Modular algorithms for polynomial basis conversion and greatest factorial factorization

February 9, 2000

Jürgen Gerhard
Fachbereich 17 Mathematik-Informatik
Universität-GH Paderborn
D-33095 Paderborn, Germany

Abstract

We give new algorithms for converting between representations of polynomials with respect to certain kinds of bases, comprising the usual monomial basis and the falling factorial basis, for fast multiplication and Taylor shift in the falling factorial basis, and for computing the greatest factorial factorization. We analyze both the classical and the new algorithms in terms of arithmetic coefficient operations. For the special case of polynomials with integer coefficients, we present modular variants of these methods and give cost estimates in terms of word operations.

1 Introduction

In difference algebra, one is interested in difference (or recurrence) equations of the form

$$a_n(x)f(x+n)+a_{n-1}(x)f(x+n-1)+\cdots+a_1(x)f(x+1)+a_0(x)f(x)=g(x),$$

where $a_0, a_1, \ldots, a_n, g$ are given functions, often polynomials, and f is to be determined. A popular scheme for finding a polynomial solution f is to find first an upper bound d on the degree of f, then substitute a polynomial of degree d with undetermined coefficients in the above equation, and solve the resulting system of linear equations. In some of these algorithms (for example, Abramov *et al.* 1995), it turns out to be useful not to represent the a_i, f, and g with respect to the usual basis $1, x, x^2, x^3, \ldots$ for polynomials, but to use instead the *falling factorial basis* $1, x, x^{\underline{2}} = x\cdot(x-1), x^{\underline{3}} = x\cdot(x-1)(x-2), \ldots$. A special case is the symbolic summation of polynomials, where a polynomial g is given and we want to find an antidifference f, such that $f(x+1)-f(x)=g(x)$ (see §2.6 in Graham *et al.* 1994 or §23.1 in von zur Gathen & Gerhard 1999).

In Section 2 below, we discuss and analyze conversion algorithms between the falling factoring basis and the usual monomial basis. More generally, these algorithms apply to polynomial bases $p_0, p_1, p_2, p_3, \ldots$ of the form $p_0 = 1$ and $p_i = (x+b_i)p_{i-1}$ for $i \geq 1$, where the b_i are fixed constants. Besides the falling factorials, this includes other special cases: the *Newton representation*, where all b_i's are distinct, and the *Taylor shift*, where all b_i's are identical. Several algorithms for the latter problem have been discussed in von zur Gathen & Gerhard (1997). Our basis conversion algorithms are closely related to the fast algorithms for multipoint evaluation and interpolation by Borodin & Moenck (1974). The latter algorithms are in the style of Lagrange interpolation, while ours are more in the style of Newton interpolation. For polynomials of degree at most n, our algorithms take $O(\mathsf{M}(n)\log n)$ arithmetic operations on coefficients, where M is a multiplication time (see below), while the classical algorithms take $O(n^2)$. For polynomials with integer coefficients, we also discuss and analyze modular variants of these algorithms.

In Section 3, we present a new algorithm that reduces multiplication in the falling factorial basis to ordinary polynomial multiplication in the monomial basis, and thus yields an $O(\mathsf{M}(n))$ algorithm for multiplication in the falling factorial basis. We also give an $O(\mathsf{M}(n))$ algorithm for Taylor shift in the falling factorial basis, which is an adaption of an algorithm by Aho *et al.* (1975) for Taylor shifts in the monomial basis. The classical algorithms for both problems take time $O(n^2)$.

The *greatest factorial factorization* (gff) has been introduced in Paule (1995) as a discrete analog of the squarefree factorization (see also §23.3 in von zur Gathen & Gerhard 1999). In Section 4, we discuss and analyze a new fast algorithm for computing the gff, which is an adaption of Yun's (1976) algorithm for the squarefree factorization. Both algorithms take $O(\mathsf{M}(n)\log n)$ arithmetic operations on coefficients for polynomials of degree n, while Paule's (1995) algorithm for gff computation takes $O(n\,\mathsf{M}(n)\log n)$. Moreover, we give a new modular algorithm for computing the gff in $\mathbb{Z}[x]$. An analogous modular squarefree factorization algorithm appears in Gerhard (1999).

The three sections are essentially independent. In the remainder of this section, we will fix notation and state some known results for later use. The reader may skip this at first reading and look up details later if necessary.

In our cost analyses, we use two different cost measures. In the *arithmetic cost* model, we count only additions, multiplications, or divisions in the underlying ring. For the ring $\mathbb{Z}$, we refer to a more realistic machine model: we think of a hypothetical processor that executes "simple" operations on *single precision* integers or *words*, which are integers with a fixed number ω of bits. Common values are $\omega = 32$ or $\omega = 64$. We count the number of such *word operations*. These usually include basic arithmetic operations, memory access, program control, etc. In the latter model, the *word length* of an integer $a \in \mathbb{Z}$ is the number of words it occupies in storage, namely, $1 + \lfloor(\log_2|a|)/\omega\rfloor$. We employ these two models quite informally. A *single precision primes* is a prime $p \in \mathbb{N}$ fitting precisely into one machine word, so that $2^{\omega-1} < p < 2^{\omega}$.

We use throughout the notion of a *multiplication time* M. This is a function $\mathbb{N} \longrightarrow \mathbb{R}_{\geq 0}$ such that polynomials of degree n over a ring R or integers of word length at most n can be multiplied with $\mathsf{M}(n)$ arithmetic operations in R or word operations, respectively (see Section 8.3 in von zur Gathen & Gerhard 1999). We assume that M is *subadditive*, so that $\mathsf{M}(n+m) \geq \mathsf{M}(n) + \mathsf{M}(m)$ for all $n, m \in \mathbb{N}$, and that $\mathsf{M}(n) \geq n$ for all n. The *classical* multiplication algorithm that we learned in school has $\mathsf{M}(n) \in O(n^2)$. The time estimate for Karatsuba's algorithm (Karatsuba & Ofman 1962) is $\mathsf{M}(n) \in O(n^{1.59})$, and the fastest currently known algorithm has $\mathsf{M}(n) \in O(n \log n \operatorname{loglog} n)$ (Schönhage & Strassen 1971, Schönhage 1977, Cantor & Kaltofen 1991).

We employ the following well-known facts. Proofs and references for them can be found, for example, in von zur Gathen & Gerhard (1999), Sections 2.4, 3.3, 5.4, 8.4, 9.1, 10.1–10.3, and 11.1.

Fact 1.1 *(i) We can multiply two polynomials in $\mathbb{Z}[x]$ of degree at most n and with coefficients of word length at most λ using $O(n^2\lambda^2)$ word operations with classical arithmetic and $O(\mathsf{M}(n\lambda))$ with fast arithmetic.*

(ii) Division with remainder for polynomials of degree at most n (integers of word length at most n) takes $O(n^2)$ arithmetic operations (word operations) with classical arithmetic, and $O(\mathsf{M}(n))$ with fast arithmetic.

(iii) The Extended Euclidean Algorithm for polynomials of degree at most n (integers of length at most n) can be performed with $O(n^2)$ arithmetic operations (word operations) using classical arithmetic, and $O(\mathsf{M}(n)\log n)$ with fast arithmetic.

(iv) We can multiply at most $r \leq n$ nonconstant polynomials (nonzero integers), whose product has degree n (word length n), taking $O(n^2)$ arithmetic operations (word operations) with classical arithmetic and $O(\mathsf{M}(n)\log r)$ with fast arithmetic.

(v) We can evaluate a polynomial of degree less than n at $r \geq n$ points (an integer of word length at most n modulo $r \geq n$ single precision integers) using $O(rn)$ arithmetic operations (word operations) with classical arithmetic and $O(\mathsf{M}(r)\log r)$ with fast arithmetic.

(vi) We can interpolate a polynomial of degree less than n at n points (execute the Chinese Remainder Algorithm for n single precision moduli) using $O(n^2)$ arithmetic operations (word operations) with classical arithmetic, and $O(\mathsf{M}(n)\log n)$ with fast arithmetic.

We note that in all of the above cost estimates for fast arithmetic, the logarithmic factors can be dropped when using a strictly superlinear multiplication time, such as given by Karatsuba's algorithm, with $\mathsf{M}(n) \in \Omega(n^{1+\varepsilon})$ for some positive ε. This remark also applies to all cost estimates below.

For a polynomial $f = \sum_{0\leq i\leq n} f_i x^i \in \mathbb{Z}[x]$, we denote its *max-norm* by $\|f\|_\infty = \max\{|f_i| : 0 \leq i \leq n\}$. We also make use of the *two-norm* $\|f\|_2 = (\sum_{0\leq i\leq n} f_i^2)^{1/2}$, the *one-norm* $\|f\|_1 = \sum_{0\leq i\leq n} |f_i|$, and the inequalities $\|f\|_\infty \leq \|f\|_2 \leq (n+1)^{1/2}\|f\|_\infty$ and $\|f\|_2 \leq \|f\|_1 \leq (n+1)\|f\|_\infty$. The following well-known fact is from Mignotte (1989), Theorème IV.4.4.

Fact 1.2 *Let $f, g_1, \ldots, g_r \in \mathbb{Z}[x]$ be nonconstant such that $f = g_1 \cdots g_r$, and $n = \deg f$. Then*

$$\prod_{1\leq i\leq r} \|g_i\|_1 \leq 2^n \|f\|_2 \leq (n+1)^{1/2} 2^n \|f\|_\infty.$$

We stress that the cost estimates for all our modular algorithms are not strictly asymptotic estimates, since they require a certain number of single precision primes and there are only finitely many of them for a fixed ω. It is possible to derive proper asymptotic estimates by allowing for primes of arbitrary precision and using the prime number theorem to bound their word length. The corresponding estimates are more complicated, and we do not state them here

2 Change of basis

In this section, we discuss conversion algorithms for univariate polynomials in $R[x]$, where R is a commutative ring with 1. These algorithms convert between a basis of the form

$$\mathcal{M}_{\mathbf{b}} = \Big((x+b_1)(x+b_2)\cdots(x+b_i)\Big)_{i\in\mathbb{N}}, \tag{2.1}$$

where $\mathbf{b} = (b_1, b_2, \ldots) \in R^{\mathbb{N}}$ is an infinite sequence of arbitrary constants from R, and the usual *monomial basis* $\mathcal{M} = \mathcal{M}_{\mathbf{0}} = (x^i)_{i\in\mathbb{N}}$. This comprises important special cases such as the *shifted monomial basis* $((x+b)^i)_{i\in\mathbb{N}}$, for some $b \in R$, the *falling factorial basis* $(x^{\underline{i}})_{i\in\mathbb{N}}$, where $x^{\underline{i}} = x(x-1)\cdots(x-i+1)$ is the *ith falling factorial*, and the *rising factorial basis* $(x^{\overline{i}})_{i\in\mathbb{N}}$, where $x^{\overline{i}} = x(x+1)\cdots(x+i-1)$ is the *ith rising factorial*. We give cost estimates for our algorithms in terms of arithmetic operations in R and, in the case $R = \mathbb{Z}$, also in word operations. For the shifted monomial basis, the conversion is simply a *Taylor shift*, and six algorithms for this problem have been discussed in von zur Gathen & Gerhard (1997), where the reader also finds some references.

