A new Truncated Fourier Transform algorithm*

Andrew Arnold
Symbolic Computation Group
University of Waterloo
a.arnold55@gmail.com
http://cs.uwaterloo.ca/~a4arnold/

ABSTRACT
Truncated Fourier Transforms (TFTs), first introduced by Van der Hoeven, refer to a family of algorithms that attempt to smooth "jumps" in complexity exhibited by FFT algorithms. We present an in-place TFT whose time complexity, measured in terms of ring operations, is comparable to existing not-in-place TFT methods. We also describe a transformation that maps between two families of TFT algorithms that use different sets of evaluation points.

1. INTRODUCTION

Let \( R \) be a ring containing an \( N \)-th principal root of unity \( \omega \). Given two polynomials \( f, g \in R[z] \), \( \deg(fg) < N \), we can compute \( fg \) by way of the Discrete Fourier Transform (DFT): a linear, invertible map which evaluates a given polynomial at the powers of \( \omega \).

Computing the DFT naively is quadratic-time. If, however, \( N \) is comprised of strictly small prime factors, one can compute a DFT using \( \mathcal{O}(N \log N) \) arithmetic operations by way of the Fast Fourier Transform (FFT). The most widely-used FFT, the radix-2 FFT, requires that \( N \) is a power of two. To compute the DFT of an input of arbitrary size, one typically appends zeroes to the input to give it power-of-two length, then applies a radix-2 FFT. This results in significant jumps in the time and space complexities of the radix-2 FFT.

Truncated Fourier Transforms (TFTs) smooth these jumps in complexity. A TFT takes a length-\( n \) input, \( n \leq N \), and returns a size-\( n \) subset of its length-\( N \) DFT, with time complexity that grows comparatively smoothly with \( n \log n \).

Typically one chooses the first \( n \) entries of the DFT, with the DFT sorted in bit-reversed order. This is natural choice as it comprised the first \( n \) entries of the output of an in-place FFT, if no re-sorting is performed. We will call such a TFT the bit-reversed TFT.

Van der Hoeven [10] showed how one could obtain a polynomial \( f(z) \) from its bit-reversed TFT, provided one knows the terms of \( f(z) \) with degree at least \( n \). This allows for faster FFT-based polynomial multiplication, particularly for products whose degree is a power of two or slightly larger. Harvey and Roche showed further how the bit-reversed TFT transform can be made in-place, at the cost of a constant factor additional ring multiplications.

Mateer [7] devised a TFT algorithm based on a series of modular reductions, that acts as a preprocessor to the FFT. Mateer's TFT algorithm, which we discuss in section 3, breaks an input \( f(z) \) with \( \deg(f) < n \) into its images modulo cyclotomic polynomials of the form \( z^{k} + 1 \), \( k \) a power-of-two. We will call this TFT the cyclotomic TFT. In [8], Sergeev showed how the cyclotomic TFT can be made in-place, with time complexity comparable to not-in-place TFT algorithms. In section 4 we restate Sergeev's algorithm. In section 5 we present a new in-place algorithm, related to Sergeev's, for computing the cyclotomic TFT.

One caveat of the cyclotomic TFT is that different-sized inputs may use entirely different sets of evaluation points. This is problematic in applications to multivariate polynomial multiplication. In section 6 we show how an algorithm that computes the cyclotomic TFT can be modified to compute a bit-reversed TFT by way of an affine transformation.

As a proof of concept we implemented the algorithms introduced in this paper in python. These implementations can be found at the author's website at:

http://cs.uwaterloo.ca/~a4arnold/tft

2. PRELIMINARIES

2.1 The Discrete Fourier Transform

The Discrete Fourier Transform (DFT) of a polynomial \( f(z) \) is its vector of evaluations at the distinct powers of a root of unity. Specifically, if \( f(z) = \sum_{i=0}^{N-1} a_i z^i \) is a polynomial over a ring \( R \) containing an \( N \)-th primitive root of unity \( \omega \), then we define the discrete Fourier transform of \( f(z) \) as

\[
\text{DFT}_\omega(f) = \left( f(\omega^j) \right)_{0 \leq j < N}.
\]  

(1)

We treat the polynomial \( f \) and its vector of coefficients \( a = (a_0, a_1, \ldots, a_{N-1}) \) as equivalent and use the notation \( \text{DFT}_\omega(a) \) and \( \text{DFT}_\omega(f) \) interchangeably. If we take addition and multiplication to be pointwise in \( R^N \), then the map \( \text{DFT}_\omega : \mathbb{R}[z]/(z^N - 1) \to \mathbb{R}^N \) forms a ring homomorphism. If \( \omega \) is a principal root of unity, that is, for \( j \) not divisible by \( N \), \( \sum_{i=0}^{N-1} \omega^{ij} = 0 \), then \( \text{DFT}_\omega \) has an inverse.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ISSAC ’13 Boston, Massachusetts USA
Copyright 20XX ACM X-XXXXXX-XX-X/XXX ...$15.00.
map IDFT, : \( \mathbb{R}^N \to \mathbb{R}[z]/(z^N - 1) \), defined by
\[
\text{IDFT,}_\omega(\hat{a}) = \frac{1}{N} \text{DFT,}_\omega^{-1}(\hat{a}),
\]
where \( \hat{a} \in \mathbb{R}^N \) and we again treat a polynomial as equivalent to its vector of coefficients. This suggests a multiplication algorithm for \( f, g \in \mathbb{R}[z] \).

**Theorem 1** (The Convolution Theorem). Suppose \( \mathcal{R} \) is a ring containing an \( N \)-th principal root of unity \( \omega \) and let \( f, g \in \mathcal{R}[z] \). Then
\[
fg \equiv (\text{IDFT,}_\omega(f \cdot \text{DFT,}_\omega(g))) (\text{mod } (\mathcal{R}[z]/(z^N - 1)))
\]
where \( \equiv \) is the vector component-wise product, and, given two polynomials \( g(z), h(z) \in \mathcal{R}[z] \), \( g(z) \equiv h(z) \mod (z) \equiv \) the unique polynomial \( r(z) \) such that \( h(z) \equiv r(z) \mod (z) \).

Thus to multiply \( f \) and \( g \), we can choose \( N > \deg(fg) \) and an \( N \)-th principal root of unity \( \omega \in \mathcal{R} \), compute the length- \( N \) DFTs of \( f \) and \( g \), take their pointwise product, and take the inverse DFT of the pointwise product.

### 2.2 The Fast Fourier Transform

We can compute the Discrete Fourier \( \hat{a} \) transform of \( f(z) \), \( f \) reduced modulo \( (z^N - 1) \), by way of a Fast Fourier Transform (FFT). The FFT is believed to have been first discovered by Cooley and Tukey [1]. For a detailed history of the FFT we refer the reader to [3].

