Algebraic Signal Processing Theory: Cooley-Tukey Type Algorithms for Real DFTs

Yevgen Voronenko, *Member, IEEE,* and Markus Püschel, *Senior Member, IEEE*

***Abstract*— In this paper we systematically derive a large class of fast general-radix algorithms for various types of real discrete Fourier transforms (real DFTs) including the discrete Hartley transform (DHT) based on the algebraic signal processing theory. This means that instead of manipulating the transform definition, we derive algorithms by manipulating the polynomial algebras underlying the transforms using one general method. The same method yields the well-known Cooley-Tukey fast Fourier transform (FFT) as well as general radix discrete cosine and sine transform algorithms. The algebraic approach makes the derivation concise, unifies and classifies many existing algorithms, yields new variants, enables structural optimization, and naturally produces a human-readable structural algorithm representation based on the Kronecker product formalism. We show, for the first time, that the general-radix Cooley-Tukey and the lesser known Bruun algorithms are instances of the same generic algorithm. Further, we show that this generic algorithm can be instantiated for all four types of the real DFT and the DHT.**

***Index Terms*— Discrete Fourier transform, fast algorithm, polynomial algebra, Chinese remainder theorem**

I. Introduction

It is well-known that the discrete Fourier transform (DFT) of a real signal is conjugate-symmetric. Hence, as one may expect, the required computation can be reduced by roughly a factor of two. As for the DFT, there is a large number of publications on fast algorithms for this "real DFT" (RDFT) and its variants, such as the discrete Hartley transform (DHT) and others [1]–[27]. Knowing the entire space of available algorithms is not just of academic interest but crucial for real-world implementations: many applications spend the bulk of their runtime computing DFTs, different computing platforms usually require different algorithms, and the complexity of modern processors imposes many structural and other requirements on an algorithm in order to run efficiently [28], [29].

Unfortunately, it is extraordinarily difficult to obtain an overview on the available real fast Fourier transforms. Arguably, this has two reasons. First, there is a lack of theory that explains the algorithms and simplifies their derivation. Second, the typical representation of algorithms as nested summations involving complicated cosine and sine expressions is hard to parse by a human. For complex fast Fourier transforms (FFTs) the situation is different as shown by excellent books on both FFT theory and structural FFT representation using the Kronecker product formalism [30]–[32].

One commonly used algorithm for the real DFT utilizes a half-size complex DFT and a post-processing step [33]. We provide the exact form in (72) in the Appendix. However, it is only applicable to even transform sizes, and the post-processing step has to traverse the entire dataset, which can be very expensive on modern machines with deep memory hierarchies and multiple processor cores. Thus, finding direct (i.e., without a conversion) real FFTs is still a relevant problem.

Most real FFTs are derived by lengthy manipulations of the transform definition using trigonometric identities. This method has produced many important algorithms; however, it does not explain the existence of the algorithms, provides no insight into the structure and degrees of freedom, and leaves the question open whether all algorithms have been found. Notable exceptions include the derivation of Winograd type real FFTs using the Chinese remainder theorem and other techniques [13], [14], the derivation of DHT algorithms by projecting FFTs using finite field algebra [15], and the derivation of real Bruun FFTs [5], [19].

Contributions of this paper. In this paper we complete our preliminary work in [34] and derive real FFTs *algebraically.* This means that we first associate with each real DFT a polynomial algebra as explained in [35] to obtain a uniform description. Then we derive fast algorithms by manipulating the polynomial algebra, instead of the transform itself, using one general method or theorem. This method produces a large class of general-radix algorithms that we call Cooley-Tukey type since the same method produces the (complex) Cooley-Tukey FFT as well as general radix discrete cosine and sine transform (DCT and DST) algorithms [36]. Our method is an extension of the techniques used in [30]. The occurrence and relevance of polynomial algebras for complex and real DFTs and for other transforms in signal processing is clarified by the algebraic signal processing theory introduced in [35], [37].

Our method makes the algorithm derivation concise, transparent, and naturally produces the algorithms in the form of structured matrix factorizations that visualize the structure of the algorithms. Our method identifies degrees of freedom and unifies many well known algorithms in the literature. For example, Bergland's [1] and Bruun's [5] real DFT algorithms, and Bracewell's DHT algorithm [38], are instantiations of our general method. Further, we use the polynomial algebra framework to structurally optimize algorithms and derive, and hence explain, many known identities between real DFTs. We

Manuscript received December 24, 2007; revised June 24, 2008. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Soontorn Oraintara. This work was supported by NSF through awards 0310941, 0325687, and 0634967 and by DARPA through the Department of Interior Grant NBCH1050009. The authors are with the Department of Electrical and Computer Engineering, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA 15213. E-mail: {yvoronen,pueschel}@ece.cmu.edu.

provide a detailed literature review in Section VI.

We also made an effort to fully specify all occurring algorithms and identities in a form suitable as a reference for implementation developers.

The method in this paper does not produce all existing real FFTs; e.g., prime-factor [39], [40] and Rader-type [41], [42] algorithms are excluded.

Organization. Section II introduces polynomial algebras, explains their connection to transforms, and characterizes complex and real DFTs in this framework. In Section III we explain our general method to produce Cooley-Tukey type algorithms using polynomial algebras. The method is then applied to the real DFTs in Section IV. We show first a unified algorithm description and then show two classes of instantiations. The structural optimization and identities between real DFTs are explained and derived in Section V. Section VI analyzes the derived algorithms and gives a detailed literature review. We conclude in Section VII.

II. Polynomial Algebras and Transforms

In this section we introduce polynomial algebras and explain how they are associated to transforms. Then we identify this connection for various versions of complex and real DFTs. Later we exploit this algebraic interpretation of real DFTs to derive their Cooley-Tukey type algorithms.

We also introduce the matrix notation that we will use to describe fast transform algorithms.

For further background on the mathematics in this section and polynomial algebras in particular, we refer to [43].

A. Polynomial Algebras and Transforms

Polynomial algebra. An algebra $\mathcal{A}$ is a vector space that also provides a multiplication of its elements. Examples include the sets of complex or real numbers $\mathbb{C}$ or $\mathbb{R}$, and the sets of complex or real polynomials $\mathbb{C}[x]$ or $\mathbb{R}[x]$.

The key structure in this paper is the *polynomial algebra*. Given a fixed polynomial $p(x)$ of degree $\deg(p) = n$, we define a polynomial algebra as the set

$$\mathbb{C}[x]/p(x) = \{s(x) \mid \deg(s) < \deg(p)\}$$

of polynomials of degree smaller than n with addition and multiplication modulo p. Viewed as a vector space, $\mathbb{C}[x]/p(x)$ hence has dimension n.

As a simple example we consider $\mathcal{A} = \mathbb{C}[x]/(x^2-1)$, which has dimension 2. A possible basis is $b = (1, x)$. In $\mathcal{A}$, for example, $x \cdot (x+1) = x^2 + x \equiv x+1 \bmod (x^2-1)$, obtained by replacing x^2 with 1.

Chinese remainder theorem (CRT). Assume $p(x) = q(x)r(x)$ factorizes into two coprime (no common factors) polynomials q and r. Then the Chinese remainder theorem (CRT) for polynomials is the linear mapping[1]

$$\begin{aligned} \Delta:\ \mathbb{C}[x]/p(x) &\to \mathbb{C}[x]/q(x) \oplus \mathbb{C}[x]/r(x), \\ s(x) &\mapsto (s(x) \bmod q(x), s(x) \bmod r(x)). \end{aligned}$$

[1]More precisely, isomorphism of algebras.

Here, $\oplus$ is the direct sum of vector spaces with elementwise operation. If we choose bases b, c, d in the three polynomial algebras, then Δ can be expressed as a matrix. This matrix is obtained by mapping every element of b with Δ, expressing it in the concatenation $c \cup d$ of the bases c and d, and writing the results into the columns of the matrix.