Given the coefficients $f_0, \ldots, f_n \in R$ of a polynomial $f = \sum_{0\le i\le n} f_i x^i \in R[x]$, we want to find $g_0, \ldots, g_n \in R$ such that

$$f = \sum_{0\le i\le n} g_i \cdot (x+b_1)\cdots(x+b_i), \tag{2.2}$$

or vice versa. If all b_i's are distinct, then the right hand side in (2.2) is the representation of f that we obtain from *Newton interpolation* at the sample points $-b_1, \ldots, -b_n$, and the g_i's are the divided differences with respect to these points.

The main idea of the following algorithms is to employ *Horner's rule*

$$f = g_0 + (x+b_1)\Big(g_1 + \cdots + (x+b_{n-1})\Big(g_{n-1} + (x+b_n)g_n\Big)\cdots\Big). \tag{2.3}$$

To analyze the bit cost of the algorithms, we need some upper bounds. If h is a polynomial and $i \in \mathbb{Z}$, then we denote the coefficient of x^i in h by $[x^i]h$, with the convention that it be zero if $i < 0$ or $i > \deg h$.

Lemma 2.1 *Let $R = \mathbb{Z}$, $n \in \mathbb{N}_{\ge 1}$, and $b_1, \ldots, b_n \in \mathbb{Z}$ of absolute value at most b. Moreover, let $u_n = (x+b_1)\cdots(x+b_n)$, $u_n^* = \mathrm{rev}(u_n) = (1+b_1x)\cdots(1+b_nx)$, and $v_n \in \mathbb{Z}[x]$ of degree at most n such that $u_n^* v_n \equiv 1 \bmod x^{n+1}$.*

(i) $\|u_n^*\|_\infty = \|u_n\|_\infty \leq \|u_n\|_1 \leq (b+1)^n$ *and* $\|v_n\|_\infty \leq (4b)^n$.

(ii) *If* $f_0, \ldots, f_n, g_0, \ldots, g_n \in \mathbb{Z}$ *are such that (2.2) holds,* $A_\infty = \max\{|f_i|: 0 \leq i \leq n\}$, $B_\infty = \max\{|g_i|: 0 \leq i \leq n\}$, *and* $B_1 = \sum_{0\leq i\leq n} |g_i|$, *then* $A_\infty \leq (b+1)^n B_1 \leq (n+1)(b+1)^n B_\infty$ *and* $B_\infty \leq (n+1)(4b)^n A_\infty$.

Proof. The claim about $\|u_n\|_1$ follows since the one-norm is submultiplicative. We have $v_n \equiv \prod_{1\leq i\leq n} h_i \bmod x^{n+1}$, where $h_i = 1/(1+b_i x) = \sum_{j\geq 0}(-b_i)^j x^j \in R[[x]]$ for $1 \leq i \leq n$. Let $v_n = \sum_{0\leq k\leq n} w_k x^k$, with all $w_k \in \mathbb{Z}$. Then

$$|w_k| = \left| \sum_{j_1+\cdots+j_n=k} (-b_1)^{j_1} \cdots (-b_n)^{j_n} \right| \leq \binom{n+k-1}{n-1} b^k \leq \binom{2n-1}{n-1} b^n \leq (4b)^n$$

for $0 \leq k \leq n$. This concludes the proof of (i).

For (ii), we obtain from (2.2) and (i) that

$$A_\infty = \|f\|_\infty \leq \sum_{0\leq i\leq n} |g_i|\, \|u_i\|_\infty \leq (b+1)^n B_1 \leq (n+1)(b+1)^n B_\infty.$$

Let $f^* = \mathrm{rev}(f)$ and $0 \leq j < n$. To prove the second inequality, we claim that $g_j = [x^{n-j}](f^* v_{j+1})$. If $n \geq i > j$, then

$$u_i^* v_{j+1} \equiv \frac{(1+b_1 x)\cdots(1+b_i x)}{(1+b_1 x)\cdots(1+b_{j+1}x)} = (1+b_{j+2}x)\cdots(1+b_i x) \bmod x^{n+1},$$

and hence $[x^{i-j}](u_i^* v_{j+1}) = 0$. If $i = j$, then $[x^{i-j}](u_i^* v_{j+1}) = u_j^*(0) v_{j+1}(0) = 1$, and if $i < j$, then $[x^{i-j}](u_i^* v_{j+1}) = 0$, by definition. Thus (2.2) yields

$$\begin{aligned} [x^{n-j}](f^* v_{j+1}) &= [x^{n-j}] \sum_{0\leq i\leq n} g_i u_i^* x^{n-i} v_{j+1} = \sum_{0\leq i\leq n} g_i [x^{n-j}](u_i^* v_{j+1} x^{n-i}) \\ &= \sum_{0\leq i\leq n} g_i [x^{i-j}](u_i^* v_{j+1}) = g_j [x^{j-j}](u_j^* v_{j+1}) = g_j, \end{aligned}$$

and the claim is proved. Finally,

$$|g_j| \leq \|f^* v_{j+1}\|_\infty \leq (n+1)\|f\|_\infty \|v_{j+1}\|_\infty \leq (n+1)(4b)^n A_\infty,$$

by (i). The second inequality in (ii) now follows since $|g_n| = |f_n| \leq A_\infty$. □

Theorem 2.2 *Let* $f \in R[x]$ *of degree* $n \geq 1$ *and* $\mathbf{b} = (b_1, b_2, \ldots) \in R^{\mathbb{N}}$. *Using classical arithmetic, we can compute the coefficients* $g_0, \ldots, g_n$ *of* f *with respect to* $\mathcal{M}_\mathbf{b}$ *from the usual coefficients, and also vice versa, with* $O(n^2)$ *additions and multiplications in* R. *If* $R = \mathbb{Z}$, $|b_i| < 2^d$ *for all* $i \leq n$, *and* $\|f\|_\infty < 2^\lambda$, *then the cost for computing the* g_i *from the usual coefficients is* $O(n^2 d(nd+\lambda))$ *word operations. The same estimate is valid for computing the usual coefficients from the* g_i *if* $|g_i| < 2^\lambda$ *for all* i.

Proof. For computing the usual coefficients from $g_0, \ldots, g_n$, we employ the Horner-like scheme (2.3). In n steps, we compute

$$f^{(n)} = g_n, \quad f^{(i)} = (x + b_{i+1}) f^{(i+1)} + g_i \text{ for } i = n-1, \ldots, 0, \tag{2.4}$$

and obtain $f = f^{(0)}$. Then the invariant

$$f = f^{(i)} \cdot (x + b_1) \cdots (x + b_i) + \sum_{0 \le j < i} g_j \cdot (x + b_1) \cdots (x + b_j) \tag{2.5}$$

holds for $0 \le i \le n$. The cost for step $n - i$ is i multiplications and $i - 1$ additions in R, and summing up yields a total cost of $O(n^2)$ ring operations.

For the other conversion, we simply reverse the scheme (2.4), by letting $f^{(0)} = f$, obtaining $f^{(i+1)}$ and g_i as quotient and remainder in the division of $f^{(i)}$ by $x + b_{i+1}$, and finally $g_n = f^{(n)}$. This takes exactly the same number of arithmetic operations as the Horner-like scheme (each addition in the former corresponds to a subtraction in the latter).

If $R = \mathbb{Z}$, then we have to estimate the size of the coefficients of the $f^{(i)}$. By (2.5), we have

$$f^{(i)} = \sum_{i \le j \le n} g_j \cdot (x + b_{i+1}) \cdots (x + b_j) \tag{2.6}$$

for all i. If we start with $g_0, \ldots, g_n$ of absolute value less than 2^λ, then Lemma 2.1 (ii) shows that $\|f^{(i)}\|_\infty < (n+1)2^{nd+\lambda}$, and hence the word length of the coefficients of all $f^{(i)}$ is $O(nd + \lambda)$. Thus in step $n - i$, we have i integer multiplications of size $O(d) \times O(nd + \lambda)$ and $i - 1$ additions of integers of length $O(nd + \lambda)$, for $0 \le i \le n$. This yields a total cost of $O(n^2 d(nd + \lambda))$ word operations.

For the reverse transformation, when the usual coefficients of f are given and of absolute value less than 2^λ, then Lemma 2.1 (ii) implies that $|g_i| < (n+1)2^{n(d+2)+\lambda}$ for $0 \le i \le n$. Now (2.6) together with another application of the same Lemma shows that $\|f^{(i)}\|_\infty \le (n+1)^2 2^{2n(d+1)+\lambda}$, and hence the coefficients of all $f^{(i)}$ are of word length $O(nd + \lambda)$. This yields an overall cost estimate of $O(n^2 d(nd + \lambda))$ word operations. □

For the falling (or rising) factorial coefficients, we have $b_i = \mp(i - 1)$ for all i and $d = \lceil \log_2 n \rceil$.

Corollary 2.3 *Let $f \in R[x]$ of degree $n \ge 1$. Using classical arithmetic, we can compute the falling (or rising) factorial coefficients $g_0, \ldots, g_n$ of f from its usual coefficients, and also vice versa, with $O(n^2)$ additions and multiplications in R. If $R = \mathbb{Z}$ and $\|f\|_\infty < 2^\lambda$, then the cost for computing the falling (or rising) factorial coefficients from the usual coefficients is $O(n^2 \log n \cdot (n \log n + \lambda))$ word operations. The same estimate is valid for computing the usual coefficients from the falling (or rising) factorial coefficients if $|g_i| < 2^\lambda$ for all i.*

The following two theorems analyze conversion algorithms applying the divide-and-conquer technique from the fast evaluation and interpolation algorithms by Borodin & Moenck (1974) (see also Strassen 1973, and Strassen 1974 and §4.5 in Borodin & Munro (1975) for a survey) to the Horner-like scheme (2.3) (see also method E in von zur Gathen & Gerhard 1997 and Chapter 10 in von zur Gathen & Gerhard 1999).

