Proc. ISSAC '97, Maui, 1997, pp. 40-47
(some errors are corrected in this version)

# Fast algorithms for Taylor shifts and certain difference equations

Joachim von zur Gathen and Jürgen Gerhard

Fachbereich 17 Mathematik-Informatik
Universität-GH Paderborn
D-33095 Paderborn, Germany
e-mail: {gathen,jngerhar}@uni-paderborn.de
www: http://www.uni-paderborn.de/cs/gathen.html

Extended Abstract

**Abstract**

We analyze six algorithms for computing integral Taylor shifts for polynomials with integral coefficients. We present and analyze a new algorithm for solving the "key equation" which occurs in many rational and hypergeometric summation algorithms. In a special case, our algorithm is asymptotically faster than previously known methods. We give experimental results for our algorithms.

## 1 Introduction

The basic tasks in symbolic summation, namely the summation of rational and hypergeometric functions, have been solved for quite a while. An elegant framework for several such solutions is presented by Paule (1995), and we refer to that paper for the extensive literature on the subject.

In another area of Computer Algebra, namely the factorization of polynomials over finite fields, the theoretical progress of recent years has given rise to software that can handle problems of gigantic size, completely beyond the reach of methods a few years ago (Montgomery 1991, Kaltofen & Lobo 1994, Reischert 1995, Shoup 1995, Fleischmann & Roelse 1996). At the heart of these achievements lies the systematic use of fast arithmetic, pioneered by Shoup (1993). This Extended Abstract presents a first step towards applying fast methods in symbolic summation, namely to two problems that occur in several summation algorithms (but may also be of interest in other areas of Computer Algebra): computing integral Taylor shifts and solving linear first order difference equations.

The paper is organized as follows. In Section 2, we analyze six algorithms for computing integral Taylor shifts $f(x+a)$ for a polynomial $f \in \mathbb{Z}[x]$ and an integer $a$. In Section 3, we present and analyze a new algorithm for solving the "key equation" $a \cdot u(x+1) - b \cdot u(x) = c$ for $u \in F[x]$, where $F$ is a field of characteristic 0 and $a, b, c \in F[x]$ are given, which occurs in many rational and hypergeometric summation algorithms. Both sections include some experimental results.

## 2 Computing integral Taylor shifts

For a polynomial $f = \sum_{0 \le i \le n} f_i x^i \in \mathbb{Z}[x]$ and an integer $a \in \mathbb{Z}$, we want to compute the coefficients $g_0, \ldots, g_n \in \mathbb{Z}$ of the Taylor expansion

$$g = \sum_{0 \le k \le n} g_k x^k = E^a f = f(x+a) = \sum_{0 \le i \le n} f_i (x+a)^i, \quad (1)$$

where $E$ is the shift operator $Ef = f(x+1)$. This is one ingredient of algorithms for symbolic summation (Abramov 1971, Gosper 1978, Paule 1995), and is a basic operation in many computer algebra systems (e.g., `translate` in Maple).

Writing out (1) explicitly, for $0 \le k \le n$ we have

$$g_k = \sum_{k \le i \le n} \binom{i}{k} f_i a^{i-k}. \quad (2)$$

An important special case is $a = \pm 1$.

The following lemma says how the coefficient size of a polynomial increases at most by a Taylor shift.

**Lemma 2.1** *Let $f \in \mathbb{Z}[x]$ be nonzero of degree $n \in \mathbb{N}$ and $a \in \mathbb{Z}$. If the coefficients of $f$ are bounded in absolute value by $B \in \mathbb{N}$, then the coefficients of $g = f(x+a) \in \mathbb{Z}[x]$ are absolutely bounded by $B(|a|+1)^n$.*

For $f, g: \mathbb{N} \longrightarrow \mathbb{R}_{\ge 0}$, we have the usual notation $f \in O(g)$ if there are constants $N, c \in \mathbb{N}$ such that $f(n) \le cg(n)$ for all $n \ge N$. Similarly, $f \in \Omega(g)$ and $f \in \Theta(g)$, and $f \in O^\sim(g)$ if $f \in g \cdot (\log(2+g))^{O(1)}$.

Let $\mathsf{M}: \mathbb{N} \longrightarrow \mathbb{N}$ be a multiplication time for $\mathbb{Z}$ and $\mathbb{Z}[x]$, i.e., two $l$-bit integers can be multiplied using $O(\mathsf{M}(l))$ bit operations, and two polynomials in $\mathbb{Z}[x]$ of degree at most $n$ with $l$-bit coefficients can be multiplied using $O(\mathsf{M}(nl))$ bit operations. By Schönhage & Strassen (1971) (see also Schönhage *et al.* 1994), we may assume that $\mathsf{M}(n) = n \log n \operatorname{loglog} n \in O^\sim(n)$. For practically fast algorithms, see Shoup (1995). Using standard reductions (see Aho *et al.* 1974), we may further assume that division with remainder of $l$-bit integers may be done in time $O(\mathsf{M}(l))$.

In the next two subsections, we discuss several computational methods and analyze their costs.

### 2.1 Using classical polynomial arithmetic

A. *Horner's method*: We compute

$$g(x) = f_0 + (x+a)\Big(f_1 + \cdots + (x+a)f_n \cdots\Big)$$

in $n$ steps

$$g^{(0)} = f_n, \quad g^{(i)} = (x+a)g^{(i-1)} + f_{n-i} \text{ for } 1 \le i \le n,$$

and obtain $g = g^{(n)}$.

B. *Shaw & Traub's (1974) method*: Compute $f^* = f(ax)$, $g^* = f^*(x+1)$, using A, and then $g = g^*(x/a)$.

C. *Multiplication-free method*: Suppose that $a > 0$. We write $a = \sum_{0 \le j < d} a_j 2^j$ in binary, with $a_j \in \{0,1\}$ for all $j$. Then $E^a = E^{a_{d-1}2^{d-1}} \circ \cdots \circ E^{a_1 \cdot 2} \circ E^{a_0}$. For $j = 0, 1, \ldots, d-1$, successively apply method A with $a_j 2^j$. The case $a < 0$ is handled similarly.

We note that both methods B and C boil down to Horner's method if $a = \pm 1$.

For a nonzero polynomial $f = \sum_{0 \le i \le n} f_i x^i \in \mathbb{Z}[x]$, we let

$$\lambda(f) = \log_2 \max\{|f_i| \colon 0 \le i \le n\}.$$

Then $\lfloor \lambda(f) \rfloor + 2$ is the binary length (including the sign) of the largest coefficient of $f$. In particular, $\lambda(a) = \log_2 |a|$ for a nonzero integer $a$.