The simplest and most widely-used FFT, the radix-2 FFT, assumes \( N = 2^p \) for some \( p \in \mathbb{Z}_{>0} \). We describe the radix-2 FFT in terms of modular reductions. Let \( N \) be a power of two. We break \( f \) into images modulo polynomials of decreasing degree until we have the images \( f \equiv f(z^i) \equiv f(\omega^i), 0 \leq i < N \). At the start of the first iteration we have \( f \) reduced modulo \( z^N - 1 \). After \( i \) iterations, we will have the \( 2^i \) images
\[
f \equiv f(z^i) \equiv f(\omega^i), \quad 0 \leq j < 2^i,
\]
where \( u = N/2^i \). If \( i = p \) this gives us the DFT of \( f \). Consider then the image \( f' = f(z^i) \mod (z^u - \omega^{2ui}) = \sum_{b=0}^{u-1} b_i, \) for some \( j, 0 \leq j < \frac{N}{2^i} \). We break this image into two images \( f_0 \) and \( f_1 \), where
\[
\begin{align*}
f_0 &= f' \mod (z^u - \omega^{uj}), \\
f_1 &= f' \mod (z^u + \omega^{uj}) = f' \mod (z^u - \omega^{uj+i+N/2}),
\end{align*}
\]
If we write \( f' = \sum_{k=0}^{u-1} b_k z^k \), then we can write \( f_0 \) and \( f_1 \) in terms of the coefficients \( b_k \):
\[
\begin{align*}
f_0 &= \sum_{k=0}^{u-1} (b_k + \omega^{uj} b_{k+u}) z^k, \\
f_1 &= \sum_{k=0}^{u-1} (b_k - \omega^{uj} b_{k+u}) z^k.
\end{align*}
\]
Thus, given an array containing the coefficients \( b_k \) of \( f' \), we can write \( f_0 \) and \( f_1 \) in place of \( f' \) by way of operations
\[
\begin{bmatrix} b_k \\ b_{k+u} \end{bmatrix} \leftarrow \begin{bmatrix} 1 & \omega^{uj} \\ 1 & -\omega^{uj} \end{bmatrix} \begin{bmatrix} b_k \\ b_{k+u} \end{bmatrix}, \quad 0 \leq i \leq u.
\]

The pair of assignments \( \mathbf{8} \) are known as a butterfly operation, and can be performed with a ring multiplication by the twiddle factor \( \omega^{uj} \), and two additions. Note \( f_0 \) and \( f_1 \) are in a similar form as \( f' \), and if \( u > 1 \) we can break those images into smaller images in the same fashion. Starting this method with input \( f \mod (z^N - 1) \), will give us \( f \mod (z - \omega^j) = f(\omega^j), 0 \leq j < N \).

If the butterfly operations are performed in place, the resulting evaluations \( f(\omega^j) \) will be written in bit-reversed order. More precisely, if we let \( \lfloor j \rfloor_p \) denote the integer resulting from reversing the first \( p \) bits of \( j \), \( 0 \leq j < 2^p \), we have that \( f(\omega^j) \) will be written in place of \( a_k \), where \( k = \lfloor j \rfloor_p \) and \( \log \) is taken to be base-2 throughout. As an example,
\[
[11]_5 = [011011]_5 = 110102 = 16 + 8 + 2 = 26.
\]

We can make the FFT entirely in-place by computing the powers of \( \omega^u \) sequentially at every iteration. This entails traversing the array in a non-sequential order. Procedure FFT describes such an implementation.

If we observe that
\[
\begin{bmatrix} 1 & \omega^{uj} \\ 1 & -\omega^{uj} \end{bmatrix}^{-1} = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ -\omega^{-uj} & -1 \end{bmatrix},
\]
then we can implement an inverse FFT by inverting the butterfly operations in reversed order. We can, moreover, delay multiplications by powers of \( \omega \) until the end of the inverse FFT computation. This entails multiplying each coefficient by \( \frac{1}{2} \).

**Procedure** FFT(\( a, N, \omega \)), an in-place implementation of the radix-2 Cooley-Tukey FFT

**Input:**
- \( a = (a_0, \ldots, a_{N-1}) \), an array vector \( a \in \mathbb{R}^N \).
- \( \omega \), a root of \( z^N - 1 \), with \( N = 2^p \)

**Result:** \( \text{DFT,}_\omega(a) \) is written to the array \( a \) in bit-reversed order.

\[
\begin{array}{l}
\text{1 for } i \leftarrow 1 \text{ to } p \\
\text{2 \hspace{1em} u \leftarrow N/2^i} \\
\text{3 for } j \leftarrow 0 \text{ to } N/2 \text{ do} \\
\text{4 \hspace{1em} t \leftarrow \text{revs}(j)} \\
\text{5 \hspace{1em} for } k \leftarrow 0 \text{ to } u - 1 \text{ do} \\
\text{6 \hspace{1em} \begin{bmatrix} a_{i+k} \\ a_{i+k+u} \end{bmatrix} \leftarrow \begin{bmatrix} 1 & \omega^{uj} \\ 1 & -\omega^{uj} \end{bmatrix} \begin{bmatrix} a_{i+k} \\ a_{i+k+u} \end{bmatrix}} \\
\end{array}
\]

**Theorem 2.** Let \( N \) be a power of two and \( \omega \) an \( N \)-th root of unity. Then \( \text{DFT,}_\omega(f) \) can be computed in place using \( \frac{1}{2} N \log N + O(N) \) ring multiplications. The inverse FFT can interpolate \( f(z) \) from its DFT with similar complexity.

Using the radix-2 FFT, if \( d = \deg(fg) \), we choose \( N \) to be the least power of 2 exceeding \( d \). This entails appending zeros to the input arrays containing the coefficients of \( f \) and \( g \) respectively. By this method, computing a product of degree \( 2^p \) costs roughly double that a product of degree \( 2^{p-1} \). Crandall’s devil’s convolution algorithm \( \mathbf{2} \) somewhat flattens these jumps in complexity, though not entirely. It works by reducing a discrete convolution of arbitrary length into more easily computable convolutions. More recently, truncated Fourier transform (TFT) algorithms, described hereafter, have addressed this issue.
2.3 Truncated Fourier Transforms

In many applications, it is useful to compute a pruned DFT, a subset of a length-N DFT, at a cost less than that computing a complete DFT. In 2004, van der Hoeven showed in [9] that, given some knowledge of the form of the input, one can invert some pruned DFTs. The inverse transform relies on the observation that, given any two of the inputs/outputs to a butterfly operation [9], one can compute the other two values. Suppose \( n \in Z \) is arbitrary and \( N \) is the least power of two at least \( n \). For \( \omega \), a primitive root of unity of order \( N = 2^n \), van der Hoeven showed how to invert the length-\( n \) bit-reversed Truncated Fourier Transform,

\[
\text{TFT}_{\omega,n}(f) = \left( f(\omega^i) \right)_{0 \leq i < n},
\]

when we know the coefficients of \( f(z) \) of degree at least \( n \) (e.g. when \( \deg(f) < n \)). To distinguish this particular TFT we will call it the bit-reversed TFT.