Theorem 2.4 *Let $f \in R[x]$ of degree less than $n \geq 2$ and $\mathbf{b} = (b_1, b_2, \ldots)$ in $R^{\mathbb{N}}$. Using fast arithmetic, we can compute the coefficients $g_0, \ldots, g_{n-1}$ of f with respect to $\mathcal{M}_{\mathbf{b}}$ using $O(\mathsf{M}(n) \log n)$ additions and multiplications in R. If $R = \mathbb{Z}$, $\|f\|_\infty < 2^\lambda$, and $|b_i| < 2^d$ for $i < n$, then the cost is $O(\mathsf{M}(n^2 d + n\lambda) \log n)$ or $O^\sim(n^2 d + n\lambda)$ word operations, where the $O^\sim$ notation suppresses logarithmic factors.*

Proof. We assume that $n = 2^k$ is a power of two. In a precomputation stage, we first compute recursively the subproducts

$$m_{ij} = (x + b_{j2^i+1}) \cdots (x + b_{(j+1)2^i}) \tag{2.7}$$

of $(x+b_1) \cdots (x+b_n)$, for $0 \leq i < k$ and $0 \leq j < 2^{k-i}$. This takes $O(\mathsf{M}(n) \log n)$ ring operations, by Fact 1.1 (iv). In the main stage, we divide f with remainder to obtain $f^{(0)}, f^{(1)} \in R[x]$ of degrees less than $n/2$ such that $f = f^{(0)} + m_{k-1,0} \cdot f^{(1)}$, and proceed recursively to compute the coefficients with respect to $\mathcal{M}_{\mathbf{b}}$ of

$$\begin{aligned} f^{(0)} &= \sum_{0 \leq i < n/2} g_i \cdot (x + b_1) \cdots (x + b_i), \\ f^{(1)} &= \sum_{0 \leq i < n/2} g_{n/2+i} \cdot (x + b_{n/2+1}) \cdots (x + b_{n/2+i}). \end{aligned} \tag{2.8}$$

Then

$$\begin{aligned} f &= f^{(0)} + (x + b_1) \cdots (x + b_{n/2}) \cdot f^{(1)} \\ &= \sum_{0 \leq i < n/2} g_i \cdot (x + b_1) \cdots (x + b_i) + \sum_{0 \leq i < n/2} g_{n+i} \cdot (x + b_1) \cdots (x + b_{n/2+i}) \\ &= \sum_{0 \leq i < n} g_i \cdot (x + b_1) \cdots (x + b_i). \end{aligned}$$

The cost for the division with remainder is $O(\mathsf{M}(n))$ ring operations. If $T(n)$ denotes the cost of the main stage, then we have $T(1) = 0$ and $T(n) \in 2T(n/2) + O(\mathsf{M}(n))$, and unravelling the recursion yields $T(n) \in O(\mathsf{M}(n) \log n)$. This dominates the cost for the precomputation stage.

If $R = \mathbb{Z}$, then $\|m_{ij}\|_\infty \leq 2^{nd}$, by Lemma 2.1 (i), for all i, j. Thus we can compute all m_{ij} with $O(\mathsf{M}(n^2 d) \log n)$ word operations, by Fact 1.1 (i) and (iv). Lemma 2.1 (ii) shows that $|g_i| < n 2^{n(d+2)+\lambda}$ for all i. As in the proof of Theorem 2.2, another application of the same lemma yields $\|f^{(0)}\|_\infty, \|f^{(1)}\|_\infty < n^2 2^{2n(d+1)+\lambda}$, and hence the coefficients of $f^{(0)}$ and $f^{(1)}$ are of word size $O(nd + \lambda)$. We perform the division with remainder by first computing $\mathrm{rev}(m_{k-1,0})^{-1}$ modulo $x^{n/2}$ using Newton iteration, then multiplying the result by $\mathrm{rev}(f)$ modulo $x^{n/2}$ to get $\mathrm{rev}(f^{(1)})$, and finally obtain $f^{(0)} = f - m_{k-1,0} f^{(1)}$, as described, for example, in Section 9.1 of von zur Gathen & Gerhard (1999). Lemma 2.1 (i) implies that the coefficients of $\mathrm{rev}(m_{k-1,0})^{-1} \bmod x^{n/2}$ have word length $O(nd)$, and hence the coefficients of all intermediate results in the computation have word length $O(nd+\lambda)$. Thus the cost for the division with remainder is $O(\mathsf{M}(n^2d+n\lambda))$ word operations. If $T(n)$ denotes the cost in word operations for the main stage, then $T(1)=0$ and $T(n) \in 2T(n/2)+O(\mathsf{M}(n^2d+n\lambda))$, and we obtain $T(n) \in O(\mathsf{M}(n^2d+n\lambda)\log n)$. $\square$

Theorem 2.5 *Let $n \geq 2$ and $b_1,\ldots,b_{n-1},g_0,\ldots,g_{n-1} \in R$. We can compute the coefficients of $f=\sum_{0\leq i<n} g_i \cdot (x+b_1)\cdots(x+b_i) \in R[x]$ with $O(\mathsf{M}(n)\log n)$ additions and multiplications in R, using fast arithmetic,. If $R=\mathbb{Z}$, $|g_i|<2^\lambda$, and $|b_i|<2^d$ for all $i<n$, then the cost is $O(\mathsf{M}(n^2d+n\lambda)\log n)$ or $O^\sim(n^2d+n\lambda)$ word operations.*

Proof. The proof parallels the proof of Theorem 2.4. Again, we assume that $n=2^k$ is a power of two, and precompute the coefficients of the subproducts m_{ij} for $0 \leq i < k$ and $0 \leq j < 2^{k-i}$, as defined in (2.7), using $O(\mathsf{M}(n)\log n)$ ring operations. In the main stage, we recursively compute the coefficients of $f^{(0)}, f^{(1)}$ as in (2.8), and then obtain $f=f^{(0)}+m_{k-1,0}\cdot f^{(1)}$. Multiplying $m_{k-1,0}$ by $f^{(1)}$ and adding $f^{(0)}$ takes $O(\mathsf{M}(n))$ additions and multiplications in R. If $T(n)$ denotes the cost of the main stage, then we have $T(1)=0$ and $T(n) \in 2T(n/2)+O(\mathsf{M}(n))$, and unravelling the recursion yields $T(n) \in O(\mathsf{M}(n)\log n)$. This dominates the cost for the precomputation stage.

If $R=\mathbb{Z}$, then $\|m_{ij}\|_\infty \leq 2^{nd}$ for all i,j, as in the proof of Theorem 2.4, and computing all m_{ij} takes $O(\mathsf{M}(n^2d)\log n)$ word operations. Now $\|f^{(0)}\|_\infty$ and $\|f^{(1)}\|_\infty$ are less than $n2^{nd+\lambda}$, by Lemma 2.1 (ii), and their word size is $O(nd+\lambda)$. Thus multiplying $m_{k-1,0}$ by $f^{(1)}$ and adding $f^{(0)}$ takes $O(\mathsf{M}(n^2d+n\lambda))$ word operations. If now $T(n)$ denotes the cost of the main stage in word operations, then we have $T(1)=0$ and $T(n) \in 2T(n/2)+O(\mathsf{M}(n^2d+n\lambda))$, and as usual, we find $T(n) \in O(\mathsf{M}(n^2d+n\lambda)\log n)$. $\square$

The output size for both conversion problems is $O(n^2d+n\lambda)$ if the input size is $\Theta(n\lambda)$, by Lemma 2.1 (ii), and hence the estimates of Theorems 2.4 and 2.5 are—up to logarithmic factors—optimal for those inputs where the output size is close to the upper bound.

Corollary 2.6 *Let $f \in R[x]$ of degree less than $n \geq 2$. Using fast arithmetic, we can compute the falling (or rising) factorial coefficients of f from the usual ones using $O(\mathsf{M}(n)\log n)$ ring operations. If $R=\mathbb{Z}$ and $\|f\|_\infty < 2^\lambda$, then the cost is $O(\mathsf{M}(n^2\log n+n\lambda)\log n)$ or $O^\sim(n^2+n\lambda)$ word operations. The same estimates are valid for computing the usual coefficients from the falling (or rising) factorial coefficients if the latter are absolutely less than 2^λ.*

For the special case of Taylor shift, where all b_i are equal, the *convolution method* (Aho *et al.* 1975, see also Schönhage *et al.* 1994, §9.3, and von zur Gathen & Gerhard 1997, method F) reduces the computation to essentially one polynomial multiplication and thus saves a factor of $\log n$ in comparison to the estimates of Theorems 2.4 and 2.5. For later reference, we state this fact without proof.

Fact 2.7 *Let $f \in R[x]$ of degree $n \geq 1$ and $b \in R$. Then we can compute the coefficients of $f(x+b)$ using $O(n^2)$ additions and multiplications in R using classical arithmetic, and $O(\mathsf{M}(n))$ with fast arithmetic.*

We now present modular algorithms for converting between $\mathcal{M}$ and $\mathcal{M}_{\mathbf{b}}$. We recall that ω is the word size of our hypothetical processor.