**Theorem 2.2** *Let $f \in \mathbb{Z}[x]$ of degree $n \in \mathbb{N}_{\ge 1}$ and $a \in \mathbb{Z} \setminus \{0\}$ with $\lambda(f) \le l$ and $\lambda(a) \le d$ for some $l, d \in \mathbb{N}$. Then the cost in bit operations to compute $E^a f = f(x+a) \in \mathbb{Z}[x]$ for the three methods above is*

*A: $O(n^2 d(nd+l))$ with classical and $O(n^2\, \mathsf{M}(nd+l))$ with fast integer arithmetic,*

*B: $O(n^2 d(nd+l))$ with classical and $O(n^2(nd+l) + n\,\mathsf{M}(nd+l))$ with fast integer arithmetic,*

*C: $O(n^2 d(nd+l))$.*

**Proof.** A. In step $i$, we have at most $i-1$ additions and $i$ multiplications by $a$ in $\mathbb{Z}$, in total $O(n^2)$ arithmetic operations in $\mathbb{Z}$. Since the bit size of the integers involved is $O(nd+l)$, by Lemma 2.1, the cost for one addition is $O(nd+l)$, and the cost for one multiplication by $a$ is $O(d(nd+l))$ with classical multiplication and $O(\mathsf{M}(nd+l))$ with fast multiplication. Thus we get a total cost of $O(n^2 d(nd+l))$ and $O(n^2\,\mathsf{M}(nd+l))$ bit operations, respectively.

B. We have $n-1$ multiplications of size $O(d) \times O(nd)$ for the computation of $a^2, \ldots, a^n$, plus $n$ multiplications of size $O(l) \times O(nd)$ for the computation of $f_1 a, \ldots, f_n a^n$ which yield $f^*$, plus $O(n^2)$ additions of integers of size $O(nd+l)$, by A. and Lemma 2.1, plus $n$ exact divisions of numbers of size $O(nd+l)$ by numbers of size $O(nd)$. With classical integer arithmetic, the cost for dividing an integer $b$ by an integer $c$ is $O(\lambda(c)\lambda(\lfloor b/c \rfloor))$ bit operations, and we obtain a total of $O(n^2 d(nd+l))$ bit operations. With fast integer arithmetic, the overall cost is $O(n^2(nd+l) + n\,\mathsf{M}(nd+l))$.

**Corollary 2.3** *With the assumptions of Theorem 2.2, the cost for the three algorithms in bit operations is*

*(i) $O(n^2(n+l))$ if $a = \pm 1$,*

*(ii) $O(n^3 l^2)$ with classical integer arithmetic if $d \in O(l)$,*

*(iii) $O^{\sim}(n^3 l)$ for A and $O(n^3 l)$ for B, respectively, with fast integer arithmetic if $d \in O(l)$.*

| $n$ | small | large |
|---|---|---|
| 128 | 0.004 | 0.006 |
| 256 | 0.019 | 0.036 |
| 512 | 0.102 | 0.244 |
| 1024 | 0.637 | 1.788 |
| 2048 | 4.700 | 13.897 |
| 4096 | 39.243 | 111.503 |

Table 1: Running times with method A for $a = 1$, degree $n-1$, "small" coefficients between $-n$ and $n$, and "large" coefficients between $-2^n$ and $2^n$.

| $n$ | A | B | C |
|---|---|---|---|
| 128 | 0.04 | 0.02 | 0.04 |
| 256 | 0.33 | 0.18 | 0.31 |
| 512 | 2.74 | 1.59 | 2.56 |
| 1024 | 22.91 | 14.10 | 19.08 |
| 2048 | 220.32 | 148.17 | 253.57 |

Table 2: Running times with methods A,B,C for degree $n-1$ and "small" coefficients and values of $a$ between $-n$ and $n$.

| $n$ | A | B | C |
|---|---|---|---|
| 128 | 2.18 | 2.02 | 5.89 |
| 256 | 60.63 | 60.70 | 180.57 |
| 512 | 1875.04 | 1890.94 | 5735.17 |

Table 3: Running times with methods A,B,C for degree $n-1$ and "large" coefficients and values of $a$ between $-2^n$ and $2^n$.

Tables 1, 2, and 3 show the performances of methods A,B, and C in our experiments. Running times are given in average CPU seconds for 10 pseudorandomly chosen inputs on a Sparc Ultra with 167 MHz. Our software is written in `C++`. For arithmetic in $\mathbb{Z}$, we have used Victor Shoup's highly optimized `C++` library `NTL` for integer and polynomial arithmetic, which is in parts described in Shoup (1995). It uses Karatsuba & Ofman's (1962) method for multiplying large integers.

### 2.2 Asymptotically fast methods

D. *Paterson & Stockmeyer's (1973) method*: We assume that $(n+1) = m^2$ is a square (padding $f$ with zeroes if necessary), and write $f = \sum_{0 \le i < m} f^{(i)} x^{mi}$, with polynomials $f^{(i)} \in \mathbb{Z}[x]$ of degree less than $m$ for $0 \le i < m$.

1. Compute $(x+a)^i$ for $1 \le i \le m$.
2. For $0 \le i < m$, compute $f^{(i)}(x+a)$ as a linear combination of $1, (x+a), (x+a)^2, \ldots, (x+a)^{m-1}$.

3. Compute

$$g(x) = \sum_{0 \le i < m} f^{(i)}(x+a) \cdot (x+a)^{mi}$$

in a Horner-like fashion.

E. *Divide & conquer method* (von zur Gathen 1990, see also Bini & Pan 1994): We assume that $n+1 = 2^m$ is a power of two. In a precomputation stage, we compute $(x+a)^{2^i}$ for $0 \le i < m$. In the main stage, we write $f = f^{(0)} + x^{(n+1)/2} f^{(1)}$, with polynomials $f^{(0)}, f^{(1)} \in \mathbb{Z}[x]$ of degree less than $(n+1)/2$. Then

$$g(x) = f^{(0)}(x+a) + (x+a)^{(n+1)/2} f^{(1)}(x+a),$$

where we compute $f^{(0)}(x+a)$ and $f^{(1)}(x+a)$ recursively.

F. *The convolution method* (Aho *et al.* 1975, see also Schönhage *et al.* 1994, §9.3): After multiplying both sides of (2) by $k!n!$, we obtain

$$n!k!g_k = \sum_{k \le i \le n} (i!f_i) \cdot \frac{n!a^{i-k}}{(i-k)!}$$

in $\mathbb{Z}$. If we let $u = \sum_{0 \le i \le n} i!f_i x^{n-i}$ and $v = n! \sum_{0 \le j \le n} a^j x^j / j!$ in $\mathbb{Z}[x]$, then $n!k!g_k$ is the coefficient of $x^{n-k}$ in the product polynomial $uv$.

**Theorem 2.4** *Let $f \in \mathbb{Z}[x]$ of degree $n \in \mathbb{N}_{\ge 1}$, $a \in \mathbb{Z} \setminus \{0\}$, $\lambda(f) \le l$, and $\lambda(a) \le d$ for some $l, d \in \mathbb{N}$. The cost in bit operations to compute $E^a f = f(x+a) \in \mathbb{Z}[x]$ for the three methods above is*

*D:* $O(n^{1/2}\,\mathsf{M}(n^2 d + nl))$ *or* $O^{\sim}(n^{2.5} l)$,

*E:* $O(\mathsf{M}(n^2 d + nl) \log n)$ *or* $O^{\sim}(n^2 l)$,

*F:* $O(\mathsf{M}(n^2 \log n + n^2 d + nl))$ *or* $O^{\sim}(n^2 l)$,

*where the $O^{\sim}$-estimates are valid if $d \in O^{\sim}(l)$.*

We note that the input size is $\Theta(nl + d)$, and by Lemma 2.1, the size of the output $f(x+a)$ is $O(n(nd+l))$, or $O(n^2 l)$ if $d \in O(l)$. Thus Algorithms E and F are—up to logarithmic factors—asymptotically optimal. For $a = \pm 1$, the output size is $O(n(n+l))$.

