Fast transforms over finite fields of characteristic two

Nicholas Coxon

FAST TRANSFORMS OVER FINITE FIELDS OF CHARACTERISTIC TWO

NICHOLAS COXON

Abstract. We describe new fast algorithms for evaluation and interpolation on the "novel" polynomial basis over finite fields of characteristic two introduced by Lin, Chung and Han (FOCS 2014). Fast algorithms are also described for converting between their basis and the monomial basis, as well as for converting to and from the Newton basis associated with the evaluation points of the evaluation and interpolation algorithms. Combining algorithms yields a new truncated additive fast Fourier transform (FFT) and inverse truncated additive FFT which improve upon some previous algorithms when the field possesses an appropriate tower of subfields.

1. Introduction

Let $\mathbb{F}$ be a finite field of characteristic two, and $\beta = (\beta_0, \ldots, \beta_{n-1}) \in \mathbb{F}^n$ have entries that are linearly independent over $\mathbb{F}_2$. Enumerate the $\mathbb{F}_2$-linear subspace of $\mathbb{F}$ generated by the entries of the vector by setting

$$\omega_i = \sum_{k=0}^{n-1} [i]_k \beta_k$$

for $i \in \{0, \ldots, 2^n - 1\}$, where $[\,\cdot\,]_k : \mathbb{N} \to \{0, 1\}$ for $k \in \mathbb{N}$ such that $i = \sum_{k \in \mathbb{N}} 2^k [i]_k$ for $i \in \mathbb{N}$. Define polynomials

$$X_i = \prod_{k=0}^{n-1} \prod_{j=0}^{2^k [i]_k - 1} \frac{x - \omega_j}{\omega_{2^k} - \omega_j} \quad \text{and} \quad N_i = \prod_{j=0}^{i-1} \frac{x - \omega_j}{\omega_i - \omega_j}$$

for $i \in \{0, \ldots, 2^n - 1\}$. Then the definition of the functions $[\,\cdot\,]_k$ implies that X_i has degree equal to i, while it is clear that N_i also has degree equal to i. Letting $\mathbb{F}[x]_\ell$ denote the space of polynomials over $\mathbb{F}$ with degree strictly less than ℓ, it follows that $\{X_0, \ldots, X_{\ell-1}\}$ and $\{N_0, \ldots, N_{\ell-1}\}$ are bases of $\mathbb{F}[x]_\ell$ over $\mathbb{F}$ for $\ell \in \{1, \ldots, 2^n\}$. The former basis was introduced by Lin, Chung and Han [21], and is referred to hereafter as the Lin–Chung–Han basis, or simply the LCH basis, of $\mathbb{F}[x]_\ell$ associated with β. The polynomials $N_0, \ldots, N_{2^n-1}$ are scalar multiples of the Newton basis polynomials associated with the points $\omega_0, \ldots, \omega_{2^n-1}$, with the scalars chosen such that $N_i(\omega_i) = 1$. However, we simply refer to $\{N_0, \ldots, N_{\ell-1}\}$ as the Newton basis of $\mathbb{F}[x]_\ell$ associated with β. The space $\mathbb{F}[x]_\ell$ also comes equipped with the monomial basis $\{1, x, \ldots, x^{\ell-1}\}$.

In this paper, we describe new fast algorithms for evaluation and interpolation on the Lin–Chung–Han basis, and for converting between the Lin–Chung–Han basis and each of the Newton and monomial bases. These algorithms may in turn be

This work was supported by Nokia in the framework of the common laboratory between Nokia Bell Labs and INRIA.

combined to obtain fast algorithms for evaluation and interpolation on the Newton or monomial bases, and for converting between the Newton and monomial bases. The resulting algorithm for evaluation on the monomial basis provides a new additive fast Fourier transform (FFT). The designation as "additive" reflects the fact that FFTs have traditionally evaluated polynomials at each element of a cyclic multiplicative group, whereas the evaluation points $\omega_0, \ldots, \omega_{2^n-1}$ of our FFT form an additive group. To avoid confusion, we refer to algorithms that evaluate over a multiplicative group as multiplicative FFTs hereafter. Additive FFTs have been investigated as an alternative to multiplicative FFTs for use in multiplication algorithms for binary polynomials [30, 5, 24, 7, 8, 18], and have also found applications in coding theory and cryptography [4, 3, 9, 10, 1].

Additive FFTs first appeared in the 1980s with the of algorithm of Wang and Zhu [31], which was subsequently rediscovered by Cantor [6]. For characteristic two finite fields, the Wang–Zhu–Cantor algorithm requires β to be a so-called Cantor basis: its entries must satisfy $\beta_0 = 1$ and $\beta_{k-1} = \beta_k^2 - \beta_k$ for $k \in \{1, \ldots, n-1\}$. The algorithm then takes the coefficients on the monomial basis of a polynomial in $\mathbb{F}[x]_{2^n}$ and evaluates it at each of the points $\omega_0, \ldots, \omega_{2^n-1}$ with $\mathcal{O}(2^n n^{\log_2 3})$ additions in $\mathbb{F}$, and $\mathcal{O}(2^n n)$ multiplications in $\mathbb{F}$. Gao and Mateer [14] subsequently improve upon this complexity by describing an algorithm that performs $\mathcal{O}(2^n n \log n)$ additions and $\mathcal{O}(2^n n)$ multiplications. However, as for the Wang–Zhu–Cantor algorithm, this complexity is only obtained in a limited setting, since a finite field of characteristic two admits a Cantor basis of dimension n if and only if it contains $\mathbb{F}_{2^{2^{\lceil \log_2 n \rceil}}}$ as a subfield [14, Appendix A].

The additive FFT of von zur Gathen and Gerhard [30] removes the restriction that β must be a Cantor basis, allowing the vector to be chosen subject only to the requirement of linear independence of its entries. Their algorithm performs $\mathcal{O}(2^n n^2)$ additions in $\mathbb{F}$, and $\mathcal{O}(2^n n^2)$ multiplications in $\mathbb{F}$. Gao and Mateer [14] subsequently describe an algorithm that performs $\mathcal{O}(2^n n^2)$ additions and only $\mathcal{O}(2^n n)$ multiplications. Bernstein, Chou and Schwabe [4] in turn generalise the algorithm of Gao and Mateer so that time is not wasted manipulating coefficients that are known to be zero when the polynomial being evaluated by the transform belongs to $\mathbb{F}[x]_\ell$ for some $\ell < 2^n$. They also describe how to replace some multiplications in their algorithm with less time consuming additions. Bernstein and Chou [3] contribute several more improvements to the algorithm in the case that β is a Cantor basis.

The generalisation of Bernstein, Chou and Schwabe is obtained by reducing to the case $\ell = 2^n$ and disregarding parts of the algorithm that involve manipulating coefficients that are known to be zero. This technique is often referred to as truncation (or pruning [23, 25]). This term is also applied when only part of a transform is computed by performing only those steps of the algorithm that are relevant to the computation of the desired outputs. Truncation is used in FFT-based polynomial multiplication to ensure that running times vary relatively smoothly in the length of the problem. To achieve such behaviour it is also necessary to invert truncated transforms. However, simply examining the output of a truncated transform may not allow its inversion. An elegant solution to this problem is provided by van der Hoeven [27, 26] who took the crucial step of augmenting the output with information about known zero coefficients in the input, allowing him to provide a multiplicative truncated Fourier transform together with its corresponding inverse truncated Fourier transform (see also [15, 16, 17]). While truncated additive FFTs

have been investigated [30, 24, 5, 4, 3, 7, 8, 18], existing methods for their inversion lack the effectiveness and elegance of the approach introduced by van der Hoeven.

Alongside the introduction of their basis, Lin, Chung and Han [21] describe fast algorithms for evaluation and interpolation on the basis that yield lower complexities than additive FFTs. Their evaluation algorithm takes the coefficients on the LCH basis of a polynomial in $\mathbb{F}[x]_{2^n}$ and evaluates it at each of the points $\omega_0, \ldots, \omega_{2^n-1}$ with $\mathcal{O}(2^n n)$ additions in $\mathbb{F}$, and $\mathcal{O}(2^n n)$ multiplications in $\mathbb{F}$. Their interpolation algorithm inverts this transformation with the same complexity. Lin, Chung and Han then demonstrate the usefulness of their "novel" basis by using the algorithms to provide fast encoding and decoding algorithms for Reed–Solomon codes. This application is further explored in the subsequent works of Lin, Al-Naffouri and Han [19] and Lin, Al-Naffouri, Han and Chung [20], while Ben-Sasson et al. [2] utilise the algorithms within their zero-knowledge proof system.

Lin, Al-Naffouri, Han and Chung [20] additionally consider the problem of converting between the LCH basis and the monomial basis. They provide a pair of algorithms that allow polynomials in $\mathbb{F}[x]_{2^n}$ to be converted between the two bases with $\mathcal{O}(2^n n^2)$ additions in $\mathbb{F}$, and $\mathcal{O}(2^n n)$ multiplications in $\mathbb{F}$. Moreover, they provide a second pair of algorithms that allow the conversions to be performed with $\mathcal{O}(2^n n \log n)$ additions and no multiplications in the case that β is a Cantor basis. Their algorithms use ideas introduced by Gao and Mateer [14], and they note that combining their algorithms with the evaluation algorithm of Lin, Chung and Han [21] yields two additive FFTs, one for arbitrary β and one for Cantor bases, that are "algebraically similar" to the two provided by Gao and Mateer. The additive FFT for Cantor bases is applied, with impressive results, to the problem of binary polynomial multiplication in a series of papers [7, 8, 18].

The techniques developed for additive FFTs have yet to be applied to evaluation/interpolation and conversions problems involving the Newton basis associated with their evaluation points. In the realm of multiplicative FFTs, one has the fast algorithms of van der Hoeven and Schost [28], which convert between the monomial basis and the Newton basis associated with the radix-2 truncated Fourier transform points [27, 26]. Fast conversion between the two bases is a necessary requirement of multivariate evaluation and interpolation algorithms [28, 12] and their application to systematic encoding of Reed–Muller and multiplicity codes [12]. For applications in coding theory, characteristic two finite fields are particularly interesting due to their fast arithmetic. However, the algorithms of van der Hoeven and Schost are not suited to such fields as they require the existence of roots of unity with order equal to a power of two. It is likely that this problem may be partially overcome by generalising their algorithm in a manner analogous to the generalisation of the radix-2 truncated Fourier transform [27, 26] to mixed radices by Larrieu [17], so that the algorithms only require the existence of roots of unity with smooth orders. The multivariate evaluation and interpolation algorithms of van der Hoeven and Schost [28] and the author [12] are readily modified to work with the (scaled) Newton basis introduced at the beginning of the section. Thus, using techniques developed for additive FFTs to provide fast algorithms for converting between the Newton and monomial bases yields a complementary solution to the problem.

Our contribution. The evaluation/interpolation and conversion problems considered in this paper all involve the LCH basis. Consequently, we begin in Section 2

by reviewing properties of the basis that are utilised in the development of our algorithms, with proofs provided to give a consistent and self-contained presentation.

In Section 3, we describe algorithms for converting between the Newton and LCH bases. The algorithms follow the divide-and-conquer paradigm that is characteristic of FFTs, with the length ℓ conversion problems for polynomials in $\mathbb{F}[x]_\ell \subseteq \mathbb{F}[x]_{2^n}$ each reduced to two shorter problems of lengths $2^{\lceil \log_2 \ell \rceil - 1}$ and $\ell - 2^{\lceil \log_2 \ell \rceil - 1}$. The algorithms allow conversion in either direction to be performed with at most $(\lfloor \ell/2 \rfloor - 1)\lceil \log_2 \ell \rceil + \mathcal{O}(\log^2 \ell)$ additions in $\mathbb{F}$, and at most $\lfloor \ell/2 \rfloor \lceil \log_2 \ell \rceil - \ell + \mathcal{O}(\log^2 \ell)$ multiplications in $\mathbb{F}$. Here, the big-O terms account for operations that may be performed as precomputations. Moreover, each algorithm requires only $\mathcal{O}(\log^2 \ell)$ field elements to be stored in auxiliary space, i.e., in the space used by the algorithm in addition to the space required to store its inputs or outputs.

In Section 4, we generalise the algorithms of Lin, Chung and Han [21] to allow fast evaluation and interpolation on the LCH basis for polynomials in $\mathbb{F}[x]_\ell \subseteq \mathbb{F}[x]_{2^n}$. The generalisations are obtained by reducing to the case $\ell = 2^n$ while taking advantage of known zero coefficients of the polynomials when written on the basis, in a manner analogous to van der Hoeven's truncated FFT and inverse truncated FFT. Given the coefficients on the LCH basis of a polynomial $f \in \mathbb{F}[x]_\ell \subseteq \mathbb{F}[x]_{2^n}$ and an element of the field λ, the evaluation algorithm returns $f(\omega_0 + \lambda), \ldots, f(\omega_{\ell-1} + \lambda)$. The interpolation algorithm inverts this transformation, returning the coefficients of a polynomial when given its evaluations at the ℓ points and λ. Both algorithms perform at most

$$\min\Big((\ell - 1)(\lceil \log_2 \ell \rceil + 2), \big(2^{\lceil \log_2 \ell \rceil} - 1\big)(\lceil \log_2 \ell \rceil + 1)\Big) + \mathcal{O}(\log^2 \ell)$$

additions in $\mathbb{F}$, and at most

$$\min\left(\frac{\ell - 1}{2}(\lceil \log_2 \ell \rceil + 1), 2^{\lceil \log_2 \ell \rceil - 1}\lceil \log_2 \ell \rceil\right) + \mathcal{O}(\log^2 \ell)$$

multiplications in $\mathbb{F}$, where the big-O terms once again account for operations that may be performed as precomputations. Moreover, each algorithm requires $2^{\lceil \log_2 \ell \rceil} - \ell + \mathcal{O}(\log^2 \ell)$ field elements to be stored in auxiliary space. As the required space may be large when ℓ is slightly larger than a power of two, we describe variants of the algorithms that require only $\mathcal{O}(\log^2 \ell)$ field elements to be stored in auxiliary space, while performing at most $2^{\lceil \log_2 \ell \rceil}$ extra additions in $\mathbb{F}$, and at most $2^{\lceil \log_2 \ell \rceil} - \ell$ extra multiplications in $\mathbb{F}$.

In the final section of the paper, Section 5, we describe algorithms for converting between the LCH and monomial bases. The algorithms assume the existence of a tower of subfields

$$\mathbb{F}_2 = \mathbb{F}_{2^{d_0}} \subset \mathbb{F}_{2^{d_1}} \subset \cdots \subset \mathbb{F}_{2^{d_m}} \subseteq \mathbb{F}$$

such that $d_{m-1} < n \leq d_m$, and $\beta_i / \beta_{d_t \lfloor i/d_t \rfloor} \in \mathbb{F}_{2^{d_t}}$ for $i \in \{0, \ldots, n-1\}$ and $t \in \{0, \ldots, m-1\}$. Then for polynomials in $\mathbb{F}[x]_\ell \subseteq \mathbb{F}[x]_{2^n}$ such that $2^{d_s} < \ell \leq 2^{d_{s+1}}$ for some $s \in \{0, \ldots, m-1\}$, the algorithms allow conversion in either direction to be performed with at most

$$\frac{1}{2}\left\lfloor \frac{\ell}{2} \right\rfloor \left(\lceil \log_2 \ell \rceil \left(\left\lceil \frac{\log_2 \ell}{d_s} \right\rceil - 1 \right) + \sum_{t=0}^{s-1} d_t \left\lceil \frac{\log_2 \ell}{d_t} \right\rceil \left(\frac{d_{t+1}}{d_t} - 1 \right) \right) + \mathcal{O}(\log^2 \ell)$$

additions in $\mathbb{F}$, and at most $(\ell-1)(\lceil\log_2 \ell\rceil+1)+\mathcal{O}(\log^2 \ell)$ multiplications in $\mathbb{F}$. Once again, big-O terms represent operations that may be moved to precomputations, while the algorithms require only $\mathcal{O}(\log \ell)$ field elements to be stored in auxiliary space. Excluding the trivial case of $\mathbb{F}=\mathbb{F}_2$, where the two bases coincide, it is always possible to take $m=1$ and $d_m=[\mathbb{F}:\mathbb{F}_2]$. The algorithms then perform at most $\lfloor \ell/2 \rfloor \binom{\lceil \log_2 \ell \rceil}{2} + \mathcal{O}(\log^2 \ell)$ additions, matching the big-O complexities of existing algorithms. However, the algorithms enjoy a reduction in the number of additions and, to a lesser extent, the number of multiplications they perform if the tower contains more than one subfield of degree less than $\log_2 \ell$. To take advantage of this improvement, we describe a simple method of constructing a vector β that satisfies the necessary criteria when given its desired dimension n and an appropriate tower. We also show how to leverage freedom in the construction in order to eliminate some multiplications from the algorithms in the case that the tower contains some quadratic extensions. In the case that all extensions are quadratic, the construction may be used to eliminate all multiplications from the algorithms. The algorithms also perform at most $\lfloor \ell/2 \rfloor (\lceil \log_2(\ell/2) \rceil \lceil \log_2 \log_2 \max(\ell,2) \rceil + \lceil \log_2 \ell \rceil)/2$ additions in this case, matching the big-O complexity obtained for Cantor bases by existing algorithms.

2. Properties of the Lin–Chung–Han basis

It follows from the definition of the LCH basis associated with β that

$$X_{2^k} = \prod_{i=0}^{2^k-1} \frac{x-\omega_i}{\omega_{2^k}-\omega_i} \quad \text{for } k \in \{0,\dots,n-1\}.$$

Thus, the roots of X_{2^k} form an $\mathbb{F}_2$-linear subspace of $\mathbb{F}$, generated by $\beta_0,\dots,\beta_{k-1}$. As most work on additive transforms pre-dates the introduction of the LCH basis, it is typical in the literature to study the properties of the subspace (vanishing) polynomials associated with these subspaces. We instead choose to study the properties of the basis polynomials $X_{2^0},\dots,X_{2^{n-1}}$, which are simply scalar multiples of the subspace polynomials: the subspace polynomial of a subspace $W \subseteq \mathbb{F}$ is defined to be $\prod_{\omega\in W}(x-\omega)$. Consequently, the properties of the LCH basis presented in the section are either found in [21, 19, 20], or are analogous to properties of subspace polynomials found in [6, 30, 24].

An important property of subspace polynomials, which is inherited by the polynomials $X_{2^0},\dots,X_{2^{n-1}}$, is that they are linearised. A polynomial in $\mathbb{F}[x]$ is $\mathbb{F}_q$-linearised (alternatively, a q-polynomial) if it can be written in the form

$$\sum_{i\in\mathbb{N}} f_i x^{q^i}$$

with $f_0, f_1, \dots \in \mathbb{F}$. The following lemma shows that the polynomials $X_{2^0},\dots,X_{2^{n-1}}$ are $\mathbb{F}_2$-linearised, and establishes several additional properties of the LCH basis that are required for our algorithms.

Lemma 2.1. *The following properties hold for $k \in \{0,\dots,n-1\}$:*

(1) $X_{2^k+j} = X_{2^k} X_j$ *for* $j \in \{0,\dots,2^k-1\}$,
(2) $X_{2^k}(\omega_{2^k i+j}) = i$ *for* $i \in \{0,1\}$ *and* $j \in \{0,\dots,2^k-1\}$,

(3) $X_{2^k} = x/\beta_0$ *if* $k = 0$, *and*

$$X_{2^k} = \frac{X_{2^{k-1}}(x)^2 - X_{2^{k-1}}(x)}{X_{2^{k-1}}(\beta_k)^2 - X_{2^{k-1}}(\beta_k)}$$

otherwise,

(4) X_{2^k} *is* $\mathbb{F}_2$*-linearised,*

(5) $X_{2^k}(x + \lambda) = X_{2^k}(x) + X_{2^k}(\lambda)$ *for* $\lambda \in \mathbb{F}$.

Proof. Let $k \in \{0, \dots, n-1\}$. Then

$$[2^k i]_t = \begin{cases} 0 & \text{if } t < k, \\ [2^k i + j]_t & \text{if } t \geq k, \end{cases} \quad \text{and} \quad [j]_t = \begin{cases} [2^k i + j]_t & \text{if } t < k, \\ 0 & \text{if } t \geq k, \end{cases}$$

for $i \in \{0, 1\}$, $j \in \{0, \dots, 2^k - 1\}$ and $t \in \{0, \dots, n-1\}$. Thus,

$$X_{2^k+j} = \prod_{t=0}^{n-1} \left(\prod_{s=0}^{2^\ell [2^k]_t - 1} \frac{x - \omega_s}{\omega_{2^t} - \omega_s} \right) \left(\prod_{s=0}^{2^\ell [j]_t - 1} \frac{x - \omega_s}{\omega_{2^t} - \omega_s} \right) = X_{2^k} X_j$$

and

$$\omega_{2^k i + j} = \sum_{t=0}^{n-1} [2^k i + j]_t \beta_t = \sum_{t=0}^{n-1} [2^k i]_t \beta_t + \sum_{t=0}^{n-1} [j]_t \beta_t = \omega_{2^k i} + \omega_j$$

for $i \in \{0, 1\}$ and $j \in \{0, \dots, 2^k - 1\}$. As $\{\omega_0, \dots, \omega_{2^k - 1}\}$ forms a group under addition, it follows that

$$X_{2^k}(\omega_{2^k i + j}) = \prod_{t=0}^{2^k - 1} \frac{\omega_{2^k i} + \omega_j - \omega_t}{\omega_{2^k} - \omega_t} = \prod_{t=0}^{2^k - 1} \frac{\omega_{2^k i} - \omega_t}{\omega_{2^k} - \omega_t} = i$$

for $i \in \{0, 1\}$ and $j \in \{0, \dots, 2^k - 1\}$. Therefore, properties (1) and (2) hold.

If $k = 0$, then

$$X_{2^k} = \frac{x - \omega_0}{\omega_1 - \omega_0} = \frac{x - 0}{\beta_0 - 0} = \frac{x}{\beta_0}.$$

If $k \in \{1, \dots, n-1\}$, then property (2) implies that $X_{2^{k-1}}(\omega_i)^2 - X_{2^{k-1}}(\omega_i) = 0$ and $X_{2^k}(\omega_i) = 0$ for $i \in \{0, \dots, 2^k - 1\}$. As X_{2^k} and $X_{2^{k-1}}^2 - X_{2^{k-1}}$ both have degree equal to 2^k, it follows that $X_{2^k} = (X_{2^{k-1}}^2 - X_{2^{k-1}})/\delta$ for some nonzero element $\delta \in \mathbb{F}$. Observing that $X_{2^k}(\beta_k) = X_{2^k}(\omega_{2^k}) = 1$ then shows that $\delta = X_{2^{k-1}}(\beta_k)^2 - X_{2^{k-1}}(\beta_k)$, completing the proof of property (3).

Property (4) follows from property (3) since it shows that X_1 is $\mathbb{F}_2$-linearised, and the recursive formula implies that if $X_{2^{k-1}}$ is $\mathbb{F}_2$-linearised for some $k \in \{1, \dots, n-1\}$, then so too is X_{2^k}. Property (5) follows from property (4) since $\mathbb{F}$ has characteristic equal to two. □

Recall that β is a Cantor basis if $\beta_0 = 1$ and $\beta_{k-1} = \beta_k^2 - \beta_k$ for $k \in \{1, \dots, n-1\}$. The later requirement is contrary to the recursive formula of property (3) of Lemma 2.1, from which follow several additional properties of the LCH basis associated with a Cantor basis.

Corollary 2.2. *The following properties hold if* β *is a Cantor basis:*

(1) $X_{2^k}(\beta_i) = \beta_{i-k}$ *for* $k \in \{0, \dots, n-1\}$ *and* $i \in \{k, \dots, n-1\}$,

(2) $X_{2^k} = \sum_{j=0}^{k-i} \binom{k-i}{j} X_{2^i}^{2^j}$ *for* $k \in \{0, \dots, n-1\}$ *and* $i \in \{0, \dots, k\}$,

(3) *if $d < n$ is a power of two, then*

$$X_{2^{d(k+1)}} = X_{2^{dk}}^{2^d} - X_{2^{dk}}$$

for $k \in \{0, \ldots, \lceil n/d \rceil - 2\}$.