As an example, we consider again the polynomial $p(x) = x^2 - 1 = (x-1)(x+1)$ and the CRT decomposition

$$\Delta:\ \mathbb{C}[x]/(x^2-1) \to \mathbb{C}[x]/(x-1) \oplus \mathbb{C}[x]/(x+1).$$

As bases, we choose $b = (1, x),\ c = (1),\ d = (1)$. $\Delta(1) = (1, 1)$ with the same coordinate vector in $c \cup d = (1, 1)$. Further, because of $x \equiv 1 \bmod (x-1)$ and $x \equiv -1 \bmod (x+1)$, $\Delta(x) = (x, x) \equiv (1, -1)$ with the same coordinate vector. Thus, Δ in matrix form is the so-called butterfly matrix

$$F_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{1}$$

Polynomial transforms. Assume $p(x) \in \mathbb{C}[x]$ is separable, i.e., it has pairwise distinct zeros $\alpha = (\alpha_0, \ldots, \alpha_{n-1})$. Then the CRT decomposes $\mathbb{C}[x]/p(x)$ completely into its *spectrum*:

$$\begin{aligned} \Delta:\ \mathbb{C}[x]/p(x) &\to \mathbb{C}[x]/(x-\alpha_0) \oplus \ldots \oplus \mathbb{C}[x]/(x-\alpha_{n-1}), \\ s(x) &\mapsto (s(\alpha_0), \ldots, s(\alpha_{n-1})). \end{aligned} \tag{2}$$

Here we used that $s(x) \equiv s(\alpha_k) \bmod (x - \alpha_k)$. If we choose a basis $b = (p_0, \ldots, p_{n-1})$ in $\mathbb{C}[x]/p(x)$ and bases $b_i = (1)$ in each $\mathbb{C}[x]/(x-\alpha_i)$, then the corresponding matrix is given by

$$\mathcal{P}_{b,\alpha} = [p_\ell(\alpha_k)]_{0 \le k,\ell < n}$$

and is called the *polynomial transform* for $\mathcal{A} = \mathbb{C}[x]/p(x)$ with basis b.

If, in general, we choose $b_i = (\beta_i)$ as spectral basis, then the matrix corresponding to the decomposition (2) is the *scaled polynomial transform*

$$\mathrm{diag}_{0 \le \ell < n}(1/\beta_\ell)\mathcal{P}_{b,\alpha}.$$

We jointly refer to polynomial transforms, scaled or not, as Fourier transforms.

DFT as a polynomial transform. For example, the DFT of size n (viewed as a matrix [31], [32]) is the polynomial transform for $\mathcal{A} = \mathbb{C}[x]/(x^n - 1)$ with basis $b = t_n = (1, x, \ldots, x^{n-1})$. Namely, $x^n - 1 = \prod_{0 \le k < n}(x - w_{k/n})$ where we use the notation

$$w_r = e^{-2\pi j r},$$

which implies that $w_{1/n}$ is a primitive nth root of unity and $w_r^k = w_{kr}$. With this notation,

$$\mathcal{P}_{b,\alpha} = \left[w_{k/n}^\ell\right]_{0 \le k,\ell < n} = \left[w_{1/n}^{k\ell}\right]_{0 \le i,jn} = \mathrm{DFT}_n\,.$$

Other types of DFTs were introduced in [44], [45], named type 1–4 in [4] (type 1 is the standard DFT), and described algebraically in [35]. Namely, type 3 is obtained by choosing $\mathcal{A} = \mathbb{C}[x]/(x^n + 1)$ as algebra with the same basis t_n to get

$$\mathcal{P}_{b,\alpha} = \left[w_{1/n}^{(k+1/2)\ell}\right]_{0 \le k,\ell n} = \text{DFT-3}_n\,.$$

The DFTs of type 2 and 4 are scaled polynomial transforms as shown by

$$\text{DFT-2}_n = \text{diag}_{0\le k<n}(w_{1/n}^{k/2})\,\text{DFT}_n,$$
$$\text{DFT-4}_n = \text{diag}_{0\le k<n}(w_{1/n}^{(k+1/2)/2})\,\text{DFT-3}_n\,.$$

For example, for the DFT-2_n this implies that $(w_{1/n}^{-k/2})$ is chosen as basis in the spectral component $\mathbb{C}[x]/(x - w_{1/n}^k)$, $0 \le k < n$.

Discussion. The interpretation of the DFT as polynomial transform has long been known and been used to derive and understand its fast algorithms [30], [46]. Recently, in the context of the algebraic signal processing theory (ASP) [35], [37], [47], we have shown that this interpretation is also natural from a signal processing point of view. Namely, associated with the DFT is $\mathcal{A}$ as filter algebra, $\mathcal{M} = \mathcal{A}$ as signal $\mathcal{A}$-module, and $\Phi : \ (s_0, \ldots, s_{n-1}) \mapsto \sum_{0\le\ell n} s_\ell x^\ell$ as *finite z-transform*. The polynomial $x^n - 1$ captures the periodic boundary condition associated with the DFT, and filtering is the multiplication of polynomials $h(x) \in \mathcal{A}$ and $s(x) \in \mathcal{M}$ modulo $x^n - 1$ (which is equivalent to the circular convolution of the coefficients).

$(\mathcal{A}, \mathcal{M}, \Phi)$ is an example of a *signal model* in ASP, the basic concept on which ASP is axiomatically built. Among other things, ASP shows that a signal model supports shift-invariance if $\mathcal{A}$ is commutative. In the finite-dimensional case this naturally leads to polynomial algebras, which explains their appearance in SP. For example, we have shown that most trigonometric transforms are associated with signal models built from polynomial algebras [35], [37]. This signal models associated with the 16 discrete cosine and sine transforms do not support time but space signal processing [37].

The main insight that ASP provides for this paper is that knowing the signal model, i.e., polynomial algebra, associated with a transform makes the algorithm derivation straightforward and transparent.

B. Real Polynomial Algebras and Transforms

To relate various versions of real DFTs to polynomial algebras, we have to generalize (2) as explained in [35].

Real polynomial transforms. We assume that $p(x)$ is real, which guarantees that its roots are either real or form complex conjugate pairs. Then, instead of decomposing $\mathbb{C}[x]/p(x)$ over $\mathbb{C}$ as in (2), we decompose $\mathbb{R}[x]/p(x)$ over $\mathbb{R}$ using the complete real factorization $p(x) = \prod_{0\le k<m} f_k(x)$. The polynomials f_k have either degree one (in the case of a real root of p) or two (in the case of a pair of conjugate complex roots of p). Consequently, the CRT yields the *real* spectral decomposition

$$\begin{aligned}\Delta :\ \mathbb{R}[x]/p(x) &\to \mathbb{R}[x]/f_1(x) \oplus \ldots \oplus \mathbb{R}[x]/f_m(x),\\ s(x) &\mapsto (s(x) \bmod f_1(x), \ldots, s(x) \bmod f_m(x)).\end{aligned} \tag{3}$$

To express Δ as a matrix, we choose again a basis b in $\mathbb{R}[x]/p(x)$ and bases in the spectral components. For the latter, there is now a larger set of choices. We call each transform obtained this way a real polynomial transform, or real Fourier transform, or simply Fourier transform. Note that if all roots of p are real, then every real Fourier transform is also a complex one. This is the case for the DCTs and DSTs [37].

Real DFTs as polynomial transforms. We consider $\mathbb{R}[x]/(x^n \pm 1)$, which underlie the real DFTs as expected [35]. To simplify notation, we set

$$c_r = \cos(2\pi r), \quad s_r = \sin(2\pi r),$$
$$\mathit{cas}_r = c_r + s_r, \quad \mathit{cms}_r = c_r - s_r.$$

and introduce the polynomials (for $0 < r < 1$)

$$p_{2n,r}(x) = (x^n - w_r)(x^n - w_{-r}) = x^{2n} - 2c_r x^n + 1. \tag{4}$$

Using this notation, we have the following complete *real* factorizations:

$$x^n - 1 = (x-1)(x+1) \prod_{0\le i<n/2-1} p_{2,(i+1)/n}(x), \quad n \text{ even},$$
$$x^n - 1 = (x-1) \prod_{0\le i<n/2} p_{2,(i+1)/n}(x), \quad n \text{ odd},$$
$$x^n + 1 = \prod_{0\le i<n/2} p_{2,(i+1/2)/n}(x), \quad n \text{ even},$$
$$x^n + 1 = \prod_{0\le i<n/2} p_{2,(i+1/2)/n}(x)\ \cdot (x+1), \quad n \text{ odd},$$
$$p_{2n,r}(x) = \prod_{0\le i<n} p_{2,(i+r)/n}(x). \tag{5}$$

The first four factorizations yield the following complete *real* algebra decompositions of $\mathbb{R}[x]/(x^n - 1)$ (for n even and odd, respectively):

$$\mathbb{R}[x]/(x-1) \oplus \mathbb{R}[x]/(x+1) \oplus \bigoplus_{0\le i<n/2-1} \mathbb{R}[x]/p_{2,(i+1)/n}, \tag{6}$$

$$\mathbb{R}[x]/(x-1) \oplus \bigoplus_{0\le i<n/2} \mathbb{R}[x]/p_{2,(i+1)/n}, \tag{7}$$

and the following decompositions of $\mathbb{R}[x]/(x^n+1)$ (for n even and odd, respectively):

$$\bigoplus_{0\le i<n/2} \mathbb{R}[x]/p_{2,(i+1/2)/n}, \tag{8}$$

$$\bigoplus_{0\le i<n/2} \mathbb{R}[x]/p_{2,(i+1/2)/n} \oplus \mathbb{R}[x]/(x+1). \tag{9}$$

The factorization (5) yields the decomposition of $\mathbb{R}[x]/p_{2n,r}(x)$ as

$$\bigoplus_{0\le i<n} \mathbb{R}[x]/p_{2,(i+r)/n}(x), \tag{10}$$

and defines the skew real DFTs used later.