**Proof.** D. In step 1, we have $O(m^2)$ multiplications and additions of integers of size $O(md)$, by Lemma 2.1, or $O(n\,\mathsf{M}(n^{1/2} d))$ bit operations. The computation of each $f^{(i)}(x+a)$ for $0 \le i < m$ in step 2 uses $O(m^2)$ integer multiplications and additions of size $O(md+l)$, and the total cost of step 2 is $O(n^{3/2}\,\mathsf{M}(n^{1/2} d + l))$ bit operations. Finally, we have at most $m$ polynomial multiplications and additions of degree at most $n$ with coefficients of size $O(nd+l)$, all together $O(n^{1/2}\,\mathsf{M}(n^2 d + nl))$ bit operations. This dominates the cost of the other two steps.

F. The size of the coefficients of $u$, $v$, and $uv$ is $O(n \log n + l)$, $O(n(\log n + d))$, and $O(n(\log n + d) + l)$, respectively (the last estimate follows from Lemma 2.1 and the fact that $n!k!g_k$ is the $k$th coefficient of $uv$), and the computation of $uv$ amounts to $O(\mathsf{M}(n(n(\log n + d) + l)))$ bit operations. The coefficients of $u$ and $v$ can be computed using $O(n\,\mathsf{M}(n(\log n + d) + l))$ bit operations, and the same number suffices to recover the $g_k$ from the coefficients of $uv$. Thus the total cost is $O(\mathsf{M}(n^2 \log n + n^2 d + nl))$ bit operations. $\square$

**Corollary 2.5** *Let $f \in \mathbb{Z}[x]$ of degree $n \in \mathbb{N}_{\ge 1}$ with $\lambda(f) \le l \in \mathbb{N}$. Then the cost in bit operations for computing $Ef = f(x+1)$ or $E^{-1} f = f(x-1)$ using the above algorithms is*

*D:* $O(n^{1/2}\,\mathsf{M}(n^2 + nl))$,

*E:* $O(\mathsf{M}(n^2 + nl) \log n)$,

*F:* $O(\mathsf{M}(n^2 \log n + nl))$.

*If $l \in O^{\sim}(n)$, then the cost is $O^{\sim}(n^{2.5})$ for D and $O^{\sim}(n^2)$ for E and F.*

If we want to compute integral shifts of the same polynomial for several values $a_1, \ldots, a_k \in \mathbb{Z}$ of absolute value less than $2^d$, then the output size is $O(kn(nd+l))$, and hence the simple idea of applying method E or F $k$ times independently is—up to logarithmic factors—asymptotically optimal.

| $n$ | D | E | F |
|---|---|---|---|
| 128 | 0.06 | 0.08 | 0.26 |
| 256 | 0.64 | 0.44 | 1.64 |
| 512 | 7.43 | 2.48 | 11.45 |
| 1024 | 87.57 | 15.53 | 86.09 |
| 2048 | 1387.39 | 102.64 | 713.20 |

Table 4: Running times with methods D,E,F for degree $n-1$ and "small" coefficients and values of $a$ between $-n$ and $n$.

| $n$ | D | E | F |
|---|---|---|---|
| 128 | 7.88 | 4.81 | 6.34 |
| 256 | 241.54 | 76.21 | 107.0 |
| 512 | 7453.69 | 1289.73 | |

Table 5: Running times with methods D,E,F for degree $n-1$ and "large" coefficients and values of $a$ between $-2^n$ and $2^n$. For method $F$ with $n = 512$, our program was aborted due to lack of memory; the output is about 8 MB.

Tables 4 and 5 give running times of methods D,E, and F in our experiments in average CPU seconds for 10 pseudorandomly chosen inputs on a Sparc Ultra with 167 MHz. Integer arithmetic is again taken from `NTL`, but the polynomial multiplication of `NTL`, which is based on a modular approach and Chinese remaindering, turned out to be too slow for polynomials with "large" coefficients as they occurred in our experiments. For those polynomials, we have used a straightforward implementation of FFT-multiplication modulo Fermat numbers $2^{2^k} + 1$, as used by Schönhage & Strassen (1971), and `NTL` for the coefficient arithmetic. For polynomials with moderately sized coefficients, we used the polynomial multiplication of `NTL`. The conclusion is that in our computing environment method B is the best choice for small problems, and method E for large ones.

## 3 Solving linear first order difference equations

Let $F$ be a field of characteristic 0 (say $F = \mathbb{Q}$). In many algorithms for symbolic summation (Abramov 1971, Gosper 1978, Paule 1995), a linear first order difference equation of the form

$$a \cdot Eu - b \cdot u = c, \tag{3}$$

often called *key equation*, with given polynomials $a, b, c \in F[x]$, has to be solved for a polynomial $u \in F[x]$. For example, the algorithms of Gosper (1978) and Paule (1995) reduce the quest for a *hypergeometric* solution $f$ of the comparably simple difference equation $Ef - f = g$ with constant coefficients and hypergeometric right hand side $g$ to a somewhat more complicated equation like (3), of which a *polynomial* solution $u$ is sought.

Often (3) is solved by first determining an upper bound on the degree of a possible solution $u$ and then solving the linear system in the unknown coefficients of $u$ equivalent to (3). It can be shown that the coefficient matrix of the linear system is triangular with at most one nonzero diagonal entry, and thus the linear algebra approach takes $O(n^2)$ arithmetic operations in $F$ if the matrix has at most $n$ rows and colums, while our algorithm uses only $O^\sim(n)$ operations.

Other algorithms for this problem are due to Abramov (1989) and Abramov *et al.* (1995), who study linear difference, $q$-difference, and differential equations of arbitrary order. In our situation of first order difference equations, the algorithm of Abramov *et al.* (1995) transforms the linear system corresponding to (3) into an equivalent one whose coefficient matrix is also triangular, has the same number of rows, and bandwidth at most $\max\{\deg a, \deg b\} + 2$. This system can be particularly efficiently solved if the bandwidth is small in comparison to the number of rows, and in that case their algorithm seems to be superior to ours. In our algorithm, no change of basis is necessary.

The following example shows where equations of the form (3) occur in symbolic summation algorithms.