**Theorem 3** (Van der Hoeven, [10]). Let \( N = 2^p \), \( n \in (N/2, N] \), \( f(z) \in \mathbb{R}[z] \) and let \( \omega \in \mathbb{R} \) be a principal \( N \)-th root of unity. Suppose \( \deg(f) < n \). Then \( \text{TFT}_{\omega,n}(f) \) can be computed using \( n \log n + \mathcal{O}(n) \) ring additions and \( \frac{1}{2} n \log n + \mathcal{O}(n) \) multiplications by powers of \( \omega \).

\( f(z) \) can be recovered from \( \text{TFT}_{\omega,n}(f) \) using \( n \log n + \mathcal{O}(n) \) shifted ring additions and \( \frac{1}{2} n \log n + \mathcal{O}(n) \) multiplications.

A shifted ring addition merely means in this context an addition plus a multiplication by \( \frac{1}{2} \). Van der Hoeven’s algorithm generalizes to allow us to compute arbitrary subsets of the DFT. Given a subset \( S \subseteq \{0, 1, \ldots, N - 1\} \), we define

\[
\text{TFT}_{\omega,S}(f) = \left( f(\omega^i) \right)_{i \in S},
\]

where \( f \) we now assume \( f \) is of the form \( f(z) = \sum_{i \in S} a_i z^i \). However, such a transform may have a greater complexity than stated in theorem [3] taking \( n = \#S \). Moreover, such a map is not necessarily invertible.

**Example 4.** Taking \( \omega \) an \( N = 8 \)-th principal root of unity with \( S = \{0, 3, 4, 5\} \) gives an uninvertible map. To see that the map \( \text{TFT}_{\omega,S} \) is not invertible, one can check that the polynomial \( f(z) = (1 + \omega^2)z^3 - z^2 + (1 - \omega^2)z^2 - 1 \) evaluates to 0 for \( z = \omega^k \), \( k \in \{0, 3, 8, 4, 9, 3, 5\} = \{0, 6, 1, 5\} \).

Van der Hoeven’s method still exhibits significant jumps in space complexity, as it space for \( N \) ring elements regardless of \( n \). In 2010, Harvey and Roche [3] introduced an in-place TFT algorithm, requiring \( n + \mathcal{O}(1) \) ring elements plus an additional \( \mathcal{O}(1) \) bounded-precision integers to compute \( \text{TFT}_{\omega,n}(f) \). Their method potentially requires evaluating polynomials using linear-time methods. This results in a constant factor additionally many ring multiplications in the worst-case.

**Theorem 5** (Roche, Theorem 3.5 of [3]). Let \( N, n, f \) and \( \omega \) be as in theorem [3]. Then \( \text{TFT}_{\omega,n}(f) \) can be computed in-place using at most \( \frac{1}{2} n \log n + \mathcal{O}(n) \) ring multiplications and \( \mathcal{O}(n \log n) \) ring additions.

The inverse in-place transform entails similarly many ring multiplications and \( \mathcal{O}(n \log n) \) shifted ring additions. As an application, Harvey and Roche used this transform towards asymptotically fast in-place polynomial multiplication.

3. THE CYCLOTOMIC TFT

3.1 Notation

We use the following notation throughout section [3] and thereafter. Suppose now that we have a polynomial \( f(z) = \sum_{i=0}^{n-1} a_i z^i \in \mathbb{R}[z] \) of degree at most \( n - 1 \), where \( n \) can be any positive integer. For the remainder of this paper, we write \( n = \sum_{i=1}^{s} n_i \) where \( n_i = 2^{n_i}, n_i(i) \in \mathbb{Z}_{>=0} \) and \( n_i > n_j \) for \( 1 \leq i < j \leq s \). Again \( N \) is the least power of two at least \( n \). For \( 1 \leq i \leq s \), we let

\[
\Phi_i = z^{n_i} + 1.
\]

The TFT algorithms of sections [3] and thereafter will compute the evaluations of \( f(z) \) at the roots of \( \Phi_i \). Namely, if we fix a canonical root \( \omega_i = \omega_i, \) then let, for \( 2 \leq i \leq s \), \( \omega_i = \omega_i^{n_i} \), a root of \( \Phi_i \), these algorithms will compute

\[
f_i(\omega_i^{2j+1}) \quad \text{for} \quad 0 \leq j < n_i, \quad 1 \leq i \leq s.
\]

It is the evaluation of \( f(z) \) at the roots of the cyclotomic polynomials \( \Phi_i \). As such, we will call it here the cyclotomic TFT and denote it by \( \text{TFT}_{\omega,i}(f) \). The choice of \( \omega \) only affects the ordering of the elements of the cyclotomic TFT. If we let

\[
S(n) = \{ n_i \leq k < 2 n_i : 1 \leq i \leq s \},
\]

then we have that \( \text{TFT}_{\omega,i}(f) \) uses the same set of evaluation points as \( \text{TFT}_{\omega,S(n)}(f) \).

The roots of unity used as evaluation points in \( \text{TFT}_{\omega,i}(f) \) will differ from those used in \( \text{TFT}_{\omega,n}(f) \). For instance, \( \text{TFT}_{\omega,i}(f) \) will always use all roots-of-unity of order dividing \( n_i \), whereas \( \text{TFT}_{\omega,n}(f) \) will instead use all roots of unity of order \( 2n_1 \). We define, for \( 1 \leq i \leq s \), the images

\[
f_i = f \bmod \Phi_i.
\]

The algorithms for computing a cyclotomic TFT all follow a similar template: we will produce the images \( f_i \) sequentially, and then write the evaluation of \( f \) at the roots of \( \Phi_i \) in place of each image \( f_i \).

3.2 Discrete Weighted Transforms

Given the images \( f_i = f \bmod \Phi_i \), one can evaluate \( f \) at the roots of \( \Phi_i \) by way of a Discrete Weighted Transform (DWT), which comprises an affine transformation followed by an FFT [3].

In a more general setting, suppose we have an image \( f^* = f(z) \bmod \left( z^N - c \right) \), where \( N \) is a power-of-two. Assuming \( c \) has an \( N \)-th root over our ring \( R \), the roots of \( f(z) \) are all of the form \( c^{i/N} \), \( 0 \leq i < N \), where \( \gamma \) is an \( N \)-th primitive root of unity. Thus to evaluate \( f \) at the roots of \( (z^N - c) \) one can replace \( f^*(z) \) with \( f^*(c^{i/N} z) \), and then compute \( \text{DFT}_w(f^*(c^{i/N} z)) \). Replacing \( f^*(z) \) with \( f^*(c^{i/N} z) \) merely entails multiplying the degree-\( k \) term of \( f^* \) by \( c^{i/N} \), for \( 0 \leq k < N \). This, combined with computing the powers of \( c^{i/N} \) sequentially, entails fewer than \( 2N \) ring multiplications.