As we explain next, real DFTs are associated with $\mathcal{A} = \mathbb{R}[x]/(x^n \pm 1)$ with basis $t_n = (1, x, \ldots, x^{n-1})$ and different choices of spectral bases. The most important cases are listed in Table I, which also introduces names for the chosen bases and a unified notation for real DFTs. The decomposition properties are explained later but already listed here for completeness. The exact form of these transforms can be found in Table XIII in the appendix.

TABLE I

REAL DFTS: (a) ASSOCIATED POLYNOMIAL ALGEBRAS; (b) NOTATION FOR BASES; (c) NOTATION FOR SPECTRAL BASES.

(a)

Transform	**Algebra**	**Basis b**	**Spectral basis f**	**Unified notation**
RDFT_n	x^n-1	t	e	$\mathcal{F}_n(t\to e)$
RDFT-2_n	x^n-1	t	$e^{1/2}$	$\mathcal{F}_n(t\to e^{1/2})$
RDFT-3_n	x^n+1	t	e	$\mathcal{G}_n(t\to e)$
RDFT-4_n	x^n+1	t	$e^{1/2}$	$\mathcal{G}_n(t\to e^{1/2})$
DHT_n	x^n-1	t	h	$\mathcal{F}_n(t\to h)$
DHT-2_n	x^n-1	t	$h^{1/2}$	$\mathcal{F}_n(t\to h^{1/2})$
DHT-3_n	x^n+1	t	h	$\mathcal{G}_n(t\to h)$
DHT-4_n	x^n+1	t	$h^{1/2}$	$\mathcal{G}_n(t\to h^{1/2})$
BRDFT_n	x^n-1	t	s	$\mathcal{F}_n(t\to s)$
BRDFT-2_n	x^n-1	t	$s^{1/2}$	$\mathcal{F}_n(t\to s^{1/2})$
BRDFT-3_n	x^n+1	t	s	$\mathcal{G}_n(t\to s)$
BRDFT-4_n	x^n+1	t	$s^{1/2}$	$\mathcal{G}_n(t\to s^{1/2})$
skew real DFT	$p_{2n,r}(x)$	b	f	$\mathcal{S}_{2n,r}(b\to f)$

(b)

Polynomials	**Decomposition property**
x^n-1	$x^{km}-1=(x^m)^k-1$
x^n+1	$x^{km}+1=(x^m)^k+1$
$p_{2n,r}=x^{2n}-2c_rx^n+1$	$p_{2km,r}=p_{2k,r}(x^m)$
Bases in polynomial algebras	**Decomposition property**
$t_n=(1,x,\dots,x^{n-1})$	$t_{km}=t_k(x^m)\star t_m$
$s_n=(x^{-\lfloor n/2\rfloor},\dots,1,\dots,x^{\lceil n/2\rceil-1})$	$s_{km}=s_k(x^m)\star t_m$
$e_{2,r}=\left(1,\frac{c_r-x}{s_r}\right)$	
$e_{2n,r}=e_{2,r}(x^n)\star t_n$	$e_{2km,r}=e_{2k,r}(x^m)\star t_m$
$h_{2,r}=\left(\frac{-cms_r+x}{2s_r},\frac{cas_r-x}{2s_r}\right)$	
$h_{2n,r}=h_r(x^n)\star t_n$	$h_{2km,r}=h_{2k,r}(x^m)\star t_m$

(c)

Type	$\mathbb{R}[x]/x\pm1$	$\mathbb{R}[x]/p_{2,r}(x)$
e	(1)	$e_{2,r}=\left(1,\frac{c_r-x}{s_r}\right)$
h	(1)	$h_{2,r}=\left(\frac{-cms_r+x}{2s_r},\frac{cas_r-x}{2s_r}\right)$
s	(1)	$s_{2,r}=(2c_r-x,1)=(x^{-1},1)$
$e^{1/2}$	(1)	$x^{-1/2}e_{2,r}=\left(\frac{s_{3r/2}-xs_{r/2}}{s_r},\frac{c_{3r/2}-xc_{r/2}}{s_r}\right)$
$h^{1/2}$	(1)	$x^{-1/2}h^*_{2,r}=\left(\frac{-cms_{3r/2}+xcms_{r/2}}{2s_r},\frac{-cas_{3r/2}+xcas_{r/2}}{2s_r}\right)$
		$(a,b)^*=(a,-b)$

Most important for signal processing applications are the standard RDFT (type 1) and the discrete Hartley transform (DHT), also of type 1.

Standard RDFTs. The standard real DFT (type 1) is defined[2] by the matrix

$$\mathrm{RDFT}_n=\begin{bmatrix}1 & 1 & \dots & 1 & 1\\ 1 & -1 & \dots & 1 & -1\\ \begin{bmatrix}c_{k\ell/n}\\ -s_{k\ell/n}\end{bmatrix}_{1\le k<n/2,\,0\le\ell<n}\end{bmatrix}. \tag{11}$$

This notation means that below the first two rows, we have an $n/2-1\times n$ matrix, each entry of which is a 2×1 block containing a cosine and a negative sine of the same angle as shown.

For a real input signal of length n, it computes the real and imaginary parts (interleaved) of the complex spectrum, which is conjugate symmetric. The second row of (11) computes the highest frequency spectral component and is only present for even n. It is known, and expected, that using the RDFT instead of the DFT saves roughly half of the operations.

Algebraically, the RDFT is a real Fourier transform for the decompositions (6) or (7) with respect to the spectral basis called e in Table I(c). Namely, e means that the list (1) is chosen as basis in the one-dimensional spectral components $\mathbb{R}[x]/(x\pm1)$ and the list

$$e_{2,r}=(1,\ c_r/s_r-1/s_r\cdot x).$$

in the two-dimensional components $\mathbb{R}[x]/p_{2,r}(x)$. This choice yields $x^\ell\equiv c_{\ell r}\cdot1-s_{\ell r}\cdot(c_r/s_r-1/s_r\cdot x)$ mod $p_{2,r}$ and hence $(c_{\ell r},-s_{\ell r})$ as coordinate vector as required by (11) (where $r=k/n$).

The RDFT of type 3 [9], [35] is obtained analogously from (8) and (9) as shown in Table I(a)

As for the DFT we have to modify the basis in the spectral components to obtain the RDFTs of types 2 and 4 [9], [35]. The spectral components of DFT-2 and DFT-4 are of the form $\mathbb{C}[x]/(x-w_r)$ and have bases $(w_r^{-1/2})$. To obtain the RDFT equivalent, we observe that in $\mathbb{C}[x]/(x-w_r)$, $w_r^{-1/2}\equiv x^{-1/2}$, i.e., the spectral basis is shifted by $x^{-1/2}$. Hence, we compute the bases of the corresponding $\mathbb{R}[x]/p_{2,r}$ analogously. We compute $x^{-1/2}$ in $\mathbb{R}[x]/p_{2,r}$. First, we notice that $x^{-1/2}=x^{-1}\cdot x^{1/2}$, and compute $x^{-1}=2c_r-x$. Next we solve $y^2=x$ for y in the algebra, which yields two possible solutions $y=\pm\frac{1+x}{2c_{u/2}}$. We set $x^{1/2}=\frac{1+x}{2c_{u/2}}$ to obtain the desired transform matrices. Hence

$$x^{-1/2}=x^{-1}\cdot x^{1/2}=\frac{1+2c_r-x}{2c_{r/2}}=\frac{s_{3r/2}-xs_{r/2}}{s_r}. \tag{12}$$

The basis of $\mathbb{R}[x]/p_{2,r}$ is now $x^{-1/2}\cdot e_{2,r}$. The final result, after replacing $x^{-1/2}$ using (12), is shown in Table I(c). The complete spectral basis is denoted with $e^{1/2}$.

DHTs. Choosing the basis h instead of e and $h^{1/2}$ instead of $e^{1/2}$ [see Table I(c)] in the spectral components yields the well-known discrete Hartley transforms (DHTs) of types 1–4. "The" DHT is type 1 and due to [38], [48]; the other types were introduced in [27]. Note that $h_{2,r}$ consists of the difference and the sum of the two elements of $e_{2,r}$. Similarly, $h_{2,r}^{1/2}$ consists of the difference and the negative sum of the two elements of $e_{2,r}^{1/2}$.[3]

BRDFTs. It seems that the most natural choice of basis in the two-dimensional spectral components is $(1,x)$. However, the associated real DFT is not considered in the literature. Bruun's FFT [5] uses (implicitly, since no polynomial algebras are used) a close variant by choosing $s_{2,r}=(x^{-1},1)\equiv(2c_r-x,1)$. This choice hence yields four types of BRDFTs as we

[2]Note that the definitions of the RDFT in the literature may differ by a permutation of the rows compared to (11) as explained later.