Proof. Suppose that β is a Cantor basis. We prove properties (1) and (2) by induction on k. If $k = 0$, then property (2) holds trivially, while property (1) follows from property (3) of Lemma 2.1 since $\beta_0 = 1$. Suppose now that the two properties hold for some $k \in \{0, \ldots, n-2\}$. Then

$$X_{2^k}(\beta_{k+1})^2 - X_{2^k}(\beta_{k+1}) = \beta_1^2 - \beta_1 = \beta_0 = 1.$$

Consequently, property (3) of Lemma 2.1 implies that $X_{2^{k+1}} = X_{2^k}^2 - X_{2^k}$. Combining with the induction hypothesis, it follows that

$$X_{2^{k+1}}(\beta_i) = X_{2^k}(\beta_i)^2 - X_{2^k}(\beta_i) = \beta_{i-k}^2 - \beta_{i-k} = \beta_{i-k-1}$$

for $i \in \{k+1, \ldots, n-1\}$, and

$$X_{2^{k+1}} = X_{2^i} + \sum_{j=1}^{k+1-i} \left(\binom{k-i}{j} + \binom{k-i}{j-1} \right) X_{2^i}^{2^j} = \sum_{j=0}^{k+1-i} \binom{k+1-i}{j} X_{2^i}^{2^j},$$

for $i \in \{0, \ldots, k+1\}$. Hence, properties (1) and (2) follow by induction.

Lucas' lemma [22, p. 230] (see also [13]) implies that

$$\binom{i}{j} \equiv \prod_{k \in \mathbb{N}} \binom{[i]_k}{[j]_k} \pmod 2 \quad \text{for } i, j \in \mathbb{N}.$$

Thus, if $d < n$ is a power of two, then property (2) and Lucas' lemma imply that

$$X_{2^{d(k+1)}} = \sum_{j=0}^{d} \binom{d}{j} X_{2^{dk}}^{2^j} = X_{2^{dk}}^{2^d} - X_{2^{dk}}$$

for $k \in \{0, \ldots, \lceil n/d \rceil - 2\}$. Therefore, property (3) holds. □

3. Conversion between the Newton and LCH bases

We base our algorithms for converting between the Newton and LCH bases on the following lemma.

Lemma 3.1. *Let $\ell \in \{2, \ldots, 2^n\}$ and $k = \lceil \log_2 \ell \rceil - 1$. Then*

$$\sum_{i=0}^{\ell-1} f_i N_i(x + \lambda) = \sum_{i=0}^{\ell-1} h_i X_i(x) \tag{3.1}$$

for $f_0, \ldots, f_{\ell-1}, \lambda, h_0, \ldots, h_{\ell-1} \in \mathbb{F}$ if and only if

$$\sum_{i=0}^{2^k-1} f_i N_i(x + \lambda) = \sum_{i=0}^{2^k-1} h_i X_i(x) + X_{2^k}(\lambda) \sum_{i=0}^{\ell-2^k-1} h_{2^k+i} X_i(x) \tag{3.2}$$

and

$$\sum_{i=0}^{\ell-2^k-1} f_{2^k+i} N_i(x + \lambda + \beta_k) = \sum_{i=0}^{\ell-2^k-1} h_{2^k+i} X_i(x). \tag{3.3}$$

Proof. Let $k \in \{0, \ldots, n-1\}$ and $i \in \{0, \ldots, 2^k - 1\}$. Then it follows from the definition of the Newton basis polynomials that $N_{2^k+i}(\omega_j) = 0$ for $j \in \{0, \ldots, 2^k + i - 1\}$, and $N_{2^k+i}(\omega_{2^k+i}) = 1$. By combining the definition with property (2) of Lemma 2.1, we also have $X_{2^k}(\omega_j)N_i(\omega_j + \beta_k) = 0$ for $j \in \{0, \ldots, 2^k - 1\}$, $X_{2^k}(\omega_{2^k+j})N_i(\omega_{2^k+j} + \beta_k) = N_i(\omega_j) = 0$ for $j \in \{0, \ldots, i-1\}$, and $X_{2^k}(\omega_{2^k+i})N_i(\omega_{2^k+i} + \beta_k) = N_i(\omega_i) = 1$. It follows that $N_{2^k+i}(x)$ and $X_{2^k}(x)N_i(x+\beta_k)$ are equal, since they agree on 2^k+i+1 distinct values and each have degree equal to $2^k + i$.

Suppose now that $\ell \in \{2, \ldots, 2^n\}$ and $k = \lceil \log_2 \ell \rceil - 1$. Then $k \in \{0, \ldots, n-1\}$ and $\ell - 2^k \leq 2^{k+1} - 2^k = 2^k$. Thus, for $i \in \{0, \ldots, \ell - 2^k - 1\}$ and $\lambda \in \mathbb{F}$, we have just shown that

$$N_{2^k+i}(x + \lambda) = X_{2^k}(x + \lambda)N_i(x + \lambda + \beta_k),$$

while properties (1) and (5) of Lemma 2.1 imply that

$$X_{2^k+i} = X_{2^k}(x + \lambda + \lambda)X_i(x) = X_{2^k}(x + \lambda)X_i(x) + X_{2^k}(\lambda)X_i(x).$$

It follows that for $f_0, \ldots, f_{\ell-1}, \lambda, h_0, \ldots, h_{\ell-1} \in \mathbb{F}$, the polynomials on either side of equation (3.2), which each have degree less than $\max(\ell - 2^k, 2^k) = 2^k$, are equal to the remainder upon division by $X_{2^k}(x + \lambda)$ of the polynomial on their respective side of equation (3.1). Similarly, the polynomials on either side of equation (3.3) are equal to the quotient upon division by $X_{2^k}(x + \lambda)$ of the polynomial on their respective side of equation (3.1). Uniqueness of the quotient and remainder therefore implies that (3.1) holds for $f_0, \ldots, f_{\ell-1}, \lambda, h_0, \ldots, h_{\ell-1} \in \mathbb{F}$ if and only if (3.2) and (3.3) hold. □

Lemma 3.1 suggests recursive algorithms for converting polynomials in $\mathbb{F}[x]_\ell \subseteq \mathbb{F}[x]_{2^n}$ between the LCH basis and the basis of shifted Newton polynomials $\{N_i(x + \lambda) \mid i \in \{0, \ldots, 2^n - 1\}\}$ for a given shift parameter $\lambda \in \mathbb{F}$. Given the coefficients f_i on the left-hand side of (3.1), recursive calls can be made on the polynomials (3.2) and (3.3) to compute their coefficients on the LCH basis. These coefficients can in turn be used to compute the coefficients h_i on the right-hand side of (3.1) by performing $\ell - 2^k$ additions, and $\ell - 2^k$ multiplications by $X_{2^k}(\lambda)$. For the inverse conversion, where we start with the coefficients h_i on the right-hand side of (3.1), performing the same additions and multiplications yields the coefficients of the polynomials (3.2) and (3.3) on the LCH basis. Then recursive calls can be made to obtain their coefficients on the shifted Newton bases $\{N_i(x+\lambda) \mid i \in \{0, \ldots, 2^n - 1\}\}$ and $\{N_i(x + \lambda + \beta_k) \mid i \in \{0, \ldots, 2^n - 1\}\}$, respectively, and thus the coefficients f_i on the left-hand side of (3.1). For conversions in both directions, the initial shift parameter λ is augmented by the addition of β_k for the recursive call made on the polynomial (3.3), which necessitates the inclusion of the shift parameter in the conversion problem.

To efficiently compute the elements $X_{2^k}(\lambda)$ by which we multiply during the algorithms, we take advantage of the property (5) of Lemma 2.1 and the observation that the initial shift parameter is only augmented for the recursive calls by the addition of entries of β. To this end, we assume that $X_{2^i}(\beta_j)$ for $0 \leq i < j < \lceil \log_2 \ell \rceil$ have been precomputed. The pseudocode for the conversion from a shifted Newton basis to the LCH basis is presented in Algorithm 1, while the pseudocode for the inverse conversion is presented in Algorithm 2. Each algorithm operates on a vector $(a_i)_{0 \leq i < \ell}$ of field elements that initially contains the coefficients of a polynomial on

the input basis, and has it entries overwritten by the algorithm with the coefficients of the polynomial on the output basis.

Algorithm 1 $\mathsf{NewtonToLCH}(\ell, (X_{2^i}(\lambda))_{0\le i<\lceil \log_2 \ell\rceil}, (a_i)_{0\le i<\ell})$

Input: an integer $\ell \in \{1,\dots,2^n\}$; the vector $(X_{2^i}(\lambda))_{0\le i<\lceil \log_2 \ell\rceil}$ for some $\lambda \in \mathbb{F}$; and the vector $(a_i)_{0\le i<\ell}$ such that $a_i = f_i \in \mathbb{F}$ for $i \in \{0,\dots,\ell-1\}$.
Output: $a_i = h_i \in \mathbb{F}$ for $i \in \{0,\dots,\ell-1\}$ such that (3.1) holds.
1: **if** $\ell = 1$ **then return**
2: $k \leftarrow \lceil \log_2 \ell\rceil - 1$, $\ell' \leftarrow \ell - 2^k$, $k' \leftarrow \lceil \log_2 \ell'\rceil$
3: $\mathsf{NewtonToLCH}(2^k, (X_{2^i}(\lambda))_{0\le i<k}, (a_i)_{0\le i<2^k})$
4: $\mathsf{NewtonToLCH}(\ell', (X_{2^i}(\lambda) + X_{2^i}(\beta_k))_{0\le i<k'}, (a_{2^k+i})_{0\le i<\ell'})$
5: **for** $i = 0,\dots,\ell'-1$ **do**
6: $\quad a_i \leftarrow a_i + X_{2^k}(\lambda) a_{2^k+i}$

Algorithm 2 $\mathsf{LCHToNewton}(\ell, (X_{2^i}(\lambda))_{0\le i<\lceil \log_2 \ell\rceil}, (a_i)_{0\le i<\ell})$

Input: an integer $\ell \in \{1,\dots,2^n\}$; the vector $(X_{2^i}(\lambda))_{0\le i<\lceil \log_2 \ell\rceil}$ for some $\lambda \in \mathbb{F}$; and the vector $(a_i)_{0\le i<\ell}$ such that $a_i = h_i \in \mathbb{F}$ for $i \in \{0,\dots,\ell-1\}$.
Output: $a_i = f_i \in \mathbb{F}$ for $i \in \{0,\dots,\ell-1\}$ such that (3.1) holds.
1: **if** $\ell = 1$ **then return**
2: $k \leftarrow \lceil \log_2 \ell\rceil - 1$, $\ell' \leftarrow \ell - 2^k$, $k' \leftarrow \lceil \log_2 \ell'\rceil$
3: **for** $i = 0,\dots,\ell'-1$ **do**
4: $\quad a_i \leftarrow a_i + X_{2^k}(\lambda) a_{2^k+i}$
5: $\mathsf{LCHToNewton}(2^k, (X_{2^i}(\lambda))_{0\le i<k}, (a_i)_{0\le i<2^k})$
6: $\mathsf{LCHToNewton}(\ell', (X_{2^i}(\lambda) + X_{2^i}(\beta_k))_{0\le i<k'}, (a_{2^k+i})_{0\le i<\ell'})$

Theorem 3.2. *Algorithms 1 and 2 are correct.*

Proof. We prove correctness for Algorithm 1 by induction on ℓ. The proof of correctness for Algorithm 2 uses similar arguments, and is omitted here. Alternatively, one may note that the transformation performed on the vector $(a_i)_{0\le i<\ell}$ by the for-loop in Algorithm 1 is an involution. Algorithm 2 reverses the order of these transformations, thus performing the inverse transformation to Algorithm 1 overall.

Algorithm 1 is correct for all inputs with $\ell = 1$, since $X_0 = N_0 = 1$. Therefore, suppose that for some $\ell \in \{2,\dots,2^n\}$, the algorithm produces the correct output for all inputs with smaller values of ℓ. Moreover, suppose that the algorithm is given ℓ as an input, together with $(X_{2^i}(\lambda))_{0\le i<\lceil \log_2 \ell\rceil}$ for some $\lambda \in \mathbb{F}$, and the vector $(a_i)_{0\le i<\ell}$ with $a_i = f_i \in \mathbb{F}$ for $i \in \{0,\dots,\ell-1\}$. Let $k = \lceil \log_2 \ell\rceil - 1$, $\ell' = \ell - 2^k$ and $k' = \lceil \log_2 \ell'\rceil$, as computed in Line 2 of the algorithm, and $h_0,\dots,h_{\ell-1} \in \mathbb{F}$ be the unique elements that satisfy (3.1). Then Lemma 3.1 implies that (3.2) and (3.3) both hold. Therefore, as $2^k < \ell$, (3.2) and the induction hypothesis imply that Line 3 sets $a_i = h_i + h_{2^k+i} X_{2^k}(\lambda)$ for $i \in \{0,\dots,\ell'-1\}$, and $a_i = h_i$ for $i \in \{\ell',\dots,2^k-1\}$. Property (5) of Lemma 2.1 implies that $(X_{2^i}(\lambda)+X_{2^i}(\beta_k))_{0\le i<k'} = (X_{2^i}(\lambda+\beta_k))_{0\le i<k'}$. Consequently, as $\ell' < \ell$, (3.3) and the induction hypothesis imply that Line 4 sets $a_i = h_i$ for $i \in \{2^k,\dots,\ell-1\}$. It follows that Lines 5 and 6 set $a_i = (h_i + h_{2^k+i} X_{2^k}(\lambda)) + h_{2^k+i} X_{2^k}(\lambda) = h_i$ for $i \in \{0,\dots,\ell'-1\}$. Thus, the algorithm terminates with $a_i = h_i$ for $i \in \{0,\dots,\ell-1\}$, as required. □

When implementing Algorithms 1 and 2, subvectors of the input vector $(a_i)_{0\leq i<\ell}$ may be represented by a pointer to their first entry, rather than by replicating them in memory. Moreover, if $(X_{2^i}(\lambda))_{0\leq i<\lceil\log_2\ell\rceil}$ and $(X_{2^i}(\lambda)+X_{2^i}(\beta_k))_{0\leq i<k'}$ are always cleared from memory when the algorithm returns, then storing the vectors and those of any subsequent recursive call requires only $\mathcal{O}(\log^2\ell)$ fields elements to be stored in auxiliary space at all times. It follows that Algorithms 1 and 2 can be implemented so that only $\mathcal{O}(\log^2\ell)$ field elements are stored in auxiliary space when they are used to convert polynomials in $\mathbb{F}[x]_\ell$ between the Newton and LCH bases.

The recurrence relation of property (3) of Lemma 2.1 allows $X_{2^i}(\beta_j)$ for $0\leq i<j<\lceil\log_2\ell\rceil$ to be computed with $\mathcal{O}(\log^2\ell)$ additions and multiplications in $\mathbb{F}$. Subsequently, the recurrence relation may again be used to compute $(X_{2^i}(\lambda))_{0\leq i<\lceil\log_2\ell\rceil}$ for a desired $\lambda\in\mathbb{F}$ with a further $\mathcal{O}(\log\ell)$ additions and multiplications, where the implied constant is smaller if the previously computed denominators of the recurrence relation have been stored. In the case that $\lambda=0$, the vector simply contains all zeros. It follows that when Algorithms 1 and 2 are used to convert polynomials in $\mathbb{F}[x]_\ell$ between the Newton and LCH bases, all precomputations for algorithms can be performed with $\mathcal{O}(\log^2\ell)$ operations in $\mathbb{F}$. The number of operations then performed by the algorithms themselves is bounded by the following theorem.

Theorem 3.3. *Algorithms 1 and 2 perform at most* $(\lfloor\ell/2\rfloor-1)\lceil\log_2\ell\rceil+\ell-1$ *additions in* $\mathbb{F}$*, and at most* $\lfloor\ell/2\rfloor\lceil\log_2\ell\rceil$ *multiplications in* $\mathbb{F}$*.*

Proof. We prove the bounds for Algorithm 1 only, since it is clear that the two algorithms perform the same number of operations when given identical values of ℓ. If $\ell=1$, then Algorithm 1 performs no additions or multiplications, matching the bounds of the theorem. Proceeding by induction, suppose that the algorithm is called with $\ell\in\{2,\dots,2^n\}$, and that the two bounds hold for all smaller values of ℓ. Let $k=\lceil\log_2\ell\rceil-1$, $\ell'=\ell-2^k$ and $k'=\lceil\log_2\ell'\rceil$, as computed in Line 2 of the algorithm. Then, as $2^k<\ell$, the induction hypothesis implies that Line 3 performs at most $(2^{k-1}-1)k+2^k-1$ additions and at most $2^{k-1}k$ multiplications. The computation of the vector $(X_{2^i}(\lambda)+X_{2^i}(\beta_k))_{0\leq i<k'}$ in Line 4 requires k' additions, where $k'\leq k$ since $\ell'=\ell-2^k\leq 2^{k+1}-2^k=2^k$. Thus, as $\ell'<\ell$, the induction hypothesis implies that Line 4 performs at most $\lfloor\ell'/2\rfloor k'+\ell'-1\leq(\lfloor\ell/2\rfloor-2^{k-1})k+\ell-2^k-1$ additions and at most $\lfloor\ell'/2\rfloor k'\leq(\lfloor\ell/2\rfloor-2^{k-1})k$ multiplications. Lines 5 and 6 perform $\ell'=\ell-\lceil 2^{k+1}/2\rceil\leq\ell-\lceil\ell/2\rceil=\lfloor\ell/2\rfloor$ additions and multiplications. Summing these bounds, it follows that the algorithms performs at most $(\lfloor\ell/2\rfloor-1)(k+1)+\ell-1=(\lfloor\ell/2\rfloor-1)\lceil\log_2\ell\rceil+\ell-1$ additions, and at most $\lfloor\ell/2\rfloor(k+1)=\lfloor\ell/2\rfloor\lceil\log_2\ell\rceil$ multiplications. □

As noted above, the input $(X_{2^i}(\lambda))_{0\leq i<\lceil\log_2\ell\rceil}$ of Algorithm 1 contains only zeros if $\lambda=0$. In this case, the vector $(X_{2^i}(\lambda))_{0\leq i<k}$ that is passed to the recursive call in Line 3 of the algorithm also contains only zeros, while its counterpart $(X_{2^i}(\lambda)+X_{2^i}(\beta_k))_{0\leq i<k'}$ in Line 4 contains only precomputed elements. Moreover, Lines 5 and 6 of the algorithm leave the vector $(a_i)_{0\leq i<\ell}$ unchanged. The algorithm is readily modified to avoid performing unnecessary operations in this case, e.g., by the inclusion of an additional parameter that indicates whether $\lambda=0$, saving at least $\ell-1+\binom{\lfloor\log_2\ell\rfloor}{2}$ additions and at least $\ell-1$ multiplications overall. Similar modifications to Algorithm 2 yield the same savings, leading to the following corollary of Theorem 3.3.

Corollary 3.4. *For polynomials in $\mathbb{F}[x]_\ell \subseteq \mathbb{F}[x]_{2^n}$, conversion between the Newton and LCH bases can be performed with $(\lfloor \ell/2 \rfloor - 1)\lceil \log_2 \ell \rceil + \mathcal{O}(\log^2 \ell)$ additions in $\mathbb{F}$, and $\lfloor \ell/2 \rfloor \lceil \log_2 \ell \rceil - \ell + \mathcal{O}(\log^2 \ell)$ multiplications in $\mathbb{F}$.*

Remark 3.5. If β is a Cantor basis, then property (5) of Lemma 2.1 and property (1) of Corollary 2.2 imply that $X_{2^k}(\omega_i) = \omega_{\lfloor i/2^k \rfloor}$ for $k \in \{0, \ldots, n-1\}$ and $i \in \{0, \ldots, 2^n - 1\}$. It follows in this case that if the elements of the field are represented with respect to an extension of β to a basis of $\mathbb{F}/\mathbb{F}_2$, and Algorithm 1 or 2 is initially called with $\lambda = \omega_i$ for some $i \in \{0, \ldots, 2^n - 1\}$, as is the case when λ is zero, then only simple index arithmetic (e.g., bit shifts) is required to compute $X_{2^k}(\lambda)$. The same will hold for all recursive calls made by the algorithm, since the initial shift parameter is only augmented by the addition of entries from β for these calls. Thus, it is only necessary to provide the algorithm with λ rather than the whole vector $(X_{2^i}(\lambda))_{0 \leq i < \lceil \log_2 \ell \rceil}$ in this case, simplifying the algorithm. Similar modifications may be made to the algorithms of the next section.

4. Evaluation and interpolation on the LCH basis

Lin, Chung and Han [21] propose quasi-linear time algorithms for evaluation and interpolation with respect to the LCH basis for polynomials in $\mathbb{F}[x]_{2^n}$. In this section, we generalise their algorithms to take advantage of known zero coefficients of polynomials in $\mathbb{F}[x]_\ell$ for $\ell < 2^n$ when written on the basis, leading to algorithms with lower complexities. The generalisations are based on the following lemma, for which the case $\ell = 2^{k+1}$ is due to Lin, Chung and Han.

Lemma 4.1. *Let $\ell \in \{1, \ldots, 2^n\}$, $k \in \{0, \ldots, n-1\}$ such that $k \geq \lceil \log_2 \ell \rceil - 1$, and*

$$f = \sum_{i=0}^{\ell-1} h_i X_i \tag{4.1}$$

for $h_0, \ldots, h_{\ell-1} \in \mathbb{F}$. Then, for $t \in \{0, 1\}$, the polynomial

$$f_t = \sum_{i=0}^{\min(\ell, 2^k)-1} h_i X_i + (t + X_{2^k}(\lambda)) \sum_{i=0}^{\max(\ell - 2^k, 0)-1} h_{2^k+i} X_i \tag{4.2}$$

satisfies $f_t(\omega_s + \lambda + t\beta_k) = f(\omega_{2^k t + s} + \lambda)$ for $s \in \{0, \ldots, 2^k - 1\}$.

Proof. Let $\ell \in \{1, \ldots, 2^n\}$, $k \in \{0, \ldots, n-1\}$ such that $k \geq \lceil \log_2 \ell \rceil - 1$. Then it follows from the definition of the ω_i that $X_i(\omega_{2^k t+s} + \lambda) = X_i(\omega_s + \lambda + t\beta_k)$ for $i \in \{0, \ldots, \min(\ell, 2^k) - 1\}$, $t \in \{0, 1\}$ and $s \in \{0, \ldots, 2^k - 1\}$. The choice of k implies that $\ell - 2^k \leq 2^k$. Thus, properties (1), (2) and (5) of Lemma 2.1 imply that

$$\begin{aligned} X_{2^k+i}(\omega_{2^k t+s} + \lambda) &= X_{2^k}(\omega_{2^k t+s} + \lambda) X_i(\omega_{2^k t+s} + \lambda) \\ &= (X_{2^k}(\omega_{2^k t+s}) + X_{2^k}(\lambda)) X_i(\omega_s + \lambda + t\beta_k) \\ &= (t + X_{2^k}(\lambda)) X_i(\omega_s + \lambda + t\beta_k) \end{aligned}$$

for $i \in \{0, \ldots, \max(\ell - 2^k, 0) - 1\}$, $t \in \{0, 1\}$ and $s \in \{0, \ldots, 2^k - 1\}$. Therefore, the lemma holds if $f = X_i$ for some $i \in \{0, \ldots, \ell - 1\}$, and, thus, for all f of the form (4.1) by linearity. □

We use Lemma 4.1 to provide fast evaluation and interpolation algorithms with respect to the LCH basis for polynomials in $\mathbb{F}[x]_\ell \subseteq \mathbb{F}[x]_{2^n}$ and evaluations points $\omega_0 + \lambda, \ldots, \omega_{\ell-1} + \lambda$ for some $\lambda \in \mathbb{F}$. At a first glance, the lemma suggests reductions

for $\ell > 1$ to shorter instances of the problems corresponding to the polynomials f_0 and f_1 defined by (4.2) with $k = \lceil \log \ell \rceil - 1$. However, difficulties arise when ℓ is not a power of two, since f_0 and f_1 may each have degree equal to $2^k - 1$, while only $\ell < 2^{k+1}$ evaluations of f are either given or are required to be computed. We address these difficulties by generalising the problems, following the approach used in the development of truncated fast Fourier transforms.