Thus, to evaluate \( f \) at the roots of \( \Phi_i(z) = z^{n_i} - \omega_i^{n_i} \), one would write \( f_i(\omega_i z) \) in place of \( f_i(z) \), then compute \( \text{DFT}_w(f_i(\omega_i z)) \) by way of the FFT. As both the FFT and the affine transformation are invertible, a Discrete Weighted Transform is easily invertible as well.

3.3 Mateer’s TFT algorithm
we have that the number of multiplications becomes
\[
\sum_{i=1}^{s} \frac{1}{2} n_i \log n_i + \mathcal{O}(n_i) < \frac{1}{2} n \log n + \mathcal{O}(n). \tag{19}
\]

A similar analysis for the ring additions due to the DWTs gives us the following complexity:

**Lemma 6 (Mateer).** \(\text{TFT}'_{\omega,n}(f)\) can be computed using \(\frac{1}{2} n \log n + \mathcal{O}(n)\) ring multiplications plus \(\mathcal{O}(n \log n)\) ring additions.

Since each of the steps of Mateer’s method amounts to a series of butterfly operation, the algorithm is invertible with comparable complexity. One advantage of Mateer’s TFT is that inverting it is relatively straightforward: we invert the DWTs and then invert the butterflies.

Mateer’s method, however, requires space for \(N\) ring elements, as the images \(f \mod (z^{N/2} - 1)\) and \(f \mod (z^{N/2} + 1)\) may have maximal degree.

### 4. IN-PLACE METHODS FOR COMPUTING THE CYCLOTONIC TFT

In order to compute the cyclotomic TFT in-place, it appears, unlike the Mateer TFT, that we need to use some of the information from the images \(f_1, \ldots, f_i\) towards producing the image \(f_{i+1}\). Both Sergeev’s TFT and the new algorithm presented thereafter work in this manner.

Let, for \(1 \leq i \leq s\),
\[
\Gamma_i(z) = \prod_{j=1}^{i} \Phi_i(z) \quad \text{and} \quad C_i = f \mod \Gamma_i. \tag{20}
\]

We call \(C_i\) the combined image of \(f\), as it is the result of Chinese remaindering on the images \(f_1, \ldots, f_i\). We also define
\[
q_i = \begin{cases} f & \text{if } i = 0, \\ f \ mod \ \Gamma_i & \text{if } 1 \leq i \leq s \end{cases} \tag{21}
\]

the quotient produced dividing \(f\) by \(\Gamma_i\), as well as
\[
n_i = \begin{cases} n & \text{if } i = 0, \\ n \ mod \ n_i & \text{if } 1 \leq i \leq s \end{cases} \tag{22}
\]

Note that, as \(\Phi_i \ mod \ \Phi_j = 2 \text{ for } j \leq i\), we also have \(\Gamma_i \ mod \ \Phi_j = 2^i \text{ for } j \geq i\). Similarly, \(\Gamma_i \ mod \ (z^k - 1) = 2^i \text{ for } K\), a power of two at most \(n_i\).

For any choice of \(1 \leq i \leq s\), we have
\[
f(z) = C_i + \Gamma_i q_i. \tag{23}
\]

It is straightforward to obtain \(q_i\), given \(f\). Note that the degrees of any two distinct terms of \(\Gamma_i\) differ by at least \(n_i\), and that \(\deg(q_i) < n_i\). Thus, as \(\Gamma_i\) is monic, we have that the coefficients of \(q_i\) merely comprise the coefficients of the higher-degree terms of \(f\). More precisely,
\[
q_i = \sum_{j=0}^{n_i - 1} a_{n_i - n_i + j} z^j. \tag{24}
\]

By a similar argument, we also have that, for \(1 < i \leq s\),
\[
q_i = q_{i-1} \ mod \ \Phi_i. \tag{25}
\]

We note that \(q_s = 0\).

\[\boxed{\text{Algorithm 1: Mateer's algorithm for computing} \ TFT'_{\omega,n}(f)}\]

**Input:** \(n < N, f(z) \in \mathbb{R}[z]\), \(f\) reduced modulo \(z^N - 1\)

**Result:** \(\text{TFT}'_{\omega,n}(f)\) is written in place of \(f\)

1. \(i, K \leftarrow (1, N)\)
2. while \(K > n_i\) do
3. | Separate \(f\) mod \((z^K - 1)\) into the images \(f_{\mod (z^{K/2} - 1)}\) and \(f_{\mod (z^{K/2} + 1)}\).
4. | if \(K/2 = n_i\) then
5. | Note \(\Phi_i = z^{K/2} + 1\). Use a DWT to write the evaluation of \(f\) at the roots of \(\Phi_i\) in place of \(f_i = f \ mod \Phi_i\).
6. | \(i \leftarrow i + 1\)
7. | \(K \leftarrow K/2\)

On input we are given \(f(z)\) reduced modulo \(z^N - 1\). If \(N = n = n_1\) we can just perform a DWT on \(f\) to get \(\text{TFT}'_{\omega,n}(f)\), so suppose \(n\) is not a power of two and \(n < N\). We reduce the image into two images \(f(z) \ mod (z^{N/2} - 1)\) and \(f(z) \ mod (z^{N/2} + 1)\), in the same manner as described in the FFT. This process strictly entails ring additions. If \(f(z) = \sum_{i=0}^{N/2} a_i z^i\), then \(f_0\) and \(f_1\) can be computed by way of \(N/2\) butterfly operations.

\[
f_0(z) = \sum_{i=0}^{N/2} (a_i + a_{i+N/2}) z^i, \tag{17}
\]

\[
f_1(z) = \sum_{i=0}^{N/2} (a_i - a_{i+N/2}) z^i. \tag{18}
\]

It should be noted that these particular butterfly operations will not require ring multiplications, as their twiddle factors are all 1. As \(n_1 = N/2\), we keep the image \(f_1 = f \ mod (z^{N/2} + 1)\), and use a DWT on \(f_1\) to evaluate \(f\) at the roots of \(\Phi_1\). We then break \(f\) mod \((z^{N/2} - 1)\) into the images \(f_{\mod (z^{N/2} - 1)}\) and \(f_{\mod (z^{N/2} + 1)}\) in a similar fashion. If \(n_1 < N/4 < n_2\), we ignore the image \(f \ mod (z^{N/4} + 1)\) for the remainder of the computation. We continue in this fashion until we have \(f_i = f \ mod (z^{n_i} + 1)\), after which we should have \(\text{TFT}'_{\omega,n}(f)\) in place of \(f\), albeit scattered throughout our array of working space.