Algorithm 2.8 *Modular conversion from $\mathcal{M}$ to $\mathcal{M}_{\mathbf{b}}$.*
Input: *A polynomial $f \in \mathbb{Z}[x]$ of degree $n \geq 1$ and max-norm less than 2^λ, and $b_1, \ldots, b_n \in \mathbb{Z}$ of absolute value less than 2^d.*
Output: *Integers $g_0, \ldots, g_n$ such that $f = \sum_{0 \leq i \leq n} g_i \cdot (x+b_1) \cdots (x+b_i)$.*

1. $C \leftarrow (n+1)2^{n(d+2)+\lambda}$, $\quad r \leftarrow \lceil (\log_2 2C)/(\omega - 1) \rceil$
   choose odd single precision primes $p_1 < \cdots < p_r$
2. *for $1 \leq j \leq r$ compute $g_{0j}, \ldots, g_{nj} \in \mathbb{Z}$ of absolute value less than $p_j/2$ such that $f \equiv \sum_{0 \leq i \leq n} g_{ij} \cdot (x+b_1) \cdots (x+b_i) \bmod p_j$*
3. *for $0 \leq i \leq n$ use the Chinese Remainder Algorithm to compute $g_i \in \mathbb{Z}$ of absolute value less than $(\prod_{1 \leq j \leq r} p_j)/2$ such that $g_i \equiv g_{ij} \bmod p_j$ for $1 \leq j \leq r$*
4. *return $g_0, \ldots, g_n$*

Algorithm 2.9 *Modular conversion from $\mathcal{M}_{\mathbf{b}}$ to $\mathcal{M}$.*
Input: *Integers $g_0, \ldots, g_n$ of absolute value less than 2^λ, for some $n \geq 1$, and $b_1, \ldots, b_n \in \mathbb{Z}$ of absolute value less than 2^d.*
Output: *The coefficients of the polynomial $\sum_{0 \leq i \leq n} g_i \cdot (x+b_1) \cdots (x+b_i)$.*

1. $C \leftarrow (n+1)2^{nd+\lambda}$, $\quad r \leftarrow \lceil (\log_2 2C)/(\omega - 1) \rceil$
   choose odd single precision primes $p_1 < \cdots < p_r$
2. *for $1 \leq j \leq r$ compute $f_j \in \mathbb{Z}[x]$ of max-norm less than $p_j/2$ such that $f_j \equiv \sum_{0 \leq i \leq n} g_i \cdot (x+b_1) \cdots (x+b_i) \bmod p_j$*
3. *use the Chinese Remainder Algorithm to compute $f \in \mathbb{Z}[x]$ of max-norm less than $(\prod_{1 \leq j \leq r} p_j)/2$ such that $f \equiv f_j \bmod p_j$ for $1 \leq j \leq r$*
4. *return f*

Theorem 2.10 *Algorithms 2.8 and 2.9 work correctly as specified. Steps 2 and 3 take $O(n^3d^2 + n^2d\lambda + n\lambda^2)$ word operations with classical arithmetic and $O(\lambda\,\mathsf{M}(n)\log n + n\,\mathsf{M}(nd+\lambda)\log(nd+\lambda))$ or $O^\sim(n^2d + n\lambda)$ with fast arithmetic.*

Proof. Let $m = \prod_{1 \leq i \leq r} p_i > 2^{(\omega-1)r} \geq 2C$. In Algorithm 2.9, we have $f \equiv \sum_{0 \leq i \leq n} g_i \cdot (x+b_1) \cdots (x+b_i) \bmod m$, both sides of the congruence have max-norms less than $m/2$, by Lemma 2.1 (ii), and hence they are equal. The proof that Algorithm 2.8 works correctly is completely analogous.

In step 2 of both algorithms, we first reduce the g_i or the coefficients of f, respectively, and the b_i modulo all primes, taking $O(r(\lambda + d))$ word operations with classical arithmetic and $O(\mathsf{M}(r)\log r)$ with fast arithmetic. Then we perform the corresponding conversion modulo each prime, taking $O(rn^2)$ word operations with classical arithmetic and $O(r\,\mathsf{M}(n)\log n)$ with fast arithmetic, by Theorems 2.3, 2.4, and 2.5. Finally, the cost for the Chinese remaindering in step 3 is $O(r^2)$ per coefficient with classical arithmetic and $O(\mathsf{M}(r)\log r)$ with fast arithmetic, in total $O(nr^2)$ and $O(n\,\mathsf{M}(r)\log r)$, respectively. The estimates now follow from $r \in O(nd+\lambda)$. $\square$

We note that for particular bases $\mathcal{M}_{\mathbf{b}}$, the bound C in Algorithms 2.8 and 2.9, arising from the estimate in Lemma 2.1, may be much too large. For example, for the falling factorials, we may take $d = \log_2 n$, and then the bound C is $(n+1)n^n 2^{2n+\lambda}$ for Algorithm 2.8 and $(n+1)n^n 2^\lambda$ for Algorithm 2.9. In fact, one can show that $C = (n+1)!2^\lambda$ is sufficient in both cases.

Corollary 2.11 *Let $f \in \mathbb{Z}[x]$ of degree $n \geq 1$. If we ignore the cost for prime finding, then we have a modular algorithm for computing the falling (or rising) factorial coefficients of f from the usual coefficients. If $\|f\|_\infty < 2^\lambda$, then it takes $O(n^3 \log^2 n + n^2 \log n \cdot \lambda + n\lambda^2)$ word operations with classical arithmetic and $O(\lambda\,\mathsf{M}(n)\log n + n\,\mathsf{M}(n \log n + \lambda)\log(n+\lambda))$ or $O^\sim(n^2+n\lambda)$ with fast arithmetic. The same estimate is valid for computing the usual coefficients from the falling (or rising) factorial coefficients if the latter are absolutely less than 2^λ.*

The following table summarizes the cost estimates for the basis conversion algorithms that we have presented so far. The costs are to be read as O-estimates for polynomials of degree n with λ bit coefficients, and d is the bit size of the b_i. For the modular algorithms, we have neglected the cost for prime finding. We note that the estimates for the modular algorithms are not asymptotic estimates; see the remark at the end of Section 1.

algorithm	cost	$n = d = \lambda$
classical (Theorem 2.2)	$n^3d^2 + n^2d\lambda$	n^5
classical modular (Theorem 2.10)	$n^3d^2 + n^2d\lambda + n\lambda^2$	n^5
fast (Theorems 2.4 and 2.5)	$\mathsf{M}(n^2d + n\lambda)\log n$	$\mathsf{M}(n^3)\log n$
fast modular (Theorem 2.10)	$\lambda\,\mathsf{M}(n)\log n +$ $n\,\mathsf{M}(nd+\lambda)\log(nd+\lambda)$	$n\,\mathsf{M}(n^2)\log n$

3 Fast multiplication in the falling factorial basis

In this section, we give some fast algorithms for polynomials in the *falling factorial basis*

$$\mathcal{F} = (x^{\underline{i}})_{i\in\mathbb{N}} = \mathcal{M}_{(0,-1,-2,-3,\ldots)},$$

namely for multiplication and Taylor shift. In principle, both problems can be solved by first converting the input polynomial(s) to the monomial basis $\mathcal{M}$, applying the corresponding fast algorithm for the monomial basis, and finally converting back to the falling factorial basis. However, the following algorithms are more direct and faster by a logarithmic factor. We note that the algorithms can be easily generalized to the basis $\mathcal{M}_{\mathbf{b}}$, where $\mathbf{b} = (b_1, b_1 + h, b_1 + 2h, b_1 + 3h, \ldots)$ is an arbitrary arithmetic progression, so in particular to the rising factorial basis $\mathcal{M}_{(0,1,2,3,\ldots)}$. Our algorithms assume that the characteristic of the coefficient ring R is "big enough". We give cost estimates in arithmetic operations.

We start with the fast multiplication algorithm. The idea is to use the well-known evaluation-interpolation scheme: evaluate the input polynomials at a suitable set of points, multiply the results pointwise, and interpolate the product polynomial. The most profitable use of this scheme is in FFT multiplication algorithms, where the points are the powers of a primitive root of unity. In our case of the falling factorial basis representation, it is not surprising that the integers $0, 1, 2, 3, \ldots$ (in the more general case: the ring elements $-b_1, -b_1 - h, -b_1 - 2h, -b_1 - 3h, \ldots$) are a suitable set of evaluation points.

Algorithm 3.1 *Evaluation in the falling factorial basis.*
Input: *$f_0, \ldots, f_n \in R$, where $n \geq 1$ and R is a ring of characteristic zero or coprime to $n!$.*
Output: *The values $f(0), f(1), \ldots, f(n)$, where $f = \sum_{0 \leq i \leq n} f_i x^{\underline{i}}$.*

1. *for* $0 \leq i \leq n$ *do* $g_i \leftarrow 1/i!$
2. $\sum_{0 \leq i \leq 2n} h_i x^i \leftarrow (\sum_{0 \leq i \leq n} f_i x^i) \cdot (\sum_{0 \leq i \leq n} g_i x^i)$
3. *return* $h_0, h_1, 2h_2, \ldots, n! \cdot h_n$

Theorem 3.2 *Algorithm 3.1 works correctly as specified and takes $O(\mathsf{M}(n))$ additions and multiplications plus $O(n)$ divisions by integers in R.*

Proof. Let $0 \leq k \leq n$. Then

$$f(k) = \sum_{0 \leq i \leq n} f_i k^{\underline{i}} = \sum_{0 \leq i \leq k} f_i \frac{k!}{(k-i)!} = k! \sum_{0 \leq i \leq k} f_i g_{k-i} = k!\, h_k, \tag{3.1}$$

and the correctness follows. The cost for steps 1 and 3 is $O(n)$ divisions and multiplications, respectively, and step 2 takes $O(\mathsf{M}(n))$ additions and multiplications. □

Algorithm 3.3 *Interpolation in the falling factorial basis.*
Input: *$u_0, \ldots, u_n \in R$, where $n \in \mathbb{N}$ and R is a field of characteristic zero or coprime to $n!$.*
Output: *The falling factorial coefficients $f_0, \ldots, f_n \in R$ of the interpolating polynomial $f = \sum_{0 \leq i \leq n} f_i x^{\underline{i}}$ such that $u_i = f(i)$ for $0 \leq i \leq n$.*

1. *for* $0 \leq i \leq n$ *do* $h_i \leftarrow u_i/i!, \quad v_i \leftarrow (-1)^i/i!$
2. $\sum_{0 \leq i \leq 2n} w_i x^i \leftarrow (\sum_{0 \leq i \leq n} h_i x^i) \cdot (\sum_{0 \leq i \leq n} v_i x^i)$
3. *return* $w_0, w_1, \ldots, w_n$

Theorem 3.4 *Algorithm 3.3 works correctly as specified and takes $O(\mathsf{M}(n))$ additions and multiplications plus $O(n)$ divisions by integers in R.*

Proof. Let $g = \sum_{0\le i\le n} x^i/i!$ and $v = \sum_{0\le i\le n}(-1)^i x^i/i!$. For $1 \le k \le n$, the coefficient of x^k in gv is

$$\sum_{0\le i\le k} \frac{(-1)^{k-i}}{i!\,(k-i)!} = \frac{1}{k!}\sum_{0\le i\le k}(-1)^{k-i}\binom{k}{i} = \frac{(1+(-1))^k}{k!} = 0,$$

and hence $gv \equiv 1 \bmod x^{n+1}$. (This also follows from the fact that g and v are initial segments of the formal exponential series and its inverse, respectively.) If we now let $h = \sum_{0\le i\le n} h_i x^i$ and $w = \sum_{0\le i\le n} w_i x^i$, then (3.1) shows that

$$\sum_{0\le k\le n} \frac{w(k)}{k!}x^k \equiv gw \equiv gvh \equiv h = \sum_{0\le k\le n}\frac{u_k}{k!}x^k \bmod x^{n+1},$$

and the correctness follows from the uniqueness of the interpolating polynomial. The cost for step 1 is $O(n)$ multiplications and divisions, and step 2 takes $O(\mathsf{M}(n))$ additions and multiplications. □

Corollary 3.5 *Let $f_0,\ldots,f_n,g_0,\ldots,g_n \in R$, where $n\in\mathbb{N}$ and R is a ring of characteristic zero or coprime to $(2n)!$. Then we can compute $h_0,\ldots,h_{2n}\in R$ such that*

$$\sum_{0\le i\le 2n} h_i x^{\underline{i}} = \Big(\sum_{0\le i\le n} f_i x^{\underline{i}}\Big)\Big(\sum_{0\le i\le n} g_i x^{\underline{i}}\Big)$$

using $O(\mathsf{M}(n))$ additions and multiplications plus $O(n)$ divisions by integers in R.