4.1. **Evaluation on the LCH basis.** Given the coefficients of a polynomial $f \in \mathbb{F}[x]_\ell \subseteq \mathbb{F}[x]_{2^n}$ on the LCH basis, Lemma 4.1 suggests a recursive algorithm for evaluating the polynomial at $c \in \{1, \ldots, 2^n\}$ evaluation points of the form $\omega_0 + \lambda, \ldots, \omega_{c-1} + \lambda$ for some $\lambda \in \mathbb{F}$. Letting $k = \lceil \log_2 \max(\ell, c) \rceil - 1$ and taking f_0 and f_1 to be the two polynomials defined by (4.2), the lemma allows the problem to be reduced to recursively evaluating f_0 at the points $\omega_0 + \lambda, \ldots, \omega_{\min(c,2^k)-1} + \lambda$ and, if $c > 2^k$, evaluating f_1 at the points $\omega_0 + (\lambda + \beta_k), \ldots, \omega_{c-2^k-1} + (\lambda + \beta_k)$. The coefficients of f_0 on the LCH basis can be computed with $\max(\ell - 2^k, 0)$ additions and multiplications by $X_{2^k}(\lambda)$. Then, if needed, the coefficients of f_1 can be computed with a further $\max(\ell - 2^k, 0)$ additions. In particular, if $c > 2^{\lceil \log_2 \ell \rceil}$, then no field operations are required to compute the coefficients of f_0 and f_1 as they are both equal to f.

The pseudocode for the evaluation algorithm is presented in Algorithm 3. We use the techniques introduced in Section 3 to compute the values $X_{2^k}(\lambda)$ required by the algorithm. Consequently, we once again assume that $X_{2^i}(\beta_j)$ for $0 \leq i < j < \lceil \log_2 \ell \rceil$ have been precomputed for the input value of ℓ. The algorithm operates on a vector $(a_i)_{0 \leq i < \max(\ell, 2^{\lceil \log_2 c \rceil})}$ of field elements that initially contains the coefficients of some polynomial $f \in \mathbb{F}[x]_\ell$ on the LCH basis, and overwrites its first c entries with $f(\omega_0 + \lambda), \ldots, f(\omega_{c-1} + \lambda)$ for a given $\lambda \in \mathbb{F}$. The entries of the vector that are unspecified by the input requirements of algorithm can be thought of as initially containing zeros. However, this is not assumed by the algorithm. While only $\max(\ell, c)$ entries are needed to store the inputs and outputs of the algorithm, a longer vector is used for certain parameters so that there is space to simultaneously store the coefficients of the polynomials f_0 and f_1. In Section 4.3, we show how to avoid using additional space when $c \leq \ell$, but at the cost of a higher complexity.

Theorem 4.2. *Algorithm 3 is correct.*

Proof. We prove the theorem by induction on $\lceil \log_2 \max(\ell, c) \rceil$. If $\lceil \log_2 \max(\ell, c) \rceil = 0$, then $\ell = c = 1$ and the algorithm produces the correct output since the polynomial f defined in (4.1) is the constant polynomial $h_0 X_0 = h_0$. Therefore, for some $k \in \{0, \ldots, n-1\}$, suppose that the algorithm produces the correct output for all inputs with $\lceil \log_2 \max(\ell, c) \rceil \leq k$. Moreover, suppose that the algorithm is called with inputs $\ell, c \in \{1, \ldots, 2^n\}$ such that $\lceil \log_2 \max(\ell, c) \rceil = k + 1$, the vector $(X_{2^i}(\lambda))_{0 \leq i < \lceil \log_2 \ell \rceil}$ for some $\lambda \in \mathbb{F}$, and the vector $(a_i)_{0 \leq i < \max(\ell, 2^{\lceil \log_2 c \rceil})}$ such that $a_i = h_i \in \mathbb{F}$ for $i \in \{0, \ldots, \ell - 1\}$. Let $f \in \mathbb{F}[x]_\ell$ be the polynomial defined in (4.1), and f_t be the polynomial defined in (4.2) for $t \in \{0, 1\}$. Let $\ell' = \min(\ell, 2^k)$, $\ell'' = \ell - \ell'$, $c_0 = \min(c, 2^k)$ and $c_1 = c - c_0$, as computed in Line 2 of the algorithm. Then $f_0, f_1 \in \mathbb{F}[x]_{\ell'}$, $2^k + \ell'' = \max(\ell, 2^k) \leq \max(\ell, \max(\ell, c)) = \max(\ell, c)$, and if $c_1 > 0$, then $2^k + \ell' \leq 2^{k+1} = 2^{\lceil \log_2 c \rceil}$. The inequalities imply that the entries of $(a_i)_{0 \leq i < \max(\ell, 2^{\lceil \log_2 c \rceil})}$ accessed by Lines 3 to 9 of the algorithm all have indices less than $\max(\ell, 2^{\lceil \log_2 c \rceil})$. We also note that the subvector $(a_i)_{0 \leq i < 2^k}$ in Line 11 has the correct length since $2^k \geq \max(\ell', 2^{\lceil \log_2 c_0 \rceil}) \geq \max(\ell', c_0) = 2^k$.

Algorithm 3 $\mathsf{LCHEval}(\ell, c, (X_{2^i}(\lambda))_{0\le i<\lceil\log_2 \ell\rceil}, (a_i)_{0\le i<\max(\ell,2^{\lceil\log_2 c\rceil})})$

Input: integers $\ell, c \in \{1,\dots,2^n\}$; the vector $(X_{2^i}(\lambda))_{0\le i<\lceil\log_2 \ell\rceil}$ for some $\lambda \in \mathbb{F}$; and the vector $(a_i)_{0\le i<\max(\ell,2^{\lceil\log_2 c\rceil})}$ such that $a_i = h_i \in \mathbb{F}$ for $i \in \{0,\dots,\ell-1\}$.
Output: $a_i = f(\omega_i + \lambda)$ for $i \in \{0,\dots,c-1\}$, where f is the polynomial in (4.1).

1: **if** $\ell = 1$ **and** $c = 1$ **then return**
2: $k \leftarrow \lceil\log_2 \max(\ell, c)\rceil - 1$, $\ell' \leftarrow \min(\ell, 2^k)$, $\ell'' \leftarrow \ell - \ell'$, $c_0 \leftarrow \min(c, 2^k)$, $c_1 \leftarrow c - c_0$
3: **for** $i = 0,\dots,\ell''-1$ **do**
4: $\quad a_i \leftarrow a_i + X_{2^k}(\lambda) a_{2^k+i}$
5: **if** $c_1 > 0$ **then**
6: $\quad$ **for** $i = 0,\dots,\ell''-1$ **do**
7: $\quad\quad a_{2^k+i} \leftarrow a_i + a_{2^k+i}$
8: $\quad$ **for** $i = \ell'',\dots,\ell'-1$ **do**
9: $\quad\quad a_{2^k+i} \leftarrow a_i$
10: $\quad \mathsf{LCHEval}(\ell', c_1, (X_{2^i}(\lambda) + X_{2^i}(\beta_k))_{0\le i<\lceil\log_2 \ell'\rceil}, (a_{2^k+i})_{0\le i<\max(\ell',2^{\lceil\log_2 c_1\rceil})})$
11: $\mathsf{LCHEval}(\ell', c_0, (X_{2^i}(\lambda))_{0\le i<\lceil\log_2 \ell'\rceil}, (a_i)_{0\le i<2^k})$

Lines 3 and 4 of the algorithm set $a_i = h_i + X_{2^k}(\lambda) h_{2^k+i}$ for $i \in \{0,\dots,\ell''-1\}$. As $\ell'' = \max(\ell - 2^k, 0)$, the subvector $(a_i)_{0\le i<\ell'}$ subsequently contains the coefficients of f_0 on the LCH basis. If $c_1 > 0$, then Lines 6 to 9 set $a_{2^k+i} = h_{2^k+i} + (h_i + X_{2^k}(\lambda) h_{2^k+i}) = h_i + (1 + X_{2^k}(\lambda)) h_{2^k+i}$ for $i \in \{0,\dots,\ell''-1\}$, and $a_{2^k+i} = h_i$ for $i \in \{\ell'',\dots,\ell'-1\}$. Thus, the subvector $(a_{2^k+i})_{0\le i<\ell'}$ subsequently contains the coefficients of f_1 on the LCH basis. We have $f_0, f_1 \in \mathbb{F}[x]_{\ell'}$, $\lceil\log_2 \max(\ell', c_0)\rceil = k$ and $\lceil\log_2 \max(\ell', c_1)\rceil \le \lceil\log_2 \max(2^k, 2^{k+1} - 2^k)\rceil = k$. Therefore, the induction hypothesis and Lemma 4.1 imply that Line 11 sets $a_i = f_0(\omega_i + \lambda) = f(\omega_i + \lambda)$ for $i \in \{0,\dots,c_0-1\}$, and if $c_1 > 0$, then Line 10 sets $a_{2^k+i} = f_1(\omega_i + \lambda + \beta_k) = f(\omega_{2^k+i} + \lambda)$ for $i \in \{0,\dots,c_1-1\}$. Hence, the algorithm terminates with $a_i = f(\omega_i + \lambda)$ for $i \in \{0,\dots,c_0-1\} \cup \{2^k,\dots,2^k + c_1 - 1\} = \{0,\dots,c-1\}$, as required. □

As noted in Section 3, the precomputation of $X_{2^i}(\beta_j)$ for $0 \le i < j < \lceil\log_2 \ell\rceil$ can be performed with $\mathcal{O}(\log^2 \ell)$ field operations. Moreover, Algorithm 3 can be implemented so that only $\max(2^{\lceil\log_2 c\rceil} - \max(\ell, c), 0) + \mathcal{O}(\log^2 \ell)$ field elements are stored in auxiliary space. The number of field operations performed by the algorithm itself is bounded by the following theorem.

Theorem 4.3. *Algorithm 3 performs at most*

$$\min\big((c-1)(\lceil\log_2 \min(\ell, c)\rceil + 1) + \ell - 1, \big(2^{k+1} - 1\big)(\lceil\log_2 \ell\rceil + 1)\big)$$

additions in $\mathbb{F}$*, where* $k = \lceil\log_2 \max(\ell, c)\rceil - 1$*, and at most*

$$\min\left(\frac{c-1}{2}\lceil\log_2 \min(\ell, c/2)\rceil + \ell - 1, 2^k\lceil\log_2 \ell\rceil\right)$$

multiplications in $\mathbb{F}$*.*

Proof. We prove the theorem by induction on $\lceil\log_2 \max(\ell, c)\rceil$. If $\lceil\log_2 \max(\ell, c)\rceil = 0$, then $\ell = c = 1$ and Algorithm 3 performs no additions or multiplications, matching the bounds of the theorem. Therefore, for some $k \in \{0,\dots,n-1\}$, suppose that the bounds stated in the theorem hold for all inputs with $\lceil\log_2 \max(\ell, c)\rceil \le k$. Moreover, suppose that the algorithm is called with inputs $\ell, c \in \{1,\dots,2^n\}$ such that $\lceil\log_2 \max(\ell, c)\rceil = k + 1$. Let $\ell' = \min(\ell, 2^k)$, $\ell'' = \ell - \ell'$, $c_0 = \min(c, 2^k)$ and

$c_1 = c - c_0$, as computed in Line 2 of the algorithm. Then, as shown in the proof of Theorem 4.2, $\lceil \log_2 \max(\ell', c_i) \rceil \leq k$ for $i \in \{0, 1\}$, allowing the induction hypothesis to be used to bound the number of additions and multiplications performed by Lines 10 and 11. We begin by proving the bounds $(2^{k+1} - 1)(\lceil \log_2 \ell \rceil + 1)$ and $2^k \lceil \log_2 \ell \rceil$ on the number of additions and multiplications performed by the algorithm.

Lines 3 to 9 of Algorithm 3 perform at most $2\ell''$ additions, and at most ℓ'' multiplications. The induction hypothesis implies that Line 11 performs at most $(2^k - 1)(\lceil \log \ell' \rceil + 1)$ additions, and at most $2^{k-1} \lceil \log_2 \ell' \rceil$ multiplications. Similarly, Line 10 performs at most $(2^k - 1)(\lceil \log \ell' \rceil + 1) + \lceil \log_2 \ell' \rceil$ additions, and at most $2^{k-1} \lceil \log_2 \ell' \rceil$ multiplications. Summing these bounds, it follows that Algorithm 3 performs at most $(2^{k+1} - 1)(\lceil \log_2 \ell' \rceil + 1) + 2\ell'' - 1$ additions, and at most $2^k \lceil \log_2 \ell' \rceil + \ell''$ multiplications. If $\ell \leq 2^k$, then $\ell' = \ell$ and $\ell'' = 0$. If $\ell > 2^k$, then $\lceil \log_2 \ell' \rceil = k = \lceil \log_2 \ell \rceil - 1$ and $\ell'' = \ell - 2^k \leq 2^k$. In either case, we find that Algorithm 3 performs at most $(2^{k+1} - 1)(\lceil \log_2 \ell \rceil + 1)$ additions, and at most $2^k \lceil \log_2 \ell \rceil$ multiplications.

To prove the remaining bounds of the theorem, we consider the cases $c_1 = 0$ and $c_1 > 0$ separately. Therefore, suppose for now that $c_1 = 0$. Then $c \leq 2^k < \ell \leq 2^{k+1}$, $\ell' = 2^k$, $c_0 = c$, $\lceil \log_2 \min(\ell', c_0) \rceil = \lceil \log_2 \min(\ell, c) \rceil$ and $\lceil \log_2 \min(\ell', c_0/2) \rceil = \lceil \log_2 \min(\ell, c/2) \rceil$. Thus, Lines 3 and 4 perform $\ell'' = \ell - 2^k$ additions and multiplications, while the induction hypothesis implies that Line 11 performs at most $(c-1)(\lceil \log_2 \min(\ell, c) \rceil + 1) + 2^k - 1$ additions, and at most $(c-1)\lceil \log_2 \min(\ell, c/2) \rceil + 1)/2 + 2^k - 1$ multiplications. Summing these bounds then shows that Algorithm 3 performs at most $(c - 1)(\lceil \log_2 \min(\ell, c) \rceil + 1) + \ell - 1$ additions, and at most $(c - 1)\lceil \log_2 \min(\ell, c/2) \rceil / 2 + \ell - 1$ multiplications.

Suppose now that $c_1 > 0$. Then Lines 3 to 9 of the algorithm perform $2\ell''$ additions and ℓ'' multiplications. As $\lceil \log_2 \min(\ell', c_1/2) \rceil \leq \lceil \log_2 \min(\ell', c_1) \rceil \leq \lceil \log_2 \ell' \rceil$, the induction hypothesis implies that Line 10 performs at most

$$(c_1 - 1)(\lceil \log_2 \ell' \rceil + 1) + \ell' - 1 + \lceil \log_2 \ell' \rceil$$

additions, and at most $(c_1 - 1)\lceil \log_2 \ell' \rceil / 2 + \ell' - 1$ multiplications. Similarly, as $\ell' \leq 2^k$ and $c_0 = 2^k$, the induction hypothesis implies that Line 11 performs at most $(c_0 - 1)(\lceil \log_2 \ell' \rceil + 1)$ additions, and at most $c_0 \lceil \log_2 \ell' \rceil / 2$ multiplications. Summing these bounds, it follows that the algorithm performs at most

$$(c - 1)(\lceil \log_2 \ell' \rceil + 1) + \ell - 1 + \ell'' - 1$$

additions, and at most $(c - 1)\lceil \log_2 \ell' \rceil / 2 + \ell - 1$ multiplications. If $\ell \leq 2^k$, then $\ell'' = 0$ and $\lceil \log_2 \ell' \rceil = \lceil \log_2 \min(\ell, c) \rceil = \lceil \log_2 \min(\ell, c/2) \rceil$. If $\ell > 2^k$, then $\ell'' = \ell - 2^k \leq 2^k \leq c - 1$ and $\lceil \log_2 \ell' \rceil = \lceil \log_2 \min(\ell, c) \rceil - 1 \leq \lceil \log_2 \min(\ell, c/2) \rceil$. In either case, we find that Algorithm 3 performs at most $(c-1)(\lceil \log_2 \min(\ell, c) \rceil + 1) + \ell - 1$ additions, and at most $(c - 1)\lceil \log_2 \min(\ell, c/2) \rceil / 2 + \ell - 1$ multiplications. □

Theorem 4.3 does not account for the cost of Lines 8 and 9 of Algorithm 3. A straightforward induction argument shows that the lines copy a total of $c - r + (\lceil r / 2^{\lceil \log_2 \ell \rceil - 1} \rceil - 1)(2^{\lceil \log_2 \ell \rceil} - \ell)$ field elements over the duration of the algorithm, where r is the least positive residue of c modulo $2^{\lceil \log_2 \ell \rceil}$. Like the algorithms of Section 3, Algorithm 3 is readily modified so that it avoids performing unnecessary operations when $\lambda = 0$. Doing so saves at least $\ell - 1 + \binom{\lfloor \log_2 \min(\ell, c) \rfloor}{2}$ additions and at least $\ell - 1$ multiplications overall.

4.2. **Interpolation on the LCH basis.** Lemma 4.1 suggests a recursive algorithm for computing the coefficients on the LCH basis of a polynomial $f \in \mathbb{F}[x]_\ell \subseteq \mathbb{F}[x]_{2^n}$ when given the ℓ evaluations $f(\omega_0 + \lambda), \ldots, f(\omega_{\ell-1} + \lambda)$ for some $\lambda \in \mathbb{F}$. Letting $k = \lceil \log_2 \ell \rceil - 1$ and defining $f_0, f_1 \in \mathbb{F}[x]_{2^k}$ by (4.2), the lemma implies that $f(\omega_i + \lambda) = f_0(\omega_i + \lambda)$ for $i \in \{0, \ldots, 2^k - 1\}$, allowing the coefficients of f_0 on the LCH basis to be recursively computed. The lemma further implies that $f(\omega_{2^k+i} + \lambda) = f_1(\omega_i + \lambda + \beta_k)$ for $i \in \{0, \ldots, \ell - 2^k - 1\}$. It follows that if ℓ is a power of two, and thus $\ell - 2^k = 2^k$, then the coefficients of f_1 may also be recursively computed, after-which it is straightforward to compute the coefficients of f.

If ℓ is not a power of two, then we lack the evaluations needed to recursively compute the coefficients of f_1. However, having already computed those of f_0, we know the coefficients of $X_{2^k-\ell}, \ldots, X_{\ell-1}$ in f_1, as they are common to both polynomials. By augmenting the $\ell - 2^k$ evaluations with this additional information, we then possess sufficient information to compute the remaining coefficients of f_1. To incorporate this information into the algorithm, we follow the approach introduced by van der Hoeven [27] for his inverse truncated FFT by generalising the algorithm to include an additional parameter $c \in \{1, \ldots, \ell\}$, and requiring as inputs the c evaluations $f(\omega_0 + \lambda), \ldots, f(\omega_{c-1} + \lambda)$ for some $\lambda \in \mathbb{F}$, and the $\ell - c$ coefficients of $X_c, \ldots, X_{\ell-1}$ in f. The algorithm is then required to compute the c unknown coefficients of $X_0, \ldots, X_{c-1}$ in f, and should be initially called with $c = \ell$.

Pseudocode for the generalised interpolation algorithm is presented in Algorithm 4. Once again the algorithm assumes that $X_{2^i}(\beta_j)$ for $0 \leq i < j < \lceil \log_2 \ell \rceil$ have been precomputed for the input value of ℓ.

Remark 4.4. As noted above, Algorithm 4 should initially be called with $c = \ell$. However, the ability to initially take $c < \ell$ makes the algorithm suitable for use in the fast Hermite interpolation algorithm proposed by the author in [11]. For this application, one is given the higher order coefficients of a polynomial on the monomial basis, rather than on the LCH basis. However, Algorithm 9 of Section 5 can be used to compute the coefficients required by Algorithm 4. Similarly, the evaluation algorithm of the preceding section can be combined with Algorithm 9 for use in the fast Hermite evaluation algorithm proposed in [11].

Theorem 4.5. *Algorithm 4 is correct.*

Proof. We prove the theorem by induction on ℓ. If $\ell = 1$, then $c = 1$ and Algorithm 4 produces the correct output since $f = h_0 X_0 = h_0$ is a constant polynomial. Therefore, for some $\ell \in \{2, \ldots, 2^n\}$, suppose that the algorithm produces the correct output for all inputs with smaller values of ℓ. Let $f \in \mathbb{F}[x]_\ell$ and $h_0, \ldots, h_{\ell-1} \in \mathbb{F}$ such that (4.1) holds. Suppose that the algorithm is called with ℓ as an input, together with $c \in \{1, \ldots, \ell\}$, $(X_{2^i}(\lambda))_{0 \leq i < \lceil \log_2 \ell \rceil}$ for some $\lambda \in \mathbb{F}$, $a_i = f(\omega_i + \lambda)$ for $i \in \{0, \ldots, c-1\}$, and $a_i = h_i$ for $i \in \{c, \ldots, \ell - 1\}$. Let $k = \lceil \log_2 \ell \rceil - 1$, $\ell'' = \ell - 2^k$, $c_0 = \min(c, 2^k)$ and $c_1 = c - c_0$, as computed in Line 2 of the algorithm. Then $\{0, \ldots, c_0 - 1\} \cup \{2^k, \ldots, 2^k + c_1 - 1\} = \{0, \ldots, c-1\}$ and $\{c_0, \ldots, 2^k - 1\} \cup \{2^k + c_1, \ldots, 2^k + \ell'' - 1\} = \{c, \ldots, \ell - 1\}$. Finally, let f_0 and f_1 be the polynomials defined in (4.2). Then $f_0, f_1 \in \mathbb{F}[x]_{2^k}$ and Lemma 4.1 implies

Algorithm 4 $\mathsf{LCHInterp}(\ell, c, (X_{2^i}(\lambda))_{0\leq i<\lceil\log_2 \ell\rceil}, (a_i)_{0\leq i<2^{\lceil\log_2 \ell\rceil}})$

Input: integers $\ell, c \in \{1, \dots, 2^n\}$ such that $c \leq \ell$; the vector $(X_{2^i}(\lambda))_{0\leq i<\lceil\log_2 \ell\rceil}$ for some $\lambda \in \mathbb{F}$; and the vector $(a_i)_{0\leq i<2^{\lceil\log_2 \ell\rceil}}$ such that for some $f \in \mathbb{F}[x]_\ell$, $a_i = f(\omega_i + \lambda)$ for $i \in \{0, \dots, c-1\}$, and $a_i = h_i$ for $i \in \{c, \dots, \ell-1\}$, where $h_0, \dots, h_{\ell-1} \in \mathbb{F}$ are the unique elements that satisfy (4.1).

Output: $a_i = h_i$ for $i \in \{0, \dots, c-1\}$.