Performing the aforementioned modular reductions amounts to \(\mathcal{O}(n)\) additions. Accounting for the cost of the DWTs,
4.1 Computing images of \( C \), without explicitly computing \( C_i \)

We will express the combined image \( C_i \), \( C_i \) reduced modulo \( \Phi_{i+1} \) or a polynomial \( z^k-1 \) with \( k = 2^i \), in terms of the coefficients of the images \( f_1, \ldots, f_i \). To this end we introduce the following notation. Given an integer \( e \), we will let \( e[i] \) refer to the \( i \)-th bit of \( e \), i.e.

\[
e = \sum_{i=0}^{\lfloor \log(e) \rfloor} e[i]2^i, \quad e[i] \in \{0, 1\}.
\]

(26)

Sergeev’s TFT relies on the following property, albeit stated differently than in [8].

**Lemma 7.** Fix \( i, j \) and \( k \), \( 1 \leq j \leq i < k \leq s \). Suppose that \( f_{j} = z^{e} \) and \( f_{i} = 0 \) for \( l \neq j \), \( 0 \leq l \leq i \). Let \( m \) be a power of two at most \( n_i \). Then \( C_i \mod z^{m}-1 \) is nonzero only if \( e[n_j(l)] = 1 \) for \( j < l \leq i \), in which case

\[
C_i \mod (z^m-1) = 2^{i-j} z^e \mod (z^m-1)
\]

(27)

\[
= 2^{i-j} z^e \mod m.
\]

(28)

As \( \Phi_k, k > i \), divides \( z^{n_i-1} \). Lemma 7 gives us the following corollary.

**Corollary 8.** Fix \( i, j \) and \( k \), \( 1 \leq j \leq i < k \leq s \). Suppose that \( f_{j} = z^{e} \) and \( f_{i} = 0 \) for \( l \neq j \), \( 0 \leq l \leq i \). Then \( C_i \mod \Phi_k \) is nonzero only if \( e[n_j(l)] = 1 \) for \( j < l \leq i \), in which case

\[
C_i \mod \Phi_k = 2^{i-j} z^e \mod \Phi_k.
\]

(29)

Given that \( z^n \mod \Phi_k = -1 \), we have that

\[
2^{i-j} z^e \mod \Phi_k = (-1)^{e[n(k)]} 2^{i-j} z^e \mod n_k,
\]

(30)

where \( e \mod n_k \) is the integer \( e^* \) such that \( n_k | (e - e^*) \) and \( 0 \leq e^* < n_k \). The values \( e[n_j(l)] \) can be determined from \( n_i = 2^n(0) \) and \( e \) by way of a bitwise “and” operation.

**Example 9.** (Example of Corollary 8). Suppose \( n = 86 \) \((= 64 + 16 + 4 + 2)\), and suppose that

\[
f_1 = f \mod z^{64} + 1 = z^2, \quad f_2 = 0, \quad f_3 = 0,
\]

and \( \deg(f) < 64 + 16 + 4 \). In this example,

\[
C_3 = f \mod (z^{64} + 1)(z^{16} + 1)(z^4 + 1).
\]

(31)

Let \( g(z) = C_3 \mod (z^2 + 1) \). Then by lemma 8

\[
g(z) = \begin{cases} 
0 & \text{if } e \in [0, 20] \cup [24, 28] \cup [32, 52] \cup [56, 60), \\
4 & \text{if } e \in [20, 28, 52, 56]. \\
4z & \text{if } e \in [21, 29, 53, 57]. \\
-4z & \text{if } e \in [22, 30, 54, 58]. \\
-4z & \text{if } e \in [23, 31, 55, 59].
\end{cases}
\]

Remark 10. Let \( 1 \leq j \leq i \leq s \). A proportion of \( 2^{i-1} \) terms of \( f_j \) have an exponent satisfying the nonzero criterion of lemmas 8 and 7. Moreover, this nonzero criterion may be easily checked using the bitwise “and” operation.

**Proof of Lemma 7.** We fix \( e \) and \( j \) and prove by induction on \( i \).

**Base case:** Let \( i = j \). As in the proof of lemma 8 we have \( C_i = 2^{i-j} \Gamma_{i+1} z^{e} \). As \( \Phi_i \mod z^m - 1 = 2^i \) for a power of two dividing \( n_i \), it follows that \( C_i \mod z^m - 1 = z^e \mod z^m - 1 \).

**Inductive step:** Suppose now that the lemma holds for a fixed \( i \geq j \), and consider \( C_{i+1} \mod z^{m-1}, m \) a power of two dividing \( n_{i+1} \). We suppose that \( f_{j}^{*} = 0 \) for \( 1 \leq l \neq j \leq i + 1 \) and \( f_{j}^{*} = z^{e} \). By Chinese remaindering,

\[
C_{i+1} = C_{i} - \Gamma_{i} (\Gamma_{i+1}^{-1} C_{i} \mod \Phi_{i+1}) \cdot \frac{2^{i-j} z^{e}}{\Phi_{i+1}}.
\]

(32)

Reducing this expression modulo \( z^{m-1} - 1 \) gives us

\[
C_{i+1} = C_{i} - \Gamma_{i} (\Gamma_{i+1}^{-1} C_{i} \mod \Phi_{i+1}) \mod z^{m-1} - 1.
\]

(33)

Case 1: \( e[n_j(l)] = 0 \) for some \( l, j < l \leq i \), then by the induction hypothesis, \( C_i \mod z^{m-1} = 0 \). By lemma 8 \( C_i \mod \Phi_{i+1} = 0 \). Thus we have \( C_{i+1} \mod z^{m-1} = 0 \) as well.

Case 2: \( e[n_j(l)] = 0 \) for \( j < l \leq i \), then by the induction hypothesis, \( C_i \mod z^{m-1} = 2^{i-j} z^e \mod z^{m-1} \). By lemma 8 \( C_i \mod \Phi_{i+1} = z^{e} \mod \Phi_{i+1} \)

\[
C_{i+1} \mod z^{m-1} = 2^{i-j} z^{e} - \Gamma_{i} (\Gamma_{i+1}^{-1} z^{e} \mod \Phi_{i+1}) \mod z^{m-1} - 1.
\]

(34)

\[
= 2^{i-j} z^{e} \mod m - (z^{e} \mod \Phi_{i+1}) \mod z^{m-1} - 1.
\]

(35)

\[
= 2^{i-j} \left(1 + (-1)^{e[n_i(i+1)]}\right) z^{e} \mod m.
\]

(36)

As \( 1 + (-1)^{e[n_i(i+1)]} \) evaluates to 2 if \( e[n_i(i+1)] = 1 \) and 0 otherwise, this completes the proof. \( \square \)

**Lemma 7** can be derived from lemma 1 in [8].

4.2 Sergeev’s in-place cyclotomic TFT

In [8], Sergeev describes an arithmetic circuit for computing a cyclotomic TFT. Such a circuit for computing TFT for \( f \) has a depth of \( n + O(1) \). That is, it requires space for \( n + O(1) \) ring elements. This algorithm, like Mateer’s breaks \( f \) into the images \( f_{j} \), in linear time, and then applies a DWT on each image. Algorithm 2 describes how \( f \) is broken into the images \( f_{j}^{*} \).