[3]This slightly inconsistent definition of $h^{1/2}$ makes the DHTs of types 2 and 3 mutual transposes up to a permutation, see (63)–(70) in Section VI.

TABLE II
RELATIONSHIP BETWEEN REAL AND COMPLEX DFTS FOR EVEN n.

$$
\begin{aligned}
\mathrm{DHT}_n &= \left(I_2 \oplus \left(I_{n/2-1} \otimes \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}\right)\right) \mathrm{RDFT}_n, \\
\text{DHT-2}_n &= \left(I_2 \oplus \left(I_{n/2-1} \otimes \begin{bmatrix} 1 & -1 \\ -1 & -1 \end{bmatrix}\right)\right) \text{RDFT-2}_n, \\
\text{DHT-3}_n &= \left(I_{n/2} \otimes \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}\right) \text{RDFT-3}_n, \\
\text{DHT-4}_n &= \left(I_{n/2} \otimes \begin{bmatrix} 1 & -1 \\ -1 & -1 \end{bmatrix}\right) \text{RDFT-4}_n, \\
\mathrm{RDFT}_n &= \left(I_2 \oplus \left(I_{n/2-1} \otimes_i \begin{bmatrix} c_{(i+1)/n} & 1 \\ s_{(i+1)/n} & 0 \end{bmatrix}\right)\right) \mathrm{BRDFT}_n, \\
\text{RDFT-3}_n &= \left(I_{n/2} \otimes_i \begin{bmatrix} c_{(i+1/2)/n} & 1 \\ s_{(i+1/2)/n} & 0 \end{bmatrix}\right) \text{BRDFT-3}_n.
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{DFT}_n &= U_n \left(I_2 \oplus \left(I_{(n-2)/2} \otimes \begin{bmatrix} 1 & j \\ 1 & -j \end{bmatrix}\right)\right) \mathrm{RDFT}_n, \\
\text{DFT-2}_n &= U_n \left(\begin{bmatrix} 1 & j \end{bmatrix} \oplus \left(I_{(n-2)/2} \otimes \begin{bmatrix} 1 & j \\ -1 & j \end{bmatrix}\right)\right) \text{RDFT-2}_n, \\
\text{DFT-3}_n &= V_n \left(I_{n/2} \otimes \begin{bmatrix} 1 & j \\ 1 & -j \end{bmatrix}\right) \text{RDFT-3}_n, \\
\text{DFT-4}_n &= V_n \left(I_{n/2} \otimes \begin{bmatrix} 1 & j \\ -1 & j \end{bmatrix}\right) \text{RDFT-4}_n .
\end{aligned}
$$

call them. Only BRDFT and BRDFT-3 will prove useful in the algorithm derivation later.

Skew real DFTs. Table I(a) also introduces *skew* real DFTs associated with (10). These will arise as building blocks in the real DFT algorithms.

Relationships. Since the RDFTs, DHTs, and BRDFTs only differ in the choice of spectral basis, they can be converted into each other using a base change in the spectrum. It takes the form of a block-diagonal matrix; the block sizes correspond to the dimensions of the spectral components. For example, for even n, and $t \in \{1, 2, 3, 4\}$, it takes the form of the first six equations in Table II. (the notation is explained in the next section)

Further, the DFT can be computed by first decomposing $\mathbb{C}[x]/(x^n - 1)$ over $\mathbb{R}$ using any real DFT and then completely by decomposing the two-dimensional components. For even n we obtain this way the last four equations in Table II. U and V are permutations that also occur in the spectral format conversion below. They are defined in (16) and (17) below.

Format. We use for all real DFTs the same ordering of the spectrum corresponding to (8) and (9), i.e., the one-dimensional spectral components are first, and the two-dimensional components are kept together. Other ways of ordering hence require a permutation of the rows of the transform matrices. Our ordering corresponds to the `Perm` format of the RDFT in the Intel IPP library [49]. This is different from FFTW [50] which pads one-dimensional spectral components with an extra 0, and is the equivalent of `CCS` format in IPP.

For the RDFTs, our ordering is common (e.g., already used in [1]). For the DHTs it is different from the original "standard" definitions in [38] and [27]. The relationship is as follows

$$\text{Standard-DHT-t}_n = U_n \, \text{DHT-t}_n, \; t \in \{1, 2\}, \tag{13}$$

$$\text{Standard-DHT-t}_n = V_n \, \text{DHT-t}_n, \; t \in \{3, 4\}. \tag{14}$$

The permutations U and V are defined below in (16) and (17).

We provide the explicit form of the RDFTs, DHTs, and BRDFTs as used in this paper in Table XIII in the appendix.

C. Matrix Notation

In the remainder of the paper we derive fast algorithms represented as factorizations of the transform matrix as products of structured sparse matrices. The notation follows [31].

Basic matrices. We denote the $n \times n$ identity matrix with I_n. If the columns of I_n are reversed, we get J_n. The 2×2 butterfly matrix F_2 was defined in (1) and a 2×2 rotation by angle $2\pi r$ is defined as

$$R_{2\pi r} = \begin{bmatrix} c_r & -s_r \\ s_r & c_r \end{bmatrix}.$$

Further, $\mathrm{diag}_{0 \le i < n}(\alpha_i)$ is the diagonal matrix with diagonal entries α_i and we define

$$D_n = \mathrm{diag}_{0 \le i < n}((-1)^i). \tag{15}$$

Finally, L^n_m, $n = km$, is the stride permutation matrix, which has 1s at positions $(jk + i, im + j)$, and 0s elsewhere.

Matrix operators. Further, if $A = [a_{i,j}]$ and B are matrices, then the direct sum and the tensor or Kronecker product are respectively defined as

$$A \oplus B = \begin{bmatrix} A & \\ & B \end{bmatrix}, \quad A \otimes B = [a_{i,j} B].$$

Since $I_k \otimes A = A \oplus \ldots \oplus A$, we will write the direct sum of different $k \times k$ matrices A_i, $0 \le i < k$, as

$$I_k \otimes_i A_i = A_0 \oplus \ldots \oplus A_{k-1}.$$

The spectral reordering permutations U and V in Table II can now be expressed as

$$U_n = \begin{cases} (I_{(n+2)/2} \oplus J_{(n-2)/2}) L^n_2, & n \text{ even}, \\ (I_{(n+1)/2} \oplus J_{(n-1)/2})(I_1 \oplus L^{n-1}_2), & n \text{ odd}. \end{cases} \tag{16}$$

$$V_n = \begin{cases} (I_{n/2} \oplus J_{n/2}) L^n_2, & n \text{ even}, \\ (I_{(n-1)/2} \oplus J_{(n+1)/2})(L^{n-1}_2 \oplus I_1), & n \text{ odd}. \end{cases} \tag{17}$$

III. FAST ALGORITHMS: THEORY

The connection to polynomial algebras can be used to derive fast algorithms for the associated transforms using a general method that builds on but extends the early work [30]. In short, we derive general-radix Cooley-Tukey algorithms by performing the decompositions (2) or (3) *in steps* based on a *decomposition of* p if one exists. We have already shown that this method yields the standard Cooley-Tukey FFT for the DFT and a large class of general-radix algorithms (including many novel ones) for the 16 DCTs and DSTs [36], [51].

In this paper, we first generalize the method to work with real decompositions and then apply it to the various real DFTs. Together with [36], this shows that one method spawns most known algorithms for trigonometric transforms.

Notation. To state the general method we first introduce the product of bases of polynomials. Let $b = (p_0, \ldots, p_{k-1})$ and $c = (q_0, \ldots, q_{m-1})$ be two lists of polynomials. Then their *product* is the list of length km

$$\begin{aligned} b \star c \;=\; & (p_0 q_0, \ldots, p_0 q_{m-1}, \\ & \ldots\ldots \\ & p_{k-1} q_0, \ldots, p_{k-1} q_{m-1}). \end{aligned}$$

Further, if b is as above, and $r(x)$ is any polynomial, then we denote with

$$b(r(x)) = (p_0(r(x)), \ldots, p_{k-1}(r(x)))$$

the same list but with $r(x)$ inserted for x.

Method. We first need the following lemma, a variation of a theorem we already used in [51] to derive DCT/DST algorithms.