1: **if** $\ell = 1$ **then return**
2: $k \leftarrow \lceil\log_2 \ell\rceil - 1$, $\ell'' \leftarrow \ell - 2^k$, $c_0 \leftarrow \min(c, 2^k)$, $c_1 \leftarrow c - c_0$
3: **for** $i = c_0, \dots, \ell'' - 1$ **do**
4: $\quad a_i \leftarrow a_i + X_{2^k}(\lambda) a_{2^k+i}$
5: $\mathsf{LCHInterp}(2^k, c_0, (X_{2^i}(\lambda))_{0\leq i<k}, (a_i)_{0\leq i<2^k})$
6: **if** $c_1 = 0$ **then**
7: $\quad$ **for** $i = 0, \dots, \min(\ell'', c_0) - 1$ **do**
8: $\quad\quad a_i \leftarrow a_i + X_{2^k}(\lambda) a_{2^k+i}$
9: $\quad$ **return**
10: **for** $i = c_1, \dots, \ell'' - 1$ **do**
11: $\quad b \leftarrow X_{2^k}(\lambda) a_{2^k+i}$
12: $\quad a_{2^k+i} \leftarrow a_i + a_{2^k+i}$
13: $\quad a_i \leftarrow a_i + b$
14: **for** $i = \ell'', \dots, 2^k - 1$ **do**
15: $\quad a_{2^k+i} \leftarrow a_i$
16: $\mathsf{LCHInterp}(2^k, c_1, (X_{2^i}(\lambda) + X_{2^i}(\beta_k))_{0\leq i<k}, (a_{2^k+i})_{0\leq i<2^k})$
17: **for** $i = 0, \dots, c_1 - 1$ **do**
18: $\quad a_{2^k+i} \leftarrow a_i + a_{2^k+i}$
19: $\quad a_i \leftarrow a_i + X_{2^k}(\lambda) a_{2^k+i}$

that at the beginning of the algorithm, the following hold for $i \in \{0, \dots, 2^k - 1\}$:

$$a_i = \begin{cases} f_0(\omega_i + \lambda) & \text{if } i < c_0, \\ h_i & \text{otherwise,} \end{cases} \qquad a_{2^k+i} = \begin{cases} f_1(\omega_i + \lambda + \beta_k) & \text{if } i < c_1, \\ h_{2^k+i} & \text{if } c_1 \leq i < \ell'', \\ * & \text{otherwise,} \end{cases}$$

where an asterisks denotes an entry that is unspecified by the algorithm. It follows that Lines 3 and 4 of the algorithm set $a_i = h_i + X_{2^k}(\lambda) h_{2^k+i}$ for $i \in \{c_0, \dots, \ell'' - 1\}$. As $2^k < \ell$, the induction hypothesis and the definition of f_0 imply that for $i \in \{0, \dots, c_0 - 1\}$, Line 5 sets

$$a_i = \begin{cases} h_i + X_{2^k}(\lambda) h_{2^k+i} & \text{if } i < \min(\ell'', c_0), \\ h_i & \text{if } \ell'' \leq i < c_0. \end{cases}$$

The entries $a_{2^k}, \dots, a_{\ell-1}$ are so far unchanged by the algorithm. Thus, if $c_1 = 0$, then Lines 7 and 8 set $a_i = h_i$ for $i \in \{0, \dots, \min(\ell'', c_0) - 1\}$, and the algorithm terminates with $a_i = h_i$ for $i \in \{0, \dots, c_0 - 1\} = \{0, \dots, c - 1\}$, as required.

Suppose now that $c_1 > 0$. Then $c_1 = c - 2^k \leq \ell - 2^k = \ell'' \leq 2^k = c_0$. Therefore, after Lines 10 to 15 have been performed, the following hold for $i \in \{0, \dots, 2^k - 1\}$:

$$a_i = \begin{cases} h_i + X_{2^k}(\lambda) h_{2^k+i} & \text{if } i < c_1, \\ h_i & \text{otherwise,} \end{cases} \qquad a_{2^k+i} = \begin{cases} f_1(\omega_i + \lambda + \beta_k) & \text{if } i < c_1, \\ h_i + (1 + X_{2^k}(\lambda)) h_{2^k+i} & \text{if } c_1 \leq i < \ell'', \\ h_i & \text{otherwise.} \end{cases}$$

As $2^k < \ell$, the induction hypothesis and the definition of f_1 imply that Line 16 sets $a_{2^k+i} = h_i + (1 + X_{2^k}(\lambda))h_{2^k+i}$ for $i \in \{0, \ldots, c_1 - 1\}$. Thus, Lines 17 to 19 then set $a_{2^k+i} = (h_i + X_{2^k}(\lambda)h_{2^k+i}) + (h_i + (1 + X_{2^k}(\lambda))h_{2^k+i}) = h_{2^k+i}$ and $a_i = (h_i + X_{2^k}(\lambda)h_{2^k+i}) + X_{2^k}(\lambda)h_{2^k+i} = h_i$ for $i \in \{0, \ldots, c_1 - 1\}$. After-which, the algorithm terminates with $a_i = h_i$ for $i \in \{0, \ldots, 2^k + c_1 - 1\} = \{0, \ldots, c - 1\}$, as required. □

Algorithm 4 can be implemented so that only $2^{\lceil \log_2 \ell \rceil} - \ell + \mathcal{O}(\log^2 \ell)$ field elements are stored in auxiliary space. All precomputations for the algorithm can be performed with $\mathcal{O}(\log^2 \ell)$ field operations, while the number of field operations performed by the algorithm itself is bounded by the following theorem.

Theorem 4.6. *Algorithm 4 performs at most*

$$\min\Big((c-1)(\lceil \log_2 c \rceil + 1) + \ell - 1, \Big(2^{\lceil \log_2 \ell \rceil} - 1\Big)(\lceil \log_2 \ell \rceil + 1)\Big)$$

additions in $\mathbb{F}$*, and at most*

$$\min\left(\frac{c-1}{2}(\lceil \log_2 c \rceil - 1) + \ell - 1, 2^{\lceil \log_2 \ell \rceil - 1} \lceil \log_2 \ell \rceil\right)$$

multiplications in $\mathbb{F}$*.*

Proof. Suppose that Algorithms 3 and 4 are called with identical inputs $\ell, c \in \{1, \ldots, 2^n\}$ such that $c \leq \ell$. Then the values k, ℓ'', c_0 and c_1 computed in Line 2 of both algorithms are identical, while the value ℓ' computed in Line 2 of Algorithm 3 is equal to 2^k. Thus, if $c_1 = 0$, then the recursive calls of Line 5 of Algorithm 4 and Line 11 of Algorithm 3 are made with identical inputs ℓ and c, while the remaining lines of either algorithm perform ℓ'' additions and ℓ'' multiplications. If $c_1 > 0$, then the recursive calls of Line 5 of Algorithm 4 and Line 11 of Algorithm 3 are once again made with identical inputs ℓ and c, the recursive calls of Line 16 of Algorithm 4 and Line 10 of Algorithm 3 are similarly made with identical inputs ℓ and c, and the remaining lines of either algorithm perform $2\ell''$ additions and ℓ'' multiplications. As both algorithms perform no additions or multiplications if $\ell = 1$ and $c = 1$, it follows by induction on ℓ that Algorithms 3 and 4 perform the same number of additions and the same number of multiplications when given identical inputs $\ell, c \in \{1, \ldots, 2^n\}$ such that $c \leq \ell$. Thus, the theorem follows from Theorem 4.3. □

Lines 14 and 15 of Algorithm 4 copy $2^k - \ell'' = 2^{\lceil \log_2 \ell \rceil} - \ell$ field elements if $c_1 > 0$. Thus, no elements are copied during recursive calls made by the algorithm, since they always have ℓ equal to a power of two. It follows that the algorithm copies $(\lceil c/2^{\lceil \log_2 \ell \rceil - 1} \rceil - 1)(2^{\lceil \log_2 \ell \rceil} - \ell)$ field elements in total. Modifying Algorithm 4 so that it avoids performing unnecessary operations when $\lambda = 0$ saves at least $\ell - 1 + \binom{\lfloor \log_2 c \rfloor}{2}$ additions and at least $\ell - 1$ multiplications overall.

4.3. **Space-efficient algorithms.** For values of ℓ that are slightly larger than a power of two, the vector $(a_i)_{0 \leq i < 2^{\lceil \log_2 \ell \rceil}}$ on which Algorithm 4 operates has length that is almost double the length ℓ of the interpolation problem it is solving. This additional space is filled by Lines 14 and 15 of the algorithm with copies of the $2^{\lceil \log_2 \ell \rceil} - \ell$ coefficients of f_0 on the LCH basis that are common to f_1. The copies are then passed to the recursive call of Line 16. Duplicating the coefficients is necessary since the output requirements of the algorithm do not guarantee that

the original coefficients will be preserved if they are passed to the recursive call, in which case they cannot be used to compute the final output of the algorithm. Thus, modifying the algorithm so that it preserves all coefficients it is given as inputs allows it to be further modified to operate on a vector of length ℓ. By making these modifications, we obtain Algorithm 5.

Algorithm 5 is once again presented under the assumption that $X_{2^i}(\beta_j)$ for $0 \leq i < j < \lceil \log_2 \ell \rceil$ have been precomputed for the input value of ℓ. In Line 9 of the algorithm, $(a_i)_{2^k \leq i < \ell} \,\|\, (a_i)_{\ell'' \leq i < 2^k}$ denotes the concatenation of the subvectors $(a_i)_{2^k \leq i < \ell}$ and $(a_i)_{\ell'' \leq i < 2^k}$. The resulting vector is the cyclic right shift by ℓ'' positions of the subvector $(a_i)_{\ell'' \leq i < \ell}$. Thus, Line 9 of the algorithm can realised with only $\mathcal{O}(1)$ additional field elements stored in auxiliary space by using an in-place algorithm to permute the entries of this subvector before passing it to the recursive call, then subsequently inverting the permutation. These permutations are only required if ℓ is not a power of two, and, in particular, are not required by any recursive calls made by the algorithm.

Algorithm 5 $\mathsf{LowSpaceLCHInterp}(\ell, c, (X_{2^i}(\lambda))_{0 \leq i < \lceil \log_2 \ell \rceil}, (a_i)_{0 \leq i < \ell})$

Input: integers $\ell, c \in \{1, \ldots, 2^n\}$ such that $c \leq \ell$; the vector $(X_{2^i}(\lambda))_{0 \leq i < \lceil \log_2 \ell \rceil}$ for some $\lambda \in \mathbb{F}$; and the vector $(a_i)_{0 \leq i < \ell}$ such that for some $f \in \mathbb{F}[x]_\ell$, $a_i = f(\omega_i + \lambda)$ for $i \in \{0, \ldots, c-1\}$, and $a_i = h_i$ for $i \in \{c, \ldots, \ell - 1\}$, where $h_0, \ldots, h_{\ell-1} \in \mathbb{F}$ are the unique elements that satisfy (4.1).

Output: $a_i = h_i$ for $i \in \{0, \ldots, \ell - 1\}$.

1: **if** $\ell = 1$ **then return**
2: $k \leftarrow \lceil \log_2 \ell \rceil - 1$, $\ell'' \leftarrow \ell - 2^k$, $c_0 \leftarrow \min(c, 2^k)$, $c_1 \leftarrow c - c_0$
3: **for** $i = c_0, \ldots, \ell'' - 1$ **do**
4: $\quad a_i \leftarrow a_i + X_{2^k}(\lambda) a_{2^k + i}$
5: $\mathsf{LowSpaceLCHInterp}(2^k, c_0, (X_{2^i}(\lambda))_{0 \leq i < k}, (a_i)_{0 \leq i < 2^k})$
6: **if** $c_1 > 0$ **then**
7: $\quad$ **for** $i = c_1, \ldots, \ell'' - 1$ **do**
8: $\quad\quad a_{2^k + i} \leftarrow a_i + a_{2^k + i}$
9: $\quad \mathsf{LowSpaceLCHInterp}(2^k, c_1, (X_{2^i}(\lambda) + X_{2^i}(\beta_k))_{0 \leq i < k}, (a_i)_{2^k \leq i < \ell} \,\|\, (a_i)_{\ell'' \leq i < 2^k})$
10: $\quad$ **for** $i = 0, \ldots, \ell'' - 1$ **do**
11: $\quad\quad a_{2^k + i} \leftarrow a_i + a_{2^k + i}$
12: **for** $i = 0, \ldots, \ell'' - 1$ **do**
13: $\quad a_i \leftarrow a_i + X_{2^k}(\lambda) a_{2^k + i}$

Theorem 4.7. *Algorithm 5 is correct.*

Proof. If $\ell = 1$, then $c = 1$ and Algorithm 4 produces the correct output since $f = h_0$ is a constant polynomial. Proceeding by induction, assume for some $\ell \in \{2, \ldots, 2^n\}$ that the algorithm produces the correct output for all inputs with smaller values of ℓ. Let $f \in \mathbb{F}[x]_\ell$ and $h_0, \ldots, h_{\ell-1} \in \mathbb{F}$ such that (4.1) holds. Suppose that the algorithm is called with ℓ as an input, together with $c \in \{1, \ldots, \ell\}$, $(X_{2^i}(\lambda))_{0 \leq i < \lceil \log_2 \ell \rceil}$ for some $\lambda \in \mathbb{F}$, $a_i = f(\omega_i + \lambda)$ for $i \in \{0, \ldots, c-1\}$, and $a_i = h_i$ for $i \in \{c, \ldots, \ell - 1\}$. Define integers k, ℓ'', c_0 and c_1 as in Line 2 of the algorithm, and let f_0, f_1 be the polynomials defined in (4.2). Then $f_0, f_1 \in \mathbb{F}[x]_{2^k}$ and Lemma 4.1 implies that at the beginning of the algorithm, the following hold

for $i \in \{0, \ldots, 2^k - 1\}$ and $j \in \{0, \ldots, \ell'' - 1\}$:

$$a_i = \begin{cases} f_0(\omega_i + \lambda) & \text{if } i < c_0, \\ h_i & \text{otherwise,} \end{cases} \qquad a_{2^k+j} = \begin{cases} f_1(\omega_j + \lambda + \beta_k) & \text{if } j < c_1, \\ h_{2^k+j} & \text{otherwise.} \end{cases}$$

It follows that Lines 3 and 4 of the algorithm set $a_i = h_i + X_{2^k}(\lambda)h_{2^k+i}$ for $i \in \{c_0, \ldots, \ell'' - 1\}$. The induction hypothesis and the definition of f_0 imply that Line 5 sets $a_i = h_i + X_{2^k}(\lambda)h_{2^k+i}$ for $i \in \{0, \ldots, \min(\ell'', c_0) - 1\}$, and $a_i = h_i$ for $i \in \{\ell'', \ldots, c_0 - 1\}$, without modifying a_i for $i \in \{c_0, \ldots, 2^k - 1\}$. Thus, after Line 5, the following holds for $i \in \{0, \ldots, 2^k - 1\}$:

$$a_i = \begin{cases} h_i + X_{2^k}(\lambda)h_{2^k+i} & \text{if } i < \ell'', \\ h_i & \text{otherwise.} \end{cases}$$

Therefore, if $c_1 = 0$, then Lines 6 to 11 have no effect, while Lines 12 and 13 set $a_i = (h_i + X_{2^k}(\lambda)h_{2^k+i}) + X_{2^k}(\lambda)h_{2^k+i} = h_i$ for $i \in \{0, \ldots, \ell'' - 1\}$. It follows that the algorithm terminates with $a_i = h_i$ for $i \in \{0, \ldots, \ell - 1\}$ in this case, as required.

Suppose now that $c_1 > 0$. Then $c_1 \leq \ell'' \leq 2^k$. Therefore, after Lines 7 and 8 have been performed, the following holds for $i \in \{0, \ldots, \ell'' - 1\}$:

$$a_{2^k+i} = \begin{cases} f_1(\omega_i + \lambda + \beta_k) & \text{if } i < c_1, \\ h_i + (1 + X_{2^k}(\lambda))h_{2^k+i} & \text{otherwise.} \end{cases}$$

The induction hypothesis and the definition of f_1 imply that Line 9 sets $a_{2^k+i} = h_i + (1 + X_{2^k}(\lambda))h_{2^k+i}$ for $i \in \{0, \ldots, c_1 - 1\}$, without modifying a_i for $i \in \{\ell'', \ldots, 2^k - 1\} \cup \{2^k + c_1, \ldots, 2^k + \ell'' - 1\}$. Thus, Lines 10 and 11 set $a_{2^k+i} = (h_i + X_{2^k}(\lambda)h_{2^k+i}) + (h_i + (1 + X_{2^k}(\lambda))h_{2^k+i}) = h_{2^k+i}$ for $i \in \{0, \ldots, \ell'' - 1\}$, after-which Lines 12 and 13 set $a_i = (h_i + X_{2^k}(\lambda)h_{2^k+i}) + X_{2^k}(\lambda)h_{2^k+i} = h_i$ for $i \in \{0, \ldots, \ell'' - 1\}$. Hence, the algorithm once again terminates with $a_i = h_i$ for $i \in \{0, \ldots, \ell - 1\}$ in this case. □

Rather than prove an analogue of Theorem 4.6 for Algorithm 5, we instead bound the difference in the number of field operations performed by Algorithms 4 and 5 when given identical inputs.

Theorem 4.8. *When compared to Algorithm 4 for identical inputs, Algorithm 5 performs at most* $2^{\lceil \log_2 \ell \rceil} - c(\lceil \log_2 \ell \rceil - \lceil \log_2 c \rceil + \lfloor c/2^{\lceil \log_2 \ell \rceil} \rfloor)$ *more additions in* $\mathbb{F}$*, and at most* $2^{\lceil \log_2 \ell \rceil} - c(\lceil \log_2 \ell \rceil - \lceil \log_2 c \rceil + 1)$ *more multiplications in* $\mathbb{F}$*.*

Proof. It is clear that the bounds hold if $\ell = 1$, since c must equal one and both algorithms will terminate immediately without performing any field operations. Proceeding by induction, let $\ell \in \{2, \ldots, 2^n\}$ and suppose that the bounds stated in the theorem hold for all smaller values of ℓ. Moreover, suppose that Algorithms 4 and 5 are called with identical inputs that include ℓ and $c \in \{1, \ldots, \ell\}$. Let $k = \lceil \log_2 \ell \rceil - 1$, $\ell'' = \ell - 2^k$, $c_0 = \min(c, 2^k)$ and $c_1 = c - c_0$, as computed in Line 2 of both algorithms.

Suppose for now that $c_1 = 0$. Then $c_0 = c \leq 2^k$. Thus, the induction hypothesis implies that Line 5 of Algorithm 5 performs at most

$$2^k - c\Big(k - \lceil \log_2 c \rceil + \Big\lfloor \frac{c}{2^k} \Big\rfloor\Big) \leq 2^{\lceil \log_2 \ell \rceil} - c\Big(\lceil \log_2 \ell \rceil - \lceil \log_2 c \rceil + \Big\lfloor \frac{c}{2^{\lceil \log_2 \ell \rceil}} \Big\rfloor\Big) - \big(2^k - c\big)$$

more additions, and at most $2^{\lceil \log_2 \ell \rceil} - c(\lceil \log_2 \ell \rceil - \lceil \log_2 c \rceil + 1) - (2^k - c)$ more multiplications, than Line 5 of Algorithm 4. The remaining lines of Algorithm 5 perform $\max(\ell'' - c_0, 0) \leq \max(2^{k+1} - 2^k - c, 0) = 2^k - c$ more additions and

multiplications than the remaining lines of Algorithm 4. Consequently, the bounds stated in the theorem hold if $c_1 = 0$.

Suppose now that $c_1 > 0$. Then $2^k < c \leq \ell \leq 2^{k+1} = 2^{\lceil \log_2 \ell \rceil}$, $c_0 = 2^k$ and $c_1 = c - 2^k \leq 2^k$. Thus, the induction hypothesis implies that Line 5 of Algorithm 5 performs the same number of operations in $\mathbb{F}$ as Line 5 of Algorithm 4. Moreover, we have $k - \lceil \log_2 c_1 \rceil \geq 0 = \lceil \log_2 \ell \rceil - \lceil \log_2 c \rceil$ and $\lfloor c_1 / 2^k \rfloor = \lfloor c / 2^{\lceil \log_2 \ell \rceil} \rfloor$. Therefore, the induction hypothesis implies that Line 9 of Algorithm 5 performs at most

$$2^{\lceil \log_2 \ell \rceil} - c\Big(\lceil \log_2 \ell \rceil - \lceil \log_2 c \rceil + \Big\lfloor \frac{c}{2^{\lceil \log_2 \ell \rceil}} \Big\rfloor\Big) + 2^k \Big(\Big\lfloor \frac{c}{2^{\lceil \log_2 \ell \rceil}} \Big\rfloor - 1\Big)$$

more additions, and at most $2^{\lceil \log_2 \ell \rceil} - c(\lceil \log_2 \ell \rceil - \lceil \log_2 c \rceil + 1)$ more multiplications, than Line 16 of Algorithm 4. The remaining lines of Algorithm 5 perform $\ell'' - c_1 = \ell - c$ more additions than those of Algorithm 4, and the same number of multiplications. Therefore, the bounds stated in the theorem hold if $c_1 > 0$, since $2^k(\lfloor c / 2^{\lceil \log_2 \ell \rceil} \rfloor - 1) + \ell - c$ is equal to zero if $c = 2^{\lceil \log_2 \ell \rceil}$, and equal to $\ell - c - 2^k \leq 2^{k+1} - (2^k + 1) - 2^k < 0$ otherwise. □

Each for-loop in Algorithm 5 performs a transformation on the vector $(a_i)_{0 \leq i < \ell}$ that is an involution. Thus, performing these transformations in reverse order inverts the overall transformation performed by the algorithm, yielding an evaluation algorithm with the same complexity and space requirements as Algorithm 5. Pseudocode for this evaluation algorithm is presented in Algorithm 6. From the proof of Theorem 4.6, we know that Algorithms 3 and 4 perform the same number of additions and multiplications when given identical inputs $\ell, c \in \{1, \ldots, 2^n\}$ such that $c \leq \ell$. Consequently, the bounds of Theorem 4.8 also hold when comparing Algorithm 6 to Algorithm 3 for identical inputs.

Algorithm 6 $\mathsf{LowSpaceLCHEval}(\ell, c, (X_{2^i}(\lambda))_{0 \leq i < \lceil \log_2 \ell \rceil}, (a_i)_{0 \leq i < \ell})$

Input: integers $\ell, c \in \{1, \ldots, 2^n\}$ such that $c \leq \ell$; the vector $(X_{2^i}(\lambda))_{0 \leq i < \lceil \log_2 \ell \rceil}$ for some $\lambda \in \mathbb{F}$; and the vector $(a_i)_{0 \leq i < \ell}$ such that for some $f \in \mathbb{F}[x]_\ell$, $a_i = h_i$ for $i \in \{0, \ldots, \ell - 1\}$, where $h_0, \ldots, h_{\ell-1} \in \mathbb{F}$ are the unique elements that satisfy (4.1).

Output: $a_i = f(\omega_i + \lambda)$ for $i \in \{0, \ldots, c - 1\}$; and $a_i = h_i$ for $i \in \{c, \ldots, \ell - 1\}$.