**Algorithm 2:** Sergeev’s algorithm for computing \( f_{1}, \ldots, f_{s} \) in place of \( f \)

**Input:** \( n < N \), \( f(z) \in \mathbb{R}[z] \), \( f = \sum_{i=0}^{n-1} a_{i} z^{i} \)

**Result:** \( f_{1}, \ldots, f_{s} \) are written in place of \( f \)

1. \( (i, K) \leftarrow (0, N) \)

**while** \( K > n \) **do**

3. \( \text{Loop invariant: We have the images } f_{j}, 1 \leq j \leq i, \text{ and the first } n_{i} \text{ coefficients of } f \mod (z^{K} - 1) \).

4. \( \text{if } K/2 = n_{i+1} \text{ then} \)

5. \( \text{Compute } f_{i+1} \text{ and the first } n_{i+1} \text{ coefficients of } f \mod (z^{K/2} - 1) \text{ in place of the first } n_{i}^{*} \text{ coefficients of } f \mod (z^{K} - 1) \).

6. \( i \leftarrow i + 1 \)

7. \( \text{else} \)

8. \( \text{Write the first } n_{i}^{*} \text{ coefficients of } f \mod (z^{K/2} - 1) \text{ in place of the coefficients of } f \mod (z^{K} - 1) \).

9. \( K \leftarrow K/2 \)

Suppose we have \( f_{1}, \ldots, f_{i} \), and the first \( n_{i}^{*} \) coefficients of \( g = f \mod (z^{K} - 1) \), where \( K \leq n_{i} \) is a power of two. Note
that $K$ may be greater than $n_i^*$. If we write
\[ f(z) \equiv (z^K - 1) \equiv \sum_{j=0}^{K-1} b_j z^j, \quad (37) \]
then
\[ f(z) \equiv (z^{K/2} - 1) = \sum_{j=0}^{K/2-1} (b_j + b_{j+K/2}) z^{j}, \quad (38) \]
\[ f(z) \equiv (z^{K/2} + 1) = \sum_{j=0}^{K/2-1} (b_j - b_{j+K/2}) z^{j}. \quad (39) \]

If $K/2 > n_i^*$, Sereegev’s algorithm will write the first $n_i^*$ coefficients of $f \equiv (z^{K/2} - 1)$ in place of $f \equiv (z^{K/2} - 1)$, i.e. we want to add $b_{i+K/2}$ to $b_i$ for $0 \leq i < n_i^*$. Reducing $f = C_i + \Gamma q_i$ modulo $z^K - 1$, we have
\[ f \equiv (z^K - 1) = C_i + 2\Gamma q_i \equiv (z^{K/2} - 1). \quad (40) \]

Remark 11. As $\deg(q_i) < n_i^*/K$, the coefficients $b_{i+K/2}$, $0 \leq j < n_i^*$ depends strictly on $C_i$, $\equiv (z^{K/2} - 1)$.

We use lemma 7 to determine the coefficient of each $f_{j}$, $1 \leq j \leq i$, towards $b_{i+K/2}$. Namely, if a term $cz^i$ appearing in $f_{j}$ satisfies the nonzero criterion of lemma 7, then it will have a contribution of $2^{i-j}c$ towards $b_{i+K/2}$. For $0 \leq i < n_i^*$, we compute the sum of these contributions towards $b_{i+K/2}$ and then add that to $b_i$. In order to minimize the number of multiplications necessary in order to compute the weights $2^{-j}$, we sum the contributions from $f_{j}$, then multiply that sum by 2, then add to that all the contributions from $f_{j}$, again multiply that by 2, and so forth.

If instead, $K \leq n_i^*$, we have that $K$ must equal $n_{i+1}$, we write $f_{i+1}$ and the first $n_{i+1}$ coefficients of $f \equiv (z^{K/2} - 1)$ in place of $b_j$, $0 \leq j < n_i^*$. For $0 \leq j < n_{i+1}$, we have both $b_j$ and $b_{j+K/2}$, and can compute the $z^i$ coefficient of $f_{i+1}$ and $f \equiv (z^{K/2} - 1)$ in their places accordingly. What remains is to compute the last $n_i - n_i^*$ coefficients of $f_{i+1}$. This amounts to computing $b_{i+K/2}$ by way of lemma 7 as in the first case.

5. A NEW IN-PLACE CYCLOTONIC TFT ALGORITHM

In order to compute a new coefficient of an image of $f$ in Sereegev’s algorithm, one has to effectively make a pass through the array in order to sum the contributions from images $f_j$. Instead of summing the contributions and then adding that into our array, we will add contributions directly back into our array of coefficients. In order to do this, we will work with the weighted images,
\[ f_i^*(z) = 2^{i+1} f_i, \quad (41) \]
this allows us to add all the contributions to a new polynomial image with a single pass over our array. Moreover, unlike Sereegev’s algorithm, we will forego producing part of the images $f \equiv (z^{K/2} - 1)$, $K$, a power of two. We will compute $f_{i+1}$ immediately after producing $f_i$.

Our transform has three steps: we first iteratively compute the remainders produced by dividing $q_i - 1$ by $\Phi_i$,
\[ r_i = q_i - 1 \mod \Phi_i, \quad 1 \leq i \leq s. \quad (42) \]
in place of $f_i$; we then iteratively write $f_i^*$ in place of $r_i$ for $i = 1, 2, \ldots, s$; lastly we rewrite the weighted images $f_i^*$ to get $f_i$ and compute a DWT of each image $f_i$ separately to give us the weighted evaluation points.

5.1 Breaking $f$ into the remainders $r_1, \ldots, r_s$

We first break $f = q_0$ into its quotient and remainder dividing by $z^{n_1} + 1$,
\[ r_1 = f \equiv (z^{n_1} + 1) = \sum_{i=0}^{n_1-1} (a_i - a_{i+n_1}) z^i, \quad (43) \]
\[ q_1 = f \equiv (z^{n_1} + 1) = \sum_{i=0}^{n_1-1} a_{i+n_1} z^i, \quad (44) \]
where $a_i = 0$ for $i \geq n$. This can be done in place with $n_1$ subtractions in $R$. We then similarly break $q_1$ into $r_2$ and $q_2$, then $q_2$ into $r_3$ and $q_3$, and continue until we have $r_1, \ldots, r_{s-1}$ and $q_{s-1}$. Since $\deg(q_{s-1}) < n_s = \deg(\Phi_s)$, $r_s$ is exactly $q_{s-1}$.