**Example 3.1** *Let $t \in \mathbb{N}_{\geq 1}$, and suppose that we want to compute a closed form for the sum $\sum_{1\leq k<n}(k^2+tk)^{-1}$. Such a closed form can be obtained by determining—if possible—a rational function $u/v$, with $u, v \in F[x]$ and $v$ nonzero and monic, satisfying the difference equation*

$$\frac{Eu}{Ev} - \frac{u}{v} = \frac{1}{x^2+tx} \qquad (4)$$

*in $F(x)$, and then*

$$\sum_{1\leq k<n} \frac{1}{k^2+tk} = \frac{u(n)}{v(n)} - \frac{u(1)}{v(1)}$$

*(see, e.g., Graham* et al. *1994). Using Abramov's (1971) or Gosper's (1978) algorithm, this leads to solving an equation of the form (3) with $a = x$, $b = x + t$, and $c = (x+1)(x+2)\cdots(x+t-1)$. Its unique solution is $u = -v'/t$, where $v = x(x+1)(x+2)\cdots(x+t-1)$ and $'$ denotes the formal derivative with respect to $x$, and $u/v$ satisfies (4). The rational function $u/v$ is in reduced form since $v$ is squarefree, and the degree $\deg u = t-1$ is exponential in the bit size of the input $(x^2+tx)^{-1}$, which is about $\log t$.*

We first restate the following well-known lemma about the degree bound. A proof can, e.g., be found in Graham *et al.* (1994), §5.7. Here and in the sequel, $\mathrm{lc}(f)$ denotes the leading coefficient of a nonzero polynomial $f \in F[x]$, and we will assume that the degree of the zero polynomial is $-\infty$.

**Lemma 3.2** *Let $a, b, c, u \in F[x]$ be nonzero polynomials satisfying (3), with degrees $n, m, k, d$, respectively. Furthermore, let $\delta \in F$ be the coefficient of $x^{n-1}$ in $\mathrm{lc}(a)^{-1}(b-a)$, with $\delta = 0$ if $n = 0$. Then $\deg(a-b) < \max\{n, m\}$ or $\delta \neq k-n+1$. Moreover,*

*(i) $d = k - \deg(a-b)$ if $\deg(a-b) \geq \max\{n, m\}$,*

*(ii) $d = k-n+1$ if $\deg(a-b) < \max\{n, m\}$ and $\delta \notin \mathbb{N}$ or $\delta < k-n+1$,*

*(iii) $d \in \{k-n+1, \delta\}$ if $\deg(a-b) < \max\{n, m\}$, $\delta \in \mathbb{N}$, and $\delta > k-n+1$.*

### 3.1 The generic case

The main observation for our algorithm is that the highest coefficients of a possible solution $u \in F[x]$ of (3) only depend on the highest coefficients of $a, b$, and $c$. The idea is, similarly to the asymptotically fast Euclidean algorithm (see Aho *et al.* 1974, Strassen 1983), to compute first the "upper half" of $u$, only using the "upper halfs" of $a, b$, and $c$, plugging the obtained partial solution into (3), and recursively solving it for the "lower half". We first illustrate this in an example. For a polynomial $f \in F[x]$ and an integer $k$, we denote by $f \upharpoonright k$ the polynomial part of $x^{-k}f$. Thus $f \upharpoonright k$ is equal to the quotient of $f$ on division by $x^k$ if $k \geq 0$, and to $x^{-k}f$ if $k < 0$, and $\mathrm{lc}(f \upharpoonright k) = \mathrm{lc}(f)$ if both polynomials are nonzero. We have $\deg(f \upharpoonright k) = \deg f - k$ if $f \upharpoonright k \neq 0$, and $\deg(x^k(f \upharpoonright k) - f) < k$.

**Example 3.3** *We study the generic example with $m = n = 3$ and $k = 6$. So let $a = \sum_{0\leq i\leq 3} a_i x^i$, $b = \sum_{0\leq i\leq 3} b_i x^i$, $c = \sum_{0\leq i\leq 6} c_i x^i$ and $u = \sum_{0\leq i\leq 3} u_i x^i$ in $F[x]$ such that (3) holds. Comparing coefficients on both sides yields the linear system*

$$\begin{aligned}
c_6 &= b_3^* u_3, \\
c_5 &= b_3^* u_2 + (3a_3 + b_2^*)u_3, \\
c_4 &= b_3^* u_1 + (2a_3 + b_2^*)u_2 + (3a_3 + 3a_2 + b_1^*)u_3, \\
c_3 &= b_3^* u_0 + (a_3 + b_2^*)u_1 + (a_3 + 2a_2 + b_1^*)u_2 \\
&\quad + (a_3 + 3a_2 + 3a_1 + b_0^*)u_3, \\
c_2 &= b_2^* u_0 + (a_2 + b_1^*)u_1 + (a_2 + 2a_1 + b_0^*)u_2 \\
&\quad + (a_2 + 3a_1 + 3a_0)u_3, \\
c_1 &= b_1^* u_0 + (a_1 + b_0^*)u_1 + (a_1 + 2a_0)u_2 + (a_1 + 3a_0)u_3, \\
c_0 &= b_0^* u_0 + a_0 u_1 + a_0 u_2 + a_0 u_3,
\end{aligned}$$

*where $b_i^*$ is shorthand for $a_i - b_i$ for $0 \leq i \leq 3$. If $3 = \deg(a-b) = \max\{\deg a, \deg b\}$, then $b_3^* \neq 0$, and the first four equations uniquely determine $u_0, \ldots, u_3$, and $u_2$ and $u_3$ can already be computed from the first two equations. Let $U = u \upharpoonright 2 = u_3 x + u_2$, $A = (x+1)^2 a \upharpoonright 4 = a_3 x + 2a_3 + a_2$, $B = x^2 b \upharpoonright 4 = b_3 x + b_2$, and $C = c \upharpoonright 4 = c_6 x^2 + c^5 x + c_4$. Then*

$$\begin{aligned}
A \cdot EU - B \cdot U &= b_3^* u_3 x^2 + \Big(b_3^* u_2 + (3a_3 + b_2^*)u_3\Big)x + \cdots \\
&= c_6 x^2 + c^5 x + \cdots,
\end{aligned}$$

*i.e., $U$ is the unique polynomial satifying $\deg(A \cdot EU - B \cdot U - C) \leq \deg C - 2$, or equivalently, $\deg(a \cdot E(Ux^2) - b \cdot Ux^2 - c) \leq \deg c - 2$.*

*If we have determined $U$, we write $u = Ux^2 + V$, with $V = u_1 x + u_0$, and plug this into (3):*

$$\begin{aligned}
a \cdot EV - b \cdot V &= a(EU \cdot (x+1)^2 + EV) - b(Ux^2 + V) \\
&\quad - ((x+1)^2 a \cdot EU - x^2 b \cdot U) \\
&= c - ((x+1)^2 a \cdot EU - x^2 b \cdot U).
\end{aligned}$$

*This is again a linear first order difference equation for $V$ which can then be solved. In general, the degrees of $U$ and $V$ are about half the degree of $u$, and they can be determined recursively.*

In the sequel, it is convenient to denote by $\varphi_{a,b}\colon F[x] \longrightarrow F[x]$ the $F$-linear operator $\varphi_{a,b}(u) = a \cdot Eu - b \cdot u$. Then (3) can be rewritten in the equivalent form

$$\varphi_{a,b}(u) = c. \tag{5}$$

Let $\mathsf{M}(n)$ be a multiplication time for $F[x]$, such that polynomials in $F[x]$ of degree at most $n$ can be multiplied using $O(\mathsf{M}(n))$ operations in $F$. By Schönhage & Strassen (1971), we may take $\mathsf{M}(n) = n \log n \operatorname{loglog} n \in O^{\sim}(n)$.

The following algorithm works in case (i) of Lemma 3.2. In this case, the coefficient matrix of the linear system equivalent to (5) is triangular, with all diagonal entries equal to the leading coefficient of $a - b$, as in Example 3.3.