1: **if** $\ell = 1$ **then return**
2: $k \leftarrow \lceil \log_2 \ell \rceil - 1$, $\ell'' \leftarrow \ell - 2^k$, $c_0 \leftarrow \min(c, 2^k)$, $c_1 \leftarrow c - c_0$
3: **for** $i = 0, \ldots, \ell'' - 1$ **do**
4: $\quad a_i \leftarrow a_i + X_{2^k}(\lambda) a_{2^k + i}$
5: **if** $c_1 > 0$ **then**
6: $\quad$ **for** $i = 0, \ldots, \ell'' - 1$ **do**
7: $\quad\quad a_{2^k + i} \leftarrow a_i + a_{2^k + i}$
8: $\quad \mathsf{LowSpaceLCHEval}(2^k, c_1, (X_{2^i}(\lambda) + X_{2^i}(\beta_k))_{0 \leq i < k}, (a_i)_{2^k \leq i < \ell} \,\|\, (a_i)_{\ell'' \leq i < 2^k})$
9: $\quad$ **for** $i = c_1, \ldots, \ell'' - 1$ **do**
10: $\quad\quad a_{2^k + i} \leftarrow a_i + a_{2^k + i}$
11: $\mathsf{LowSpaceLCHEval}(2^k, c_0, (X_{2^i}(\lambda))_{0 \leq i < k}, (a_i)_{0 \leq i < 2^k})$
12: **for** $i = c_0, \ldots, \ell'' - 1$ **do**
13: $\quad a_i \leftarrow a_i + X_{2^k}(\lambda) a_{2^k + i}$

5. Conversion between the LCH and monomial bases

We consider the problem of converting between the LCH and monomial bases under the assumption that there exists a tower of subfields

$$\mathbb{F}_2 = \mathbb{F}_{2^{d_0}} \subset \mathbb{F}_{2^{d_1}} \subset \cdots \subset \mathbb{F}_{2^{d_m}} \subseteq \mathbb{F} \tag{5.1}$$

such that $d_{m-1} < n \leq d_m$, and $\beta_i/\beta_{d_t\lfloor i/d_t \rfloor} \in \mathbb{F}_{2^{d_t}}$ for $i \in \{0, \ldots, n-1\}$ and $t \in \{0, \ldots, m-1\}$. We can assume that $\mathbb{F}$ is not equal to $\mathbb{F}_2$, since otherwise $\beta = (1)$ and its associated LCH basis coincides with the monomial basis. Then the existence of such a tower is established by taking $m = 1$ and $d_m = [\mathbb{F} : \mathbb{F}_2]$, so we have not imposed any restrictions on the vector β. However, our algorithms enjoy a reduction in their complexities when m is greater than one. To the tower we associate a family of bases of $\mathbb{F}[x]_{2^n}$ that includes the LCH basis and the twisted monomial basis $\{1, x/\beta_0, \ldots, (x/\beta_0)^{2^n-1}\}$. We then show how to efficiently convert between its members, allowing for rapid conversion between the LCH and twisted monomial bases by traversing the family. Conversion between the LCH and monomial bases, in either direction, then requires only linearly many additional multiplications to be performed. We begin by introducing the family of bases and some supporting notation.

For $t \in \{0, \ldots, m-1\}$ and $k \in \{0, \ldots, \lceil n/d_t \rceil - 1\}$, define $e_{t,k} = \lceil d_t(k+1)/d_{t+1} \rceil$ so that $d_{t+1}e_{t,k}$ is the least multiple of d_{t+1} that is greater than or equal to $d_t(k+1)$. For $t \in \{0, \ldots, m-1\}$, $k \in \{0, \ldots, \lceil n/d_t \rceil - 1\}$ and $i \in \{0, \ldots, 2^n - 1\}$, define

$$Y_i^{(t,k)} = \left(\prod_{s=0}^{k} X_{2^{d_t s}}^{i_s} \right) \left(\prod_{s'=e_{t,k}}^{\lceil n/d_{t+1} \rceil - 1} X_{2^{d_{t+1}s'}}^{i'_{s'}} \right),$$

where

$$i_s = \sum_{r=0}^{d_t-1} 2^r [i]_{d_t s + r}, \quad i_k = \sum_{r=0}^{d_{t+1}e_{t,k} - d_t k - 1} 2^r [i]_{d_t k + r} \quad \text{and} \quad i'_{s'} = \sum_{r=0}^{d_{t+1}-1} 2^r [i]_{d_{t+1}s' + r}$$

for $s \in \{0, \ldots, k-1\}$ and $s' \in \{e_{t,k}, \ldots, \lceil n/d_{t+1} \rceil - 1\}$. Then each polynomial $Y_i^{(t,k)}$ has degree equal to i. It follows that $\{Y_0^{(t,k)}, \ldots, Y_{\ell-1}^{(t,k)}\}$ is a basis of $\mathbb{F}[x]_\ell$ for $\ell \in \{1, \ldots, 2^n\}$, $t \in \{0, \ldots, m-1\}$ and $k \in \{0, \ldots, \lceil n/d_t \rceil - 1\}$. In Section 5.1, we show that the LCH basis is obtained by taking $t = 0$ and $k = \lceil n/d_0 \rceil - 1$, while the twisted monomial basis corresponds to $t = m-1$ and $k = 0$.

For $t \in \{0, \ldots, m-1\}$ and $k \in \{0, \ldots, \lceil n/d_t \rceil - 2\}$, define

$$\delta_{t,k} = X_{2^{d_t k}}\big(\beta_{d_t(k+1)}\big)^{2^{d_t}} - X_{2^{d_t k}}\big(\beta_{d_t(k+1)}\big).$$

Then $\delta_{0,0}, \ldots, \delta_{0,n-2}$ are the denominators that appear in the recurrence relation from property (3) of Lemma 2.1. In Section 5.1, we show that our assumption on the quotients $\beta_i/\beta_{d_t\lfloor i/d_t \rfloor}$ leads to higher order recurrence relations between the polynomials $X_{2^0}, \ldots, X_{2^{n-1}}$ of the LCH basis for which the $\delta_{t,k}$ once again appear as denominators. For $t \in \{0, \ldots, m-1\}$, $k \in \{0, \ldots, \lceil n/d_t \rceil - 1\}$ and $\ell \in \{1, \ldots, 2^n\}$, define

$$I_{t,k,\ell} = \big\{0, \ldots, \min\big(\lceil \ell/2^{d_t k} \rceil, 2^{d_{t+1}e_{t,k} - d_t k}\big) - 1\big\},$$

and

$$J_{t,k,\ell} = \big\{j \in \{0, \ldots, \ell-1\} \mid [j]_{d_t k} = \cdots = [j]_{d_{t+1}e_{t,k}-1} = 0\big\}.$$

For $t \in \{0, \ldots, m-1\}$ and $k \in \{0, \ldots, \lceil n/d_t \rceil - 1\}$, the sets $\{2^{d_t k} i + j \mid i \in I_{t,k,\ell-j}\}$ for $j \in J_{t,k,\ell}$ then form a partition of $\{0, \ldots, \ell-1\}$.

Having introduced the family of bases and the requisite notation, we are now ready to state the main technical lemma upon which we base our algorithms for converting between the LCH and monomial bases.

Lemma 5.1. *Let $\ell \in \{1, \dots, 2^n\}$ and $f \in \mathbb{F}[x]_\ell$. Then, for $t \in \{0, \dots, m-1\}$ and $k \in \{0, \dots, \lceil n/d_t \rceil - 1\}$, there exist unique elements $f_0^{(t,k)}, \dots, f_{\ell-1}^{(t,k)} \in \mathbb{F}$ such that*

$$f = \sum_{i=0}^{\ell-1} f_i^{(t,k)} Y_i^{(t,k)}. \tag{5.2}$$

Moreover, the following properties hold for $t \in \{0, \dots, m-1\}$ and $k \in \{0, \dots, \lceil n/d_t \rceil - 1\}$:

1. *if $t = m-1$ and $k = 0$, then $f = \sum_{i=0}^{\ell-1} f_i^{(t,k)} (x/\beta_0)^i$,*
2. *if $k < \lceil n/d_t \rceil - 1$ and d_{t+1}/d_t divides $k+1$, then $f_i^{(t,k)} = f_i^{(t,k+1)}$ for $i \in \{0, \dots, \ell-1\}$,*
3. *if $k < \lceil n/d_t \rceil - 1$ and d_{t+1}/d_t does not divide $k+1$, then*

$$\sum_{i=0}^{|I_{t,k,\ell-j}|-1} f_{2^{d_t k}i+j}^{(t,k)} x^i = \sum_{i=0}^{|I_{t,k,\ell-j}|-1} f_{2^{d_t k}i+j}^{(t,k+1)} x^{i-2^{d_t}\lfloor i/2^{d_t} \rfloor} \left(\frac{x^{2^{d_t}} - x}{\delta_{t,k}} \right)^{\lfloor i/2^{d_t} \rfloor}$$

for $j \in J_{t,k,\ell}$, where

$$\delta_{t,k} = \mathrm{Tr}_{\mathbb{F}_{2^{d_{t+1}}}/\mathbb{F}_{2^{d_t}}} \left(\frac{\beta_{d_t(k+1)}}{\beta_{d_t k}} \right)$$

if $d_{t+1}/d_t = 2$,

4. *if $t > 0$ and $k \geq \lceil (\log_2 \ell)/d_t \rceil - 1$, then $f_i^{(t,k)} = f_i^{(t-1,0)}$ for $i \in \{0, \dots, \ell-1\}$,*
5. *if $t = 0$ and $k \geq \lceil (\log_2 \ell)/d_t \rceil - 1$, then $f = \sum_{i=0}^{\ell-1} f_i^{(t,k)} X_i$.*

Lemma 5.1 is proved in Section 5.1. Properties (1) and (5) of the lemma identify the twisted monomial and LCH bases amongst the family of bases of $\mathbb{F}[x]_\ell$, while properties (2), (3) and (4) provide a means of traversing its members. Of the later three properties, only property (3) requires computation, which can be performed efficiently by applying the generalised Taylor expansion algorithm of Gao and Mateer [14, Section II] or its inverse algorithm, depending on the direction of conversion. These algorithms are recalled in Section 5.2.

The number of times property (3) must be applied is invariant under the choice of tower: if $\ell \in \{2, \dots, 2^{d_m}\}$, then

$$\sum_{t=0}^{m-1} \sum_{\substack{k=0 \\ d_{t+1}/d_t \nmid k+1}}^{\lceil (\log_2 \ell)/d_t \rceil - 2} 1 = \sum_{t=0}^{m-1} \left(\left\lceil \frac{\log_2 \ell}{d_t} \right\rceil - \left\lceil \frac{\log_2 \ell}{d_{t+1}} \right\rceil \right) = \lceil \log_2 \ell \rceil - 1. \tag{5.3}$$

However, by using the algorithms of Gao and Mateer, the number of additions and multiplications required by a Taylor expansion step decreases as d_t grows. Thus, our algorithms benefit by the presence of subfields of degree less than $\log_2 \ell$. No multiplications are performed during a Taylor expansion step if $\delta_{t,k} = 1$, which in the case that $\mathbb{F}_{2^{d_{t+1}}}/\mathbb{F}_{2^{d_t}}$ is a quadratic extension occurs if and only if $\mathrm{Tr}_{\mathbb{F}_{2^{d_{t+1}}}/\mathbb{F}_{2^{d_t}}}(\beta_{d(k+1)}/\beta_{dk}) = 1$. Multiplications are also saved if $\beta_0 = 1$, since the twisted monomial basis coincides with the monomial basis. We cannot guarantee that any of these desirable properties are satisfied by an arbitrary choice of β.

Furthermore, some of the properties are precluded when the field does not contain subfields of appropriate degree. However, the following proposition shows that when we have freedom to choose β, and subfields of appropriate degree do exist, then a standard basis construction can be used to obtain a vector with the desired properties.

Proposition 5.2. *Suppose there exists a tower of subfields*

$$\mathbb{F}_2 = \mathbb{F}_{2^{d_0}} \subset \mathbb{F}_{2^{d_1}} \subset \cdots \subset \mathbb{F}_{2^{d_m}} \subseteq \mathbb{F}.$$

Let $\{\alpha_{t,0}, \ldots, \alpha_{t,d_{t+1}/d_t-1}\}$ *be a basis of* $\mathbb{F}_{2^{d_{t+1}}}/\mathbb{F}_{2^{d_t}}$ *for* $t \in \{0, \ldots, m-1\}$*, and*

$$\beta_i = \prod_{t=0}^{m-1} \alpha_{t,i_t} \quad \text{such that} \quad \sum_{t=0}^{m-1} i_t d_t = i$$

for $i \in \{0, \ldots, d_m - 1\}$*. Then the following properties hold:*

1. $\beta_0, \ldots, \beta_{d_m-1}$ *are linearly independent over* $\mathbb{F}_2$*,*
2. $\beta_i/\beta_{d_t\lfloor i/d_t \rfloor} \in \mathbb{F}_{2^{d_t}}$ *for* $i \in \{0, \ldots, d_m - 1\}$ *and* $t \in \{0, \ldots, m-1\}$*,*
3. *if* $\alpha_{0,0} = \cdots = \alpha_{m-1,0} = 1$*, then* $\beta_0 = 1$*,*
4. *if* $d_{t+1}/d_t = 2$ *for some* $t \in \{0, \ldots, m-1\}$*, then* $\beta_{d_t(k+1)}/\beta_{d_t k} = \alpha_{t,1}/\alpha_{t,0}$ *for even* $k \in \{0, \ldots, d_m/d_t - 2\}$*.*

Proof. Let $i \in \{0, \ldots, d_m - 1\}$ and write $i = \sum_{t=0}^{m-1} i_t d_t$ with $i_t \in \{0, \ldots, d_{t+1}/d_t - 1\}$ for $t \in \{0, \ldots, m-1\}$. Then

$$\frac{\beta_i}{\beta_{d_t\lfloor i/d_t \rfloor}} = \frac{\alpha_{0,i_1} \cdots \alpha_{t-1,i_{t-1}} \alpha_{t,i_t} \cdots \alpha_{m-1,i_{m-1}}}{\alpha_{0,0} \cdots \alpha_{t-1,0} \alpha_{t,i_t} \cdots \alpha_{m-1,i_{m-1}}} = \frac{\alpha_{0,i_0} \cdots \alpha_{t-1,i_{t-1}}}{\alpha_{0,0} \cdots \alpha_{t-1,0}} \in \mathbb{F}_{2^{d_t}}$$

for $t \in \{0, \ldots, m-1\}$. Thus, property (2) holds. Similarly, we have

$$\frac{\beta_{d_t i + j}}{\beta_0} = \frac{\alpha_{t,i}}{\alpha_{t,0}} \frac{\beta_j}{\beta_0} \quad \text{and} \quad \frac{\beta_j}{\beta_0} = \frac{\beta_j}{\beta_{d_t\lfloor j/d_t \rfloor}} \in \mathbb{F}_{2^{d_t}}$$

for $i \in \{0, \ldots, d_{t+1}/d_t - 1\}$, $j \in \{0, \ldots, d_t - 1\}$ and $t \in \{0, \ldots, m-1\}$. As $\{\alpha_{t,0}/\alpha_{t,0}, \ldots, \alpha_{t,d_{t+1}/d_t-1}/\alpha_{t,0}\}$ is a basis of $\mathbb{F}_{2^{d_{t+1}}}/\mathbb{F}_{2^{d_t}}$ for $t \in \{0, \ldots, m-1\}$, it follows that if $\{\beta_0/\beta_0, \ldots, \beta_{d_t-1}/\beta_0\}$ is a basis of $\mathbb{F}_{2^{d_t}}/\mathbb{F}_2$ for some $t \in \{0, \ldots, m-1\}$, then $\{\beta_0/\beta_0, \ldots, \beta_{d_{t+1}-1}/\beta_0\}$ is a basis of $\mathbb{F}_{2^{d_{t+1}}}/\mathbb{F}_2$. Therefore, $\{\beta_0/\beta_0, \ldots, \beta_{d_m-1}/\beta_0\}$ is a basis of $\mathbb{F}_{2^{d_m}}/\mathbb{F}_2$, since $d_0 = 1$. Hence, property (1) holds.

By definition, β_0 is the product of $\alpha_{0,0}, \ldots, \alpha_{m-1,0}$, from which property (3) follows immediately. Property (4) holds since if $d_{t+1}/d_t = 2$ for some $t \in \{0, \ldots, m-1\}$, then

$$\frac{\beta_{d_t(k+1)}}{\beta_{d_t k}} = \frac{\beta_{d_{t+1}(k/2)+d_t}}{\beta_{d_{t+1}(k/2)}} = \frac{\alpha_{t,1}}{\alpha_{t,0}}$$

for even $k \in \{0, \ldots, d_m/d_t - 2\}$. □

If the tower in Proposition 5.2 contains a quadratic extension $\mathbb{F}_{2^{d_{t+1}}}/\mathbb{F}_{2^{d_t}}$, then taking $\alpha_{t,0} = 1$ and $\alpha_{t,1} \in \mathbb{F}_{2^{d_{t+1}}}$ such that $\mathrm{Tr}_{\mathbb{F}_{2^{d_{t+1}}}/\mathbb{F}_{2^{d_t}}}(\alpha_{t,1}) = 1$ yields a basis $\{\alpha_{t,0}, \alpha_{t,1}\}$ of the extension with the property that $\alpha_{t,1}/\alpha_{t,0}$ has trace equal to one. Indeed, linear independence of the two elements over $\mathbb{F}_{2^{d_t}}$ follows from the observation that $\mathrm{Tr}_{\mathbb{F}_{2^{d_{t+1}}}/\mathbb{F}_{2^{d_t}}}(1) = 0$.

5.1. **Proof of Lemma 5.1.** We begin by proving properties (1), (2), (4) and (5) of Lemma 5.1.

Lemma 5.3. *The following properties hold for* $t \in \{0, \dots, m-1\}$ *and* $k \in \{0, \dots, \lceil n/d_t \rceil - 1\}$*:*

(1) *if* $t = m-1$ *and* $k = 0$*, then* $Y_i^{(t,k)} = (x/\beta_0)^i$ *for* $i \in \{0, \dots, 2^n - 1\}$*,*
(2) *if* $k < \lceil n/d_t \rceil - 1$ *and* d_{t+1}/d_t *divides* $k+1$*, then* $Y_i^{(t,k)} = Y_i^{(t,k+1)}$ *for* $i \in \{0, \dots, 2^n - 1\}$*,*
(3) *if* $t > 0$*, then* $Y_i^{(t,k)} = Y_i^{(t-1,0)}$ *for* $i \in \{0, \dots, 2^{\min(d_t(k+1),n)} - 1\}$*,*
(4) *if* $t = 0$*, then* $Y_i^{(t,k)} = X_i$ *for* $i \in \{0, \dots, 2^{k+1} - 1\}$*.*

Proof. We have $\lceil n/d_m \rceil - 1 = 0$, $e_{m-1,0} = 1$ and

$$\sum_{r=0}^{d_m e_{m-1,0} - d_{m-1} \times 0 - 1} 2^r [i]_{d_{m-1} \times 0 + r} = \sum_{r=0}^{d_m - 1} 2^r [i]_r = i$$

for $i \in \{0, \dots, 2^n - 1\}$. Thus, $Y_i^{(m-1,0)} = X_1^i$ for $i \in \{0, \dots, 2^n - 1\}$. As $X_1 = x/\beta_0$ by property (3) of Lemma 2.1, it follows that property (1) holds.

Let $t \in \{0, \dots, m-1\}$ and $k \in \{0, \dots, \lceil n/d_t \rceil - 2\}$ such that d_{t+1}/d_t divides $k+1$. Then $e_{t,k} = d_t(k+1)/d_{t+1}$ and $e_{t,k+1} = e_{t,k} + 1$. Thus, for $i \in \{0, \dots, 2^n - 1\}$,

$$\sum_{r=0}^{d_{t+1} e_{t,k} - d_t k - 1} 2^r [i]_{d_t k + r} = \sum_{r=0}^{d_t - 1} 2^r [i]_{d_t k + r}$$

and

$$\sum_{r=0}^{d_{t+1} - 1} 2^r [i]_{d_{t+1} e_{t,k} + r} = \sum_{r=0}^{d_{t+1} e_{t,k+1} - d_t(k+1) - 1} 2^r [i]_{d_t(k+1) + r}.$$

It follows that property (2) holds.

Suppose that $t \in \{1, \dots, m-1\}$ and $k \in \{0, \dots, \lceil n/d_t \rceil - 1\}$. Then $d_{t+1} e_{t,k} \geq d_t(k+1)$ and $e_{t-1,0} = 1$. For $i \in \{0, \dots, 2^{d_t(k+1)} - 1\}$, it follows that

$$\sum_{r=0}^{d_t - 1} 2^r [i]_{d_t s + r} = 0 \quad \text{and} \quad \sum_{r=0}^{d_{t+1} - 1} 2^r [i]_{d_{t+1} s' + r} = 0$$

for $s \in \{k+1, \dots, \lceil n/d_t \rceil - 1\}$ and $s' \in \{e_{t,k}, \dots, \lceil n/d_t \rceil - 1\}$. Consequently, if $i \in \{0, \dots, 2^{\min(d_t(k+1),n)} - 1\}$ and

$$i_s = \sum_{r=0}^{d_t - 1} 2^r [i]_{d_t s + r} \quad \text{for } s \in \{0, \dots, \lceil n/d_t \rceil - 1\},$$

then

$$i_0 = \sum_{r=0}^{d_t - 1} 2^r [i]_{d_t \times 0 + r} = \sum_{r=0}^{d_t e_{t-1,0} - d_{t-1} \times 0 - 1} 2^r [i]_{d_{t-1} \times 0 + r}$$

and

$$Y_i^{(t,k)} = \prod_{s=0}^{k} X_{2^{d_t s}}^{i_s} = \left(\prod_{s=0}^{0} X_{2^{d_{t-1} s}}^{i_s} \right) \left(\prod_{s'=e_{t-1,0}}^{\lceil n/d_t \rceil - 1} X_{2^{d_t s'}}^{i_{s'}} \right) = Y_i^{(t-1,0)}.$$

Therefore, property (3) holds.

As $d_0 = 1$, we have $d_1 e_{0,k} = d_1\lceil (k+1)/d_1 \rceil \geq k+1$ for $k \in \{0, \dots, \lceil n/d_1 \rceil - 1\}$. Thus, property (1) of Lemma 2.1 implies that

$$Y_i^{(0,k)} = \left(\prod_{s=0}^{k} X_{2^s}^{[i]_s} \right) \left(\prod_{s'=e_{0,k}}^{\lceil n/d_1 \rceil - 1} X_{2^{d_1 s'}}^0 \right) = \prod_{s=0}^{k} X_{2^s [i]_s} = X_i$$

for $k \in \{0, \dots, \lceil n/d_1 \rceil - 1\}$ and $i \in \{0, \dots, 2^{k+1} - 1\}$. Hence, property (4) holds. □

Recall that a polynomial in $\mathbb{F}[x]$ is $\mathbb{F}_q$-linearised if it can be written in the form $\sum_{i \in \mathbb{N}} f_i x^{q^i}$ with $f_0, f_1, \dots \in \mathbb{F}$. The following lemma generalises properties (3) and (5) of Lemma 2.1, and shows that our assumption on the quotients $\beta_i/\beta_{d_t \lfloor i/d_t \rfloor}$ leads to higher order and, importantly, sparse recurrence relations between the polynomials $X_{2^0}, \dots, X_{2^{n-1}}$ of the LCH basis.

Lemma 5.4. *Let $\mathbb{F}_{2^d} \subseteq \mathbb{F}$ such that $\beta_i/\beta_{d\lfloor i/d \rfloor} \in \mathbb{F}_{2^d}$ for $i \in \{0, \dots, n-1\}$. Then the following properties hold for $k \in \{0, \dots, \lceil n/d \rceil - 1\}$:*

(1) *if $k < \lceil n/d \rceil - 1$, then*

$$X_{2^{d(k+1)}} = \frac{X_{2^{dk}}(x)^{2^d} - X_{2^{dk}}(x)}{X_{2^{dk}}\big(\beta_{d(k+1)}\big)^{2^d} - X_{2^{dk}}\big(\beta_{d(k+1)}\big)},$$

(2) *$X_{2^{dk}}$ is $\mathbb{F}_{2^d}$-linearised.*

Moreover, if $\mathbb{F}_{2^{2d}} \subseteq \mathbb{F}$ and $\beta_i/\beta_{2d\lfloor i/(2d) \rfloor} \in \mathbb{F}_{2^{2d}}$ for $i \in \{0, \dots, n-1\}$, then

$$X_{2^{dk}}\big(\beta_{d(k+1)}\big)^{2^d} - X_{2^{dk}}\big(\beta_{d(k+1)}\big) = \mathrm{Tr}_{\mathbb{F}_{2^{2d}}/\mathbb{F}_{2^d}}\left(\frac{\beta_{d(k+1)}}{\beta_{dk}} \right) \tag{5.4}$$

for even $k \in \{0, \dots, \lceil n/d \rceil - 2\}$.

