(d^2)$. For moderate $d$, one uses the Karatsuba multiplication [21], of complexity $O(d \log 3/\log 2)$, or some higher order Toom-Cook scheme [4, 29], of complexity $O(d^\alpha)$ with $1 < \alpha < \log 3/\log 2$. For large $d$, one uses FFT [3, 5, 27] and TFT [14, 15] multiplications, both of complexity $O(d \log d)$ whenever $A$ has sufficiently many $2^k$-th roots of unity, or the Cantor-Kaltofen multiplication [3, 27] with complexity in $O(d \log d \log \log d)$ otherwise.

Unfortunately most of the techniques known in the univariate case do not straightforwardly extend to several variables. Even for the known extensions, the efficiency thresholds are different. For instance, the naive product is generally softly optimal when sparse representations are used, because the size of the output can grow with the product of the sizes of the input. In fact, the nature of the input supports plays a major role in the design of the algorithms and the determination of their mutual thresholds. In this article, we focus on and analyze an intermediate approach to several variables, which roughly corresponds to an extension of the Karatsuba and Toom-Cook strategies, and which turns out to be more efficient than the naive multiplication for instance if the supports of the input polynomials are rather dense in their respective convex hulls.

1.1 Our contributions

The main idea in our blockwise polynomial multiplication is to cut the supports in blocks of size $b_1 \times \cdots \times b_n$ and to rewrite all polynomials as “block polynomials” in $z_1^{b_1}, \ldots, z_n^{b_n}$ with “block coefficients” in

$$B_s := \{ P \in A[z_1, \ldots, z_n] : \deg z_i P < b_1, \ldots, \deg z_n P < b_n \}.$$

These block coefficients are then manipulated in a suitable “evaluated model”, in which the multiplication is intended to be faster than with the direct naive approach. The blockwise polynomials are themselves multiplied in a naive manner. The precise algorithm is described in Section 3.1, where we also discuss the expected speedup compared to the naive product. A precise complexity analysis is subtle because it depends too much on the nature of the supports. Although the worse case bound turns out to be no better than with the naive approach, we detail, in Section 3.2, a way for quickly determining a good block size, that suits supports that are not too small, not too thin, and rather dense in their convex hull.

Our Section 4 is devoted to implementation issues. We first design a cache-friendly version of our blockwise product. Then we adapt a blockwise product for multivariate power series truncated in total degree. Finally, we discuss a technique to slightly improve the blockwise approach for small and thin supports.

In Section 5 we analyze the complexity of the blockwise product for two polynomials supported by monomials of degree at most $d - 1$: if $A$ contains $\mathbb{Q}$ in its center, then we show that their product can be computed with $O(s^{1.5337})$ operations in $A$, where $s$ denotes the number of all the monomials of degree at most $2(d - 1)$. Notice that the constant hidden behind the latter $O$ does not depend neither on $d$ nor on the number of the variables. With no hypothesis on $A$, the complexity bound we reach becomes in $O(s^{1.5930})$, by a direct extension of the Karatsuba algorithm.

* This work has been partly supported by the French ANR-09-JCJC-0098-01 MaGiX project, and by the DIGITEO 2009-36HD grant of the Région Île-de-France.
Finally, we prove similar complexity results for truncated power series

1.2 Related works

Algorithms for multiplying multivariate polynomials and series have been designed since the early ages of computer algebra [6, 7, 8, 20, 28]. Even those based on the naive techniques are a matter of constant improvements in terms of data structures, memory management, vectorization and parallelization [10, 18, 23, 24, 25, 26, 30].

With a single variable $z$, the usual way to multiply two polynomials $P$ and $Q$ of degree $d-1$ with the Karatsuba algorithm begins with splitting $P$ and $Q$ into $P_0(z) + z^h P_1(z)$ and $Q_0(z) + z^h Q_1(z)$, where $h := \lfloor d/2 \rfloor$. Then $P_0 Q_0$, $P_1 Q_1$, and $(P_0 + P_1) (Q_0 + Q_1)$ are computed recursively. This approach has been studied for several variables [22], but it turns out to be efficient mainly for plain block supports, as previously discussed in [8, Section 3].

For any kind of support, there exist general purpose sparse multiplication algorithms [2, 18]. They feature quasi-linear complexity in terms of the sizes of the supports of the input and of a given superset for the support of the product. Nevertheless the logarithmic overhead of the latter algorithms is important, and alternative approaches, directly based on the truncated Fourier transform, have been designed in [14, 15, 19] for supports which are initial segments of $N^n$ (i.e. sets of monomials which are complementary to a monomial ideal), with the same order of efficiency as the FFT multiplication for univariate polynomials.

To the best of our knowledge, the blockwise technique was introduced, in a somewhat sketchy manner, in [13, Section 6.3.3], and then refined in [16, Section 6]. In the case of univariate polynomials, its complexity has been analyzed in [12], where important applications are also presented.

2. CONVENTIONS AND NOTATION

Throughout this article, we use the sparse representation for the polynomials, and the computation tree model with the total complexity point of view [1, Chapter 4] for the complexity analysis of the algorithms. Informally speaking, this means that complexity estimates charge a constant cost for each arithmetic operation and the equality test in $\mathbb{A}$, and that all the constants are thought to be free at our disposal.

2.1 Block polynomials

Consider a multivariate polynomial $P \in \mathbb{A}[z] = \mathbb{A}[z_1, \ldots, z_n]$, also written as

$$ P = \sum_{i \in N^n} P_i z^i = \sum_{i_1, \ldots, i_n \in N} P_{i_1, \ldots, i_n} z_1^{i_1} \cdots z_n^{i_n}. $$

We define its support by $\text{supp } P = \{i \in N^n: P_i \neq 0\}$, and denote its cardinality $|\text{supp } P|$ by $s_P$. We interpret $s_P$ as the sparse size of $P$.