Lemma 1 Let $q(x)$ be separable and let $q(x) = \prod_{0 \le i < k} q_i(x)$. Further, let c and e_i, $0 \le i < k$, be bases for $\mathbb{C}[x]/q(x)$ and $\mathbb{C}[x]/q_i(x)$, respectively, and let, with respect to these bases, M be the matrix associated with the CRT decomposition

$$\mathbb{C}[x]/q(x) \to \bigoplus_{0 \le i < k} \mathbb{C}[x]/q_i(x).$$

If $r(x)$ is an arbitrary polynomial of degree m and d a basis for $\mathbb{C}[x]/r(x)$, then $M \otimes I_m$ is the matrix associated with

$$\mathbb{C}[x]/q(r(x)) \to \bigoplus_{0 \le i < k} \mathbb{C}[x]/q_i(r(x)),$$

with respect to the bases $b = c(r(x)) \star d$ and $b_i = e_i(r(x)) \star d$.

The above holds if $\mathbb{C}$ is replaced by $\mathbb{R}$ and all polynomials are real.

Proof: M is obtained by reducing $c_\ell(x) \in c$ modulo the $q_i(x)$, $0 \le i < k$, and expressing the results in the bases e_i of $\mathbb{C}[x]/q_i(x)$. The resulting vector is the ℓth column of M. If $c_\ell(x) \equiv c_{\ell,i} \bmod q_i(x)$ has the coordinate vector $v_{\ell,i}$ with respect to the basis e_i, then $c_\ell(r(x))d_j \equiv c_{\ell,i}(r(x))d_j \bmod q_i(r(x))$ and has the coordinate vector $v \otimes (0, \ldots, 0, 1, 0, \ldots, 0)^T$ (the 1 is in position j, $0 \le j < m$) with respect to the basis $e_i(r(x)) \star d$. The matrix that has these vectors as columns is $M \otimes I_m$, as desired. ∎

Our general method for deriving fast algorithms for complex and real Fourier transforms is sketched in Fig. 1 and takes slightly different forms in both cases. The key property that we require is that $p(x) = q(r(x))$ decomposes. We assume $\deg(p) = n = km$, $\deg(q) = k$, $\deg(r) = m$, and that suitable bases have been chosen.

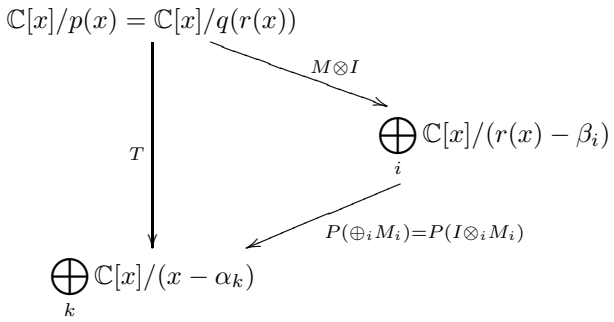
In the complex case $\mathbb{C}[x]/p(x) = \mathbb{C}[x]/q(r(x))$, we first use the CRT with respect to the decomposition of the outer polynomial $q(y) = \prod_i (y - \beta_i)$. According to Lemma 1, the associated matrix is $M \otimes I_m$, where M is a Fourier transform for $\mathbb{C}[y]/q(y)$. The smaller polynomial algebras $\mathbb{C}[x]/(r(x) - \beta_i)$ are then decomposed by the proper $m \times m$ Fourier transforms M_i. A final permutation P reorders the summands into the required form.

In the real case $\mathbb{R}[x]/p(x) = \mathbb{R}[x]/q(r(x))$, we proceed analogously. The difference is that q is now decomposed over $\mathbb{R}$ as $q(y) = \prod_i q_i(y)$, where the q_i have degree one or two. Again, the smaller algebras $\mathbb{R}[x]/q_i(r(x))$ are decomposed by their proper real Fourier transforms M_i, which now have dimensions $m \times m$ or $2m \times 2m$. The resulting spectrum is permuted with P into the proper order.

In both cases, the resulting algorithm takes the form

$$T = P\left(\bigoplus_i M_i\right)(M \otimes I_m) \tag{18}$$

Complex fast Fourier transform



Real fast Fourier transform

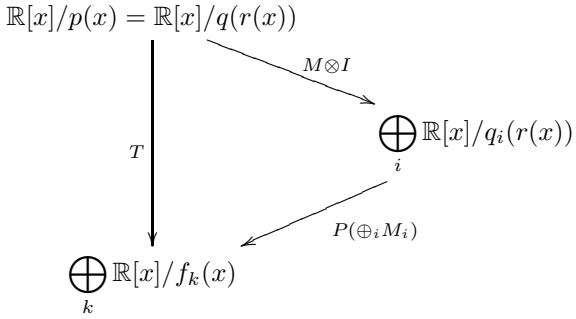


Fig. 1. Algebraic derivation of Cooley-Tukey type algorithms for complex and real Fourier transforms T. Proper bases (not shown) are chosen for the decompositions to work. In both cases, the resulting algorithm takes the form $T = P(\oplus_i M_i)(I \otimes M)$.

and we call it *Cooley-Tukey type*.

Example: Cooley-Tukey FFT. We use Lemma 1 to derive the Cooley-Tukey FFT. The polynomial algebra for DFT_n is $\mathbb{C}[x]/(x^n - 1)$ with standard basis t_n. Assuming $n = km$, then $x^n - 1 = (x^m)^k - 1$ decomposes. Applying the CRT in steps yields

$$\begin{aligned} &\mathbb{C}[x]/(x^n - 1) = \mathbb{C}[x]/((x^m)^k - 1) \\ &\to \bigoplus_{0 \le i < k} \mathbb{C}[x]/(x^m - w_{i/k}) && (22) \\ &\to \bigoplus_{0 \le i < k} \bigoplus_{0 \le j < m} \mathbb{C}[x]/(x - w_{(jk+i)/n}) && (23) \\ &\to \bigoplus_{0 \le i < n} \mathbb{C}[x]/(x - w_{i/n}). && (24) \end{aligned}$$

We read off the matrices for each decomposition step. First, we observe that $t_n = t_k(x^m) \star t_m$ [see Table I(b)]. Thus, Lemma 1 is applicable: step (22) corresponds to $(\mathrm{DFT}_k \otimes I_m)$ and t_m is the basis in each $\mathbb{C}[x]/(x^m - w_{i/k})$. The latter polynomial algebras are completely decomposed in step (23) by the polynomial transforms (called *skew DFTs*)

$$\begin{aligned} \mathrm{DFT}_m(i/k) &= [w_{(jk+i)/n}^{\ell}]_{0 \le j, \ell < n} \\ &= \mathrm{DFT}_n \cdot \mathrm{diag}_{0 \le \ell < m}(w_{i\ell/n}). \end{aligned} \tag{25}$$

The final step (24) is just the stride permutation applied to the one-dimensional algebras.

TABLE III

STEPWISE DECOMPOSITION OF $\mathbb{R}[x]/(x^n-1)$, $n=km$ AND k EVEN, AND THE CORRESPONDING MATRIX FACTORIZATION OF RDFT_n.

Stepwise algebra decomposition:

$$\mathbb{R}[x]/((x^m)^k-1)$$

$$\to \left[\mathbb{R}[x]/(x^m-1)\right] \quad \oplus \left[\mathbb{R}[x]/(x^m+1)\right] \quad \oplus \left[\bigoplus_{0<i<\frac{k}{2}} \mathbb{R}[x]/p_{2m,i/k}\right] \tag{19}$$

$$\to \left[\mathbb{R}[x]/(x-1) \oplus \mathbb{R}[x]/(x+1) \oplus \left(\bigoplus_{0<i<\frac{m}{2}} \mathbb{R}[x]/p_{2,i/m}\right)\right] \quad \oplus \left[\bigoplus_{0\le i<\frac{m}{2}} \mathbb{R}[x]/p_{2,(i+1/2)/m}\right] \quad \oplus \left[\bigoplus_{0<i<\frac{k}{2}} \bigoplus_{0<j<m} \mathbb{R}[x]/p_{2,(kj+i)/n}\right] \tag{20}$$

$$\to \mathbb{R}[x]/(x-1) \oplus \mathbb{R}[x]/(x+1) \oplus \left(\bigoplus_{0<i<\frac{km}{2}} \mathbb{R}[x]/p_{2,i/km}\right) \tag{21}$$

Corresponding matrix factorization: $\mathrm{RDFT}_{km} = P_m^{km}\Big(\mathrm{RDFT}_m \oplus \text{RDFT-3}_m \oplus \big(I_{k/2-1} \otimes_i B_{2m}^e \mathcal{S}_{2m,(i+1)/k}(e \to e)\big)\Big)\Big(\mathrm{RDFT}_k \otimes I_m\Big)$.