Proof. Let $\mathbb{F}_{2^d} \subseteq \mathbb{F}$ such that $\beta_i/\beta_{d\lfloor i/d \rfloor} \in \mathbb{F}_{2^d}$ for $i \in \{0, \dots, n-1\}$. We show that if $X_{2^{dk}}$ is $\mathbb{F}_{2^d}$-linearised for some $k \in \{0, \dots, \lceil n/d \rceil - 2\}$, then property (1) holds and $X_{2^{d(k+1)}}$ is $\mathbb{F}_{2^d}$-linearised. As property (3) of Lemma 2.1 implies that $X_{2^0} = x/\beta_0$ is $\mathbb{F}_{2^d}$-linearised, properties (1) and (2) of the lemma then follow by induction on k.

Suppose that $X_{2^{dk}}$ is $\mathbb{F}_{2^d}$-linearised for some $k \in \{0, \dots, \lceil n/d \rceil - 2\}$. Then

$$\frac{\omega_{2^{dk} i}}{\beta_{dk}} = \sum_{j=0}^{d-1} [i]_j \frac{\beta_{dk+j}}{\beta_{dk}} = \sum_{j=0}^{d-1} [i]_j \frac{\beta_{dk+j}}{\beta_{d\lfloor (dk+j)/d \rfloor}} \in \mathbb{F}_{2^d} \quad \text{for } i \in \{0, \dots, 2^d - 1\}.$$

Thus, property (5) of Lemma 2.1 and the assumption that $X_{2^{dk}}$ is $\mathbb{F}_{2^d}$-linearised imply that

$$X_{2^{dk}}\big(\omega_{2^{dk} i + j}\big) = X_{2^{dk}}(\beta_{dk}) \frac{\omega_{2^{dk} i}}{\beta_{dk}} + X_{2^{dk}}(\omega_j) = \frac{\omega_{2^{dk} i}}{\beta_{dk}} \in \mathbb{F}_{2^d}$$

for $i \in \{0, \dots, 2^d - 1\}$ and $j \in \{0, \dots, 2^{dk} - 1\}$. It follows that ω_i is a common root of the degree $2^{d(k+1)}$ polynomials $X_{2^{d(k+1)}}$ and $X_{2^{dk}}^{2^d} - X_{2^{dk}}$ for $i \in \{0, \dots, 2^{d(k+1)} - 1\}$. Hence, they are equal up to a nonzero scalar multiple. As $X_{2^{d(k+1)}}(\beta_{d(k+1)}) = 1$, it follows that property (1) holds. Property (1) then implies that $X_{2^{d(k+1)}}$ is $\mathbb{F}_{2^d}$-linearised, since $X_{2^{dk}}$ is $\mathbb{F}_{2^d}$-linearised by assumption.

Suppose now that $\mathbb{F}_{2^{2d}} \subseteq \mathbb{F}$ and $\beta_i/\beta_{2d\lfloor i/(2d) \rfloor} \in \mathbb{F}_{2^{2d}}$ for $i \in \{0, \dots, n-1\}$. Then for even $k \in \{0, \dots, \lceil n/d \rceil - 2\}$, property (1) of the lemma implies that

$X_{2^{dk}} = X_{2^{2d(k/2)}}$ is $\mathbb{F}_{2^{2d}}$-linearised, while

$$\frac{\beta_{d(k+1)}}{\beta_{dk}} = \frac{\beta_{d(k+1)}}{\beta_{2d(k/2)}} = \frac{\beta_{d(k+1)}}{\beta_{2d\lfloor d(k+1)/(2d)\rfloor}} \in \mathbb{F}_{2^{2d}}.$$

Hence, for even $k \in \{0, \dots, \lceil n/d \rceil - 2\}$, we have

$$X_{2^{dk}}\big(\beta_{d(k+1)}\big) = X_{2^{dk}}(\beta_{dk})\frac{\beta_{d(k+1)}}{\beta_{dk}} = \frac{\beta_{d(k+1)}}{\beta_{dk}},$$

from which (5.4) follows. □

If $t \in \{0, \dots, m-1\}$ and $k \in \{0, \dots, \lceil n/d_t \rceil - 2\}$ such that d_{t+1}/d_t does not divide $k+1$, then $e_{t,k+1} = e_{t,k}$. It follows in this case that each quotient $Y_i^{(t,k+1)}/Y_i^{(t,k)}$ is of the form $X^a_{2^{d_t(k+1)}}/X^b_{2^{d_t k}}$ for some $a, b \in \mathbb{N}$. Using Lemma 5.4 to express these quotients as a functions of $X_{2^{d_t k}}$ leads to property (3) of Lemma 5.1.

Lemma 5.5. *If $t \in \{0, \dots, m-1\}$ and $k \in \{0, \dots, \lceil n/d_t \rceil - 2\}$ such that d_{t+1}/d_t does not divide $k+1$, then*

$$Y^{(t,k)}_{2^{d_t k} i + j} = X^i_{2^{d_t k}} Y^{(t,k)}_j$$

and

$$Y^{(t,k+1)}_{2^{d_t k} i + j} = \left(\frac{X^{2^{d_t}}_{2^{d_t k}} - X_{2^{d_t k}}}{\delta_{t,k}}\right)^{\lfloor i/d_t \rfloor} X^{i - 2^{d_t}\lfloor i/d_t \rfloor}_{2^{d_t k}} Y^{(t,k)}_j$$

for $i \in I_{t,k,2^n}$ and $j \in J_{t,k,2^n}$.

Proof. Suppose that $t \in \{0, \dots, m-1\}$ and $k \in \{0, \dots, \lceil n/d_t \rceil - 2\}$ such that d_{t+1}/d_t does not divide $k+1$. Then, for $i \in I_{t,k,2^n}$ and $j \in J_{t,k,2^n}$, we have

$$\sum_{r=0}^{d_t - 1} 2^r [2^{d_t k} i + j]_{d_t s + r} = \sum_{r=0}^{d_t - 1} 2^r [j]_{d_t s + r}$$

for $s \in \{0, \dots, k-1\}$,

$$\sum_{r=0}^{d_{t+1}e_{t,k} - d_t k - 1} 2^r [2^{d_t k} i + j]_{d_t k + r} = \sum_{r=0}^{d_{t+1}e_{t,k} - d_t k - 1} 2^r [i]_r = i + \sum_{r=0}^{d_{t+1}e_{t,k} - d_t k - 1} 2^r [j]_{d_t k + r}$$

and

$$\sum_{r=0}^{d_{t+1} - 1} 2^r [2^{d_t k} i + j]_{d_{t+1} s' + r} = \sum_{r=0}^{d_{t+1} - 1} 2^r [j]_{d_{t+1} s' + r}$$

for $s' \in \{e_{t,k}, \dots, \lceil n/d_t \rceil - 1\}$. Consequently, $Y^{(t,k)}_{2^{d_t k} i + j} = X^i_{2^{d_t k}} Y^{(t,k)}_j$ for $i \in I_{t,k,2^n}$ and $j \in J_{t,k,2^n}$.

As d_{t+1}/d_t does not divide $k+1$, $e_{t,k+1} = e_{t,k}$ and $d_{t+1}e_{t,k+1} = d_{t+1}e_{t,k} > d_t(k+1)$. Thus, for $i \in I_{t,k,2^n}$ and $j \in J_{t,k,2^n}$, we have

$$\sum_{r=0}^{d_t - 1} 2^r [2^{d_t k} i + j]_{d_t k + r} = \sum_{r=0}^{d_t - 1} 2^r [i]_r = \left(i - 2^{d_t}\left\lfloor \frac{i}{2^{d_t}} \right\rfloor\right) + \sum_{r=0}^{d_t - 1} 2^r [j]_{d_t k + r}$$

and

$$\sum_{r=0}^{d_{t+1}e_{t,k+1}-d_t(k+1)-1} 2^r[2^{d_tk}i+j]_{d_t(k+1)+r} = \sum_{r=0}^{d_{t+1}e_{t,k}-d_t(k+1)-1} 2^r[i]_{d_t+r} + \left\lfloor \frac{i}{2^{d_t}} \right\rfloor + \sum_{r=0}^{d_{t+1}e_{t,k}-d_t(k+1)-1} 2^r[j]_{d_t(k+1)+r}.$$

It follows that

$$Y^{(t,k+1)}_{2^{d_tk}i+j} = X^{\lfloor i/2^{d_t}\rfloor}_{2^{d_t(k+1)}} X^{i-2^{d_t}\lfloor i/2^{d_t}\rfloor}_{2^{d_tk}} Y^{(t,k)}_j$$

for $i \in I_{t,k,2^n}$ and $j \in J_{t,k,2^n}$. The proof is completed by making the substitution $X_{2^{d_t(k+1)}} = (X^{2^{d_t}}_{2^{d_tk}} - X_{2^{d_tk}})/\delta_{t,k}$, which holds by Lemma 5.4 and the assumption that $\beta_i/\beta_{d_t\lfloor i/d_t\rfloor} \in \mathbb{F}_{2^{d_t}}$ for $i \in \{0,\dots,n-1\}$. □

We are now ready to prove Lemma 5.1.

Proof of Lemma 5.1. Let $\ell \in \{1,\dots,2^n\}$ and $f \in \mathbb{F}[x]_\ell$. Then, for $t \in \{0,\dots,m-1\}$ and $k \in \{0,\dots,\lceil n/d_t\rceil - 1\}$, existence and uniqueness of the elements $f^{(t,k)}_0,\dots,f^{(t,k)}_{\ell-1}$ follows from the observation that $Y^{(t,k)}_i$ has degree equal to i for $i \in \{0,\dots,\ell-1\}$. Properties (1), (2), (4) and (5) then follow immediately from Lemma 5.3. To prove property (3), suppose that $t \in \{0,\dots,m-1\}$ and $k \in \{0,\dots,\lceil n/d_t\rceil - 2\}$ such that d_{t+1}/d_t does not divide $k+1$. Then k is even if $d_{t+1}/d_t = 2$, in which case Lemma 5.4 implies that $\delta_{t,k} = \mathrm{Tr}_{\mathbb{F}_{2^{d_{t+1}}}/\mathbb{F}_{2^{d_t}}}(\beta_{d_t(k+1)}/\beta_{d_tk})$. The sets $\{2^{d_tk}i+j \mid i \in I_{t,k,\ell-j}\} = \{2^{d_tk}i+j \mid i \in \{0,\dots,|I_{t,k,\ell-j}|-1\}\}$ for $j \in J_{t,k,\ell}$ form a partition of $\{0,\dots,\ell-1\}$. Thus, there exist elements $f'_0,\dots,f'_{\ell-1} \in \mathbb{F}$ such that

$$\sum_{i=0}^{|I_{t,k,\ell-j}|-1} f^{(t,k)}_{2^{d_tk}i+j}x^i = \sum_{i=0}^{|I_{t,k,\ell-j}|-1} f'_{2^{d_tk}i+j}x^{i-2^{d_t}\lfloor i/2^{d_t}\rfloor}\left(\frac{x^{2^{d_t}}-x}{\delta_{t,k}}\right)^{\lfloor i/2^{d_t}\rfloor}$$

for $j \in J_{t,k,\ell}$. After substituting $X_{2^{d_tk}}$ for x, then multiplying each equation by $Y^{(t,k)}_j$, Lemma 5.5 implies that

$$\sum_{i=0}^{|I_{t,k,\ell-j}|-1} f^{(t,k)}_{2^{d_tk}i+j}Y^{(t,k)}_{2^{d_tk}i+j} = \sum_{i=0}^{|I_{t,k,\ell-j}|-1} f'_{2^{d_tk}i+j}Y^{(t,k+1)}_{2^{d_tk}i+j}$$

for $j \in J_{t,k,\ell}$. By summing these equations, it follows that

$$f = \sum_{j\in J_{t,k,\ell}}\sum_{i\in I_{t,k,\ell-j}} f^{(t,k)}_{2^{d_tk}i+j}Y^{(t,k)}_{2^{d_tk}i+j} = \sum_{i=0}^{\ell-1} f'_iY^{(t,k+1)}_i.$$

Thus, the uniqueness of $f^{(t,k+1)}_0,\dots,f^{(t,k+1)}_{\ell-1}$ implies that $f'_i = f^{(t,k+1)}_i$ for $i \in \{0,\dots,\ell-1\}$. Hence, property (3) follows by the choice of $f'_0,\dots,f'_{\ell-1}$. □

5.2. **Generalised Taylor expansion in characteristic two.** The generalised Taylor expansion of a polynomial $f \in \mathbb{F}[x]$ at a nonconstant polynomial $p \in \mathbb{F}[x]$, also known as its p-adic expansion, is the series expansion

$$f = f_0 + f_1p + f_2p^2 + \cdots$$

such that $f_i \in \mathbb{F}[x]_{\deg p}$ for $i \in \mathbb{N}$. Gao and Mateer [14, Section II] provide a fast algorithm for computing the coefficients of the Taylor expansion when $p = x^t - x$ for some $t \geq 2$, which is utilised as part of their additive FFT algorithms.

Their algorithm may be viewed as a specialisation of the recursive algorithm of von zur Gathen [29] that is supplemented by the easy division of polynomials in $\mathbb{F}[x]_{2^{k+1}t}$ by $(x^t - x)^{2^k} = x^{2^k t} - x^{2^k}$. We present a nonrecursive version of Gao and Mateer's algorithm modelled on the basis conversion algorithms of van der Hoeven and Schost [28, Section 2.2], and specialised to t equal to a power of two. We also present the inverse algorithm, which recovers a polynomial from the coefficients of its Taylor expansion. Finally, we derive a bound on the complexity of both algorithms that is tighter than the one provided by Gao and Mateer.

Let $f \in \mathbb{F}[x]_\ell$ and $p \in \mathbb{F}[x]$ be a nonconstant polynomial. Then, for $k \in \mathbb{N}$, there exist unique polynomials $f_{k,0}, f_{k,1}, \ldots \in \mathbb{F}[x]_{2^k \deg p}$ such that

$$f = \sum_{i \in \mathbb{N}} f_{k,i} p^{2^k i}.$$

In particular, $f_{0,0}, f_{0,1}, \ldots$ are the coefficients of the Taylor expansion of f at p, $f_{k,i} = 0$ for $i \geq \lceil \ell / (2^k \deg p) \rceil$, and $f_{k,0} = f$ for $k \geq \lceil \log_2 \lceil \ell / \deg p \rceil \rceil$. Grouping terms with indices $2i$ and $2i+1$ shows that

$$f = \sum_{i \in \mathbb{N}} \Big(f_{k,2i} + p^{2^k} f_{k,2i+1} \Big) p^{2^{k+1} i} \quad \text{for } k \in \mathbb{N}.$$

Thus, $f_{k+1,i} = f_{k,2i} + p^{2^k} f_{k,2i+1}$ for $k, i \in \mathbb{N}$. If $p = x^{2^d} - x$ for some $d \geq 1$, then

$$f_{k+1,i} = f_{k,2i} + x^{2^k} f_{k,2i+1} + x^{2^{d+k}} f_{k,2i+1} \quad \text{for } k, i \in \mathbb{N}.$$

In this case, given the coefficients of $f_{k,2i}$ and $f_{k,2i+1}$ on the monomial basis, it follows that the coefficients of $f_{k+1,i}$ on the monomial basis can be computed with $1 + \deg f_{k,2i+1} \leq 2^{d+k}$ additions. This computation is also readily inverted by performing the same number of additions. Consequently, given the Taylor coefficients $f_{0,0}, f_{0,1}, \ldots$ with respect to the monomial basis, we can efficiently compute $f = f_{\lceil \log_2 \lceil \ell / 2^d \rceil \rceil, 0}$ with respect to the monomial basis by means of the recursive formula, and vice versa. Using this observation, we obtain Algorithms 7 and 8.

Algorithm 7 $\mathsf{TaylorExpansion}(d, \ell, (a_i)_{0 \leq i < \ell})$

Input: Integers $d \geq 1$ and $\ell \geq 1$; and the vector $(a_i)_{0 \leq i < \ell}$ such that $a_i = f_i \in \mathbb{F}$ for $i \in \{0, \ldots, \ell - 1\}$.

Output: $a_i = h_i \in \mathbb{F}$ for $i \in \{0, \ldots, \ell - 1\}$ such that

$$\sum_{i=0}^{\ell-1} h_i x^{i - 2^d \lfloor i / 2^d \rfloor} \Big(x^{2^d} - x \Big)^{\lfloor i / 2^d \rfloor} = \sum_{i=0}^{\ell-1} f_i x^i. \tag{5.5}$$

1: **for** $k = \lceil \log_2 \lceil \ell / 2^d \rceil \rceil - 1, \ldots, 0$ **do**
2: $\quad \ell_1 \leftarrow \lfloor \ell / (2^{d+k+1}) \rfloor$, $\ell_2 \leftarrow \ell - 2^{d+k+1} \ell_1$
3: $\quad$ **for** $i = 0, \ldots, \ell_1 - 1$ **do**
4: $\quad\quad$ **for** $j = 2^{d+k} - 1, \ldots, 0$ **do**
5: $\quad\quad\quad a_{2^{d+k}(2i) + 2^k + j} \leftarrow a_{2^{d+k}(2i) + 2^k + j} + a_{2^{d+k}(2i+1) + j}$
6: $\quad$ **for** $j = \ell_2 - 2^{d+k} - 1, \ldots, 0$ **do**
7: $\quad\quad a_{2^{d+k}(2\ell_1) + 2^k + j} \leftarrow a_{2^{d+k}(2\ell_1) + 2^k + j} + a_{2^{d+k}(2\ell_1 + 1) + j}$

Lemma 5.6. *Algorithms 7 and 8 perform at most* $\lfloor \ell / 2 \rfloor \lceil \log_2 \lceil \ell / 2^d \rceil \rceil$ *additions in* $\mathbb{F}$.

Algorithm 8 InverseTaylorExpansion$(d, \ell, (a_i)_{0\le i<\ell})$

Input: Integers $d \ge 1$ and $\ell \ge 1$; and the vector $(a_i)_{0\le i<\ell}$ such that $a_i = h_i \in \mathbb{F}$ for $i \in \{0, \ldots, \ell-1\}$.
Output: $a_i = f_i$ for $i \in \{0, \ldots, \ell-1\}$ such that (5.5) holds.

1: **for** $k = 0, \ldots, \lceil \log_2 \lceil \ell/2^d \rceil \rceil - 1$ **do**
2: $\quad \ell_1 \leftarrow \lfloor \ell/(2^{d+k+1}) \rfloor$, $\ell_2 \leftarrow \ell - 2^{d+k+1}\ell_1$
3: $\quad$ **for** $i = 0, \ldots, \ell_1 - 1$ **do**
4: $\quad\quad$ **for** $j = 0, \ldots, 2^{d+k} - 1$ **do**
5: $\quad\quad\quad a_{2^{d+k}(2i)+2^k+j} \leftarrow a_{2^{d+k}(2i)+2^k+j} + a_{2^{d+k}(2i+1)+j}$
6: $\quad$ **for** $j = 0, \ldots, \ell_2 - 2^{d+k} - 1$ **do**
7: $\quad\quad a_{2^{d+k}(2\ell_1)+2^k+j} \leftarrow a_{2^{d+k}(2\ell_1)+2^k+j} + a_{2^{d+k}(2\ell_1+1)+j}$

Proof. For each $k \in \{0, \ldots, \lceil \log_2 \lceil \ell/2^d \rceil \rceil - 1\}$, Lines 2–7 of either algorithm perform

$$2^{d+k}\ell_1 + \max(\ell_2 - 2^{d+k}, 0) \le 2^{d+k}\ell_1 + (\ell_2 - \lceil \ell_2/2 \rceil) = 2^{d+k}\ell_1 + \lfloor \ell_2/2 \rfloor = \lfloor \ell/2 \rfloor$$

additions in $\mathbb{F}$. □

5.3. **Conversion algorithms and complexity.** By combining Lemma 5.1 with the Taylor expansion algorithms of Section 5.2, we obtain Algorithms 9 and 10 for converting between the LCH and monomial bases. Algorithm 9 is presented under the assumption that $\delta_{t,k}$ has been precomputed for $t \in \{0, \ldots, m-1\}$ and $k \in \{0, \ldots, \lceil (\log_2 \ell)/d_t \rceil - 2\}$ such that d_{t+1}/d_t does not divide $k+1$, while Algorithm 10 assumes that the inverses of these elements and $1/\beta_0$ have been precomputed. As a result, (5.3) implies that each algorithm is required to store $\mathcal{O}(\log \ell)$ precomputed elements. For each $t \in \{0, \ldots, m-1\}$, the elements $\delta_{t,k}$ for $k \in \{0, \ldots, \lceil (\log_2 \ell)/d_t \rceil - 2\}$ can be computed with $\mathcal{O}((\log^2 \ell)/d_t)$ field operations by using property (1) of Lemma 5.4 and exponentiation by repeated squaring. As $d_t \ge 2^t$ for $t \in \{0, \ldots, m-1\}$, it follows that all precomputations for the algorithms can be performed with $\mathcal{O}(\log^2 \ell)$ field operations. Each algorithm operates on a vector $(a_i)_{0\le i<\ell}$ that initially contains the coefficients of a polynomial on the input basis, and which has it entries overwritten by the coefficients of the polynomial on the output basis. The subvectors of these vectors that are passed to Algorithms 7 and 8 can be represented in practice by a pointer to their first entry and a stride parameter. In doing so, only $\mathcal{O}(\log \ell)$ field elements are required to be stored in auxiliary space by either algorithm.

Theorem 5.7. *Algorithms 9 and 10 are correct.*

Proof. Correctness is only proved for Algorithm 9, since the proof for Algorithm 10 follows along similar lines. Suppose that Algorithm 9 is called on $\ell \in \{1, \ldots, 2^n\}$ and $(a_i)_{0\le i<\ell}$, with $a_i = f_i \in \mathbb{F}$ for $i \in \{0, \ldots, \ell-1\}$. Then it is clear that the algorithm produces the correct output if $\ell = 1$, since $X_0 = 1$ and the algorithm does not modify the entries of $(a_i)_{0\le i<\ell}$ in this case. Therefore, assume that $\ell > 1$. Let $f = \sum_{i=0}^{\ell-1} f_i x^i$, and $f_0^{(t,k)}, \ldots, f_{\ell-1}^{(t,k)} \in \mathbb{F}$ satisfy (5.2) for $t \in \{0, \ldots, m-1\}$ and $k \in \{0, \ldots, \lceil n/d_t \rceil - 1\}$, which exist and are unique by Lemma 5.1.

Lines 1 to 4 of the algorithm multiply a_i by β_0^i for $i \in \{1, \ldots, \ell-1\}$ if $\beta_0 \ne 1$. If $\ell \le 2$, then the remaining lines of the algorithm have no effect on the vector $(a_i)_{0\le i<\ell}$, since $\lceil (\log_2 \ell)/d_t \rceil < 2$ for $t \in \{0, \ldots, m-1\}$. As $X_0 = 1$ and $X_0 = x/\beta_0$, it follows that the algorithm produces the correct output if $\ell \le 2$. Therefore, assume

Algorithm 9 $\mathsf{MonomialToLCH}(\ell, (a_i)_{0\leq i<\ell})$

Input: an integer $\ell \in \{1, \dots, 2^n\}$; and the vector $(a_i)_{0\leq i<\ell}$ such that $a_i = f_i \in \mathbb{F}$ for $i \in \{0, \dots, \ell-1\}$.

Output: $a_i = h_i \in \mathbb{F}$ for $i \in \{0, \dots, \ell-1\}$ such that $\sum_{i=0}^{\ell-1} h_i X_i = \sum_{i=0}^{\ell-1} f_i x^i$.