Given a vector $b = (b_1, \ldots, b_n) \in (\mathbb{N}^+)^n$ of positive integers, we define the set of block supports at order $b$ by

$$ \mathbb{B}_b = \{P \in \mathbb{A}[z_1, \ldots, z_n]: \text{deg}_{z_1} P < b_1, \ldots, \text{deg}_{z_n} P < b_n\}. $$

Any polynomial $P$ can uniquely be rewritten as a block polynomial $\tilde{P} = \sum_{i \in N^n} \tilde{P}_i z^i \in \mathbb{B}_b[\tilde{z}^n] \subseteq \mathbb{B}_b[z_1^{b_1}, \ldots, z_n^{b_n}]$, where

$$ \tilde{P}_i = \sum_{0 \leq i_1 < b_1, \ldots, 0 \leq i_n < b_n} P_{i_1, i_2, \ldots, i_n} z_1^{i_1} \cdots z_n^{i_n}. $$

Given $P, Q \in \mathbb{B}_b[\tilde{z}^n]$, we also notice that $\tilde{P} \tilde{Q} \in \mathbb{B}_{2b-1}[\tilde{z}^n]$. We let $s_b = b_1 \cdots b_n$ represent the size of the block $b$.

2.2 Evaluation-interpolation schemes

A univariate evaluation-interpolation scheme in size $d \in \mathbb{N}$ on $\mathbb{A}$ is the data of

- an algorithm for computing an evaluation function $\text{Eval}_d: \mathbb{B}_{d-1} \rightarrow \mathbb{A}^{\mathbb{N}(d)}$, where $\mathbb{B}$ is the set of polynomials in $\mathbb{A}[z]$ of degree at most $d-1$, and where $\mathbb{N}(d)$ can be seen as the number of evaluation points, and
- an algorithm for computing an interpolation function, written $\text{Eval}_d^{-1}$, but which is not necessarily exactly the inverse of $\text{Eval}_d$. $\text{Eval}_d^{-1}: \mathbb{A}^{\mathbb{N}(d)} \rightarrow \mathbb{B}_{d-1}$.

such that $\text{Eval}_d$ and $\text{Eval}_d^{-1}$ are linear and

$$ \text{Eval}_d^{-1}(\text{Eval}_d(P)) \cap \text{Eval}_d(Q)) = PQ $$

holds for all $P$ and $Q$ in $\mathbb{B}_d$, where $\cap$ denotes the entry-wise product in $\mathbb{A}^{\mathbb{N}(d)}$. We write $E(d)$ for a common bound on the complexity of $\text{Eval}_d$ and $\text{Eval}_d^{-1}$, so that two polynomials in $\mathbb{B}_d$ can be multiplied in time $3E(d) + N(d)$. For convenience we still write $\text{Eval}_d$ and $\text{Eval}_d^{-1}$ for extensions to any $\mathbb{A}^k$ seen as a ring endowed with the entry-wise product.

**Example 1.** The Karatsuba algorithm for polynomials of degree 1 corresponds to $d = 2$, $N(2) = 3$,

$$ \text{Eval}_2: P_0 + P_1 z \mapsto (P_0, P_0 + P_1, P_1), $$

$$ \text{Eval}_2^{-1} : (C_0, C_1, C_2) \mapsto C_0 + (C_1 - C_0) z + C_2 z^2. $$

$\text{Eval}_d$ can be interpreted as the evaluation of a polynomial at the values 0, 1, and $+\infty$. By induction, the Karatsuba algorithm for polynomials of degree at most $d-1$ corresponds to $N(d) = 3N(\lfloor d/2 \rfloor)$. $\text{Eval}_d P \mapsto \text{Eval}_{\lfloor d/2 \rfloor}(P)$, where $P$ is the block polynomial of $P$ with respect to $b = (2)$, and where $\text{Eval}_{\lfloor d/2 \rfloor}(P)$ actually corresponds to applying the extension of $\text{Eval}_{\lfloor d/2 \rfloor}$ to $\mathbb{B}_b[z^b] \rightarrow \mathbb{A}^{3\mathbb{N}(\lfloor d/2 \rfloor)}$. This yields $N(2^d) = 3^d$ and $E(2^d) = O(3^d)$.

Univariate evaluation-interpolation schemes can be extended to several variables as follows: if $b = (b_1, \ldots, b_n) \in (\mathbb{N}^+)^n$ then we define $N(b) = N(b_1) \cdots N(b_n)$, define the maps

$$ \text{Eval}_b: \mathbb{B}_b \rightarrow \mathbb{A}^{N(b)} $$

$$ C \mapsto \text{Eval}_{b_1, b_2, \ldots, b_n}(C), $$

and take

$$ E(b) = N(b) (E(b_1) + \cdots + E(b_n)). \tag{1} $$

where $\text{Eval}_{b_1}$ and $\text{Eval}_{b_1}^{-1}$ represent the evaluation and interpolation with respect to the variable $z_1$:

$$ \text{Eval}_{b_1} : \mathbb{A}^{N(b_1)} \otimes \mathbb{B}(b_1) \rightarrow \mathbb{A}^{N(b_1, b_1)} \otimes \mathbb{B}(b_1, b_1), $$

$$ \text{Eval}_{b_1}^{-1} : \mathbb{A}^{N(b_1)} \otimes \mathbb{B}(b_1) \rightarrow \mathbb{A}^{N(b_1)} \otimes \mathbb{B}(2b_1, \ldots, 2b_1). $$

In the sequel we will only use multivariate evaluation-interpolation schemes constructed in this way.
3. THE BASIC ALGORITHM

For what follows, we first assume that we have a strategy for picking a suitable evaluation-interpolation scheme for any given block size \( b \in (\mathbb{N}^>)^n \), and that we have a fast way to compute the corresponding functions \( N(b) \) and \( E(b) \), at least approximately. We also assume that \( N(d)/d \) and \( E(d)/d \) are increasing functions.

3.1 Blockwise multiplication

Let us first assume that we have fixed a block size \( b \).