**Algorithm 3.4**
*Input:* $a, b, c \in F[x]$ *with* $\deg(a-b) = \max\{\deg a, \deg b\} = n \in \mathbb{N}$, *and* $d = \deg c - n$.
*Output: A polynomial* $u \in F[x]$ *with* $u = 0$ *or* $\deg u = d$ *if* $d \geq 0$ *such that* $\deg(\varphi_{a,b}(u) - c) < n$.

1. *If* $d < 0$ *then return* $u = 0$.
2. *If* $d = 0$ *then return* $u = \operatorname{lc}(c)/\operatorname{lc}(a-b)$.
3. *Set* $m = \lceil d/2 \rceil$, $t = n - (d-m)$, $A_1 = (x+1)^m a \restriction t+m$, $B_1 = x^m b \restriction t+m$, *and* $C_1 = c \restriction t+m$.
4. *Recursively call the algorithm with input* $A_1, B_1, C_1$ *to obtain* $U \in F[x]$ *such that* $\deg(\varphi_{A_1,B_1}(U) - C_1) < d - m$.
5. *Set* $A_2 = a \restriction n-m$, $B_2 = b \restriction n-m$, *and* $C_2 = (c - \varphi_{a,b}(Ux^m)) \restriction n-m$.
6. *Recursively call the algorithm with input* $A_2, B_2, C_2$, *yielding* $V \in F[x]$ *with* $\deg(\varphi_{A_2,B_2}(V) - C_2) < m$.
7. *Return* $u = Ux^m + V$.

**Theorem 3.5** *Algorithm 3.4 works correctly and uses* $O(\mathsf{M}(d)\log d)$ *arithmetic operations in* $F$.

**Proof.** We prove correctness by induction on $d = \deg c - \deg(a-b)$. It is clear that the output is correct if $d \leq 0$, and we assume that $d > 0$. In step 3, the leading coefficients of $A_1$ and $B_1$ agree with those of $a$ and $b$, respectively. Moreover, $\deg A_1 = \deg a - t$ and $\deg B_1 = \deg b - t$, if these polynomials are nonzero, $\deg C_1 = \deg c - (t+m) = 2(d-m)$, and

$$\begin{aligned} \deg(A_1 - B_1) &= \max\{\deg A_1, \deg B_1\} \\ &= \max\{\deg a, \deg b\} - t \\ &= n - (n - (d-m)) = d - m, \\ \deg C_1 - \deg(A_1 - B_1) &= d - m < d. \end{aligned} \tag{6}$$

At least one of $A_1$ and $B_1$ is nonzero, however, and (6) holds in any case. Thus by induction, the output of the recursive call in step 4 is correct, i.e., $\deg U = \deg EU = d - m$ and $\deg(\varphi_{A_1,B_1}(U) - C_1) < \deg(A_1 - B_1) = d - m$ (we note that $U \neq 0$ since otherwise $d - m = \deg C_1 = \deg(\varphi_{A_1,B_1} - C_1) < d - m$).

In step 5, the leading coefficients of $A_2$ and $B_2$, if these polynomials are nonzero, agree with those of $a$ and $b$, respectively,

$$\deg A_2 = \deg a - (n-m), \quad \deg B_2 = \deg b - (n-m), \tag{7}$$

and

$$\begin{aligned} \deg(A_2 - B_2) &= \max\{\deg A_2, \deg B_2\} \\ &= \max\{\deg a, \deg b\} - (n-m) = m. \end{aligned} \tag{8}$$

If $A_2$ or $B_2$ is zero, the corresponding degree equation in (7) does not hold, but at least one of them is nonzero, and (8) is always true. Let $c^* = c - \varphi_{a,b}(Ux^m)$. By the definition of $\restriction$, the degrees of $(x+1)^m a - x^{t+m}A_1$, $x^m b - x^{t+m}B_1$, and $c - x^{t+m}C_1$ are less than $t+m$, and we conclude that

$$\begin{aligned} \deg c^* &= \deg(\varphi_{a,b}(Ux^m) - c) \\ &= \deg(\varphi_{(x+1)^m a, x^m b}(U) - c) \\ &= \deg \Big( x^{t+m}(\varphi_{A_1,B_1}(U) - C_1) \\ &\quad + ((x+1)^m a - x^{t+m}A_1)EU \\ &\quad - (x^m b - x^{t+m}B_1)U - (c - x^{t+m}C_1) \Big) \\ &< t + m + d - m = n + m. \end{aligned}$$

Hence

$$\deg C_2 - \deg(A_2 - B_2) = \deg c^* - (n-m) - m < m \leq d,$$

and by induction, the output of the recursive call in step 6 is correct as well, i.e., $\deg V \leq \deg C_2 - \deg(A_2 - B_2) < m$ and $\deg(\varphi_{A_2,B_2}(V) - C_2) < \deg(A_2 - B_2) = m$. Thus

$$\deg u = \deg(Ux^m + V) = \deg U + m = d.$$

Finally,

$$\begin{aligned} \varphi_{a,b}(u) - c &= \varphi_{a,b}(Ux^m) + \varphi_{a,b}(V) - (c^* + \varphi_{a,b}(Ux^m)) \\ &= \varphi_{a,b}(V) - c^*, \end{aligned}$$

and

$$\begin{aligned} \deg(\varphi_{a,b}(u) - c) &= \deg(\varphi_{a,b}(V) - c^*) \\ &= \deg \Big( x^{n-m}(\varphi_{A_2,B_2}(V) - C_2) \\ &\quad + (a - A_2 x^{n-m})EV \\ &\quad - (b - B_2 x^{n-m})V - (c^* - C_2 x^{n-m}) \Big) \\ &< n - m + m = n, \end{aligned}$$

where we have used that the degrees of $a - A_2x^{n-m}$, $b - B_2x^{n-m}$, and $c - C_2x^{n-m}$ are less than $n - m$.

We denote by $T(d)$ the cost of the algorithm for inputs with $\deg c - \deg(a-b) = d$. The cost of steps 1 and 2 is $O(1)$. In step 3, we first compute $(x+1)^m$ with repeated squaring, and then compute the leading $d - m + 1$ coefficients of $(x+1)^m a$, at a cost of $O(\mathsf{M}(d))$ operations. This is the total cost for step 3, since the computation of $B_1$ and $C_1$ requires no arithmetic operations. In step 4, we have $\deg C_1 - \deg(A_1 - B_1) = d - m \leq \lfloor d/2 \rfloor$, and hence the cost for the recursive call is at most $T(\lfloor d/2 \rfloor)$. In step 5, we compute the coefficients of $x^{n-m}, \ldots, x^{n+d}$ in $c^*$, similarly to step 3, at a cost of $O(\mathsf{M}(d+m))$ operations. The computation of $A_2$ and $B_2$ is for free. Finally, in step 6 we have $\deg C_2 - \deg(A_2 - B_2) \le m - 1 \le \lfloor d/2 \rfloor$, and the cost for the recursive call is at most $T(\lfloor d/2 \rfloor)$. Step 7 is essentially free.