In summary we get

$$\begin{aligned}\mathrm{DFT}_n &= L_m^n\big(I_k \otimes_i \mathrm{DFT}_m(i/k)\big)(\mathrm{DFT}_k \otimes I_m)\\ &= L_m^n(I_k \otimes \mathrm{DFT}_m)D_m^n(\mathrm{DFT}_k \otimes I_m),\end{aligned} \tag{26}$$

where D_m^n is diagonal, namely the direct sum of the diagonal matrices in (25). The algorithm is the decimation-in-frequency Cooley-Tukey FFT, its transpose is the decimation-in-time version. This also motivates why we call all algorithms of the form (18) *Cooley-Tukey type*.

Example: Cooley-Tukey type algorithms for RDFT. Using Table I(a), $\mathrm{RDFT}_n = \mathcal{F}_n(t \to e)$ is associated with $\mathbb{R}[x]/(x^n-1)$ with standard basis $t_n = (1, x, \ldots, x^{n-1})$; the spectral basis is of type e. We assume the size $n = km$ factors such that $x^n - 1$ decomposes and consider the case where both k and m are even.

Similarly to the complex DFT, we start with a stepwise algebra decomposition similar to (22)–(24), but now all decomposition steps are over the real numbers; hence the steps take the different form shown in Table III.

The first step decomposes $\mathbb{R}[y]/(y^k-1)$ over $\mathbb{R}$ as shown in (6) and inserts x^m for y. The result is (19) in Table III. Since $t_n = t_k(x^m) \star t_m$, the associated matrix is $\mathrm{RDFT}_k \otimes I_m$ using Lemma 1, and the bases are t_m in $\mathbb{R}[x]/(x^m \pm 1)$ and $e_{2,i/k} \star t_m = e_{2m,i/k}$ in $\mathbb{R}[x]/p_{2m,i/k}(x)$.

The second step (20) decomposes the summands in (19) completely. The first two are decomposed by $\mathcal{F}_m(t \to e) = \mathrm{RDFT}$ and $\mathcal{G}_m(t \to e) = \text{RDFT-3}$, respectively, as shown in (6) and (8). For the two-dimensional summands we need the skew real DFTs $\mathcal{S}_{2m,(i+1)/k}(e \to e)$ introduced in Table I(a).

Finally, (21) reorders the summands into the required order using a suitable permutation P_m^{km}. Note that when going from (20) to (21), $r = (kj+i)/n$ in (20) is renormalized to i/km which is always smaller than 1/2. This is done by mapping the values $r > 1/2$ to $1-r$. For example, 3/4 would be changed to 1/4. This does not affect $p_{2,r}$ or the polynomial algebra (because $p_{2,r} = p_{2,1-r}$, but does affect the definition of the spectral basis in Table I. Accounting for the change introduces a sparse base change matrix B in the final factorization. B is defined in later in Table VI..

The resulting Cooley-Tukey type algorithm is shown in Table III.

Note, that to implement this algorithm, one also needs fast algorithms for the occurring RDFT-3_m and $\mathcal{S}_{2m,(i+1)/k}(e \to e)$. We will obtain these algorithms using the same algebraic method in the next section. Without the algebraic framework, the appearance of the skew real DFTs is unexpected, which explains why the algorithm derivation has been more difficult than for the complex DFT.

IV. COOLEY-TUKEY TYPE ALGORITHMS FOR REAL DFTS

In this section we derive general-radix Cooley-Tukey type algorithms for the real DFTs following the method outlined in Fig. 1 in Section III. We first derive generic algorithms jointly for the transforms in Table I(a). Then we instantiate them first in a "natural" way and then in a way that improves the operations count, generalizing Bruun's FFT [5].

A. Generic Real DFT Algorithms

In Table I(a) we introduced a unified notation for real DFTs. It shows that they can be grouped into three types $\mathcal{F}, \mathcal{G}, \mathcal{S}$ corresponding to the three occurring polynomial algebras. The difference is in the choice of basis b (usually $b = t$) in the algebra and the basis f in the spectrum. This notation, together with our derivation method, enables us to derive the algorithms jointly for each of these groups with generic choices of bases.

The algorithms derivation follows closely Fig. 1 and is completely analogous to the special case of RDFT already considered in Table III. Hence we will be brief. The resulting algorithms are summarized in Table IV.

Case 1: $\mathcal{F}_n(b \to f)$. The polynomial algebra is $\mathcal{A} = \mathbb{R}[x]/(x^n-1)$ with basis b and spectral basis f. We assume $n = km$, which implies the polynomial decomposition $x^n - 1 = (x^m)^k - 1$. For even k, we obtain the polynomial algebra decomposition and the corresponding fast algorithm already shown in Table III. However, here we consider any basis b with the compatible decomposition property (i.e., $b_n = b_k(x^m) \star t_m$) and any spectral basis f.

We sketch the stepwise algebra decomposition in Fig. 2.

First, the coarse decomposition of $\mathcal{A}$ in (19) decomposes it into m- and $2m$-dimensional algebras (we only show $2m$-dimensional in Fig. 2). This step has a degree of freedom, namely the spectral basis of the coarse decomposition. We

TABLE IV

ALGORITHMS FOR GENERIC REAL DFTS WITH GENERIC INTERMEDIATE BASIS c. WE SAY THAT THE ALGORITHMS ARE OF TYPE $b \to c \to f$. THE OCCURRING PERMUTATIONS ARE SHOWN IN TABLE V.

Real DFTs, type 1 and 2: Algebra $\mathbb{R}[x]/x^n - 1$ with basis b and spectral basis f:

$$\mathcal{F}_{km}(b \to f) = P_m^{km} \cdot \left(\mathcal{F}_m(t \to f) \oplus \mathcal{G}_m(t \to f) \oplus \left(I_{k/2-1} \otimes_i B_{2m}^f \mathcal{S}_{2m,(i+1)/k}(c \to f)\right)\right) \cdot (\mathcal{F}_k(b \to c) \otimes I_m), \quad k \text{ even} \tag{27}$$

$$\mathcal{F}_{km}(b \to f) = P_m^{km} \cdot \left(\mathcal{F}_m(t \to f) \oplus \left(I_{\lfloor k/2 \rfloor} \otimes_i B_{2m}^f \mathcal{S}_{2m,(i+1)/k}(c \to f)\right)\right) \cdot \quad (\mathcal{F}_k(b \to c) \otimes I_m), \quad k \text{ odd} \tag{28}$$

Real DFTs, type 3 and 4: Algebra $\mathbb{R}[x]/x^n + 1$ with basis b and spectral basis f:

$$\mathcal{G}_{km}(b \to f) = Q_m^{km} \cdot \left(I_k \otimes_i B_{2m}^f \mathcal{S}_{2m,(i+1/2)/k}(c \to f)\right) \cdot \quad (\mathcal{G}_k(b \to c) \otimes I_m), \quad k \text{ even} \tag{29}$$

$$\mathcal{G}_{km}(b \to f) = Q_m^{km} \cdot \left(\left(I_{\lfloor k/2 \rfloor} \otimes_i B_{2m}^f \mathcal{S}_{2m,(i+1/2)/k}(c \to f)\right) \oplus \mathcal{G}_m(t \to f)\right) \cdot \quad (\mathcal{G}_k(b \to c) \otimes I_m), \quad k \text{ odd} \tag{30}$$

Skew real DFTs: Algebra $\mathbb{R}[x]/p_{n,r}$ with basis b and spectral basis f:

$$\mathcal{S}_{2km,r}(b \to f) = L_m^{2km} \cdot \left(I_k \otimes_i \mathcal{S}_{2m,(i+r)/k}(c \to f)\right) \cdot \left(\mathcal{S}_{2k,r}(b \to c) \otimes I_m\right) \tag{31}$$

TABLE V

PERMUTATIONS UNDERLYING THE PERMUTATION MATRICES IN TABLE IV. EACH PERMUTATION IS COMPRISED OF SEVERAL PIECEWISE INDEX MAPPINGS, WHICH MAP INPUT VECTOR INDICES TO OUTPUT VECTOR INDICES. GIVEN AN INDEX MAPPING $p \mapsto q$ THE CORREPOSNDING PERMUTATION MATRIX HAS 1S IN POSITIONS (q, p) AND 0S ELSEWHERE. NOTE, THAT ALL PIECEWISE INDEX MAPPINGS ARE STRIDES.