The first version of our algorithm for multiplying two multivariate polynomials \( P, Q \in \mathbb{A}[z] \) as follows:

**Algorithm block-multiply**\((P, Q, b)\)

**Input:** \( P, Q \in \mathbb{A}[z] \) and \( b \in (\mathbb{N}^>)^n \).

**Output:** \( PQ \in \mathbb{A}[z] \).

1. Rewrite \( P \) and \( Q \) as block polynomials \( P, Q \in \mathbb{B}_b \).
2. Compute \( P := \sum_i \text{Eval}_i(P_i) (z^i)^f \) and \( Q := \sum_i \text{Eval}_i(Q_i) (z^i)^f \in \mathbb{B}_b \).
3. Multiply \( \tilde{R} := \tilde{P} \tilde{Q} \in \mathbb{A}^{N(b)[z^b]} \) using the naive algorithm.
4. Compute \( \tilde{R} := \text{Eval}^{-1}(\tilde{R}) := \sum_i \text{Eval}^{-1}(\tilde{R}_i) (z)^i \in \mathbb{B}_{2^b-1}[z^b] \).
5. Reinterpret \( \tilde{R} \) as a polynomial \( R \in \mathbb{A}[z] \) and return \( R \).

**Proposition 2.** The algorithm block-multiply works correctly as specified, and performs at most
\[
N(b) (s_P s_Q + E(b) (s_P + s_Q + 2 s_R^b))
\]
orations in \( \mathbb{A} \).

**Proof.** While this correctness is clear from the definitions, Step 1 performs no operation in \( \mathbb{A} \). Steps 2 and 4 require \( E(b) (s_P + s_Q + s_R^b) \) operations. Step 3 requires \( N(b) s_P s_Q \) operations. Step 5 takes at most \( 2^b s_R^b \leq E(b) s_R^b \) operations in order to add up overlapping block coefficients, where \( b = \{ i \in \{1,...,n\} : b_i > 1 \} \).

Unfortunately, without any structural knowledge about the supports of \( P \) and \( Q \), the quantities \( s_R^b \) requires a time \( s_P s_Q \) to be computed. In the next subsection we explain a heuristic approach to find a good block size \( b \).

3.2 Heuristic computation of the block size

In order to complete our multiplication algorithm, we must explain how to set the block size. Computing quickly the block size that minimizes the number of operations in the product of two given polynomials \( P \) and \( Q \) looks like a difficult problem. In Section 6 we analyze it for asymptotically large supports, but in this subsection, we describe a heuristic for the general case.

For approximating a good block size we first need to approximate the number of operations in Proposition 2 by a function, written \( T(P, Q, b) \), which is intended to be easy to compute in terms of \( N(b) \) and \( E(b) \), and the input supports. For instance one may opt for the formula
\[
T(P, Q, b) = (N(b) + 2 E(b)) (s_P + 1) (s_Q + 1).
\]

Nevertheless, if \( P \) and \( Q \) are expected to be not too sparse, then one may assume \( s_R^b \lesssim 2^n (s_P + s_Q) \) and use the formula
\[
T(P, Q, b) = N(b) (s_P s_Q + 2^{n+1} E(b) (s_P + s_Q))
\]

Then we begin the approximating process with \( b = (1,...,1) \), and keep doubling the entry of \( b \) which reduces \( T(P, Q, b) \) as much as possible. We stop as soon as \( T(P, Q, b) \) cannot be further reduced in this way. Given \( b \in (\mathbb{N}^>)^n \) and \( i \in \{1,...,n\} \), we write \( b_i (b_i, b_{i-1}, 2 b_i, b_{i+1},...,b_n) \). In order to make explicit the dependency in \( b \) during the execution of the algorithm, we write \( P(b) \) instead of \( P \).

**Algorithm block-size**\((P, Q, b)\)

**Input:** \( P, Q \in \mathbb{A}[z] \).

**Output:** a block size \( b \) for multiplying \( P \) and \( Q \).

Set \( b := (1,...,1) \).

While \( P(b) \) and \( Q(b) \) are supported by at least two monomials repeat:

Let \( i \in \{1,...,n\} \) be such that \( T(P, Q, 2_i(b)) \) is minimal.

If \( T(P, Q, 2_i(b)) \geq T(P, Q, 2_i(b)) \), then return \( b \).

Set \( b := 2_i(b) \).

In the computation tree model over \( \mathbb{A} \), the block-size algorithm actually performs no operation in \( \mathbb{A} \). Its complexity must be analyzed in terms of bit-operations, which are here computation trees over \( \mathbb{Z}/2\mathbb{Z} \). For this purpose we assume that the supports are represented by vectors of monomials, with each monomial being stored as a vector of integers in binary representation.

**Proposition 3.** If \( P \) and \( Q \) have total degrees at most \( d \), then the algorithm block-size performs \( O(n^2 (s_P + s_Q) \log^2 d) \) bit-operations, plus \( O(n^2 \log d) \) calls to the function \( T \).

**Proof.** To run the algorithm block-size efficiently, we maintain the sets \( S_{P, b} := \text{supp} P(b) \) and \( S_{Q, b} := \text{supp} Q(b) \) throughout the main loop. Since \( S_{P, 2_i(b)} \) can be computed from \( S_{P, b} \) with \( O(s_P \log d) \) bit-operations, each iteration requires at most \( O(n (s_P + s_Q) \log d) \) bit-operations. The number of iterations is bounded by \( O(n \log d) \). \( \square \)

Remark that in favorable cases the first few iterations are expected to approximately halve \( s_P + s_Q \) at each stage. The expected total complexity thus becomes closer to \( O(n (s_P + s_Q) \log d) \).

4. VARIATIONS

4.1 Implementation issues

Several problems arise if one tries to implement the basic blockwise multiplication algorithm from Section 3.1 without any modification:

- The mere storage of \( \tilde{R}, \tilde{Q} \) and \( \tilde{R} \) involves \( (s_P + s_Q + s_R^b) \) \( N(b) \) coefficients in \( \mathbb{A} \). This number is usually much larger than \( s_P + s_Q + s_R \).
- Coefficients in \( \mathbb{A}^{N(b)} \) usually will not fit into cache memory, which makes the algorithm highly cache inefficient.

Both drawbacks can be removed by using a recursive multiplication algorithm instead, where the evaluation-interpolation technique is applied sequentially with respect to \( z_i, i \in \{1,...,n\} \).