Summarizing, we have $T(0), T(1) \in O(1)$ and

$$T(d) \le 2T(\lfloor d/2 \rfloor) + O(\mathsf{M}(d)) \text{ if } d \ge 2.$$

The running time bound now follows from unravelling the recursion. □

If $d \approx n$, then both the linear algebra approach and the algorithm of Abramov *et al.* (1995) take $\Omega(n^2)$ operations, while our algorithm uses only $O^{\sim}(n)$ operations.

In practice, dividing $a, b, c$ by their common divisor before applying Algorithm 3.4 reduces its running time. As Paule (1995) remarks, we may even achieve that $a, b, c$ are *pairwise* coprime. E.g., if $g = \gcd(a, c)$ is nonconstant and coprime to $b$ and $u$ satisfies (3), then necessarily $g$ divides $u$, and $u^* = u/g$ is a solution of the linear difference equation

$$\frac{a}{g} \cdot Eu^* - b \cdot u^* = \frac{c}{g}$$

whose coefficients $a/g$ and $c/g$ have smaller degrees than the corresponding coefficients of (3). This also applies to Algorithms 3.7 and 3.9 below.

**Corollary 3.6** *Let $a, b, c \in F[x]$ of degrees at most $n \in \mathbb{N}$ with $\deg(a-b) = \max\{\deg a, \deg b\}$. Then we have an algorithm that either computes the unique solution $u \in F[x]$ of (5), which has degree $d = \deg c - \deg(a-b)$, or correctly asserts that no solution exists. It uses $O(\mathsf{M}(n)\log n)$ arithmetic operations in $F$.*

### 3.2 The general case

Cases (ii) and (iii) of Lemma 3.2 are somewhat more involved, due to the possibility of multiple solutions of (3). Since the operator $\varphi_{a,b}$ is $F$-linear, its kernel (i.e., the set of solutions $u \in F[x]$ of the homogeneous difference equation $\varphi_{a,b}(u) = a \cdot Eu - b \cdot u = 0$) is an $F$-subspace of $F[x]$. Lisoněk *et al.* (1993) have shown that $\ker \varphi_{a,b}$ has dimension at most one (this is analogous to the situation for linear first-order differential equations), and hence the set of all solutions of the inhomogeneous equation (3) either is empty, or consists of exactly one element, or is a one-dimensional coset of $\ker \varphi_{a,b}$. They also show that if $\ker \varphi_{a,b}$ is one-dimensional, then $\deg(a-b) < \max\{\deg a, \deg b\}$ and the degree of all nonzero polynomials in $\ker \varphi_{a,b}$ equals the $\delta$ from Lemma 3.2. (The reverse direction does not hold; examples are provided, e.g., by Lisoněk *et al.* 1993, where our "$\delta$" is called "$K_0$".)

Let $a = \sum_{0 \le i \le n} a_i x^i$, $b = \sum_{0 \le i \le n} b_i x^i$, $u = \sum_{0 \le i \le d} u_i x^i$, and $c = \sum_{0 \le i \le k} c_i x^i$, with $n = \deg a = \deg b \ge 0$, $d = \deg u$, $k$ in $\mathbb{N}$, and assume that $\deg(a-b) < \max\{\deg a, \deg b\}$ (i.e., $a_n = b_n$). In the sequel, $\delta = \delta_{a,b} = a_n^{-1}(b_{n-1} - a_{n-1})$ denotes the coefficient of $x^{n-1}$ in $a_n^{-1}(b-a)$, with $\delta_{a,b} = 0$ if $n = 0$. The coefficient matrix of the linear system in the coefficients of $u$ equivalent to (3) is triangular, with the coefficient of $u_i$ in the equation corresponding to $x^{n-1+i}$ equal to $a_n i + a_{n-1} - b_{n-1} = a_n(i - \delta)$. E.g., if $a_3 = b_3$ in Example 3.3, then the linear system reads

$$\begin{aligned}
c_6 &= 0, \\
c_5 &= (3a_3 + b_2^*)u_3, \\
c_4 &= (2a_3 + b_2^*)u_2 + (3a_3 + 3a_2 + b_1^*)u_3, \\
c_3 &= (a_3 + b_2^*)u_1 + (a_3 + 2a_2 + b_1^*)u_2 \\
&\quad + (a_3 + 3a_2 + 3a_1 + b_0^*)u_3, \\
c_2 &= b_2^* u_0 + (a_2 + b_1^*)u_1 + (a_2 + 2a_1 + b_0^*)u_2 \\
&\quad + (a_2 + 3a_1 + 3a_0)u_3, \\
c_1 &= b_1^* u_0 + (a_1 + b_0^*)u_1 + (a_1 + 2a_0)u_2 + (a_1 + 3a_0)u_3, \\
c_0 &= b_0^* u_0 + a_0 u_1 + a_0 u_2 + a_0 u_3,
\end{aligned}$$

where again $b_i^* = a_i - b_i$ for $0 \le i \le 3$. In general, at most one of the subdiagonal entries $a_n(i-\delta)$ vanishes, and this can only happen if $\delta$ is a nonnegative integer. Then there may be a degree of freedom in the choice of $u_\delta$ (corresponding to a nonzero solution of the homogeneous equation $a \cdot Eu - b \cdot u = 0$), in which case we might simply choose $u_\delta = 0$. However, it may happen that $\delta \in \mathbb{N}$ and $\ker \varphi_{a,b} = \{0\}$, and in that case $u_\delta$ has to be chosen consistently with the other equations.

The following algorithm can be used in case (ii) of Lemma 3.2 if $\delta \notin \mathbb{N}$ (then (5) has at most one solution). It computes $u \in F[x]$ such that $\deg(\varphi_{a,b}(u) - c) < \deg a - 1$ and works rather similarly to Algorithm 3.4.

**Algorithm 3.7**
*Input:* $a = \sum_{0 \le i \le n} a_i x^i$, $b = \sum_{0 \le i \le n} b_i x^i$, $c$ *in* $F[x]$ *such that* $n \in \mathbb{N}$, $a_n = b_n \ne 0$, $d = 1 - n + \deg c$, *and* $\delta = \delta_{a,b} \notin \{0, 1, \ldots, d\}$.
*Output: A polynomial* $u \in F[x]$ *with* $u = 0$ *or* $\deg u = d$ *if* $d \ge 0$ *such that*

$$\deg(\varphi_{a,b}(u) - c) < n - 1. \tag{9}$$

1. *If* $d < 0$ *then return* $u = 0$.
2. *If* $d = 0$ *then return* $u = \mathrm{lc}(c)/(a_{n-1} - b_{n-1})$.
3. *Set* $m = \lceil d/2 \rceil$, $t = n - 1 - (d - m)$, $A_1 = (x+1)^m a \restriction t + m$, $B_1 = x^m b \restriction t + m$, *and* $C_1 = c \restriction t + m$.
4. *Recursively call the algorithm with input* $A_1, B_1, C_1$ *to obtain* $U \in F[x]$ *such that* $\deg(\varphi_{A_1,B_1}(U) - C_1) < d - m$.
5. *Set* $A_2 = a \restriction n - 1 - m$, $B_2 = b \restriction n - 1 - m$, *and* $C_2 = (c - \varphi_{a,b}(Ux^m)) \restriction n - 1 - m$.
6. *Recursively call the algorithm with input* $A_2, B_2, C_2$, *yielding* $V \in F[x]$ *with* $\deg(\varphi_{A_2,B_2}(V) - C_2) < m$.
7. *Return* $u = Ux^m + V$.