Permutation matrix	k	m	Index mappings (input vector $\mapsto$ output vector)	Bounds	
P_m^{km}	even	even	$mj + 2i + (0,1) \mapsto kj + 2ki + (0,1)$	$0 \le j \le 1$	$0 \le i < m/2$
			$2mj + 2i + (0,1) \mapsto \text{refl}_{2n}(2j + 2ki) + (0,1)$	$1 \le j < k/2$	$0 \le i < m$
P_m^{km}	even	odd	$(0, 2m-1) \mapsto (0,1)$		
			$1 + mj + 2i + (0,1) \mapsto kj + 2ki + (0,1)$	$0 \le j \le 1$	$0 \le i < \lfloor m/2 \rfloor$
			$2mj + 2i + (0,1) \mapsto \text{refl}_{2n}(2j + 2ki) + (0,1)$	$1 \le j < k/2$	$0 \le i < m$
P_m^{km}	odd	even	$2i + (0,1) \mapsto 2ki + (0,1)$		$0 \le i < m/2$
			$-m + 2mj + 2i + (0,1) \mapsto \text{refl}_{2n}(2j + 2ki) + (0,1)$	$1 \le j < \lfloor k/2 \rfloor$	$0 \le i < m$
P_m^{km}	odd	odd	$0 \mapsto 0$		
			$1 + 2i + (0,1) \mapsto 1 + 2ki + (0,1)$		$0 \le i < \lfloor m/2 \rfloor$
			$-m + 2mj + 2i + (0,1) \mapsto 1 + \text{refl}_{2n}(2j + 2ki) + (0,1)$	$1 \le j < \lfloor k/2 \rfloor$	$0 \le i < m$
Q_m^{km}	even	any	$2mj + 2i + (0,1) \mapsto \text{refl}_{2n-1}(2j + 2ki) + (0,1)$	$0 \le j < k/2$	$0 \le i < m$
Q_m^{km}	odd	any	$2mj + 2i + (0,1) \mapsto \text{refl}_{2n-1}(2j + 2ki) + (0,1)$	$0 \le j < \lfloor k/2 \rfloor$	$0 \le i < m$
			$km - m + 2i + (0,1) \mapsto k + 2ki + (0,1)$		$0 \le i < \lfloor m/2 \rfloor$
			$km \mapsto km$ (if m is odd)		

$$\text{refl}_{2n}(i) = \begin{cases} i, & i < n, \\ 2n - i, & \text{else} \end{cases}, \qquad \text{refl}_{2n-1}(i) = \begin{cases} i, & i < n, \\ 2n - 1 - i, & \text{else} \end{cases}$$

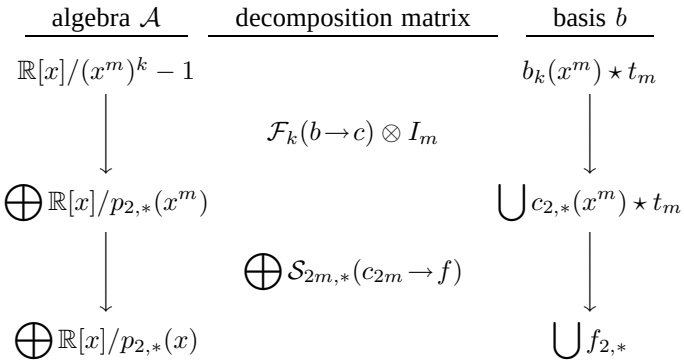


Fig. 2. Graphical representation of a generic fast algorithm for $\mathbb{R}[x]/(x^{km} - 1)$. For simplicity we omit the exact form of the parameter r in polynomials p and their bases, and use $*$ instead. We also omit m- and one-dimensional algebra summands that occur, and only show $2m$- and two-dimensional summands.

denote it with c. Possible choices for c include the same as for f and are shown in Table I(b). All these choices of c have the decomposition property $c_2(x^m) \star t_m = c_{2m}$.

Using Lemma 1, (19) is performed by $\mathcal{F}_k(b \to c) \otimes I_m$ and the basis in the first two m-dimensional summands in (19) is t_m and the basis in the algebras $\mathbb{R}[x]/p_{2m,*}(x)$ is $c_{2,*}(x^m) \star t_m = c_{2m,*}$.

Next, the complete decomposition in (20) is performed by $\mathcal{F}_m(t \to f)$ and $\mathcal{G}_m(t \to f)$ for the first two summands in (19) (not shown in Fig. 2) and by $B_{2m}^f \mathcal{S}_{2m,(i+1)/k}(c \to f)$ for the remaining summands. The matrix B was explained at the end of Section III.

The final step (21) is simply a reordering of the spectral components using a suitable permutation that is independent of the choice of bases. We omit the straightforward derivation and only show the result in Table V.

For odd k the decomposition has to be slightly adjusted since the second summands in (19)–(21) are not present.

The resulting algorithm applies to all real DFTs of type 1 and 2 and is shown in (27) and (28) in Table IV.

Case 2: $\mathcal{G}_n(b \to f)$. The algorithm derivation for $\mathcal{G}_n(b \to f)$ with underlying polynomial algebra $\mathbb{R}[x]/(x^n + 1)$ is analogous to Case 1 above but based on the decompositions (8) and (9).

As before, the basis choices considered in this paper are shown in Table I(b): $b = t$ and $f \in \{e, h, s, e^{1/2}, h^{1/2}\}$. The resulting algorithms are for real DFTs of type 3 and 4 and shown in Table IV.

Note that for even n, $x^n+1 = p_{n,1/4}(x)$, and $\mathcal{G}_n(b \to f)$ and $\mathcal{S}_{n,1/4}(b \to f)$ (discussed next) coincide up to a permutation and sign change of spectral components.

Case 3: $\mathcal{S}_{2n,r}(b \to f)$. The decomposition of the algebra $\mathbb{R}[x]/p_{2n,r}(x)$ in (10) underlying $\mathcal{S}_{2n,r}(b \to f)$ is derived analogously to the previous cases, using the decomposition property of $p_{2n,r}$ in Table I(b). The basis choices considered in this paper for these algebras are $b \in \{e, h, s\}$ and $f \in \{e, h, s, e^{1/2}, h^{1/2}\}$.

Summary. All algorithms are summarized in Table IV. We started with a transform that maps bases $b \to f$ (even though the algebras are different), and the algorithm derivation used an intermediate basis c (arising from the choice of spectral basis in the coarse decomposition). Hence we say the algorithms in Table IV are of type $b \to c \to f$; b and f define the initial transform, whereas the choice of c is a degree of freedom. From the decompositions in Table IV it is clear that the algorithms are $O(n \log(n))$ for "sufficiently composite" sizes n. The question is what choice yields the lowest exact operations count.

Below we instantiate the algorithms for two choices of c. The first we call "natural" since it breaks down real DFTs into real DFTs of a similar type. The second we call "Bruun type" since they generalize Bruun's FFT [5].

B. Natural Cooley-Tukey Type Real FFTs

For a given transform $\mathcal{F}_n(b \to f)$ it seems natural to choose the intermediate basis c in the algorithms in Table IV such that the coarse decomposition is performed by a transform of similar type. This means $c = e$ for $f \in \{e, e^{1/2}\}$ and $c = h$ for $f \in \{h, h^{1/2}\}$. Hence, the algorithms are of types $b \to e \to f$ and $b \to h \to f$. We instantiate Table IV for these choices in Table VI including the base cases needed to compute the transforms for two-power sizes. Together with Table V and size-2 and 4 base cases these equations provide self-contained algorithms specifications for all 4 types of RDFTs and DHTs for 2-power sizes. To compute other composite sizes, only the small prime size p base cases are missing for RDFTs, and size $2p$ base cases for skew transforms. The former can be obtained using arithmetic free conversions in Section V-C combined with Rader-type algorithms for the RDFT and DHT of type 1, for example, using [42] for the RDFT or an adaptation of [41] for the DHT. The latter can be obtained by converting $\mathcal{S}_{2k,r}(b \to b)$ into half-sized skew complex DFTs as explained in Section V-A.

For two-power sizes n, recursive application of the algorithms in Table VI yields an operations count of $\frac{5}{2} n \log_2(n) + O(n)$, which is known to be suboptimal. Inspection shows that the bulk of the computation is performed by the transforms $\mathcal{S}_{2k,r}(b \to b)$, where $b = e$ or $= h$. Hence, the overall cost will be dominated by the size 4 base cases for these transforms, which require 2 multiplications and 4 additions.

There are at least two ways of reducing the operations count to $2n \log_2(n) + O(n)$. The first converts the skew real DFTs $\mathcal{S}_{2k,r}(b \to b)$ into skew complex DFTs (of half size) and is explained in Section V-A. The second is explained next: it chooses different bases in the decomposition generalizing an idea by Bruun [5], which was later extended in [19], [20].

C. Bruun-Cooley-Tukey Type Real FFTs

The basic idea behind the algorithms in this section is to compute $\mathcal{F}_n(t \to f)$ and $\mathcal{G}_n(t \to f)$ with algorithms of type $t \to s \to f$. The bulk of the computation is then done by $\mathcal{S}_{2k,r}(s \to s)$ whose size 4 base case $\mathcal{S}_{4,r}(s \to s)$, also known as the "Bruun FFT butterfly," requires 2 multiplications and 2 additions, 2 less than before.