1: **if** $\beta_0 \neq 1$ **and** $\ell > 1$ **then**
2: $\quad c \leftarrow \beta_0$, $a_1 \leftarrow ca_1$
3: $\quad$ **for** $i = 2, \dots, \ell-1$ **do**
4: $\quad\quad c \leftarrow \beta_0 c$, $a_i \leftarrow ca_i$
5: **for** $t = m-1, m-2, \dots, 0$ **do**
6: $\quad$ **for** $k = 0, \dots, \lceil(\log_2 \ell)/d_t\rceil - 2$ **do**
7: $\quad\quad$ **if** $d_{t+1}/d_t \nmid k+1$ **then**
8: $\quad\quad\quad$ **for** $j \in J_{t,k,\ell}$ **do**
9: $\quad\quad\quad\quad \mathsf{TaylorExpansion}(d_t, |I_{t,k,\ell-j}|, (a_{2^{d_t k}i+j})_{0\leq i<|I_{t,k,\ell-j}|})$
10: $\quad\quad\quad$ **if** $\delta_{t,k} \neq 1$ **then**
11: $\quad\quad\quad\quad q \leftarrow \lceil |I_{t,k,\ell}|/2^{d_t}\rceil - 1$, $r \leftarrow |I_{t,k,\ell}| - 2^{d_t} q$, $c \leftarrow \delta_{t,k}$
12: $\quad\quad\quad\quad$ **for** $i_1 = 1, \dots, q-1$ **do**
13: $\quad\quad\quad\quad\quad$ **for** $i_0 = 0, \dots, 2^{d_t} - 1$ **do**
14: $\quad\quad\quad\quad\quad\quad$ **for** $j \in J_{t,k,\ell-2^{d_t k}(2^{d_t} i_1 + i_0)}$ **do**
15: $\quad\quad\quad\quad\quad\quad\quad a_{2^{d_t k}(2^{d_t} i_1 + i_0)+j} \leftarrow c a_{2^{d_t k}(2^{d_t} i_1 + i_0)+j}$
16: $\quad\quad\quad\quad\quad c \leftarrow \delta_{t,k} c$
17: $\quad\quad\quad\quad$ **for** $i_0 = 0, \dots, r-1$ **do**
18: $\quad\quad\quad\quad\quad$ **for** $j \in J_{t,k,\ell-2^{d_t k}(2^{d_t} q + i_0)}$ **do**
19: $\quad\quad\quad\quad\quad\quad a_{2^{d_t k}(2^{d_t} q + i_0)+j} \leftarrow c a_{2^{d_t k}(2^{d_t} q + i_0)+j}$

that $\ell > 2$. Then $2^{d_s} < \ell \leq 2^{d_{s+1}}$ for some $s \in \{0, \dots, m-1\}$, and properties (1) and (4) of Lemma 5.1 imply that $a_i = f_i^{(m-1,0)} = \cdots = f_i^{(s,0)}$ for $i \in \{0, \dots, \ell-1\}$ after Lines 1 to 4 of the algorithm have been performed.

Suppose that during the algorithm, Line 7 is reached and the entries of $(a_i)_{0\leq i<\ell}$ satisfy $a_i = f_i^{(t,k)}$ for $i \in \{0, \dots, \ell-1\}$, where $t \in \{0, \dots, m-1\}$ and $k \in \{0, \dots, \lceil(\log_2 \ell)/d_t\rceil - 2\}$ are the current values of the loop counter variables in Lines 5 and 6. If d_{t+1}/d_t divides $k+1$, then Lines 7 to 19 have no effect on the vector, while property (2) of Lemma 5.1 implies that its entries satisfy $a_i = f_i^{(t,k+1)}$ for $i \in \{0, \dots, \ell-1\}$. If d_{t+1}/d_t does not divide $k+1$, then after Lines 7 to 9 have been performed, the entries of $(a_i)_{0\leq i<\ell}$ satisfy

$$\sum_{i=0}^{|I_{t,k,\ell-j}|-1} f^{(t,k)}_{2^{d_t k}i+j} x^i = \sum_{i=0}^{|I_{t,k,\ell-j}|-1} a_{2^{d_t k}i+j} x^{i-2^{d_t}\lfloor i/2^{d_t}\rfloor} \left(x^{2^{d_t}} - x\right)^{\lfloor i/2^{d_t}\rfloor}$$

for $j \in J_{t,k,\ell}$. If $\delta_{t,k} \neq 1$, then Lines 10 to 19 subsequently multiply $a_{2^{d_t k}i+j}$ by $\delta_{t,k}^{\lfloor i/2^{d_t}\rfloor}$ for $i \in I_{t,k,\ell}$ and $j \in J_{t,k,\ell-2^{d_t k}i}$ such that $i \geq 2^{d_t}$, after-which property (3) of Lemma 5.1 implies that $a_i = f_i^{(t,k+1)}$ for $i \in \{0, \dots, \ell-1\}$. Therefore, regardless of whether d_{t+1}/d_t divides $k+1$, if the entries of $(a_i)_{0\leq i<\ell}$ satisfy $a_i = f_i^{(t,k)}$ for $i \in \{0, \dots, \ell-1\}$ upon reaching Line 7, then $a_i = f_i^{(t,k+1)}$ for $i \in \{0, \dots, \ell-1\}$ after Lines 7 to 19 have been performed. In particular, if $t \neq 0$ and $k = \lceil(\log_2 \ell)/d_t\rceil - 2$, then property (4) of Lemma 5.1 implies that $a_i = f_i^{(t-1,0)}$ for $i \in \{0, \dots, \ell-1\}$ after Lines 7 to 19 have been performed. As $a_i = f_i^{(t,k)}$ for $i \in \{0, \dots, \ell-1\}$ the first

Algorithm 10 LCHToMonomial$(\ell, (a_i)_{0\le i<\ell})$

Input: an integer $\ell \in \{1, \dots, 2^n\}$; and the vector $(a_i)_{0\le i<\ell}$ such that $a_i = h_i \in \mathbb{F}$ for $i \in \{0, \dots, \ell-1\}$.

Output: $a_i = f_i \in \mathbb{F}$ for $i \in \{0, \dots, \ell-1\}$ such that $\sum_{i=0}^{\ell-1} f_i x^i = \sum_{i=0}^{\ell-1} h_i X_i$.

1: **for** $t = 0, \dots, m-1$ **do**
2: **for** $k = \lceil (\log_2 \ell)/d_t \rceil - 2, \dots, 0$ **do**
3: **if** $d_{t+1}/d_t \nmid k+1$ **then**
4: **if** $\delta_{t,k} \neq 1$ **then**
5: $q \leftarrow \lceil |I_{t,k,\ell}|/2^{d_t} \rceil - 1$, $r \leftarrow |I_{t,k,\ell}| - 2^{d_t} q$, $c \leftarrow (1/\delta_{t,k})$
6: **for** $i_1 = 1, \dots, q-1$ **do**
7: **for** $i_0 = 0, \dots, 2^{d_t} - 1$ **do**
8: **for** $j \in J_{t,k,\ell-2^{d_t k}(2^{d_t} i_1 + i_0)}$ **do**
9: $a_{2^{d_t k}(2^{d_t} i_1 + i_0)+j} \leftarrow c a_{2^{d_t k}(2^{d_t} i_1 + i_0)+j}$
10: $c \leftarrow (1/\delta_{t,k}) c$
11: **for** $i_0 = 0, \dots, r-1$ **do**
12: **for** $j \in J_{t,k,\ell-2^{d_t k}(2^{d_t} q + i_0)}$ **do**
13: $a_{2^{d_t k}(2^{d_t} q + i_0)+j} \leftarrow c a_{2^{d_t k}(2^{d_t} q + i_0)+j}$
14: **for** $j \in J_{t,k,\ell}$ **do**
15: InverseTaylorExpansion$(d_t, |I_{t,k,\ell-j}|, (a_{2^{d_t k} i + j})_{0\le i<|I_{t,k,\ell-j}|})$
16: **if** $\beta_0 \neq 1$ **and** $\ell > 1$ **then**
17: $c \leftarrow (1/\beta_0)$, $a_1 \leftarrow c a_1$
18: **for** $i = 2, \dots, \ell-1$ **do**
19: $c \leftarrow (1/\beta_0) c$, $a_i \leftarrow c a_i$

time the algorithm reaches Line 7, which happens for $t = s$ and $k = 0$, it follows that the algorithm terminates with $a_i = f_i^{(0, \lceil (\log_2 \ell)/d_0 \rceil - 1)}$ for $i \in \{0, \dots, \ell-1\}$. Property (5) of Lemma 5.1 implies that this is the correct output. □

Remark 5.8. By substituting property (3) of Corollary 2.2 for Lemma 5.4 in the proof of Lemma 5.1, it is possible to show that Algorithms 9 and 10 produce the correct output when β is a Cantor basis if one takes $m = \lceil \log_2 n \rceil$ and $d_t = 2^t$ for $t \in \{0, \dots, m\}$. The substitution is necessary as the requirements on the quotients $\beta_i/\beta_{d_t \lfloor i/d_t \rfloor}$ may not be met in this case. For example, if β is a Cantor basis with $n \ge 4$, then $\beta_3 \in \mathbb{F}_{2^4} \setminus \mathbb{F}_{2^2}$ [14, Lemma 3] and, thus, $\beta_3/\beta_{2\lfloor 3/2 \rfloor} = \beta_3/\beta_2 = 1/(\beta_3 + 1) \notin \mathbb{F}_{2^2}$. For the special case of $\ell = 2^n$, the algorithms of Lin, Al-Naffouri, Han and Chung [20] are recovered, but written as iterative algorithms rather than recursive algorithms.

The following theorem bounds the number of additions and multiplications performed by Algorithms 9 and 10.

Theorem 5.9. *Suppose that* $2^{d_s} < \ell \le 2^{d_{s+1}}$ *for some* $s \in \{0, \dots, m-1\}$. *Then Algorithms 9 and 10 perform at most*

$$\frac{1}{2}\left\lfloor \frac{\ell}{2} \right\rfloor \left(\lceil \log_2 \ell \rceil \left(\left\lceil \frac{\log_2 \ell}{d_s} \right\rceil - 1 \right) + \sum_{t=0}^{s-1} d_t \left\lceil \frac{\log_2 \ell}{d_t} \right\rceil \left(\frac{d_{t+1}}{d_t} - 1 \right) \right)$$

additions in $\mathbb{F}$, *and at most* $(\ell - 1)(\lceil \log_2 \ell \rceil + 1) - 1$ *multiplications in* $\mathbb{F}$.

Theorem 5.9 is proved in Section 5.4. Recall that for an arbitrarily chosen β, the algorithms of Lin et al. [20] allow polynomials in $\mathbb{F}[x]_{2^n}$ to be converted between the LCH and monomial bases with $\mathcal{O}(2^n n^2)$ additions and $\mathcal{O}(2^n n)$ multiplications. The corresponding parameters for Algorithms 9 and 10 are $m = 1$ and $\ell = 2^n$, for which the algorithms perform the same patterns of Taylor expansions and inverse Taylor expansions as their counterparts in [20]. This resemblance between the two families of algorithms results in our algorithms yielding the same big-O complexity bounds for this case, as shown by the following corollary.

Corollary 5.10. *If $m = 1$, then Algorithms 9 and 10 perform at most $\lfloor \ell/2 \rfloor \binom{\lceil \log_2 \ell \rceil}{2}$ additions in $\mathbb{F}$, and at most $(\ell - 1)(\lceil \log_2 \ell \rceil + 1)$ multiplications in $\mathbb{F}$.*

Proof. Algorithms 9 and 10 perform no multiplications if $\ell = 1$, at most one multiplication if $\ell = 2$, and no additions for $\ell \leq 2$. Thus, the bounds hold for $\ell \leq 2$. If $m = 1$ and $\ell > 2$, then Theorem 5.9 can be applied with $s = 0$, which yields the stated bounds. □

The algorithms of Lin et al. for Cantor bases perform only $\mathcal{O}(2^n n \log n)$ additions and no multiplications. As noted in Remark 5.8, our algorithms reduce to their algorithms when applied to this case. The following corollary of Theorem 5.9 confirms that we obtain the same big-O complexities.

Corollary 5.11. *Suppose that $d_t = 2^t$ for $t \in \{0, \ldots, m\}$. Then Algorithms 9 and 10 perform at most $\lfloor \ell/2 \rfloor (\lceil \log_2(\ell/2) \rceil \lceil \log_2 \log_2 \max(\ell, 2) \rceil + \lceil \log_2 \ell \rceil)/2$ additions in $\mathbb{F}$. Moreover, if $\beta_0 = 1$ and*

$$\operatorname{Tr}_{\mathbb{F}_{2^{d_t+1}}/\mathbb{F}_{2^{d_t}}}\left(\frac{\beta_{d_t(k+1)}}{\beta_{d_t k}}\right) = 1$$

for $t \in \{0, \ldots, m-1\}$ and even $k \in \{0, \ldots, \lceil (\log_2 \ell)/d_t \rceil - 2\}$, then the algorithms perform no multiplications in $\mathbb{F}$.

Proof. Suppose that $d_t = 2^t$ for $t \in \{0, \ldots, m\}$. Then the bound on the number of additions performed by the algorithms holds trivially if $\ell \leq 2$. If $\ell > 2$, then Theorem 5.9 can be applied with $s = \lceil \log_2 \log_2 \ell \rceil - 1$, from which the stated bound is obtained by observing that

$$\sum_{t=0}^{s-1} d_t \left\lceil \frac{\log_2 \ell}{d_t} \right\rceil \leq s(\lceil \log_2 \ell \rceil - 1) + 2^s - 1 \leq \lceil \log_2(\ell/2) \rceil \lceil \log_2 \log_2 \ell \rceil$$

and $\lceil (\log_2 \ell)/d_s \rceil = 2$. If $\beta_0 = 1$, then Lines 1 to 4 of Algorithm 9 and Lines 16 to 19 of Algorithm 10 perform no multiplications. Similarly, if the trace condition of the corollary is met, then Lines 5 to 19 of Algorithm 9 and Lines 1 to 15 of Algorithm 10 perform no multiplications, since property (3) of Lemma 5.1 implies that $\delta_{t,k} = 1$ for $t \in \{0, \ldots, m-1\}$ and $k \in \{0, \ldots, \lceil (\log_2 \ell)/d_t \rceil - 2\}$ such that $d_{t+1}/d_t = 2$ does not divide $k + 1$. □

Remark 5.12. If $d_{t+1}/d_t = 2$ for some $t \in \{0, \ldots, m-1\}$, then property (3) of Lemma 5.1 implies that $\delta_{t,k} \in \mathbb{F}_{2^{d_t}}$ for even $k \in \{0, \ldots, \lceil n/d \rceil - 2\}$. Consequently, if some of these elements cannot be forced to equal one, then it may still be possible to reduce the cost of the corresponding multiplications in Lines 5 to 19 of Algorithm 9 (similarly, Lines 1 to 15 of Algorithm 10) by choosing a representation of the elements of the field that lowers the cost of multiplying by elements

of the subfield $\mathbb{F}_{2^{d_t}}$. Such optimisations have previously been shown to be beneficial in practice, particularly for multiplications by elements of small subfields, by Bernstein and Chou [3] and Chen et al. [7].

For $m = 1$, Algorithms 9 and 10 are based on the same ideas as the algorithms of Lin et al. [20], which in turn are based on ideas from the additive FFT of Gao and Mateer [14]. Thus, comparing our own algorithms against this case provides some indication of the benefits granted by the techniques of this section. The bound on the number of additions performed by Algorithms 9 and 10 provided by Theorem 5.9 is attained for $\ell = 2^{d_m}$. Thus, for sufficiently large ℓ, say for $\ell > 2^{d_{m-1}}$, the theorem suggests that the relative number of additions performed by the algorithms, when given as a fraction of the number of additions performed for the trivial choice of tower $\mathbb{F}_{2^{d'_0}} \subset \mathbb{F}_{2^{d'_1}}$ with $d'_0 = 1$ and $d'_1 = d_m$, is approximately

$$\frac{1}{d_m - 1} \sum_{t=0}^{m-1} \left(\frac{d_{t+1}}{d_t} - 1 \right). \tag{5.6}$$

The following example shows that this estimate is reasonably accurate, and provides a concrete example of the reduction in additive complexity afforded by utilising additional subfields when they are available. The example similarly demonstrates the reduction in multiplicative complexity afforded by the use of additional subfields, which is not exhibited by the somewhat crude bound of Theorem 5.9.

Example 5.13. Suppose that $\mathbb{F}_{2^{12}} \subseteq \mathbb{F}$. Then there are eight tuples of positive integers $(d_0, \ldots, d_m)$ such that $d_m = 12$ and (5.1) holds. For each such tuple, Figure 1 displays the relative number of additions performed by Algorithms 9 and 10, given as a fraction of the number performed for the tuple $(1, 12)$, as the polynomial length (ℓ) ranges over $\{3, \ldots, 4096\}$. The estimate (5.6) of the relative number of additions performed for large ℓ is equal to $6/11 \approx 0.55$ for $(d_0, \ldots, d_m) \in \{(1, 6, 12), (1, 2, 12)\}$, $5/11 \approx 0.45$ for $(d_0, \ldots, d_m) \in \{(1, 4, 12), (1, 3, 12)\}$, and $4/11 \approx 0.36$ for $(d_0, \ldots, d_m) \in \{(1, 3, 6, 12), (1, 2, 6, 12), (1, 2, 4, 12)\}$.

Figure 2 displays the relative number of multiplications performed by Algorithms 9 and 10, once again given as a fraction of the number performed for $(1, 12)$. To highlight the effect of the choice of tower, it is assumed that $\beta_0 = 1$ in all cases, since the number of multiplications by powers of β_0 or $1/\beta_0$ performed by Lines 1 to 4 of Algorithm 9 and Lines 16 to 19 of Algorithm 10 is independent of the tower. Tuples without daggers assume that $\delta_{t,k}$ is never equal to one. Tuples with daggers assume that $\delta_{t,k}$ is equal to one if and only if $d_{t+1}/d_t = 2$, in order to demonstrate the benefit of choosing β so that $\mathrm{Tr}_{\mathbb{F}_{2^{d_{t+1}}}/\mathbb{F}_{2^{d_t}}}(\beta_{d_t(k+1)}/\beta_{d_t k}) = 1$ for $t \in \{0, \ldots, m-1\}$ such that $d_{t+1}/d_t = 2$, and even $k \in \{0, \ldots, \lceil (\log_2 \ell)/d_t \rceil - 2\}$.

5.4. **Proof of Theorem 5.9.** We split the proof of Theorem 5.9 into five lemmas, with the first four lemmas dedicated to bounding the number of additions performed by Algorithms 9 and 10.

Lemma 5.14. *If $2^{d_s} < \ell \leq 2^{d_{s+1}}$ for some $s \in \{0, \ldots, m-1\}$, then Algorithms 9 and 10 perform at most*

$$\sum_{t=0}^{s} \sum_{\substack{k=0 \\ d_{t+1}/d_t \nmid k+1}}^{\lceil (\log_2 \ell)/d_t \rceil - 2} \sum_{j \in J_{t,k,\ell}} \left\lfloor \frac{|I_{t,k,\ell-j}|}{2} \right\rfloor \left\lceil \log_2 \left\lceil \frac{|I_{t,k,\ell-j}|}{2^{d_t}} \right\rceil \right\rceil$$

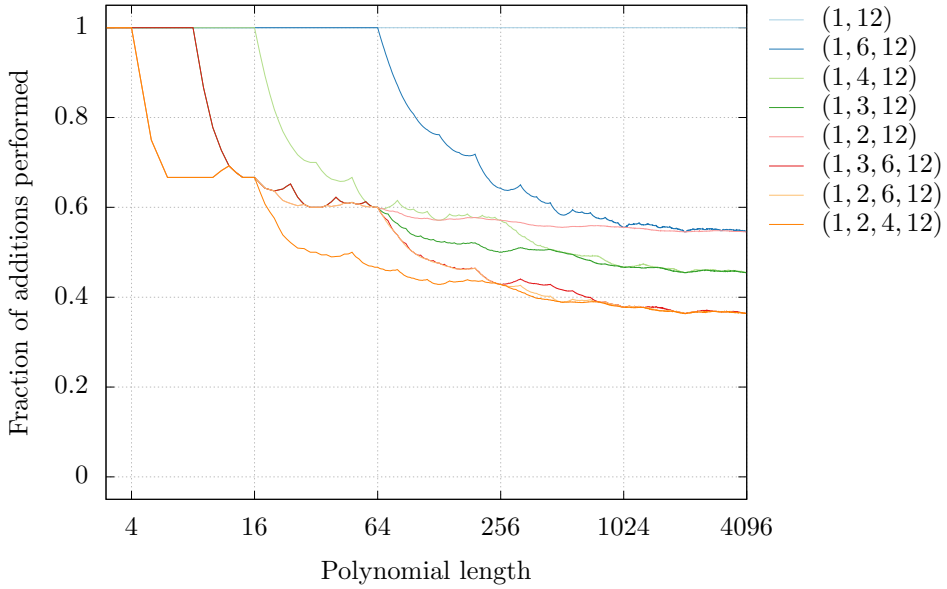

FIGURE 1. Relative number of additions for Example 5.13.

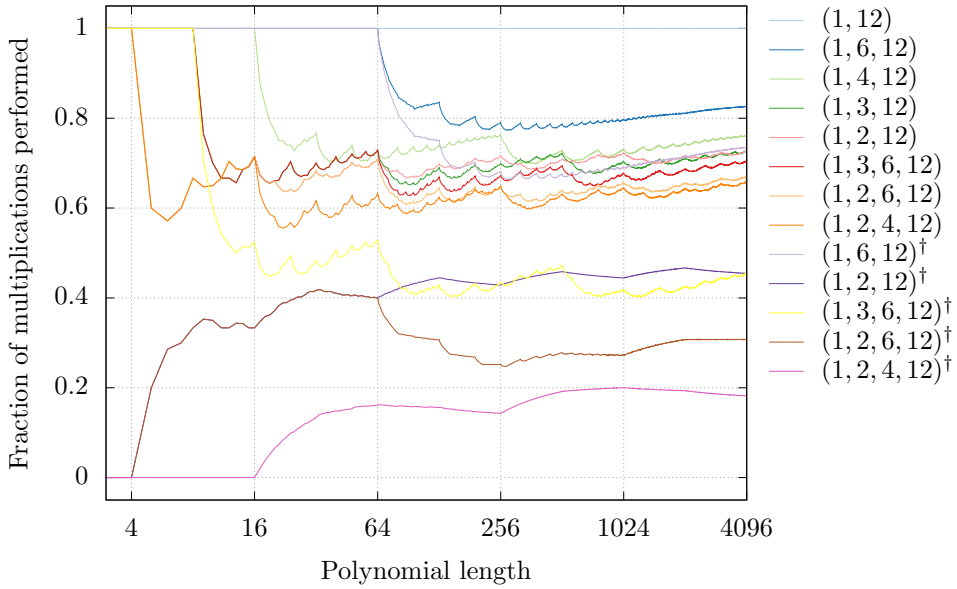

FIGURE 2. Relative number of multiplications for Example 5.13.

additions in $\mathbb{F}$.

Proof. Lemma 5.6 implies that Algorithms 9 and 10 perform at most

$$\sum_{t=0}^{m-1} \sum_{\substack{k=0 \\ d_{t+1}/d_t \nmid k+1}}^{\lceil (\log_2 \ell)/d_t \rceil - 2} \sum_{j \in J_{t,k,\ell}} \left\lfloor \frac{|I_{t,k,\ell-j}|}{2} \right\rfloor \left\lceil \log_2 \left\lceil \frac{|I_{t,k,\ell-j}|}{2^{d_t}} \right\rceil \right\rceil$$

additions in $\mathbb{F}$. If there exists $s \in \{0, \ldots, m-1\}$ such that $2^{d_s} < \ell \leq 2^{d_{s+1}}$, then the bound is equal to that of the lemma since $\lceil (\log_2 \ell)/d_t \rceil < 2$ for $t \in \{s+1, \ldots, m-1\}$. □

We now bound the inner-most sums of the bound from Lemma 5.14, before working our way out to obtain the bound of Theorem 5.9.