**Theorem 3.8** *Algorithm 3.7 works correctly and uses* $O(\mathsf{M}(d)\log d)$ *arithmetic operations in* $F$.

We now use Algorithm 3.7 to solve (5) in the case where $\deg(a-b) < \max\{\deg a, \deg b\}$. This is clear if $\delta = \delta_{a,b} \notin \mathbb{N}$; otherwise we first compute the coefficients of $x^i$ in $u$ for $i > \delta$ and then take care of a possible freedom in choosing the coefficient of $x^\delta$. The idea is to compute polynomials $U, V, W \in F[x]$ such that $\deg U < d - \delta$, $\deg V, \deg W < \delta$, and the set of all solutions of (9) is $\{Ux^{\delta+1} + V + s(x^\delta + W) : s \in F\}$, using Algorithm 3.7, and then to check whether some (or every) $s \in F$ gives a solution of (5) (this corresponds to the checking in Corollary 3.6).

**Algorithm 3.9**
*Input:* $a = \sum_{0\le i\le n} a_i x^i$, $b = \sum_{0\le i\le n} b_i x^i$, $c$ *in* $F[x]$ *such that* $n \in \mathbb{N}$, $a_n = b_n \ne 0$, *and* $d = 1-n+\deg c$ *if* $\delta = \delta_{a,b} \notin \mathbb{N}$ *and* $d = \max\{1-n+\deg c, \delta\}$ *otherwise.*
*Output: A solution* $u \in F[x]$ *of (5) of degree at most* $d$*, or "unsolvable" if (5) is unsolvable.*

1. *If* $\delta \notin \mathbb{N}$ *then call Algorithm 3.7 to compute* $u \in \mathbb{F}[x]$ *such that* $\deg(\varphi_{a,b}(u) - c) < n-1$*. If* $\varphi_{a,b}(u) = c$ *then return* $u$*, otherwise return "unsolvable".*

2. *Set* $m = \delta+1$, $t = n-1-(d-m)$, $A = (x+1)^m a \restriction t+m$, $B = x^m b \restriction t+m$*, and* $C = c \restriction t+m$*.*

3. *If* $d = \delta$*, then set* $U = 0$*; otherwise call Algorithm 3.7 with input* $A, B, C$ *to obtain* $U \in F[x]$ *such that* $\deg(\varphi_{A,B}(U) - C) < d-m$*.*

4. *Set* $c^* = c - \varphi_{a,b}(Ux^m)$*. If* $1-n+\deg c^* = \delta$ *then return "unsolvable".*

5. *Call Algorithm 3.7 with input* $a, b, c^*$ *to obtain* $V \in F[x]$ *such that* $\deg(\varphi_{a,b}(V) - c^*) < n-1$*.*

6. *Set* $h^* = \varphi_{a,b}(V) - c^*$*. If* $h^* = 0$ *then return* $u = Ux^m + V$*.*

7. *Set* $c^{**} = -\varphi_{a,b}(x^\delta)$*, and call Algorithm 3.7 with input* $a, b, c^{**}$ *to obtain* $W \in F[x]$ *such that* $\deg(\varphi_{a,b}(W) - c^{**}) < n-1$*.*

8. *Set* $h^{**} = \varphi_{a,b}(W) - c^{**}$*. If there exists some* $s \in F$ *such that* $h^* + sh^{**} = 0$ *in* $F[x]$*, then return* $u = Ux^m + V + s(x^\delta + W)$ *for the unique such* $s$*, otherwise return "unsolvable".*

**Theorem 3.10** *Algorithm 3.9 works correctly and uses* $O(\mathsf{M}(n) + \mathsf{M}(d)\log d)$ *arithmetic operations in* $F$*.*

In the nonunique case, where $\ker\varphi_{a,b} \ne \{0\}$, Algorithm 3.9 stops in step 6 if (5) is solvable, but may need to proceed until step 8 to find out that (5) is unsolvable (see Example 3.11 (ii) below). If we know somehow in advance that $\ker\varphi_{a,b} \ne \{0\}$, then we may already return "unsolvable" in step 6 if $h^* \ne 0$, since then $\ker\varphi_{a,b} = \{s(x^\delta + W) : s \in F\}$ and $h^{**} = 0$ in step 8. This may be useful in Gosper's (1978) algorithm, since Lisoněk *et al.* (1993) have shown that the condition $\ker\varphi_{a,b} = \{0\}$ corresponds to the case where Gosper's algorithm is used to compute the sum of a rational function.

We note that the value of $d$ in Algorithm 3.9 may be exponentially large in the bit size of $a$, $b$, and $c$. For example, if $a = x$, $b = x+t$ for some $t \in \mathbb{N}_{\ge 1}$, and $c$ has "small" degree and "small" coefficients as well, then $d = \max\{1-\deg a + \deg c, \delta\} = \delta = t$, and this is exponential in the bit size of $a$ and $b$, which is about $\log t$.

We believe that a modification of our algorithms also works in the case of first order linear $q$-difference and differential equations.

**Example 3.11** *(i) Let* $a = x^2$ *and* $b = x^2 - x/2 + 1$*. Then* $\delta = -1/2 \notin \mathbb{N}$*. If we let* $c = x-2$*, then we obtain* $u = 2$ *in step 1 of Algorithm 3.9. We check that*

$$\varphi_{a,b}(u) = x^2 \cdot 2 - (x^2 - \frac{1}{2}x + 1) \cdot 2 = x - 2,$$

*and* $u$ *solves (5). On the other hand, for* $c = x$*, we also get* $u = 2$ *in step 1, but now* $\varphi_{a,b}(u) = c - 2 \ne c$*, and (5) is unsolvable.*

*(ii) Let* $a = x^2$ *and* $b = x^2 + 2x + 1$*, so that* $\delta = 2$*. For* $c = x^2 + x$*, we have* $d = \max\{1 - \deg a + \deg c, \delta\} = \delta = 2$*. Thus* $U = 0$ *in step 3,* $c^* = c$ *and* $1 + \deg a - \deg c^* = 1 < \delta$ *in step 4, and in step 5 we obtain* $V = -x$*. Then*

$$\varphi_{a,b}(V) = x^2(-x-1) - (x^2+2x+1)(-x) = x^2 + x,$$

*whence* $h^* = 0$ *in step 6, and* $u = -x$ *solves (5).*

*On the other hand, for* $c = x^2 + x + 1$ *we have* $d$, $U$, $c^*$*, and* $V$ *as before, but now* $h^* = -1$ *in step 6. Then*

$$c^{**} = -\varphi_{a,b}(x^\delta) = -x^2(x+1)^2 + (x^2+2x+1)x^2 = 0$$

*and* $W = 0$ *in step 7, and* $h^{**} = 0$ *in step 8. Thus* $h^* + sh^{**} = -1 \ne 0$ *for all* $s \in F$*, and (5) is unsolvable.*