**Algorithm improved-block-multiply**\((P, Q, b, i_1, ..., i_k)\)

**Input:** \( P, Q \in \mathbb{A}[z] \), \( b \in (\mathbb{N}^>)^n \) and pairwise distinct \( i_1, ..., i_k \in \{1,...,n\} \).
choose the same complexity as the basic algorithm in terms of the memory consumption of the algorithm. ...

operations in $P, Q$ by $\Theta(1)$, so that $S = \text{supp} U = S_{n,d}$, then $s_{p,s_Q}$ can be replaced by $(2n + d - 1)$ in our complexity bound (2).

Assuming that we have a way to estimate the number of operations in $\mathbb{A}^{N_b}$ necessary to compute the truncated product $R$ using the naive algorithm, we can adapt the heuristic of Section 3.2 to determine a candidate block size. Finally let us mention that the additional implementation tricks from Section 4.1 also extend to the truncated case in a straightforward way.

4.3 Separate treatment of the border

If we increase the block size $b$ in block-multiply, then the block coefficients $P_i, Q_i$ get sparser and sparser. At a certain point, this makes it pointless to further increase $b$. One final optimization which can often be applied is to decompose $P = P_{\text{flat}} + P_{\text{border}}$ and $Q_{\text{int}} + Q_{\text{border}}$ in such a way that the multiplication $P_{\text{flat}} Q_{\text{int}}$ can be done using a larger block size than the remaining part $P_{\text{border}} Q_{\text{border}} + P_{\text{border}} Q_{\text{int}} + P_{\text{int}} Q_{\text{border}}$ of the multiplication $P Q$. Instead of providing a full and rather technical algorithm, let us illustrate this idea with the example.

The naive multiplication performs 36 products in $\mathbb{A}$. The direct use of the Karatsuba algorithm with $b = (2, 2)$, reduces to the naive product of two polynomials of 3 terms with coefficients in $\mathbb{A}$, which amounts to $3 \times 3 \times 9 = 81$ multiplications in $\mathbb{A}$. If we decompose $P$ and $Q$ into

$P := Q := 1 + z_1 + z_1^2 + (1 + z_1) z_2 + z_2^2$.

The naive multiplication performs 36 products in $\mathbb{A}$. The direct use of the Karatsuba algorithm with $b = (2, 2)$, reduces to the naive product of two polynomials of 3 terms with coefficients in $\mathbb{A}$, which amounts to $3 \times 3 \times 9 = 81$ multiplications in $\mathbb{A}$. If we decompose $P$ and $Q$ into

$P_{\text{int}} = Q_{\text{int}} = 1 + z_1 + (1 + z_1) z_2$,

$P_{\text{border}} = Q_{\text{border}} = z_1^2 + z_2^2$,

then $P_{\text{int}} Q_{\text{int}}$ takes 9 products in $\mathbb{A}$, and the naive multiplication for $P_{\text{int}} Q_{\text{border}} + P_{\text{border}} Q_{\text{int}} + P_{\text{border}} Q_{\text{border}}$ uses 20 products in $\mathbb{A}$. In this example, the separate treatment of the border saves 7 products in $\mathbb{A}$ out of the 36 of the naive algorithm, and is faster than the direct use of the blockwise algorithm.

5. UNIFORM COMPLEXITY ANALYSIS

In this section we focus on the specific and important problems of multiplying polynomials $P, Q \in \mathbb{A}[z]$ of total degree at most $d - 1$, and truncated power series at order $d$. Recall from Section 4.2 that $S_{n,d} := \{ i \in \mathbb{N}^n : i_1 + \cdots + i_n \leq d - 1 \}$.

Let

$\lambda(\alpha) := (1 + \alpha) \log (1 + \alpha) - \alpha \log \alpha$,

$\phi_\beta(\alpha) := \log (2 \beta^{-1}) + 2 \lambda(\alpha/\beta) / \lambda(2 \alpha)$. 

and, by [16, Section 6], $|S_{n,d}| = (n + d - 1)$. The naive truncated power series multiplication at order $d$ can be performed using only

$M_{n,d} := \sum_{k=0}^{d-1} \sum_{l_k=k} |S_{n,l_k}| |S_{n,l_k}| = \binom{2n+d-1}{2n} = |S_{2n,d}|$

operations, as shown in [16, Section 6]. Hence, if $S = S_{n,d}$ and $b = (\beta, \ldots, \beta)$, so that $S = \text{supp} U = S_{n,d}/b$, then $s_{p,s_Q}$ can be replaced by $(2n + d/\beta - 1)$ in our complexity bound (2).

In a similar way as in Proposition 2, we can show that truncated-block-multiply requires at most

$N(b) s_{p,s_Q} + E(b) (s_p + s_Q + 2 s_{q})$ (2)

operations in $\mathbb{A}$. However, this bound is very pessimistic since $S = \text{supp} U$ usually has a special form which allows us to perform the naive truncated multiplication in step 3 in much less than $s_{p,s_Q} \text{ operations}$. For instance, in the special and important case of truncated power series at order $d$, we have

$S = S_{n,d} := \{ i \in \mathbb{N}^n : i_1 + \cdots + i_n \leq d - 1 \}$.
Let $\eta > 0$ be the first value of $\alpha$ such that $\phi_1(\alpha) = \phi_4(\alpha)$, and define $\zeta := \phi_1(\eta)$. Numeric computations yield $2.1454 < \eta < 2.1455$ and $1.5336 < \zeta < 1.5337$.

### 5.1 Polynomial Product

**Theorem 4.** Let $\varepsilon > 0$, and assume that $A$ admits evaluation-interpolation schemes with $N(d) = 2d - 1$ and $E(d) = O(d \log (d + 1))$, for all $d \geq 1$, and for some constant $\nu \geq 0$. Then two polynomials in $A[z]$ of degree at most $d - 1$ can be multiplied with block-multiply using at most $O((|S_{n,2d-1}|^{\varepsilon})$ operations in $A$, if the sizes of the blocks $b = (\beta, \ldots, \beta)$ are chosen as follows:

- $\beta = 1$, if $d/n \leq \eta$,
- $\beta = 4$, if $\eta < d/n \leq 8$,
- $\beta = \lceil \alpha/2 \rceil$, if $8 < d/n$.