Proof. We use Algorithm 3.1 to compute $f(i)$ and $g(i)$ for $0\le i\le 2n$ and then call Algorithm 3.3 to compute $h\in R[x]$ of degree at most $2n$ such that $h(i) = f(i)g(i)$ for $0\le i\le 2n$. By Theorems 3.2 and 3.4, the cost for the two evaluations and the interpolation is $O(\mathsf{M}(n))$ additions and multiplications plus $O(n)$ divisions, and this dominates the additional cost for the $2n+1$ pointwise multiplications to compute the values $h(i)$. □

Our last algorithm in this section is for Taylor shift in the falling factorial basis. It is an adaption of the convolution method (see the discussion preceding Fact 2.7 for references). More generally, this method works for any polynomial basis satisfying a binomial identity (in the language of umbral calculus, such a sequence is called an *associated sequence*, see Roman 1984).

Theorem 3.6 *Let $f_0,\ldots,f_n,b\in R$, where $n\in\mathbb{N}$ and R is a ring of characteristic zero or coprime to $n!$. Then we can compute $g_0,\ldots,g_n\in R$ such that*

$$\sum_{0\le k\le n} g_k x^{\underline{k}} = \sum_{0\le i\le n} f_i\cdot(x+b)^{\underline{i}} \tag{3.2}$$

using $O(\mathsf{M}(n))$ additions and multiplications plus $O(n)$ divisions by integers in R.

Proof. We apply Vandermonde's convolution

$$(x+b)^{\underline{i}} = i!\binom{x+b}{i} = i!\sum_{0\le k\le i}\binom{x}{k}\binom{b}{i-k} = i!\sum_{0\le k\le i}\binom{b}{i-k}\frac{x^{\underline{k}}}{k!}$$

to (3.2) and find

$$g_k = \sum_{k\le i\le n}\frac{i!\,f_i}{k!}\binom{b}{i-k}$$

for $0 \le k \le n$. If we now let $u = \sum_{0\le i\le n} i! f_i x^{n-i}$ and $v = \sum_{0\le j\le n}\binom{b}{j}x^j$, then $k!\,g_k$ is the coefficient of x^{n-k} in the product polynomial uv. Using the recursion formula

$$\binom{b}{j+1} = \frac{b-j}{j+1}\binom{b}{j}$$

for $j \in \mathbb{N}$, computing the coefficients of u and v takes $O(n)$ additions, multiplications, and divisions in R. The same estimate is valid for recovering the g_k from the coefficients of the product polynomial uv, and the multiplication itself takes $O(\mathsf{M}(n))$ additions and multiplications. □

4 Modular greatest factorial factorization

In this section, we adapt Yun's (1976) algorithm for squarefree factorization to its discrete analog, the *greatest factorial factorization*, which has been introduced by Paule (1995). We also discuss a modular algorithm for computing this factorization. It is quite analogous to the modular algorithm in Gerhard (1999) for computing the squarefree factorization. In the following, E denotes the *shift operator*, which acts by $Ef = f(x+1)$ on a polynomial f in the indeterminate x.

Definition 4.1 *Let F be a field and $f \in F[x]$ nonzero monic of degree n. A* greatest factorial factorization *(or* gff*) of f is a sequence of monic polynomials $f_1, \ldots, f_n \in F[x]$ with the following properties.*

(i) $f = \prod_{1\le i\le n} f_i^{\underline{i}}$,

(ii) $\gcd(f_i^{\underline{i}}, Ef_j) = \gcd(f_i^{\underline{i}}, E^{-j}f_j) = 1$ *for* $1 \le i \le j \le n$.

The intuition is that f_i collects all maximal falling factorials of length i. We note that (ii) is not equivalent to $\gcd(f_i^{\underline{i}}, Ef_j) = 1$ for all i, j. For example, the sequence $x, x+1, 1$ is a gff of $x^3 + x^2$ but does not satisfy the latter condition.

Paule (1995) has shown that when F has characteristic zero, then every nonzero polynomial has a unique gff (see also Lemma 23.10 in von zur Gathen & Gerhard 1999). This is false in positive characteristic: for example, the polynomial $x^p - x = x^{\underline{p}} \in \mathbb{F}_p[x]$ has no gff. The claim is still true if we require in addition that the degree of the polynomial be less than the characteristic. An alternative would be to modify the definition, namely to require (ii) only for $i < j$ and to add the property $\gcd(f_i, Ef_i) = 1$ for $1 \le i \le n$, in analogy to the squarefree decomposition. Then the gff is still not unique: for example, $(x+1)^{\underline{p}}$ is also a gff of $x^p - x$. However, since we will not need the concept of gff when the degree exceeds the characteristic, we will stick to the original definition. The gff of the constant polynomial 1 is the empty sequence.

The following technical lemma will be used later.

Lemma 4.2 *Let F be a field, $f \in F[x]$ a nonzero monic polynomial of degree n, and $f_1, \ldots, f_n \in F[x]$ the gff of f. Moreover, let $h \in F[x]$ be an irreducible divisor of f.*

(i) Let $i, j, k, l \in \mathbb{N}$ be such that $0 \le k < i \le j$ and $0 \le l < j \le n$. If $h \mid E^{-k} f_i$ and $h \mid E^{-l} f_j$, then $0 \le l - k \le j - i$.

(ii) Let $e, i \le n$ be such that $h^e \mid f_i^{\underline{i}}$. Then there is a unique $k \in \{0, \ldots, i-1\}$ such that $h^e \mid E^{-k} f_i$.

(iii) Assume that $e \ge 1$ is such that $h^e \mid (f/f_1) = f_2^{\underline{2}} \cdots f_n^{\underline{n}}$. Then $Eh^e \mid (f/f_1)$ or $E^{-1}h^e \mid (f/f_1)$.

Proof. (i) If $0 > l - k$, then $E^{l+1}h \mid E^{l-k+1} f_i \mid f_i^{\underline{i}}$ and $E^{l+1}h \mid Ef_j$. Thus $E^{l+1}h \mid \gcd(f_i^{\underline{i}}, Ef_j)$, contradicting Definition 4.1 (ii). Similarly, if $l - k > j - i$, then $E^{l-j}h \mid E^{l-k-j} f_i \mid f_i^{\underline{i}}$ and $E^{l-j}h \mid E^{-j} f_j$. Thus $E^{l-j}h \mid \gcd(f_i^{\underline{i}}, E^{-j} f_j)$, again contradicting Definition 4.1 (ii).

(ii) This follows from (i) with $i = j$.

(iii) Let $e_j \in \mathbb{N}$ be the multiplicity of h in $f_j^{\underline{j}}$, for $2 \le j \le n$. Then $e \le e_2 + \cdots + e_n$. Moreover, let $i \ge 2$ be minimal with $e_i > 0$. By (ii), there is a unique $k \in \{0, \ldots, i-1\}$ with $h^{\underline{e_i}} \mid E^{-k} f_i$. Assume first that $k = 0$. Then $i \ge 2$ implies that $E^{-1}h^{e_i} \mid E^{-1} f_i \mid f_i^{\underline{i}}$. If $e \le e_i$, then we are done. Otherwise, let $j > i$ be such that $e_j \ge 1$. Again by (ii), there is a unique $l \in \{0, \ldots, j-1\}$ such that $h^{e_j} \mid E^{-l} f_j$. Then (i) shows that $l \le j - i < j - 1$, and hence $E^{-1}h^{e_j} \mid E^{-l-1} f_j \mid f_j^{\underline{j}}$. Since this holds for all $j > i$, we conclude that $E^{-1}h^{e_j} \mid f_j^{\underline{j}}$ for $2 \le j \le n$, and the claim follows. Similar arguments show that $Eh^{e_j} \mid f_j^{\underline{j}}$ for $2 \le j \le n$ if $k > 0$. □

The following lemma, which we quote without proof, is due to Paule (1995); see also Theorem 23.12 in von zur Gathen & Gerhard (1999). It is the analogy of a well-known property of the squarefree factorization.

Lemma 4.3 *(Fundamental Lemma) Let F be a field of characteristic $p = 0$ or $p > n$ and $f \in F[x]$ nonzero monic of degree n. If $f_1, \ldots, f_n$ is the gff of f and $g = \gcd(f, Ef)$, then*

$$g = f_2 f_3^{\underline{2}} \cdots f_n^{\underline{n-1}}, \quad \frac{f}{g} = f_1 (E^{-1} f_2) \cdots (E^{-n+1} f_n), \quad \frac{Ef}{g} = E(f_1 \cdots f_n).$$

Moreover, $f_2, \ldots, f_{1+\deg g}$ is the gff of g.

We now present two algorithms for computing the gff. The first one is due to Paule (1995). The second one appears to be new and is analogous to Yun's (1976) algorithm for computing the squarefree decomposition.

Algorithm 4.4 *Gff computation.*
Input: *A nonzero monic polynomial $f \in F[x]$ of degree n, where F is a field of characteristic $p > n$*
Output: *The gff of f.*

0. *if $f = 1$ then return the empty sequence*
1. $g \leftarrow \gcd(f, Ef)$, $\quad m \leftarrow \deg g$
2. *call the algorithm recursively to compute the gff $g_1, \ldots, g_m$ of g*
3. $h \leftarrow \dfrac{f}{g \cdot (E^{-1}g_1)(E^{-2}g_2)\cdots(E^{-m}g_m)}$
4. *return $h, g_1, \ldots, g_m, 1, \ldots, 1$*

Theorem 4.5 *Algorithm 4.4 works correctly and takes $O(n^3)$ field operations with classical arithmetic and $O(n\,\mathsf{M}(n)\log n)$ with fast arithmetic.*

Proof. Let $f_1, f_2, \ldots, f_n$ be the gff of f. Then $g = f_2 f_3^{\underline{2}} \cdots f_n^{\underline{n-1}}$, by the Fundamental Lemma 4.3, and by induction, we have $f_i = g_{i-1}$ for $2 \le i \le m+1$ and $f_i = 1$ for $m+2 \le i \le n$. Finally, $f/g = f_1 \cdot (E^{-1}f_2)\cdots(E^{-n+1}f_n)$, again by the Fundamental Lemma, so that $h = f_1$, and the correctness is proved.