*In fact,*

$$\ker\varphi_{a,b} = \{s(x^\delta + W) : s \in F\} = \{sx^2 : s \in F\},$$

*and the set of all solutions of (5) for* $c = x^2 + x$ *is*

$$Ux^{\delta+1} + V + \ker\varphi_{a,b} = \{sx^2 - x : s \in F\}.$$

*(iii) Let* $a = x^2$ *and* $b = x^2 + x + 1/4$*. Then* $\delta = 1$*, and for* $c = 4x^3 + 3x^2 + x$*, we have* $d = \max\{1 - \deg a + \deg c, \delta\} = 2 > \delta$*. Thus* $m = 2$, $t = 1$, $A = (x+1)^2 a \restriction 3 = x + 2$, $B = x^2 b \restriction 3 = x + 1$*, and* $C = c \restriction 3 = 4$ *in step 2, and in step 3 we obtain* $U = 4$*. Next, we have*

$$\begin{aligned} c^* &= c - \varphi_{a,b}(Ux^m) \\ &= 4x^3 + 3x^2 + x - x^2 \cdot 4(x+1)^2 + (x^2 + x + \frac{1}{4}) \cdot 4x^2 \\ &= x, \end{aligned}$$

*and* $1 - \deg a + \deg c^* = 0 < \delta$*. Now we compute* $V = -1$ *in step 5, and*

$$h^* = \varphi_{a,b}(V) - c^* = -x^2 + (x^2 + x + \frac{1}{4}) - x = \frac{1}{4} \ne 0$$

*in step 6. In step 7, we get*

$$c^{**} = -\varphi_{a,b}(x^\delta) = -x^2(x+1) + (x^2 + x + 1/4)x = \frac{1}{4}x,$$

*and* $W = -1/4$*. Finally,*

$$h^{**} = \varphi_{a,b}(W) - c^{**} = -\frac{1}{4}x^2 + \frac{1}{4}(x^2 + x + 1/4) - \frac{1}{4}x = \frac{1}{16}$$

*in step 8, and* $h^* + sh^{**} = 1/4 + s/16 = 0$ *if and only if* $s = -4$*. Then* $u = 4x^2 - 1 - 4\cdot(x - 1/4) = 4x(x-1)$*, and we check that*

$$\begin{aligned} \varphi_{a,b}(u) &= x^2 \cdot 4(x+1)x - (x^2 + x + 1/4)\cdot 4x(x-1) \\ &= 4x^3 + 3x^2 + x, \end{aligned}$$

*and* $u$ *solves (5).*

*If we take* $c = 4x^3 + 4x^2 + x$*, then* $d$ *and* $U$ *are as before, but now* $c^* = x^2 + x$, $1 - \deg a + \deg c^* = 1 = \delta$*, and the algorithm returns "unsolvable" in step 4.*

*In fact, the homogeneous equation* $\varphi_{a,b}(u) = 0$ *only has the trivial solution* $u = 0$*.*

| $t$ | time |
|---|---|
| 64 | 0.19 |
| 128 | 0.79 |
| 256 | 4.09 |
| 512 | 24.51 |
| 1024 | 168.62 |
| 2048 | 1250.90 |

Table 6: Running times of Algorithm 3.9 for the solution of the difference equation $x \cdot Eu - (x+t) \cdot u = (x+1)(x+2)\cdots(x+t-1)$ and various values of $t$. The solution has degree $t-1$; see Example 3.1.

Table 6 shows the running times of our implementation of Algorithm 3.9 in some experiments in CPU seconds on a Sparc Ultra with 167 MHz. Integer and polynomial arithmetic was taken from `NTL`, which we have extended so that it can also cope with polynomials over $\mathbb{Q}$, reducing polynomial multiplication in $\mathbb{Q}[x]$ to polynomial multiplication in $\mathbb{Z}[x]$ by multiplying with a common denominator. For multiplying polynomials with "large" coefficients, we have used the same algorithm as described at the end of Section 2.

We believe that in the Example of Table 6, a careful implementation of the algorithm of Abramov *et al.* (1995), which reduces the problem to solving a linear system with a 2-band coefficient matrix, using `NTL` would clearly outperform our algorithm. However, in case (i) of Lemma 3.2, Algorithm 3.4 is, at least when only counting arithmetic operations in $F$, asymptotically faster than the algorithm of Abramov *et al.* (1995), but we have not yet implemented it.

### Acknowledgements

We would like to thank the anonymous referees for many helpful comments and suggestions.

### References

S. A. Abramov, On the summation of rational functions. *Zh. vychisl. Mat. mat. Fiz.* **11** (1971), 1071–1075. English translation in USSR Computational Mathematics and Mathematical Physics.

S. A. Abramov, Problems in computer algebra involved in the search for polynomial solutions of linear differential and difference equations. *Moscow Univ. Comput. Math. and Cybernet.* **3** (1989), 63–68.

S. A. Abramov, M. Bronstein, and M. Petkovšek, On polynomial solutions of linear operator equations. In *Proc. ISSAC '95*. ACM Press, 1995, 290–296.

A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading MA, 1974.

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

D. Bini and V. Y. Pan, *Polynomial and matrix computations*, vol. 1. Birkhäuser, 1994.

P. Fleischmann and P. Roelse, Comparitive implementations of Berlekamp's and Niederreiter's polynomial factorization algorithms. In *Finite Fields and their Applications*, ed. S. Cohen and H. Niederreiter, 1996, 73–84.

J. von zur Gathen, Functional decomposition of polynomials: the tame case. *J. Symb. Comp.* **9** (1990), 281–299.

R. W. Gosper, Decision procedure for indefinite hypergeometric summation. *Proc. Natl. Acad. Sci. U.S.A.* **25** (1978), 40–42.

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

E. Kaltofen and A. Lobo, Factoring high-degree polynomials by the black box Berlekamp algorithm. In *Proc. ISSAC '94*, ed. J. von zur Gathen and M. Giesbrecht. ACM Press, 1994, 90–98.

A. Karatsuba and Y. Ofman, Умножение многозначных чисел на автоматах. *Dokl. Akad. Nauk USSR* **145** (1962), 293–294. Multiplication of multidigit numbers on automata, Soviet Physics–Doklady **7** (1963), 595-596.

P. Lisoněk, P. Paule, and V. Strehl, Improvement of the degree setting in Gosper's algorithm. *J. Symb. Comp.* **16** (1993), 243–258.

P. L. Montgomery, Factorization of $X^{216091} + X + 1 \bmod 2$ — a problem of Herb Doughty. Manuscript, 1991.

M. S. Paterson and L. Stockmeyer, On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM J. Comput.* **2** (1973), 60–66.

P. Paule, Greatest factorial factorization and symbolic summation. *J. Symbolic Computation* **20** (1995), 235–268.

D. Reischert, *Schnelle Multiplikation von Polynomen über GF(2) und Anwendungen*. Diplomarbeit, Universität Bonn, Germany, 1995.

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

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

V. Shoup, Factoring polynomials over finite fields: asymptotic complexity vs. reality. In *Proc. Int. IMACS Symp. on Symbolic Computation, New Trends and Developments*, Lille, France, 1993, 124–129.

V. Shoup, A new polynomial factorization algorithm and its implementation. *J. Symb. Comp.* **20** (1995), 363–397.

V. Strassen, The computational complexity of continued fractions. *SIAM J. Comput.* **12** (1983), 1–27.