**Proof.** We introduce $\alpha := d/n$. By Stirling’s formula, there exists a constant $K_0$ such that

$$\left| \log n! - n (\log n - 1) + \frac{\log n}{2} \right| \leq K_0, \quad \text{for all } n \geq 1.$$  

From $\log |S_{n,d}| = \log \left( \binom{n+d-1}{n} \right) = \log \left( \frac{d}{n+d} \binom{n+d}{n} \right)$, we obtain the existence of a constant $K_1$ such that the following inequality holds for all $n \geq 1$ and $d \geq 1$:  

$$\log |S_{n,d}| - n \lambda(\alpha) - \frac{\log n + \log (1 + n/d)}{2} \leq K_1.$$  

Therefore there exists a constant $K_2$ such that

$$\left| \frac{\log |S_{n,d}| - \lambda(\alpha)}{n} \right| \leq K_2 \frac{\log n}{n} \quad \text{and} \quad \left| \frac{\log |S_{n,2d-1}| - \lambda(2\alpha)}{n} \right| \leq K_2 \frac{\log n}{n}$$

hold for all $d \geq 1$ and $n \geq 1$. By Proposition 2, the cost of the multiplication is bounded by $T = T_1 + T_2$ with

$$T_1 = N(b) \cdot s \cdot |q| = N(b) |S_{n,d}|^2,$$

$$T_2 = E(b) (s \cdot p \cdot |q| + 2 \cdot s \cdot \overline{q}) = 2E(b) (|S_{n,d}| + |S_{n,2d-1}|),$$

where $\overline{d} := \lceil d/\beta \rceil$, since $\deg P \leq \overline{d} - 1$, $\deg Q \leq \overline{d} - 1$, and $\deg R \leq 2 \overline{d} - 1$. With $\alpha := d/n$, $\ell(\beta) := \log (2 - \beta)$, and using (1), there exists a constant $K_3$ such that:

$$\frac{\log T_1}{n} - \ell(\beta) - 2 \lambda(\beta) \leq K_3 \frac{\log n}{n},$$

$$\frac{\log T_2}{n} - \ell(\beta) - 2 \lambda(\beta) \leq K_3 \frac{\log n}{n} + \nu \frac{\log \log (\beta + 1)}{n}.$$

Since $\lambda(\alpha) = \log (1 + 1/\alpha)$ and since $|\overline{\alpha} - \alpha/\beta| \leq 1/n$, we can bound $|\lambda(\overline{\alpha}) - \lambda(\alpha/\beta)|$ by $\frac{\log (1 + n)}{n} \leq 2 \frac{\log n}{n}$, and deduce the existence of another constant $K_4$ such that:

$$\frac{\log T_1}{n} - \ell(\beta) - 2 \lambda(\alpha/\beta) \leq K_4 \frac{\log n}{n},$$

$$\frac{\log T_2}{n} - \ell(\beta) - 2 \lambda(\alpha/\beta) \leq K_4 \frac{\log n}{n} + \nu \frac{\log \log (\beta + 1)}{n}.$$

![Figure 1](image-url)  

Figure 1. Illustration of the curves $\phi_\beta(\alpha)$ for various $\beta$, together with $\psi(\alpha)$.  

5
From $\lambda'(\alpha) = \log (1 + 1/\alpha)$ we have that $(2 \lambda(\alpha) - \lambda(2 \alpha))^t > 0$, and since $\lim_{\alpha \to 0} \lambda(\alpha) = 0$ it follows that $2 \lambda(\alpha) - \lambda(2 \alpha) > 0$, and that there exists a constant $K_5$ such that

$$\frac{\log T}{n} - \ell(\beta) - 2 \lambda(\gamma) \leq K_5 \frac{\log n + \log \log (d+1)}{n}. \quad (3)$$

In order to express the execution time in terms of the output size, we thus have to study the function

$$\psi(\alpha) : = \min_{\beta \in (1,2,\ldots)} \phi_\beta(\alpha).$$

The first $\phi_\beta$ are plotted in Figure 1. The right limit of $\phi_1$ when $\alpha$ tends to 0 is 1, while the same limit for the other $\phi_\beta$ is $+\infty$. Since $\lambda(\alpha) > 0$ whenever $\alpha > 0$, the sign of $\phi_\beta - \phi_1$ is the same as the one of

$$\delta_\beta(\alpha) := \ell(\beta) + 2 \lambda(\alpha/\beta) - 2 \lambda(\alpha).$$

From $\delta'_\beta(\alpha) = 2/\beta \log (1 + \beta/\alpha) - 2 \log (1 + 1/\alpha) < 0$ and $\lim_{\alpha \to +\infty} \delta_\beta(\alpha) = (2 \beta - 1) - 2 \log \beta < 0$, it follows that there exists a unique zero of $\phi_\beta - \phi_1$. Since $\lambda(\alpha) > 0$ whenever $\alpha > 0$, the inequality $\phi_\beta(\alpha) \leq \lambda(\alpha)$ holds for all $\alpha > 0$. Therefore there exists a constant $A$, such that $\log T/\log |S_n, 2d - 1| < \lambda + \epsilon$ for all $d \geq 3$ and $n > A \log \log d$, or $d \leq 2$ and $n \geq A$.

It remains to bound $\log T/\log |S_n, 2d - 1|$ for $n \leq A \log \log d$ and $d$ sufficiently large. There exists a constant $K_6$ such that:

$$\log |S_n, d| - n \log d = \sum_{i=0}^{n-1} \log (1 + i/d) - \log n \leq 2 n \log n \leq K_6 (\log \log d)^2,$$

$$\log |S_n, 2d - 1| - n \log d \leq K_6 (\log \log d)^2.$$  

Therefore, taking $\beta := [\alpha/2]$, there exists a constant $K_7$ with

$$\log T \leq n \ell(\beta) + 2 n \log (d/\beta) + \nu \log \log d + K_7 (\log \log d)^2 \leq n \log (d/n + 1) + 2 n \log (2 n) + \nu \log \log d + K_7 (\log \log d)^2,$$

which implies that $\log T/\log |S_n, 2d - 1| \leq 1 + \epsilon < \zeta$ when $d$ is sufficiently large. \qed

Remark 5. In the proof of Theorem 4, it was convenient for computational purposes to explicitly choose $\beta$ in terms of $\alpha$. However our choice of $\beta$ is suboptimal. For instance, taking $\beta = [2.5 \alpha]$ for $\alpha > 4$ is generally better whenever $n$ remains in $O(\log \log d)$. In fact, we think that algorithm block-size will do the choice of the block size just as well and maybe even a little bit better from the latter bounds when using a function $T$ precise enough. A detailed proof of this claim sounds quite technical, so we have not pursued this direction any further in the present article. Nevertheless, the claim is plausible for the following reasons:

- For symmetry reasons, the block size $b$ computed by block-size should usually be of the form $b = (2k^i, \ldots, 2k^i)$ for some $k \in \mathbb{N}$ (and modulo permutation of the indeterminates).