Let $T(n)$ denote the cost of the algorithm. Including the cost for the Taylor shift to compute $Ef = f(x+1)$ (Fact 2.7), step 1 takes $O(n^2)$ field operations with classical arithmetic and $O(\mathsf{M}(n)\log n)$ with fast arithmetic. The cost for step 2 is $T(m)$. Step 3 takes $O(n^2)$ operations with classical arithmetic, by Facts 1.1 (iv) and 2.7. Again by Fact 2.7, we can compute $E^{-i}g_i$ with $O(\mathsf{M}(\deg g_i))$ operations, for each i, together $O(\mathsf{M}(m))$, by the subadditivity of M. Computing the product in the denominator takes $O(\mathsf{M}(m)\log m)$, and the division with remainder takes $O(\mathsf{M}(n))$, by Fact 1.1 (iv) and (ii), respectively. Thus the cost for step 3 with fast arithmetic is $O(\mathsf{M}(n) + \mathsf{M}(m)\log m)$. The claims now follow from using $m \le n-1$ and unravelling the recursion for T. $\square$

The upper bounds of the theorem are achieved for $f = x^{\underline{n}}$.

Algorithm 4.6 *Gff computation.*
Input: *A nonzero monic polynomial $f \in F[x]$ of degree n, where F is a field of characteristic $p > n$*
Output: *The gff of f.*

0. *if $f = 1$ then return the empty sequence*
1. $g \leftarrow \gcd(f, Ef)$, $\quad u_1 \leftarrow \dfrac{f}{E^{-1}g}$, $\quad v_1 \leftarrow \dfrac{f}{g}$
2. *for $1 \le i \le n$ do*

3. $\qquad h_i \leftarrow \gcd(u_i, v_i), \quad u_{i+1} \leftarrow \frac{v_i}{h_i}, \quad b_{i+1} \leftarrow E\frac{u_i}{h_i}$

4. return $h_1, \ldots, h_n$

Clearly the algorithm may be aborted as soon as $u_i = 1$ $(= v_i)$.

Theorem 4.7 *Algorithm 4.6 works correctly and takes $O(n^2)$ field operations with classical arithmetic and $O(\mathsf{M}(n)\log n)$ with fast arithmetic.*

Proof. Let $1 \leq i < j, k \leq n$ and $f_1, \ldots, f_n \in F[x]$ be the monic gff of f. We have $E^{i-j+1}f_j \mid f_j^{\underline{j}}$, and the properties of the gff imply that $\gcd(f_k, E^{i-j}f_j) = E^{-1}\gcd(Ef_k, E^{i-j+1}f_j) = 1$ if $j \leq k$. Similarly, we have $E^{-i}f_k \mid f_k^{\underline{k}}$ and $\gcd(f_k, E^{i-j}f_j) = E^i \gcd(E^{-i}f_k, E^{-j}f_j) = 1$ if $j < k$. From these observations, the invariants

$$u_i = \prod_{i\leq k\leq n} f_k, \quad v_i = \prod_{i\leq j\leq n} E^{i-j}f_j, \quad h_i = f_i$$

for $1 \leq i \leq n$ follow easily by induction, and the correctness is proved.

The cost for step 1 is $O(n^2)$ field operations with classical arithmetic and $O(\mathsf{M}(n)\log n)$ with fast arithmetic. Similarly, one execution of step 3 takes $O((\deg u_i)^2)$ and $O(\mathsf{M}(\deg u_i)\log(\deg u_i))$ field operations with classical and fast arithmetic, respectively. If $n_j = \deg f_j$ for all j, then $\deg u_i = \sum_{i\leq j\leq n} n_j$ for all i and $\sum_{1\leq i\leq n} \deg u_i = n$, and the cost estimates follow by the subadditivity of M. $\square$

In positive characteristic p, Algorithms 4.4 and 4.6 may get into an infinite loop or return the wrong result if $p \leq n$. For example, when $f = x^p - x = x^{\underline{p}}$, then $\gcd(f, Ef) = f$, and Algorithm 4.4 does not terminate, while Algorithm 4.6 incorrectly returns $h_1 = f$ and $h_2 = \cdots = h_n = 1$.

Let Z be a UFD with field of fractions F. A nonzero polynomial $f \in Z[x]$ is *primitive* if the greatest common divisor of its coefficients is 1. We say that a sequence $f_1, \ldots, f_n \in Z[x]$ of primitive polynomials is a *primitive gff* if $f = \prod_i f_i^{\underline{i}}$ and $f_1/\mathrm{lc}(f_1), \ldots, f_n/\mathrm{lc}(f_n)$ is the (monic) gff of $f/\mathrm{lc}(f)$ in $F[x]$. The primitive gff is unique up to multiplication by units in $Z^\times$. For $Z = \mathbb{Z}$, we say that a nonzero primitive polynomial is *normalized* if its leading coefficient is positive. In this case, we can make the primitive gff unique by requiring $f, f_1, \ldots, f_n$ to be normalized, and then call it the *normalized gff*. The *primitive squarefree decomposition* and the *normalized squarefree decomposition* are defined accordingly.

We now investigate when the modular image of a gff is again a gff. Before that, we state the corresponding result for the squarefree factorization, since we will use it for the modular gff algorithm. Its proof can be found in Gerhard (1999, Lemma 7).

Lemma 4.8 *Let Z be a UFD, $f \in Z[x]$ nonzero primitive of degree n, and $f_1, \ldots, f_n \in Z[x]$ a primitive squarefree decomposition of f. Moreover, let $I \subseteq Z$ be a maximal ideal not containing $\mathrm{lc}(f)$, denote reduction modulo I by a bar,* *and assume that* $\overline{Z} = Z/I$ *has characteristic* $p = 0$ *or* $p > n$*. Finally, assume that* $g_1, \ldots, g_n \in \overline{Z}[x]$ *is the monic squarefree decomposition of* $\overline{f/\mathrm{lc}(f)}$*. Then*

$$\deg \gcd(f, f') \le \deg \gcd(\overline{f}, \overline{f}') \text{ and } \deg(g_1 \cdots g_n) \le \deg(f_1 \cdots f_n), \qquad (4.1)$$

and for each of the two inequalities, equality holds if and only if $\overline{\mathrm{lc}(f_i)} g_i = \overline{f_i}$ *for* $1 \le i \le n$*.*

Interestingly, the straightforward analog of this lemma for the gff, where f' is replaced by Ef, does not hold. For example, let $I = 7\mathbb{Z}$ and $f = (x+8)(x+7)x \cdot (x-1)(x-2) \in \mathbb{Z}[x]$. The primitive and monic gff of f is 1, $x+8$, $x, 1, 1$ and $\gcd(f, Ef) = (x+8)x(x-1)$. Now $\gcd(\overline{f}, E\overline{f}) = (x+1)x(x-1) \bmod 7\mathbb{Z} = \overline{\gcd(f, Ef))}$, but the monic gff of $\overline{f}$ is x, 1, 1, $x+1$, 1, and this is not the modular image of the gff of f. The problem here is that the two factors $x+7$ and x are coprime in $\mathbb{Z}[x]$, but not modulo I. Under the additional assumption that any two distinct irreducible factors of f remain coprime modulo I, we can prove the following analog of Lemma 4.8, which says when the modular image of a gff is again a gff.

Lemma 4.9 *Let* Z *be a UFD,* $f \in Z[x]$ *a nonzero primitive polynomial of degree* n*, and* $f_1, \ldots, f_n \in Z[x]$ *be a primitive gff of* f*. Moreover, let* $I \subseteq Z$ *be a maximal ideal not containing* $\mathrm{lc}(f)$*, denote the residue class map modulo* I *by a bar, and assume that* $\overline{Z} = Z/I$ *has characteristic* $p = 0$ *or* $p > n$*. Finally, assume that* $g_1, \ldots, g_n \in \overline{Z}[x]$ *is the monic gff of* $\overline{f/\mathrm{lc}(f)}$*. Then*

$$\deg \gcd(f, Ef) \le \deg \gcd(\overline{f}, E\overline{f}) \text{ and } \deg(g_1 \cdots g_n) \le \deg(f_1 \cdots f_n). \qquad (4.2)$$

If in addition

$$\deg \gcd(f, f') = \deg \gcd(\overline{f}, \overline{f'}), \qquad (4.3)$$

then for each of the two inequalities, equality holds if and only if $\overline{\mathrm{lc}(f_i)} g_i = \overline{f_i}$ *for* $1 \le i \le n$*.*

We note that the condition (4.3) alone is not is not sufficient to ensure that the modular image of the gff is again a gff, as the example $f = x^2(x+8) \in \mathbb{Z}[x]$ and $I = 7\mathbb{Z}$ shows. On the other hand, the two conditions (4.2) and (4.3) are not necessary, as shown by the example $f = x(x+7) \in \mathbb{Z}[x]$ and $I = 7\mathbb{Z}$.

Proof. Let F be the field of fractions of Z. We first note that

$$\gcd(f, Ef) \sim f_2 f_3^2 \cdots f_n^{n-1}, \quad E(f_1 \cdots f_n) \sim \frac{Ef}{\gcd(f, Ef)} \qquad (4.4)$$

by the Fundamental Lemma 4.3, where $\sim$ denotes equality up to associates in $F[x]$, and similarly

$$\gcd(\overline{f}, E\overline{f}) \sim g_2 g_3^2 \cdots g_n^{n-1}, \quad E(g_1 \cdots g_n) \sim \frac{E\overline{f}}{\gcd(\overline{f}, E\overline{f})}. \qquad (4.5)$$

Now $\overline{\gcd(f, Ef)}$ is a common divisor of $\overline{f}$ and $E\overline{f}$, which implies that

$$\overline{\gcd(f, Ef)} \mid \gcd(\overline{f}, E\overline{f}) \text{ and } g_1 \cdots g_n \mid \overline{f_1 \cdots f_n}, \qquad (4.6)$$

and (4.5) shows that either both inequalities in (4.2) are strict or both are equalities. This proves the first claim.