Lemma 5.15. *If* $\ell \in \{1, \ldots, 2^n\}$, $t \in \{0, \ldots, m-1\}$ *and* $k \in \{0, \ldots, \lceil (\log_2 \ell)/d_t \rceil - 2\}$, *then*

$$\sum_{j \in J_{t,k,\ell}} \left\lfloor \frac{|I_{t,k,\ell-j}|}{2} \right\rfloor \left\lceil \log_2 \left\lceil \frac{|I_{t,k,\ell-j}|}{2^{d_t}} \right\rceil \right\rceil \leq 2^{d_{t+1}e_{t,k}-1} q(d_{t+1}e_{t,k} - d_t(k+1)) + \left\lfloor \frac{r}{2} \right\rfloor \left\lceil \log_2 \left\lceil \frac{r}{2^{d_t(k+1)}} \right\rceil \right\rceil,$$

where $q = \lceil \ell / 2^{d_{t+1}e_{t,k}} \rceil - 1$ *and* $r = \ell - 2^{d_{t+1}e_{t,k}} q$.

Proof. Suppose that $\ell \in \{1, \ldots, 2^n\}$, $t \in \{0, \ldots, m-1\}$ and $k \in \{0, \ldots, \lceil (\log_2 \ell)/d_t \rceil - 2\}$. Let $q = \lceil \ell / 2^{d_{t+1}e_{t,k}} \rceil - 1$ and $r = \ell - 2^{d_{t+1}e_{t,k}} q$. Then, as $r \geq 1$, $j \in J_{t,k,\ell}$ if and only if $j = 2^{d_{t+1}e_{t,k}} j_1 + j_0$ for some $j_1 \in \{0, \ldots, q\}$ and $j_0 \in \{0, \ldots, \min(2^{d_t k}, \ell - 2^{d_{t+1}e_{t,k}} j_1) - 1\}$. Moreover, if $j \in J_{t,k,\ell}$ is written in this form, then

$$|I_{t,k,\ell-j}| = \begin{cases} 2^{d_{t+1}e_{t,k} - d_t k} & \text{if } j_1 < q, \\ |I_{t,k,r-j_0}| & \text{if } j_1 = q. \end{cases}$$

Thus, the bound of the lemma holds if and only if

$$\sum_{j=0}^{\min(2^{d_t k}, r) - 1} \left\lfloor \frac{|I_{t,k,r-j}|}{2} \right\rfloor \left\lceil \log_2 \left\lceil \frac{|I_{t,k,r-j}|}{2^{d_t}} \right\rceil \right\rceil \leq \left\lfloor \frac{r}{2} \right\rfloor \left\lceil \log_2 \left\lceil \frac{r}{2^{d_t(k+1)}} \right\rceil \right\rceil. \tag{5.7}$$

To prove this inequality, we may assume that $r \geq 2^{d_t k}$, since otherwise both sides of the inequality are zero. Let $u = \lfloor r / 2^{d_t k} \rfloor$ and $v = r - 2^{d_t k} u$. Then, as $r \leq 2^{d_{t+1}e_{t,k}}$,

$$|I_{t,k,r-j}| = u + \left\lceil \frac{v-j}{2^{d_t k}} \right\rceil = \begin{cases} u+1 & \text{if } j < v, \\ u & \text{if } j \geq v, \end{cases}$$

for $j \in \{0, \ldots, 2^{d_t k} - 1\}$. Consequently, the left-hand side of (5.7) is equal to

$$v \left\lfloor \frac{u+1}{2} \right\rfloor \left\lceil \log_2 \left\lceil \frac{u+1}{2^{d_t}} \right\rceil \right\rceil + \left(2^{d_t k} - v\right) \left\lfloor \frac{u}{2} \right\rfloor \left\lceil \log_2 \left\lceil \frac{u}{2^{d_t}} \right\rceil \right\rceil.$$

We have $u = r / 2^{d_t k}$ if $v = 0$, and $u + 1 = \lceil r / 2^{d_t k} \rceil$ if $v \neq 0$. Thus, $\lceil u / 2^{d_t} \rceil = \lceil r / 2^{d_t(k+1)} \rceil$ if $v = 0$, and $\lceil (u+1) / 2^{d_t} \rceil = \lceil r / 2^{d_t(k+1)} \rceil$ if $v \neq 0$. It follows that the left-hand side of (5.7) is less than or equal to

$$\left\lfloor \frac{v(u+1) + (2^{d_t k} - v)u}{2} \right\rfloor \left\lceil \log_2 \left\lceil \frac{r}{2^{d_t(k+1)}} \right\rceil \right\rceil = \left\lfloor \frac{r}{2} \right\rfloor \left\lceil \log_2 \left\lceil \frac{r}{2^{d_t(k+1)}} \right\rceil \right\rceil.$$

Therefore, (5.7) holds, which completes the proof of the lemma. □

Combining the following two lemmas with Lemma 5.14 completes the proof of the addition bound of Theorem 5.9.

Lemma 5.16. *For $\ell \in \{1, \dots, 2^n\}$ and $t \in \{0, \dots, m-1\}$ such that $2^{d_{t+1}} < \ell$,*

$$\sum_{\substack{k=0 \\ d_{t+1}/d_t \nmid k+1}}^{\lceil (\log_2 \ell)/d_t \rceil - 2} \sum_{j \in J_{t,k,\ell}} \left\lfloor \frac{|I_{t,k,\ell-j}|}{2} \right\rfloor \left\lceil \log_2 \left\lceil \frac{|I_{t,k,\ell-j}|}{2^{d_t}} \right\rceil \right\rceil \leq \left\lfloor \frac{\ell}{2} \right\rfloor \frac{d_t}{2} \left\lceil \frac{\log_2 \ell}{d_t} \right\rceil \left(\frac{d_{t+1}}{d_t} - 1 \right).$$

Proof. Suppose that $\ell \in \{1, \dots, 2^n\}$ and $t \in \{0, \dots, m-1\}$ such that $2^{d_{t+1}} < \ell$. Then, for $k \in \{0, \dots, \lceil (\log_2 \ell)/d_t \rceil - 2\}$, the integers $q = \lceil \ell / 2^{d_{t+1} e_{t,k}} \rceil - 1$ and $r = \ell - 2^{d_{t+1} e_{t,k}} q$ satisfy $r \leq 2^{d_{t+1} e_{t,k}}$ and $2^{d_{t+1} e_{t,k} - 1} q + \lfloor r/2 \rfloor = \lfloor \ell / 2 \rfloor$. Thus, Lemma 5.15 implies that

$$\sum_{j \in J_{t,k,\ell}} \left\lfloor \frac{|I_{t,k,\ell-j}|}{2} \right\rfloor \left\lceil \log_2 \left\lceil \frac{|I_{t,k,\ell-j}|}{2^{d_t}} \right\rceil \right\rceil \leq \left\lfloor \frac{\ell}{2} \right\rfloor (d_{t+1} e_{t,k} - d_t(k+1)) \tag{5.8}$$

for $k \in \{0, \dots, \lceil (\log_2 \ell)/d_t \rceil - 2\}$. If $k \in \{0, \dots, \lceil (\log_2 \ell)/d_t \rceil - 2\}$ such that d_{t+1}/d_t does not divide $k+1$, then $k = k_1(d_{t+1}/d_t) + k_0$ for nonnegative integers $k_1 < (d_t/d_{t+1}) \lceil (\log_2 \ell)/d_t \rceil$ and $k_0 < d_{t+1}/d_t - 1$. Moreover, when k is written in this form, we have

$$d_{t+1} e_{t,k} - d_t(k+1) = d_{t+1} \left\lceil \frac{d_t(k_0+1)}{d_{t+1}} \right\rceil - d_t(k_0+1) = d_{t+1} - d_t(k_0+1).$$

As $(d_t/d_{t+1}) \lceil (\log_2 \ell)/d_t \rceil > (d_t/d_{t+1}) \lceil d_{t+1}/d_t \rceil = 1$, it follows by substituting into (5.8) and summing the resulting inequalities that

$$\begin{aligned} &\sum_{\substack{k=0 \\ d_{t+1}/d_t \nmid k+1}}^{\lceil (\log_2 \ell)/d_t \rceil - 2} \sum_{j \in J_{t,k,\ell}} \left\lfloor \frac{|I_{t,k,\ell-j}|}{2} \right\rfloor \left\lceil \log_2 \left\lceil \frac{|I_{t,k,\ell-j}|}{2^{d_t}} \right\rceil \right\rceil \\ &\qquad \leq \left\lfloor \frac{\ell}{2} \right\rfloor \frac{d_t}{d_{t+1}} \left\lceil \frac{\log_2 \ell}{d_t} \right\rceil \sum_{k_0=0}^{d_{t+1}/d_t - 2} d_{t+1} - d_t(k_0+1) \\ &\qquad = \left\lfloor \frac{\ell}{2} \right\rfloor \frac{d_t}{2} \left\lceil \frac{\log_2 \ell}{d_t} \right\rceil \left(\frac{d_{t+1}}{d_t} - 1 \right), \end{aligned}$$

which completes the proof of the lemma. □

Lemma 5.17. *For $\ell \in \{1, \dots, 2^n\}$ and $s \in \{0, \dots, m-1\}$ such that $2^{d_s} < \ell \leq 2^{d_{s+1}}$,*

$$\sum_{\substack{k=0 \\ d_{s+1}/d_s \nmid k+1}}^{\lceil (\log_2 \ell)/d_s \rceil - 2} \sum_{j \in J_{s,k,\ell}} \left\lfloor \frac{|I_{s,k,\ell-j}|}{2} \right\rfloor \left\lceil \log_2 \left\lceil \frac{|I_{s,k,\ell-j}|}{2^{d_s}} \right\rceil \right\rceil \leq \left\lfloor \frac{\ell}{2} \right\rfloor \frac{\lceil \log_2 \ell \rceil}{2} \left(\left\lceil \frac{\log_2 \ell}{d_s} \right\rceil - 1 \right).$$

Proof. Suppose that $\ell \in \{1, \dots, 2^n\}$ and $s \in \{0, \dots, m-1\}$ satisfy $2^{d_s} < \ell \leq 2^{d_{s+1}}$. Then, for $k \in \{0, \dots, \lceil (\log_2 \ell)/d_s \rceil - 2\}$, we have $q = \lceil \ell / 2^{d_{s+1} e_{s,k}} \rceil - 1 = 0$ and $r = \ell - 2^{d_{s+1} e_{s,k}} q = \ell > 2^{d_s(k+1)}$. Thus, Lemma 5.15 implies that

$$\begin{aligned} \sum_{j \in J_{s,k,\ell}} \left\lfloor \frac{|I_{s,k,\ell-j}|}{2} \right\rfloor \left\lceil \log_2 \left\lceil \frac{|I_{s,k,\ell-j}|}{2^{d_s}} \right\rceil \right\rceil &\leq \left\lfloor \frac{\ell}{2} \right\rfloor \left\lceil \log_2 \left\lceil \frac{\ell}{2^{d_s(k+1)}} \right\rceil \right\rceil \\ &= \left\lfloor \frac{\ell}{2} \right\rfloor (\lceil \log_2 \ell \rceil - d_t(k+1)) \end{aligned}$$

for $k \in \{0, \ldots, \lceil (\log_2 \ell)/d_s \rceil - 2\}$. Moreover, d_{s+1}/d_s does not divide $k+1$ for $k \in \{0, \ldots, \lceil (\log_2 \ell)/d_s \rceil - 2\}$, since $\lceil (\log_2 \ell)/d_s \rceil - 1 < d_{s+1}/d_s$. It follows that

$$\begin{aligned}\sum_{\substack{k=0 \\ d_{s+1}/d_s \nmid k+1}}^{\lceil (\log_2 \ell)/d_s \rceil - 2} \sum_{j \in J_{s,k,\ell}} \left\lfloor \frac{|I_{s,k,\ell-j}|}{2} \right\rfloor \left\lceil \log_2 \left\lceil \frac{|I_{s,k,\ell-j}|}{2^{d_s}} \right\rceil \right\rceil \\ \leq \left\lfloor \frac{\ell}{2} \right\rfloor \left(\left\lceil \frac{\log_2 \ell}{d_s} \right\rceil - 1 \right) \left(\lceil \log_2 \ell \rceil - \frac{d_s}{2} \left\lceil \frac{\log_2 \ell}{d_s} \right\rceil \right) \\ \leq \left\lfloor \frac{\ell}{2} \right\rfloor \left(\left\lceil \frac{\log_2 \ell}{d_s} \right\rceil - 1 \right) \frac{\lceil \log_2 \ell \rceil}{2},\end{aligned}$$

which completes the proof of the lemma. □

We now complete the proof of Theorem 5.9 by bounding the number of multiplications performed by Algorithms 9 and 10.

Lemma 5.18. *If $\ell > 1$, then Algorithms 9 and 10 perform at most $(\ell-1)(\lceil \log_2 \ell \rceil + 1) - 1$ multiplications in $\mathbb{F}$.*

Proof. Suppose that $\ell > 1$. Then Lines 1 to 4 of Algorithm 9 perform at most $2(\ell-1)-1$ multiplications, while Lines 5 to 19 perform at most

$$\sum_{t=0}^{m-1} \sum_{\substack{k=0 \\ d_{t+1}/d_t \nmid k+1}}^{\lceil (\log_2 \ell)/d_t \rceil - 2} \left(\left\lceil \frac{|I_{t,k,\ell}|}{2^{d_t}} \right\rceil - 2 + \sum_{i=2^{d_t}}^{|I_{t,k,\ell}|-1} \left| J_{t,k,\ell-2^{d_t k} i} \right| \right)$$

multiplications. The same bounds hold respectively for Lines 16 to 19 and Lines 1 to 15 of Algorithm 10. If $t \in \{0, \ldots, m-1\}$ and $k \in \{0, \ldots, \lceil (\log_2 \ell)/d_t \rceil - 2\}$ such that d_{t+1}/d_t does not divide $k+1$, then $\ell > 2^{d_t(k+1)}$ and $d_{t+1} e_{t,k} - d_t k > d_t$. For such t and k, it follows that $|I_{t,k,\ell}| > 2^{d_t}$ and $\left| J_{t,k,\ell-2^{d_t k} i} \right| \geq 2^{d_t k}$ for $i \in \{0, \ldots, 2^{d_t} - 1\}$, since $\ell - 2^{d_t k} i \geq 2^{d_t k}$ for $i \leq 2^{d_t} - 1$. Thus,

$$\sum_{i=2^{d_t}}^{|I_{t,k,\ell}|-1} \left| J_{t,k,\ell-2^{d_t k} i} \right| = \sum_{i \in I_{t,k,\ell}} \left| J_{t,k,\ell-2^{d_t k} i} \right| - \sum_{i=0}^{2^{d_t}-1} \left| J_{t,k,\ell-2^{d_t k} i} \right| \leq \ell - 2^{d_t(k+1)}$$

and

$$\left\lceil \frac{|I_{t,k,\ell}|}{2^{d_t}} \right\rceil - 1 \leq \left\lceil \frac{\lceil \ell / 2^{d_t k} \rceil}{2^{d_t}} \right\rceil - 1 \leq \frac{\ell}{2^{d_t(k+1)}}$$

for $t \in \{0, \ldots, m-1\}$ and $k \in \{0, \ldots, \lceil (\log_2 \ell)/d_t \rceil - 2\}$ such that d_{t+1}/d_t does not divide $k+1$. By combining these inequalities with (5.3), it follows that Algorithms 9 and 10 perform at most

$$(\ell-1)(\lceil \log_2 \ell \rceil + 1) - 1 + \sum_{t=0}^{m-1} \sum_{\substack{k=0 \\ d_{t+1}/d_t \nmid k+1}}^{\lceil (\log_2 \ell)/d_t \rceil - 2} \left(\frac{\ell}{2^{d_t(k+1)}} - 2^{d_t(k+1)} \right)$$

multiplications. Finally, we have

$$\sum_{t=0}^{m-1} \sum_{\substack{k=0 \\ d_{t+1}/d_t \nmid k+1}}^{\lceil (\log_2 \ell)/d_t \rceil - 2} \left(\frac{\ell}{2^{d_t(k+1)}} - 2^{d_t(k+1)} \right) = \left(\frac{\ell}{2^{\lceil \log_2 \ell \rceil}} - 1 \right) \sum_{k=1}^{\lceil \log_2 \ell \rceil - 1} 2^k \leq 0,$$

since the sets $\{d_t(k+1) \mid k \in \{0, \dots, \lceil (\log_2 \ell)/d_t \rceil - 2\}, d_{t+1}/d_t \nmid k+1\}$ for $t \in \{0, \dots, m-1\}$ are pairwise disjoint and their union is $\{1, \dots, \lceil \log_2 \ell \rceil - 1\}$. □

References

1. Eli Ben-Sasson, Iddo Bentov, Alessandro Chiesa, Ariel Gabizon, Daniel Genkin, Matan Hamilis, Evgenya Pergament, Michael Riabzev, Mark Silberstein, Eran Tromer, and Madars Virza, *Computational integrity with a public random string from quasi-linear PCPs*, Advances in cryptology—EUROCRYPT 2017. Part III, Lecture Notes in Comput. Sci., vol. 10212, Springer, Cham, 2017, pp. 551–579.
2. Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev, *Scalable, transparent, and post-quantum secure computational integrity*, Cryptology ePrint Archive, Report 2018/046, 2018, `https://eprint.iacr.org/2018/046`.
3. Daniel J. Bernstein and Tung Chou, *Faster binary-field multiplication and faster binary-field MACs*, Selected areas in cryptography—SAC 2014, Lecture Notes in Comput. Sci., vol. 8781, Springer, Cham, 2014, pp. 92–111.
4. Daniel J. Bernstein, Tung Chou, and Peter Schwabe, *McBits: Fast constant-time code-based cryptography*, Cryptographic Hardware and Embedded Systems—CHES 2013, Lecture Notes in Comput. Sci., vol. 8086, Springer, Berlin, 2013, pp. 250–272.
5. Richard P. Brent, Pierrick Gaudry, Emmanuel Thomé, and Paul Zimmermann, *Faster multiplication in* GF(2)[x], Algorithmic number theory—ANTS 2008, Lecture Notes in Comput. Sci., vol. 5011, Springer, Berlin, 2008, pp. 153–166.
6. David G. Cantor, *On arithmetical algorithms over finite fields*, J. Combin. Theory Ser. A **50** (1989), no. 2, 285–300.
7. Ming-Shing Chen, Chen-Mou Cheng, Po-Chun Kuo, Wen-Ding Li, and Bo-Yin Yang, *Faster multiplication for long binary polynomials*, 2017, `arXiv:1708.09746 [cs.SC]`.
8. Ming-Shing Chen, Chen-Mou Cheng, Po-Chun Kuo, Wen-Ding Li, and Bo-Yin Yang, *Multiplying boolean polynomials with Frobenius partitions in additive fast Fourier transform*, 2018, `arXiv:1803.11301 [cs.SC]`.
9. Ming-Shing Chen, Wen-Ding Li, Bo-Yuan Peng, Bo-Yin Yang, and Chen-Mou Cheng, *Implementing 128-bit secure MPKC signatures*, Cryptology ePrint Archive, Report 2017/636, 2017, `https://eprint.iacr.org/2017/636`.
10. Tung Chou, *McBits revisited*, Cryptographic Hardware and Embedded Systems—CHES 2017, Lecture Notes in Comput. Sci., vol. 10529, Springer, Cham, 2017, pp. 213–231.
11. Nicholas Coxon, *Fast Hermite interpolation and evaluation over finite fields of characteristic two*, J. Symbolic Comput., to appear.
12. Nicholas Coxon, *Fast systematic encoding of multiplicity codes*, J. Symbolic Comput. **94** (2019), 234–254.
13. N. J. Fine, *Binomial coefficients modulo a prime*, Amer. Math. Monthly **54** (1947), 589–592.
14. Shuhong Gao and Todd Mateer, *Additive fast Fourier transforms over finite fields*, IEEE Trans. Inform. Theory **56** (2010), no. 12, 6265–6272.
15. David Harvey, *A cache-friendly truncated FFT*, Theoret. Comput. Sci. **410** (2009), no. 27-29, 2649–2658.
16. David Harvey and Daniel S. Roche, *An in-place truncated Fourier transform and applications to polynomial multiplication*, ISSAC 2010—Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation, ACM, New York, 2010, pp. 325–329.
17. Robin Larrieu, *The truncated Fourier transform for mixed radices*, ISSAC'17—Proceedings of the 2017 ACM International Symposium on Symbolic and Algebraic Computation, ACM, New York, 2017, pp. 261–268.
18. Wen-Ding Li, Ming-Shing Chen, Po-Chun Kuo, Chen-Mou Cheng, and Bo-Yin Yang, *Frobenius additive fast fourier transform*, Proceedings of the 2018 ACM International Symposium on Symbolic and Algebraic Computation (New York, NY, USA), ISSAC '18, ACM, 2018, pp. 263–270.
19. Sian-Jheng Lin, Tareq Y. Al-Naffouri, and Yunghsiang S. Han, *FFT algorithm for binary extension finite fields and its application to Reed-Solomon codes*, IEEE Trans. Inform. Theory **62** (2016), no. 10, 5343–5358.

20. Sian-Jheng Lin, Tareq Y. Al-Naffouri, Yunghsiang S. Han, and Wei-Ho Chung, *Novel polynomial basis with fast Fourier transform and its application to Reed–Solomon erasure codes*, IEEE Trans. Inform. Theory **62** (2016), no. 11, 6284–6299.
21. Sian-Jheng Lin, Wei-Ho Chung, and Yunghsiang S. Han, *Novel polynomial basis and its application to Reed-Solomon erasure codes*, 55th Annual IEEE Symposium on Foundations of Computer Science—FOCS 2014, IEEE Computer Soc., Los Alamitos, CA, 2014, pp. 316–325.
22. Edouard Lucas, *Théorie des Fonctions Numériques Simplement Périodiques. [Continued]*, Amer. J. Math. **1** (1878), no. 3, 197–240.
23. J. Markel, *FFT pruning*, IEEE Transactions on Audio and Electroacoustics **19** (1971), no. 4, 305–311.
24. Todd Mateer, *Fast Fourier Transform algorithms with applications*, ProQuest LLC, Ann Arbor, MI, 2008, Ph.D. thesis–Clemson University.
25. H. V. Sorensen and C. S. Burrus, *Efficient computation of the DFT with only a subset of input or output points*, IEEE Transactions on Signal Processing **41** (1993), no. 3, 1184–1200.
26. J. van der Hoeven, *Notes on the Truncated Fourier Transform*, Tech. Report 2005-5, Université Paris-Sud, Orsay, France, 2005.
27. Joris van der Hoeven, *The truncated Fourier transform and applications*, ISSAC 2004—Proceedings of the 2004 international symposium on Symbolic and algebraic computation, ACM, New York, 2004, pp. 290–296.
28. Joris van der Hoeven and Éric Schost, *Multi-point evaluation in higher dimensions*, Appl. Algebra Engrg. Comm. Comput. **24** (2013), no. 1, 37–52.
29. Joachim von zur Gathen, *Functional decomposition of polynomials: the tame case*, J. Symbolic Comput. **9** (1990), no. 3, 281–299.
30. Joachim von zur Gathen and Jürgen Gerhard, *Arithmetic and factorization of polynomial over $\mathbb{F}_2$ (extended abstract)*, ISSAC '96—Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation, ACM, New York, 1996, pp. 1–9.
31. Y. Wang and X. Zhu, *A fast algorithm for the Fourier transform over finite fields and its VLSI implementation*, IEEE Journal on Selected Areas in Communications **6** (1988), no. 3, 572–577.

INRIA Saclay–Île-de-France & Laboratoire d'Informatique, École polytechnique, 91128 Palaiseau Cedex, France