- The complexity of block-multiply for $b$ of the above form is close to what we would obtain by taking $\beta = \sqrt{b_1 \cdots b_n} = 2^{k+1/n}$ in the proof of Theorem 4. When allowing for real $\beta$, the maximum of $\psi = \max_{\beta \in [1,\infty]} \phi_\beta(\alpha)$ is now reached at $\eta \approx 2.1438$, with $\phi_1(\eta) = \phi_\beta(\eta) = \zeta \approx 1.5336$ and $\beta \approx 3.7866$. For rings which do not support large evaluation-interpolation schemes we propose the following extension of Theorem 4.

Theorem 6. Let $\epsilon > 0$, and assume that $\mathcal{A}$ admits a unit. Then two polynomials in $\mathcal{A}[x]$ of degree at most $d - 1$ can be multiplied using at most $O((|S_n, 2d - 1|^{\epsilon + \epsilon})$ operations in $\mathcal{A}$.

Proof. We use the Chinese remaindering technique from [3]. It suffices to prove the theorem for $n \geq 2$. The unit of $\mathcal{A}$ is written $1$, and we introduce $\mathbb{B}_k := \mathcal{A}[\omega]/(\omega^k - 1)$ and $\mathbb{T}_l := \mathcal{A}[\omega]/(\omega^l - 1)$. Using the aforementioned TTT algorithm, we can multiply $P$ and $Q$ in $\mathbb{B}_k$ (resp. in $\mathbb{T}_l$) via Theorem 4, with $2k$ (resp. with $3l$) being the next power of 2 (resp. of 3) of $\beta$ if we do not perform the divisions by 2 (resp. by 3). We thus obtain $2k R$ and $3l R$. If $u 2k + v 3l = 1$ then we deduce $R$ as $u 2k R + v 3l R$.

Replacing all arithmetic operations in $\mathcal{A}$ by arithmetic operations in $\mathbb{B}_k \times \mathbb{T}_l$ gives rise to an additional overhead of $O(\log \beta \log \log \beta)$ with respect to the complexity analysis from the proof of Theorem 4. More precisely, we have a new constant $K_5$ such that

$$\frac{\log T}{n} - \ell(\beta) - 2 \lambda(\gamma) \leq K_5 \frac{\log n + \log d}{n}.$$  

The result is thus proved for when $n > A \log d$ for a sufficiently large constant $A$. It remains to examine the case when $n \leq A \log d$. Again, we use a similar proof as at the end of Theorem 4, with new constants $K_7$ and $K_8$ such that

$$\log T \leq n \ell(\beta) + 2 n \log (d/\beta) + \log d + K_7 (\log \log d)^2 \leq n \log (d/n + 1) + 2 n \log (2 n) + \log d + K_7 (\log \log d)^2 \leq (n + 1) \log d + K_8 n \log \log d + K_7 (\log \log d)^2.$$

Hence $\log T/\log |S_n, 2d - 1| \leq 3/2 + \epsilon < \zeta$ for $d$ sufficiently large. \qed

For rings which do not have a unit, we can use a Karatsuba evaluation-interpolation scheme. For this purpose we introduce

$$\Phi_\beta(\alpha) := \frac{\log^3 2}{\log 2} \log \beta + 2 \lambda(\alpha/\beta),$$

$$\Delta(\alpha) := \frac{\log^3 2}{\lambda(2 \alpha)} + \lambda(\tau) - \lambda(2 \tau).$$
Let \( \tau \) be the unique positive zero of \( \Delta \), and let \( \zeta_2 := \Phi_{\Phi}(32 \tau) \). Numerical computations lead to \( 1.2621 < \tau < 1.2622 \) and \( 1.5929 < \zeta_2 < 1.5930 \).

**Theorem 7.** Let \( \varepsilon > 0 \), let \( \Lambda \) be any ring. Then two polynomials in \( \Lambda[z] \) of degree at most \( d - 1 \) can be multiplied using at most \( O((|S_{n,2d-1}|^{1+\varepsilon}) \) operations in \( \Lambda \).

**Proof.** We use the Kuratsuba scheme [16, Section 2] with \( N(2^2) = 3^2 \) and \( E(2^2) = O(3^2) \) for all \( l \geq 1 \). We follow the proof of Theorem 4 and begin with a constant \( K_2 \) such that

\[
\left\lceil \frac{\log T}{n} \right\rceil - \log_2 \log \beta - 2 \lambda(\gamma) \leq K_2 \frac{n + \log \beta}{n}.
\]

In order to express the execution time in terms of the output size, we thus have to study the function

\[
\Psi(\alpha) := \min_{\beta \in \{1,2,3,\ldots,\alpha\}} \Phi_{\beta}(\alpha).
\]

The right limit of \( \Phi_1 \) when \( \alpha \) tends to 0 is 1, while the same limit for the other \( \Phi_\beta \) is \( +\infty \). Since \( \lambda(\alpha) > 0 \) whenever \( \alpha > 0 \), the sign of \( \Phi_\beta - \Phi_{\beta'} \) is the same as the one of

\[
\delta_{\beta,\beta'}(\alpha) := \frac{\log 3}{2 \log 2} \log \frac{\beta}{\beta'} + \lambda(\beta') - \lambda(\beta).
\]