The "if" direction of the second claim is true without the additional assumptions. We prove the "only if" part by induction on n. There is nothing to prove if $n = 0$. If $n > 0$, then $f_2, f_3, \ldots$ is a gff of $\gcd(f, Ef)$ and $g_2, g_3, \ldots$ is a gff of $\gcd(\overline{f}, E\overline{f})$, by the Fundamental Lemma 4.3. If equality holds in (4.2), then (4.6) yields

$$g_2 g_3^{\underline{2}} \cdots g_n^{\underline{n-1}} \sim \gcd(\overline{f}, E\overline{f}) \sim \overline{\gcd(f, Ef)} \sim \overline{f_2 f_3^{\underline{2}} \cdots f_n^{\underline{n-1}}}$$

and $g_1 \cdots g_n \sim \overline{f_1 \cdots f_n}$. Applying (4.6) to $\gcd(f, Ef)$ instead of f, we find that $g_2 \cdots g_n$ divides $\overline{f_2 \cdots f_n}$. We show below that also $g_1 \mid \overline{f_1}$. Thus $g_1 \sim \overline{f_1}$ and $g_2 \cdots g_n \sim \overline{f_2 \cdots f_n}$, and the induction hypothesis, applied to $\gcd(f, Ef)$, yields $g_i \sim \overline{f_i}$ for $2 \le i \le n$.

To prove that $g_1 \mid \overline{f_1}$, let $t \in \overline{Z}[x]$ be an irreducible factor of g_1 of multiplicity $i > 0$, and let $j \ge 0$ be its multiplicity in $\overline{f}/g_1$. Moreover, let $h \in Z[x]$ be the unique primitive irreducible factor of f such that $t \mid \overline{h}$, and let $e \ge 1$ be its multiplicity in f. Equation (4.3) and Lemma 4.8 imply that the distinct irreducible factors of f in $Z[x]$ remain squarefree and pairwise coprime modulo I. Thus $p^2 \nmid \overline{h}$ and $p \nmid \overline{f}/\overline{h}^e$, and hence $e = i + j$. Now assume that $h^i \nmid f_1$. Then $h^{j+1} \mid (f/f_1) = f_2^{\underline{2}} \cdots f_n^{\underline{n}}$, and Lemma 4.2 implies that $Eh^{j+1} \mid (f/f_1)$ or $E^{-1}h^{j+1} \mid (f/f_1)$. For simplicity, we assume that $Eh^{j+1} \mid (f/f_1)$; the arguments for the other case are analogous. Then $Ep^{j+1} \mid \overline{f}$, and hence Ep divides g_1 and $\gcd(g_1^{\underline{1}}, Eg_1)$. This contradiction to Definition 4.1 (ii) for $g_1, \ldots, g_n$ proves that our assumption is wrong, $h^i \mid f_1$, and $p^i \mid \overline{f_1}$. Since this holds for any irreducible factor of g_1, we conclude that $g_1 \mid \overline{f_1}$. $\square$

Lemma 4.9 is the main tool for the following modular gff algorithm. We assume that ω is the word size of our processor. For a nonzero polynomial $f \in \mathbb{Z}[x]$, $\mathrm{normal}(f)$ denotes the unique normalized polynomial that divides f and has the same degree.

Algorithm 4.10 *Modular gff computation.*
Input: *A normalized polynomial $f \in \mathbb{Z}[x]$ of degree $1 \le n \le 2^{\omega-1}$ and max-norm $\|f\|_\infty < 2^\lambda$.*
Output: *The normalized gff of f, or otherwise "FAIL".*

1. $b \leftarrow \mathrm{lc}(f)$, $\quad B \leftarrow \lfloor (n+1)^{1/2} 2^{n+\lambda} \rfloor$, $\quad s \leftarrow \lceil \log_2(2 \cdot 3^{n/3} bB)/(\omega - 1) \rceil$
2. *choose a set S_0 of $2s$ single precision primes*
   $S_1 \leftarrow \{p \in S_0 : p \nmid b\}$
3. *for all $p \in S_1$ do*
4. *call Algorithm 4.6 to compute the monic gff*

$$f \equiv b \prod_{1 \le i \le n} g_{p,i}^{\underline{i}} \bmod p$$

of f/b modulo p, with monic polynomials $g_{p,1}, \ldots, g_{p,n} \in \mathbb{Z}[x]$ of max-norm less than $p/2$

$e_p \leftarrow \deg\gcd(f \bmod p, f' \bmod p)$

5. { *remove the bad primes from* S_1 }
   $d \leftarrow \max\{\deg g_{p,1} + \cdots + \deg g_{p,n} : p \in S_1\}$
   $e \leftarrow \min\{e_p : p \in S_1\}$
   $S_2 \leftarrow \{p \in S_1 : \deg g_{p,1} + \cdots + \deg g_{p,n} = d$ *and* $e_p = e\}$
   if $\#S_2 \geq s$ *then remove* $\#S_2 - s$ *primes from* S_2 *else return "FAIL"*

6. *for* $0 \leq j < i \leq n$ *do*

7. *use the Chinese Remainder Algorithm to compute* $f^*_{ij} \in \mathbb{Z}[x]$ *of max-norm less than* $(\prod_{p \in S_2} p)/2$ *such that* $f^*_{ij} \equiv bg_{p,i}(x-j) \bmod p$ *for all* $p \in S_2$

8. *if* $\prod_{0 \leq j < i \leq n} \|\mathrm{normal}(f^*_{ij})\|_1 > B$ *then return "FAIL"*

9. *if* $\prod_{1 \leq i \leq n} \mathrm{lc}(\mathrm{normal}(f^*_{i0}))^i = \mathrm{lc}(f)$
   then return $\mathrm{normal}(f^*_{10}), \ldots, \mathrm{normal}(f^*_{n0})$ *else return "FAIL"*

We say that a prime p is *bad* if $\deg\gcd(f, f') < \deg\gcd(f \bmod p, f' \bmod p)$ or $\deg\gcd(f, Ef) < \deg\gcd(f \bmod p, Ef \bmod p)$ or $p \mid \mathrm{lc}(f)$; otherwise p is *good*.

Theorem 4.11 *Algorithm 4.10 succeeds if at least s of the initial primes in S_0 are good. If the algorithm does not return "FAIL", then it correctly returns the normalized gff of f.*

Proof. Let $f_1, \ldots, f_n \in \mathbb{Z}[x]$ be the normalized gff of f, let $h = \gcd(f, f')$, and let $p \in S_1$. Lemmas 4.9 and 4.8 imply that $\deg(g_{p,1} \cdots g_{p,n}) \leq \deg(f_1 \cdots f_n)$ and $e_p \geq \deg h$, respectively. By definition, we have equality in both cases if and only if p is a good prime. Thus, if at least one prime in S_0 is good, then we have $d = \deg(f_1 \cdots f_n)$ and $e = \deg h$ in step 5, and all primes in S_2 are good.

If at least s of the primes in S_0 are good, then S_2 contains s good primes after step 5. Let $m = \prod_{p \in S_2} p$ and $0 \leq j < i \leq n$. Then Lemma 4.9 implies that $bf_i(x-j)/\mathrm{lc}(f_i) \equiv f^*_{ij} \bmod m$. Since $f_i(x-j)$ divides f, Mignotte's bound (Fact 1.2) implies that $\|f_i(x-j)\|_\infty \leq B$ and $\|bf_i(x-j)/\mathrm{lc}(f_i)\|_\infty < m/2$. Thus the coefficients of $bf_i(x-j)/\mathrm{lc}(f_i)$ and f^*_{ij} are both absolutely less than $m/2$, and hence they are equal. Since f_i is normalized, so is $f_i(x-j)$, and we have $f_i(x-j) = \mathrm{normal}(f^*_{ij})$. Again by Mignotte's bound, the condition in step 8 is false, the condition in step 9 is true, and the algorithm does not return "FAIL". This proves the first assertion.

To prove the second claim, suppose that the condition in step 8 is true. Then

$$\begin{aligned} \left\| \prod_{0 \leq j < i \leq n} \mathrm{normal}(f^*_{ij}) \right\|_\infty &\leq \left\| \prod_{0 \leq j < i \leq n} \mathrm{normal}(f^*_{ij}) \right\|_1 \leq \prod_{0 \leq j < i \leq n} \|\mathrm{normal}(f^*_{ij})\|_1 \\ &\leq B < \frac{m}{2}. \end{aligned}$$

Let $0 \le j < i \le n$. Then

$$\begin{aligned} \|f_{ij}^*\|_\infty &\le \|f_{ij}^*\|_1 \le b\,\|\mathrm{normal}(f_{ij}^*)\|_1 \le bB < \frac{m}{2}, \\ \|f_{i0}^*(x-j)\|_\infty &\le (j+1)^{\deg f_{i0}^*}\|f_{i0}^*\|_1 \le i^{n/i} b\,\|\mathrm{normal}(f_{i0}^*)\|_1 \le i^{n/i} bB < \frac{m}{2}, \end{aligned}$$

where we have used Lemma 2.1 (ii) and the fact that $i^{1/i} \le 3^{1/3}$ for all $i \in \mathbb{N}_{\ge 1}$. Thus the congruence $f_{ij}^* \equiv f_{i0}^*(x-j) \bmod m$ is in fact an equality and $\mathrm{normal}(f_{ij}^*) = E^{-j}\mathrm{normal}(f_{i0}^*)$. The polynomials $\prod_{1\le i\le n} \mathrm{normal}(f_{i0}^*)^{\underline{i}}$ and f agree modulo m up to a multiplicative constant. If the condition in step 9 is true, then

$$\prod_{1\le i\le n} \mathrm{normal}(f_{i0}^*)^{\underline{i}} = \prod_{0\le j<i\le n} \mathrm{normal}(f_{ij}^*) \equiv f \bmod m,$$

and since both polynomials have max-norms less than $m/2$, they are equal. Now the $\mathrm{normal}(f_{i0}^*)$ are a gff of f, since the required gcd conditions hold modulo each prime in S_2. Since these polynomials are normalized, the uniqueness of the normalized gff implies that $f_i = \mathrm{normal}(f_{i0}^*)$ for all i, and the algorithm returns the correct result. □

The following facts are from Sections 6.10 and 6.11 in von zur Gathen & Gerhard (1999).