5.2 Writing the weighted images $f_i^*$ in place of the remainders $r_i$

We first note that $f_1^*$ is precisely $r_1$. We will iteratively produce the remaining weighted images.

Suppose, at the start of the $i$-th iteration, we have $f_1^*, \ldots, f_i^*$, and $r_{i+1}, \ldots, r_s$, and we want to write $f_{i+1}^*$ in place of $r_{i+1}$. We have
\[ f_{i+1}^* = 2^{i+1} f_i \Phi_{i+1}, \quad (45) \]
\[ = 2^{i+1} (\Gamma_i q_i + C_i) \equiv (\Phi_{i+1}), \quad (46) \]
\[ = (q_i + 2^{i+1} C_i) \equiv (\Phi_{i+1}), \quad (47) \]
\[ = r_{i+1} + (2^{i+1} C_i \Phi_{i+1}). \quad (48) \]

Unfortunately, we don’t have the combined image $C_i$, but rather the weighted images $f_j^*$, $1 \leq j \leq i$, from which we can reconstruct $C_i$. We would like to be able to compute $C_i \equiv (\Phi_{i+1})$ in place of the weighted images $f_i^*$. Corollary tells us the contribution of $f_i^*$ towards subsequent images. If $e$ satisfies the non-zero criterion of the lemma, then by [18], a term $c_0 e^z$ of $f_j^*$, $1 \leq j \leq i$ will contribute $2^{e-1} [e(n+1)] z^e \equiv (\Phi_{i+1})$ to $f_{i+1}$. In order to make the contributions have weight $\pm 1$, we instead first rewrite $r_i$ by 2 and compute $2f_i^*$, and then divide by 2 thereafter. AddContributions adds the contribution of $f_1^*, \ldots, f_{i-1}^*$ to $f_i^*$.

According to corollary S, $0 \leq j \leq i$, only a proportion of $2^{-j}$ of the terms of $f_j^*$ will have a non-zero contribution to $f_{i+1}^*$. Thus the total cost of adding contributions of $f_j^*$ towards $f_{i+1}^*$, $i > j$, is less than $2\#f_j^* = 2n_j$. It follows that the total additions and subtractions in $R$ required to add all these contributions is bounded by $2n$. Since AddContributions only scales array ring elements by $\pm 1$, we have the following complexity:

Lemma 12. Calling AddContributions(a, n, i) for $i$ from 1 to $s$ entails no more than $2n$ ring additions and no ring multiplications.

In the manner we have chosen to add these contributions, we will have to make $s - 1$ passes through our array to add them all. One way we could avoid this is to instead add
all the contributions from \( f_i \), and then all the contributions from \( f^*_j \), and so forth, adding up all the contributions from a single term at once. We could use that a term \( c_{j,e}z^e \) of \( f_j \) that does not contribute towards \( f^*_j \) will not contribute to \( f_i^* \) for any \( k > i \). This would reduce the number of passes we make through the larger portion of the array, though the cache performance of potentially writing to \( s - 1 \) images at once raises questions.

When adding contributions to \( f_j^* \), any two terms \( c_{j,e}z^e \) and \( c_{j,e'}z^{e'} \) of \( f_j^* \) whose bits \( e[n(l)], e'[n(l)] \) agree for \( j < l < i \) will both contribute to \( f_j^* \) in the same fashion (i.e. we will either add or subtract both coefficients \( c_{j,e} \) and \( c_{j,e'} \) to an image, or do nothing). Thus we need only inspect the non-zero criterion of one exponent \( e \) in a block of exponents \( k_{n_i-1} \leq e < (k + 1)n_{i-1} \). Similarly, we need only inspect one exponent in a contiguous block of \( n_i \) exponents in order to determine their shared value of \( (-1)^{e[n(i)]} \).

Given an exponent that does not satisfy the non-zero criterion, our implementation will generate the next exponent that does satisfy the non-zero criterion, by way of bit operations.

Procedure \texttt{BreakIntoImages}(a, n) breaks \( f \) into the images \( f_i \). The procedure effectively has three sections. In the first section of the algorithm, we break \( f \) into the remainders \( r_i \). Producing \( r_i \) entails \( n_i^* < n_i \) additions, and so producing all the \( r_i \) entails less than \( \sum_{i=1}^{s} n_i = n \) ring additions.

In the second section we write the weighted images \( f_i^* \) in place of the \( r_i \). Adding all the contributions, per lemma \texttt{12} requires \( 2n \) additions. We reweigh the last \( n - n_1 \) coefficients of \( f \) by 2, then by \( \frac{1}{2} \). This constitutes less than \( n \) such multiplications.

In the third section we reweigh the weighted images \( f_i^* \) to get the images \( f_i \). This entails less than \( n \) multiplications by 2. This gives us the following complexity:

\textbf{Lemma 13.} Procedure \texttt{BreakIntoImages}(a, n) entails no more than \( 3n \) additions and \( 2n \) multiplications by \( 2^{11} \).

Thus, like Mateer’s and Sergeev’s algorithms, the TFT algorithm presented amounts to linearly many ring operations plus the cost of performing FFTs of size \( n_i \), \( 1 \leq i \leq s \). Inverting the TFT is straightforward. Given the images \( f_1, \ldots, f_s \), we can reobtain the polynomial \( f \) by effectively reversing the steps of procedure \texttt{BreakIntoImages}.

Procedure \texttt{BreakIntoImages}(a, n): An in-place algorithm to compute a vector of images of \( f \)

Input: \( a \), a length-\( n \) array containing the coefficients of \( f = \sum_{i=1}^{n} a_i z^i \), where \( n = \sum_{i=1}^{n} n_i \)

Result: The images \( f_i, 1 \leq i \leq s \), are written in place of \( f \).

\begin{verbatim}
// Write \( r_1, \ldots, r_s \) in place of \( f \\
1 m \leftarrow 0 \\
2 for i \leftarrow 1 to s do \\
3 \quad for j \leftarrow m to n_i - 1 do \\
4 \quad \quad a(j) \leftarrow a(j) - a(j + n_i) \\
5 \quad m \leftarrow m + n_i \\
6 \end{verbatim}

\begin{verbatim}
// Write the weighted image \( f_i^* \) in place of \( r_i \), for \( 1 \leq i \leq s \\
7 m \leftarrow 0 \\
8 for i \leftarrow 2 to s do \\
9 \quad for j \leftarrow m to m + n_i - 1 do \\
10 \quad \quad a(j) \leftarrow 2a(j) \\
11 m \leftarrow m + n_i \\
12 end for \\
13 end for \\
14 for j \leftarrow m to n - 1 do \\
15 \quad a(j) \leftarrow 2a(j) \\
16 m \leftarrow m + n_i \\
17 end for
\end{verbatim}

\texttt{BreakIntoImages} strictly entails 1) multiplying arrays elements by nonzero scalars and 2) adding or subtracting one array element from another, so it is straightforward to undo each of the steps to get back \( f \), with similar complexity.