If \( \beta > \beta' \), then \( \delta_{\beta,\beta'}(\alpha) < 0 \) and \( \lim_{\alpha \to +\infty} \delta_{\beta,\beta'}(\alpha) = \frac{\log 3}{2 \log 2} - 1 \), which implies the existence of a unique zero of \( \Phi_\beta - \Phi_{\beta'} \), written \( \rho_{\beta,\beta'} \). In particular, with \( \beta' = \beta/2 \), we obtain \( \rho_{\beta,\beta/2} = \beta/2 \). For convenience we let \( \rho_{\beta} := \rho_{\beta,\beta/2} \) and display the first few values:

<table>
<thead>
<tr>
<th>( \beta )</th>
<th>( \rho_{\beta} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>8.04850</td>
</tr>
<tr>
<td>8</td>
<td>10.0971</td>
</tr>
<tr>
<td>16</td>
<td>20.1942</td>
</tr>
<tr>
<td>32</td>
<td>40.3884</td>
</tr>
<tr>
<td>64</td>
<td>80.7768</td>
</tr>
<tr>
<td>128</td>
<td>161.5564</td>
</tr>
</tbody>
</table>

One can verify that \( \frac{\log \beta}{2 \log 2} \log (B/\beta) + \lambda(\beta \tau/B) - \lambda(\tau) = \frac{\log \beta}{2 \log 2} \log (1/t) + \lambda(\tau - \lambda(\tau)) > 0 \) holds for all \( t := \beta/B < 1 \), which implies that \( \Phi_{\rho_{\beta}}(\beta) > \Phi_{\rho_{\beta}}(\beta) \) for all \( B > 2 \). Therefore \( \Psi(\alpha) \leq \Phi_{\rho_{\beta}}(\alpha) \) with \( \beta \) such that \( \rho_{\beta} < \alpha \leq \rho_{\beta/2} \) (with the convention that \( \rho_0 = 0 \)). Lemma 10 of the appendix provides us with \( \Psi(\alpha) \leq \Phi_{\rho_{\beta}}(\alpha) \). Therefore there exists a constant \( A \), such that \( \log T/|S_{n,2d-1}| < \zeta_2 + \varepsilon \) for all \( d > 2 \) and \( n > A \log d \), or \( d \leq 1 \) and \( n > A \).

It remains to bound \( \log T/|S_{n,2d-1}| < n > A \log d \) and \( d \) sufficiently large, as in the end of the proof of Theorem 4. In this range we take \( \beta = 2^d \) with \( t := \left[ \frac{\log \alpha}{0.3 \log 2} \right] \), so that \( \beta \leq \alpha^{1/0.3} \). There exist constants \( K_7 \) and \( K_8 \) such that

\[
\log T \leq \left( (n+1) \frac{\log 3}{\log 2} - 1 \right) \log \beta + 2n \log (d/\beta) + K_7 n (\log \log d)^2.
\]

Therefore, using \( n \geq 2 \), we have

\[
\frac{\log T}{\log |S_{n,2d-1}|} < 1.5915 < \zeta_2
\]

whenever \( d \) is sufficiently large. \( \Box \)

### 5.2 Power series product

**Theorem 8.** Let \( \varepsilon > 0 \), and assume that \( \Lambda \) admits evaluation-interpolation schemes with \( N(d) = 2d - 1 \) and \( E(d) = O(d^{\log d} + 1) \), for all \( d \geq 1 \), and for some constant \( \nu \geq 0 \). Then two power series in \( \Lambda[z] \) truncated at order \( d \) can be multiplied with truncated-block-multiply using at most \( O(|S_{n,d}|^{1+\varepsilon}) \) operations in \( \Lambda \), if the sizes of the blocks \( b = (\beta, \ldots, \beta) \) are chosen as follows:

- \( \beta = 1 \), if \( d/n \leq 2 \eta \),
- \( \beta = 4 \), if \( 2 \eta < d/n \leq 16 \),
- \( \beta = \lfloor d/n \rfloor \), if \( 16 < d/n \).

**Proof.** The proof is similar to the one of Theorem 4. In the present case, for a suitable new constant \( K_2 \) the following inequalities hold:

\[
\left| \frac{\log |S_{n,d}|}{n} - \lambda(\alpha) \right| \leq K_2 \log n n,
\]

\[
\left| \frac{\log |S_{2n,d}|}{n} - 2 \lambda(\alpha/2) \right| \leq K_2 \log n n
\]

The cost of the multiplication is bounded by \( T = T_1 + T_2 \) with \( T_1 = N(b) S_{2n,d} \) and \( T_2 = 4 E(b) S_{n,d} \). There exists a new constant \( K_4 \) such that:

\[
\left| \frac{\log T_1}{n} - \nu(\beta) - 2 \lambda \frac{\alpha}{2} \right| \leq K_4 \log n n
\]

\[
\left| \frac{\log T_2}{n} - \nu(\beta) - 2 \lambda \frac{\alpha}{2} \right| \leq K_4 \log n n + \nu \log (\log (\beta + 1) n n
\]

The present situation is similar to the one of the proof of Theorem 4, with \( \alpha/2 \) instead of \( \alpha \). Therefore by taking \( \beta = 1 \) for \( \alpha < 2 \eta \), \( \beta = 4 \) for \( 2 \eta < \alpha \leq 16 \) and \( \beta = \lfloor \alpha \rfloor \) for larger \( \alpha \), there exists a constant \( A \), such that \( \log T/|S_{n,d}| < \zeta + \varepsilon \) for all \( d \geq 3 \) and \( n \geq A \log d \), or \( d \leq 2 \) and \( n > A \).

It remains to bound \( \log T/|S_{n,d}| \) for \( n \leq A \log d \) and \( d \) sufficiently large. There exists a new constant \( K_6 \) such that:

\[
\log |S_{n,d}|/n \log d \leq K_6 (\log \log d)^2;
\]

\[
|S_{n,2d-1}|/n \log d \leq K_6 (\log \log d)^2.
\]

Taking \( \beta = [\alpha] \), we thus have a constant \( K_7 \) with

\[
\log T \leq n \log (d/\beta) + \nu \log d + K_7 (\log \log d)^2
\]

\[
\leq n (d/n + 1) + 2 n \log (n) + \nu \log d + K_7 (\log \log d)^2.
\]

We conclude that \( \log T/|S_{n,d}| \leq 1 + \varepsilon < \zeta \) for \( d \) sufficiently large. \( \Box \)

### 6. REFERENCES


APPENDIX  TECHNICAL LEMMAS

This appendix contains details on how one can bound the functions $\phi_{[\alpha/2]}(\alpha)$ and $\Psi(\alpha)$ used in Section 5. We first prove the bounds in an explicit neighbourhood of infinity and then we rely on certified ball arithmetic for the remaining compact set.