Fact 4.12 *If F is a field, $f, g \in F[x]$ are nonzero polynomials, and $\delta \le m = \deg g \le n = \deg f$, then $\sigma_\delta(f,g)$, the δth* subresultant *of f and g, is the determinant of a certain quadratic $(n+m-2\delta)$-dimensional submatrix of the Sylvester matrix of f and g. It has the property that $\sigma_\delta(f,g) = 0$ if and only if δ occurs in the degree sequence of the Euclidean algorithm of f and g. In particular, for $\delta \le \deg\gcd(f,g)$ we have $\sigma_\delta(f,g) \ne 0$ if and only if $\delta = \deg\gcd(f,g)$. If f, g are in $\mathbb{Z}[x]$ and $p \in \mathbb{N}$ is a prime not dividing $\mathrm{lc}(fg)$, then $\sigma_\delta(f \bmod p, g \bmod p) = \sigma_\delta(f,g) \bmod p$. Thus for $\delta = \deg\gcd(f,g)$, we have $\deg\gcd(f \bmod p, g \bmod p) = \delta$ if and only if $p \nmid \sigma_\delta(f,g)$.*

Theorem 4.13 *Let $\delta = \deg\gcd(f, Ef)$, $\delta' = \deg\gcd(f, f')$, let $r = \sigma_\delta(f, Ef)$, and let $r' = \sigma_{\delta'}(f, f')$. Then $0 < |rr'| \le D = n^{3n/2}(n+1)^n 2^{n(4\lambda+n+1)}$. If the number of single precision primes exceeds $2\lfloor(\log_2 D)/(\omega-1)\rfloor \in \Theta(n^2 + n\lambda)$ and the set S_0 in step 2 is chosen uniformly at random from among all subsets of cardinality $2s$ of the single precision primes, then Algorithm 4.10 returns "FAIL" with probability at most $1/2$.*

Proof. The claim that r and r' are nonzero is clear from Fact 4.12. We have $\|f\|_2 \le (n+1)^{1/2}\|f\|_\infty$,

$$\begin{aligned} \|Ef\|_2 &\le \|f(x+1)\|_1 \le \|f\|_\infty \sum_{0\le i\le n} \|(x+1)^i\|_1 < 2^{n+1}\|f\|_\infty, \\ \|f'\|_2 &\le n^{1/2}\|f'\|_\infty \le n^{3/2}\|f\|_\infty, \end{aligned}$$

and Theorem 6.50 in von zur Gathen & Gerhard (1999) implies that

$$\begin{aligned} |r| &\leq \|f\|_2^{n-\delta}\|Ef\|_2^{n-\delta} \leq (n+1)^{n/2}2^{n^2+n}\|f\|_\infty^{2n}, \\ |r'| &\leq \|f\|_2^{n-\delta'-1}\|f'\|_2^{n-\delta'} \leq (n+1)^{n/2}n^{3n/2}\|f\|_\infty^{2n}. \end{aligned}$$

We conclude that $|rr'| < D$. Now $2bB \leq D$, so that under the assumptions of the theorem, we can indeed choose $2s$ primes in step 2. The leading coefficient of f divides r (see, for example, Exercise 6.41 in von zur Gathen & Gerhard 1999), and hence a prime is bad if and only if it divides rr', by Fact 4.12. By assumption, at least half of the single precision primes do not divide rr'. Therefore the probability that at least half of the primes of a randomly chosen set of $2s$ single precision primes are good is at least $1/2$ (see von zur Gathen & Gerhard 1999, Exercise 6.31), and the claim follows from Theorem 4.11. □

Theorem 4.14 *If we neglect the cost for choosing primes in step 2, then Algorithm 4.10 takes $O(n^3 + n\lambda^2)$ word operations when using classical arithmetic, and $O(n\,\mathsf{M}(n+\lambda)\log(n+\lambda)+\lambda\,\mathsf{M}(n)\log n)$ or $O^\sim(n^2+n\lambda)$ word operations when using fast arithmetic, where the $O^\sim$-notation suppresses logarithmic factors.*

Proof. We first reduce $\mathrm{lc}(f)$ and all other coefficients of f modulo all primes in S_0 and S_1, respectively. This takes $O(n\lambda s)$ and $O(n\,\mathsf{M}(s)\log s)$ word operations with classical and fast arithmetic, respectively. The cost for step 4 is $O(n^2)$ and $O(\mathsf{M}(n)\log n)$ word operations for each of the $O(s)$ primes with classical and fast arithmetic, respectively, by Theorem 4.7 and Fact 1.1 (iii). Computing a coefficient of some f^*_{ij} in step 7 takes $O(s^2)$ word operations with classical arithmetic and $O(\mathsf{M}(s)\log s)$ with fast arithmetic. There are n non-leading coefficients in total, and hence the cost for steps 6 and 7 is $O(ns^2)$ and $O(n\,\mathsf{M}(s)\log s)$, respectively. The cost for computing the primitive parts of all f^*_{ij} in step 8 is $O(n)$ gcd's and divisions of integers of word length at most s, taking $O(ns^2)$ word operations with classical arithmetic and $O(n\,\mathsf{M}(s)\log s)$ with fast arithmetic. Computing the products in steps 8 and 9 takes $O(ns+s^2)$ word operations with classical arithmetic and $O(\mathsf{M}(s)\log n)$ with fast arithmetic, by Fact 1.1 (iv). The claims now follow from $s \in O(n+\lambda)$ and using $\mathsf{M}(n) = n\log n\,\mathrm{loglog}\,n$ by adding costs. □

Remark 4.15 (i) When using fast arithmetic, then Algorithm 4.10 is—up to logarithmic factors—asymptotically optimal in the diagonal case where $n \approx \lambda$, since its running time is essentially linear in the input size, which is about n^2 words.

(ii) The number $2s$ of initial primes in step 2 of Algorithm 4.10 is often much too large in practice, for two reasons. On the one hand, the estimate for the failure probability in Theorem 4.13 is too pessimistic when there are considerably more single precision primes than required by the theorem. On the other hand, the coefficients of the f_i are often significantly smaller than guaranteed by Mignotte's bound. One solution to overcome this is to work in an adaptive fashion: start with a reasonable number of primes (say $\lceil\lambda/(\omega-1)\rceil$ or even fewer), check whether $\prod_{0\leq j<i\leq n}\mathrm{pp}(f^*_{ij}) = f$,

and add more new primes in case of failure. In this way, the algorithm will never use more primes than needed, at the expense of an additional check after each new prime. Heuristically, most of the checks can even be avoided by first checking whether the equation holds for $x = 0$. This strategy has been successfully implemented by Shoup (1995) in his software package NTL (`http://www.shoup.net/ntl/`) for computing gcd's of integer polynomials.

(iii) We did not say anything about how to find random single precision primes in step 1 of Algorithms 2.8 and 2.9 and in step 2 of Algorithm 4.10 and what this costs. There are several possibilities, such as using probabilistic primality tests or sieving methods. In practice, a precomputed list of single precision primes will be sufficient for most purposes; see Remark 12 in Gerhard (1999) for further discussions.

References

S. A. Abramov, M. Bronstein, and M. Petkovšek, On polynomial solutions of linear operator equations. In *Proceedings of the 1995 International Symposium on Symbolic and Algebraic Computation ISSAC '95*, Montreal, Canada, ed. A. H. M. Levelt. ACM Press, 1995, 290–296.

A. V. Aho, K. Steiglitz, and J. D. Ullman, Evaluating polynomials at fixed sets of points. *SIAM Journal on Computing* **4** (1975), 533–539.

A. Borodin and R. Moenck, Fast modular transforms. *Journal of Computer and System Sciences* **8**(3) (1974), 366–386.

A. Borodin and I. Munro, *The Computational Complexity of Algebraic and Numeric Problems*. Theory of computation series **1**. American Elsevier Publishing Company, New York, 1975.

David G. Cantor and Erich Kaltofen, On fast multiplication of polynomials over arbitrary algebras. *Acta Informatica* **28** (1991), 693–701.

Joachim von zur Gathen and Jürgen Gerhard, Fast algorithms for Taylor shifts and certain difference equations. In *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation ISSAC '97*, Maui HI, ed. Wolfgang W. Küchlin. ACM Press, 1997, 40–47.

Joachim von zur Gathen and Jürgen Gerhard, *Modern Computer Algebra*. Cambridge University Press, Cambridge, UK, 1999.

Jürgen Gerhard, Fast modular algorithms for squarefree factorization and Hermite integration. Submitted. Technical report tr-ri-99-206, University of Paderborn, Germany, 1999, 22 pages,
`http://math-www.uni-paderborn.de/~jngerhar/hermite.ps.gz`, 1999.

R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics*. Addison-Wesley, Reading MA, 2nd edition, 1994. First edition 1989.

A. Karatsuba and Yu. Ofman, Умножение многозначных чисел на автоматах. Доклады Академий Наук СССР **145** (1962), 293–294. A. Karatsuba and

Yu. Ofman, Multiplication of multidigit numbers on automata, Soviet Physics–Doklady **7** (1963), 595–596.

Maurice Mignotte, *Mathématiques pour le calcul formel.* Presses Universitaires de France, Paris, 1989. English translation: *Mathematics for Computer Algebra*, Springer-Verlag, New York, 1992.

Peter Paule, Greatest factorial factorization and symbolic summation. *Journal of Symbolic Computation* **20** (1995), 235–268.

Steven Roman, *The umbral calculus.* Pure and applied mathematics **111**. Academic Press, Orlando FL, 1984.

A. Schönhage, Schnelle Multiplikation von Polynomen über Körpern der Charakteristik 2. *Acta Informatica* **7** (1977), 395–398.

A. Schönhage and V. Strassen, Schnelle Multiplikation großer Zahlen. *Computing* **7** (1971), 281–292.

Arnold Schönhage, Andreas F. W. Grotefeld, and Ekkehart Vetter, *Fast Algorithms – A Multitape Turing Machine Implementation.* BI Wissenschaftsverlag, Mannheim, 1994.

Victor Shoup, A new polynomial factorization algorithm and its implementation. *Journal of Symbolic Computation* **20** (1995), 363–397.

Volker Strassen, Die Berechnungskomplexität von elementarsymmetrischen Funktionen und von Interpolationskoeffizienten. *Numerische Mathematik* **20** (1973), 238–251.

V. Strassen, Some results in algebraic complexity theory. In *Proceedings of the International Congress of Mathematicians 1974*, Vancouver, 1974, 497–501.

David Y. Y. Yun, On square-free decomposition algorithms. In *Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation ISSAC '76,* Yorktown Heights NY, ed. R. D. Jenks. ACM Press, 1976, 26–35.