### 6. Computing the Bit-Reversed TFT by Way of the Cyclotomic TFT

The bit-reversed TFT has the property that, for \( m < n \), \( \text{TFT}_{2^m}^C(f) \) is merely the first \( m \) entries of \( \text{TFT}_{2^n}^C(f) \), whereas the cyclotomic TFT does not, in general. Hence the bit-reversed TFT lends itself more readily to multivariate polynomial arithmetic. We show how an algorithm for computing the cyclotomic TFT may be modified to compute a bit-reversed TFT.

As before, let \( L = 2^p > n = \sum_{i=1}^{s} n_i = \sum_{i=1}^{s} 2^{n(i)}, \omega \in \mathcal{R} \) be an \( N \)-th principal root of unity, and let \( \omega_i = \omega^{N/2n_i} \) be a root of \( \Phi_i \). Define

\[ \Omega_i = \prod_{l=1}^{i} \omega_l \quad \text{and} \quad \Psi_j(z) = z^{n_j} - \Omega_j^{2^{n_j}} \]

for \( 0 \leq i \leq s, 0 < j \leq s \). TFT\(_{\omega_i}(f) \) is comprised of the evaluation of \( f(z) \) at the roots of the polynomials \( \Psi_j(z) \). If \( n_1 + \ldots + n_{j-1} \leq i < n_1 + \ldots + n_j \), we have that \( \omega^{(i)}_j \) is a root of \( \Psi_j \). To see this, write \( i = n_1 + \ldots + n_{j-1} + t \), \( 0 \leq t < n_j \), and observe that

\[ \omega^{(i)}_{j,n_j} = \omega^{n_1 + \ldots + n_{j-1} - t} \omega^{t}_{n_j} \]

\[ = (\omega_1 \cdots \omega_j) \omega^{(i)}_{j,n_j} \]

\[ = \Omega_j^{n_j} \omega^{(i)}_{j,n_j} \]

(50)
As \( l' < n_j \), \( \omega|^{[l']}p \) has order dividing \( n_j \), and so \( \Phi_i \) is precisely \( (\Omega_{j-1}^{-1})^{n_j} \). Every \( n_j - th \) root of unity in \( \mathbb{R} \) is of the form \( \omega|^{[l']}p \), where \( 0 \leq k < n_j \). In particular \( \omega|^{[l']}p \) is an \( n_j - th \) root of unity. Thus \( \omega|^{[l']}p \omega|^{[l']}p \) is a root of \( \Psi_j \). As the binary representations of \( n_1 + \cdots + n_{j-1} \) and \( l' \) do not share any 1 digits at the same precision,

\[
[n_1 + \cdots + n_{j-1}]_p + [l']_p = [l]_p,
\]

and so \( \omega|^{[l']}p \) is a root of \( \Psi_j \).

Consider the affine transformation \( z \mapsto \Omega_z \). Then

\[
\Psi_i(\Omega_z) = \Omega_z^{n_i} (z^{n_i} - \Omega_z^{n_i - 1}) = \Omega_z^{n_i} \left( \prod_{j=1}^{s} \omega_j^{n_i} \right) \left( z^{n_i} - \Omega_z^{n_i - 1} \right) = -\Omega_z^{n_i} (z^{n_i} + 1) = -\Omega_z^{n_i} \Phi_i.
\]

Thus, for a polynomial \( f(z) \), \( f(z) \mod \Phi_i(z) = f(z) \mod \Psi_i(\Omega_z) \). We can break \( f(z) \) into its images modulo the polynomials \( \Phi_i \) as follows:

1. Replace \( f(z) \) with \( f(\Omega_z) \).

2. Break \( f(\Omega_z) \) into its images modulo \( \Phi_i(z) \) per a cyclotomic TFT method. Equivalently, this gives us the images \( f(\Omega_z) \mod \Psi_i(\Omega_z) \).

3. Apply the transformation \( z \mapsto \Omega_z^{-1}z \) to each image to give us \( f(z) \mod \Psi_i(z) \) in place of \( f(\Omega_z) \mod \Psi_i(\Omega_z) \).

Note that the affine transformations of steps 1 and 3 both have complexity \( O(n) \). We can get \( \text{TFT}_{o,n}(f) \) from the images \( f \mod \Phi_i \) by applying a DWT to each image \( f \mod \Phi_i \). As each step here is invertible, we can invert a bit-reversed TFT using an inverse cyclotomic TFT algorithm. We can similarly invert a bit-reversed TFT using an inverse cyclotomic TFT algorithm.

7. ACKNOWLEDGMENTS

Thanks to: Igor Sergeev for bringing [6] and [8] to my attention, and for his assistance in helping me understand his TFT algorithm; Mark Giesbrecht for feedback; Joris van der Hoeven for posing the question of non-invertible TFTs; and Dan Roche for providing his implementation of bit-reversed TFT algorithms.

8. REFERENCES

[1] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex Fourier series. Math. Comp., 19:297–301, 1965.
[2] R. Crandall. Advanced Topics in Scientific Computation. Springer-Verlag, 1996. ISBN 0-387-94473-7.
[3] R. Crandall and B. Fagin. Discrete weighted transforms and large-integer arithmetic. Math. Comp., 62(205):305–324, 1994.
[4] D. Harvey and D. S. Roche. An in-place truncated fourier transform and applications to polynomial multiplication. In Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation, ISSAC ’10, pages 325–329, New York, NY, USA, 2010. ACM.
[5] M. Heideman, D. Johnson, and C. Burrus. Gauss and the history of the fast fourier transform. ASSP Magazine, IEEE, 1(4):14–21, 1984.
[6] T. Mateer. Fast fourier transform algorithms with applications, 2008.
[7] D. Roche. Efficient Computation With Sparse and Dense Polynomials. PhD thesis, University of Waterloo, 2011.
[8] I. S. Sergeev. Regular estimates for the complexity of polynomial multiplication and truncated fourier transform, 2011. Written in Russian.
[9] J. van der Hoeven. The truncated fourier transform and applications. In J. Gutierrez, editor, Proc. ISSAC 2004, pages 290–296, Univ. of Cantabria, Santander, Spain, July 4–7 2004.
[10] J. van der Hoeven. Notes on the Truncated Fourier Transform. Technical Report 2005-5, Université Paris-Sud, Orsay, France, 2005.