**Lemma 9.** For all $\alpha > 8$ we have $\phi_{[\alpha/2]}(\alpha) < \zeta$.

**Proof.** Let $\varphi(\alpha) := \phi_{[\alpha/2]}(\alpha)$ and $\tilde{\varphi}(\alpha) := \phi_{[\alpha/2]}(\alpha)$. We shall first show that $\tilde{\varphi}(\alpha) < 0$ for all $\alpha > 0$. Let $\tilde{\varphi}(\alpha) := (\alpha - 1) \tilde{\varphi}(\alpha) \lambda^2(2 \alpha) = \lambda(2 \alpha) - 2 \lambda'(2 \alpha) (\alpha - 1) (\log(\alpha - 1) + 2 \lambda(2))$. Since $\tilde{\varphi}(8) < 0$, it suffices to show that $\tilde{\varphi}(\alpha) < 0$. Since $\tilde{\varphi}(\alpha) = -2 (2 \lambda''(2 \alpha) (\alpha - 1) + \lambda'(2 \alpha)) (\log(\alpha - 1) + 2 \lambda(2))$, we let $\tilde{\varphi}(\alpha) := 2 \lambda''(2 \alpha) (\alpha - 1) + \lambda'(2 \alpha)$, and have to prove that $\tilde{\varphi}(\alpha) < 0$. The latter inequality is correct since $\lim_{\alpha \to +\infty} \tilde{\varphi}(\alpha) = 0$, and $\tilde{\varphi}(\alpha) = -\frac{5 \alpha + 1}{\alpha(2 \alpha + 1)^3} \leq 0$ for $\alpha \geq 8$. On the other hand, for $\alpha \geq 16$ we have

$$|\varphi(\alpha) - \tilde{\varphi}(\alpha)| \leq \frac{\log(2 [\alpha/2] - 1) - \log(\alpha - 1)}{\lambda(2 \alpha)} + \frac{2 \lambda(2) - 2 \lambda(\alpha/[\alpha/2])}{\lambda(2 \alpha)} \leq \frac{\log(1 + 3/15) + 2 (\lambda(2) - \lambda(2^{16}))}{\lambda(32)} < 0.07.$$  

Since $\tilde{\varphi}(16) < 1.39$, we have shown that $\varphi(\alpha) < \zeta$ for $\alpha \geq 16$. For $\alpha$ between 8 and 16, we appeal to numeric computations to verify that $\varphi(\alpha) < \zeta$ still holds. □

**Lemma 10.** For all $\alpha > 0$ we have $\Psi(\alpha) \leq \zeta_2$.

**Proof.** Let $\kappa := \log 3/\log 2$, and $\zeta := 2 \lambda(\tau) - \kappa \log \tau \simeq 2.7365$. Let $\tilde{\varphi}(\alpha) := \Phi_{\alpha/\varphi}(\alpha)$. We shall first show that $\tilde{\varphi}(\alpha) < 0$ for all $\alpha \geq 100$. Let $\tilde{\varphi}(\alpha) := \alpha \tilde{\varphi}(\alpha) \lambda^2(2 \alpha) = \kappa \lambda(2 \alpha) - 2 \lambda'(2 \alpha) \alpha (\kappa \log \alpha + \zeta)$. Since $\tilde{\varphi}(100) < 0$, it suffices to show that $\tilde{\varphi}(\alpha) < 0$. Since $\tilde{\varphi}(\alpha) = -2 (2 \alpha \lambda''(2 \alpha) + \lambda'(2 \alpha)) (\kappa \log \alpha + \zeta)$, we let $\tilde{\varphi}(\alpha) := \alpha \lambda''(2 \alpha) + \lambda'(\alpha)$, and have to prove that $\tilde{\varphi}(\alpha) \geq 0$. The latter inequality is correct since $\lim_{\alpha \to +\infty} \tilde{\varphi}(\alpha) = 0$, and $\tilde{\varphi}(\alpha) = -\frac{\alpha}{\alpha^{2}(2 \alpha + 1)^2} \leq 0$.

Let $l := \left\lceil \frac{\log(\alpha/\varphi) - \log \tau}{\log 2} \right\rceil$ and $\beta_l := 2^l$. We introduce $\varphi(\alpha) := \Phi_{\alpha/\varphi}(\alpha)$. Since $\alpha/(2 \tau) < \beta_l \leq \alpha/\tau$, for $\alpha \geq 2^{256}$, we have

$$|\varphi(\alpha) - \tilde{\varphi}(\alpha)| \leq \frac{2 (\lambda(\alpha/\beta) - \lambda(\tau))}{\lambda(2 \alpha)} \leq \frac{\log \beta}{\lambda(2^{256})} < 0.0062.$$  

Since $\tilde{\varphi}(2^{256}) < 1.5853$, we have shown that $\varphi(\alpha) < \zeta_2$ for $\alpha \geq 2^{256}$.

For $2^{16} \leq \alpha < 2^{256}$, we symbolically computed $\Phi_{\alpha/\varphi}(\alpha)$ and straightforwardly lower bounded it by a function $\Gamma(\beta)$ in the domain $\beta \tau \leq \alpha \leq 2 \beta \tau$. For each value of $\beta = 2^{16}$, $2^{17}$, ..., $2^{255}$, we used ball arithmetic to show that $\Gamma(\beta) > 0$. For smaller values of $\alpha$, we could directly compute that $\Phi_{\alpha/\varphi}(\alpha) > 0$ whenever $\beta \tau \leq \alpha \leq 2 \beta \tau$. Finally we computed that $\Phi_{\beta}(\beta \tau) \leq \zeta_2$ for all $\beta = 2, 4, 8, ..., 256$, which concludes the proof. □