The algorithm is obtained by using $c = s$ in the general algorithms from Table IV. The choice of intermediate basis is fixed, regardless of the transform we are computing. This algorithm will then contain the BRDFTs defined in Table I(a), which themselves are expanded in the same way, i.e., via a $t \to s \to s$ algorithm.

All these algorithms are shown in Table VII including the base cases needed for two-power sizes n.

V. Algorithm Optimizations

In this section we derive three further optimizations applicable to the algorithms in Section IV:

1) the conversion of skew real DFTs into skew DFTs for savings in the operations count;
2) the regularization of the structure of type 1–2 real DFT algorithms; and
3) the arithmetic-free conversions between real DFTs.

As before, all optimizations are derived using the polynomial algebra framework, apply to the generic real DFT algorithms, and can hence be instantiated for all algorithms shown so far, in particular, for the RDFT, DHT, and BRDFT algorithms.

A. Converting Skew RDFTs Into Skew DFTs

We mentioned before that the algorithms in Section IV-B have the disadvantage that the occurring skew real DFTs $\mathcal{S}_{2k,r}(b \to b)$, $b \in \{e, h\}$, are too expensive when computed using (36). Here, we show that they can be converted into the skew DFTs in (25), which turns out to save operations. The conversion is based on the following lemma.

Lemma 2 The following mapping is an isomorphism [43] of $\mathbb{R}$-algebras, i.e., in particular, a bijective linear mapping:

$$\begin{aligned} \phi_{2n} : \mathbb{R}[x]/p_{2n,r}(x) &\to \mathbb{C}[x]/(x^n - w_r), \\ s(x) &\mapsto s(x) \bmod (x^n - w_r). \end{aligned} \tag{42}$$

TABLE VI

NATURAL COOLEY-TUKEY TYPE ALGORITHMS (TYPE $b \to e \to f$ AND $b \to h \to f$) FOR REAL DFTS OF SIZE $n = km$ AND SIZE 2 BASE CASES. EACH EQUATION HAS FOUR CASES. FOR EXAMPLE, THE FIRST CHOICE IN (32) RESTATES TABLE III.

Algorithms:

$$\begin{vmatrix} \mathrm{RDFT}_n \\ \mathrm{RDFT\text{-}2}_n \\ \mathrm{DHT}_n \\ \mathrm{DHT\text{-}2}_n \end{vmatrix} = P_m^n \cdot \left(\begin{vmatrix} \mathrm{RDFT}_m \\ \mathrm{RDFT\text{-}2}_m \\ \mathrm{DHT}_m \\ \mathrm{DHT\text{-}2}_m \end{vmatrix} \oplus \begin{vmatrix} \mathrm{RDFT\text{-}3}_m \\ \mathrm{RDFT\text{-}4}_m \\ \mathrm{DHT\text{-}3}_m \\ \mathrm{DHT\text{-}4}_m \end{vmatrix} \oplus \left(I_{k/2-1} \otimes_i \begin{vmatrix} B_{2m}^e \ \mathcal{S}_{2m,(i+1)/k}(e \to e) \\ B_{2m}^{e^{1/2}} \mathcal{S}_{2m,(i+1)/k}(e \to e^{1/2}) \\ B_{2m}^h \ \mathcal{S}_{2m,(i+1)/k}(h \to h) \\ B_{2m}^{h^{1/2}} \mathcal{S}_{2m,(i+1)/k}(h \to h^{1/2}) \end{vmatrix} \right) \right) \cdot \left(\begin{vmatrix} \mathrm{RDFT}_k \\ \mathrm{RDFT}_k \\ \mathrm{DHT}_k \\ \mathrm{DHT}_k \end{vmatrix} \otimes I_m \right), \quad k \text{ even} \tag{32}$$

$$\begin{vmatrix} \mathrm{RDFT}_n \\ \mathrm{RDFT\text{-}2}_n \\ \mathrm{DHT}_n \\ \mathrm{DHT\text{-}2}_n \end{vmatrix} = P_m^n \cdot \left(\begin{vmatrix} \mathrm{RDFT}_m \\ \mathrm{RDFT\text{-}2}_m \\ \mathrm{DHT}_m \\ \mathrm{DHT\text{-}2}_m \end{vmatrix} \oplus \left(I_{\lfloor k/2 \rfloor} \otimes_i \begin{vmatrix} B_{2m}^e \ \mathcal{S}_{2m,(i+1)/k}(e \to e) \\ B_{2m}^{e^{1/2}} \mathcal{S}_{2m,(i+1)/k}(e \to e^{1/2}) \\ B_{2m}^h \ \mathcal{S}_{2m,(i+1)/k}(h \to h) \\ B_{2m}^{h^{1/2}} \mathcal{S}_{2m,(i+1)/k}(h \to h^{1/2}) \end{vmatrix} \right) \right) \cdot \left(\begin{vmatrix} \mathrm{RDFT}_k \\ \mathrm{RDFT}_k \\ \mathrm{DHT}_k \\ \mathrm{DHT}_k \end{vmatrix} \otimes I_m \right), \quad k \text{ odd} \tag{33}$$

$$\begin{vmatrix} \mathrm{RDFT\text{-}3}_n \\ \mathrm{RDFT\text{-}4}_n \\ \mathrm{DHT\text{-}3}_n \\ \mathrm{DHT\text{-}4}_n \end{vmatrix} = Q_m^n \cdot \left(I_k \otimes_i \begin{vmatrix} B_{2m}^e \ \mathcal{S}_{2m,(i+1/2)/k}(e \to e) \\ B_{2m}^{e^{1/2}} \mathcal{S}_{2m,(i+1/2)/k}(e \to e^{1/2}) \\ B_{2m}^h \ \mathcal{S}_{2m,(i+1/2)/k}(h \to h) \\ B_{2m}^{h^{1/2}} \mathcal{S}_{2m,(i+1/2)/k}(h \to h^{1/2}) \end{vmatrix} \right) \cdot \left(\begin{vmatrix} \mathrm{RDFT\text{-}3}_k \\ \mathrm{RDFT\text{-}3}_k \\ \mathrm{DHT\text{-}3}_k \\ \mathrm{DHT\text{-}3}_k \end{vmatrix} \otimes I_m \right), \quad k \text{ even} \tag{34}$$

$$\begin{vmatrix} \mathrm{RDFT\text{-}3}_n \\ \mathrm{RDFT\text{-}4}_n \\ \mathrm{DHT\text{-}3}_n \\ \mathrm{DHT\text{-}4}_n \end{vmatrix} = Q_m^n \cdot \left(\left(I_{\lfloor k/2 \rfloor} \otimes_i \begin{vmatrix} B_{2m}^e \ \mathcal{S}_{2m,(i+1/2)/k}(e \to e) \\ B_{2m}^{e^{1/2}} \mathcal{S}_{2m,(i+1/2)/k}(e \to e^{1/2}) \\ B_{2m}^h \ \mathcal{S}_{2m,(i+1/2)/k}(h \to h) \\ B_{2m}^{h^{1/2}} \mathcal{S}_{2m,(i+1/2)/k}(h \to h^{1/2}) \end{vmatrix} \right) \oplus \begin{vmatrix} \mathrm{RDFT\text{-}3}_m \\ \mathrm{RDFT\text{-}4}_m \\ \mathrm{DHT\text{-}3}_m \\ \mathrm{DHT\text{-}4}_m \end{vmatrix} \right) \cdot \left(\begin{vmatrix} \mathrm{RDFT\text{-}3}_k \\ \mathrm{RDFT\text{-}3}_k \\ \mathrm{DHT\text{-}3}_k \\ \mathrm{DHT\text{-}3}_k \end{vmatrix} \otimes I_m \right), \quad k \text{ odd} \tag{35}$$

$$\begin{vmatrix} \mathcal{S}_{2n,r}(e \to e) \\ \mathcal{S}_{2n,r}(e \to e^{1/2}) \\ \mathcal{S}_{2n,r}(h \to h) \\ \mathcal{S}_{2n,r}(h \to h^{1/2}) \end{vmatrix} = L_m^{2n} \cdot \left(I_k \otimes_i \begin{vmatrix} \mathcal{S}_{2m,(i+r)/k}(e \to e) \\ \mathcal{S}_{2m,(i+r)/k}(e \to e^{1/2}) \\ \mathcal{S}_{2m,(i+r)/k}(h \to h) \\ \mathcal{S}_{2m,(i+r)/k}(h \to h^{1/2}) \end{vmatrix} \right) \cdot \left(\begin{vmatrix} \mathcal{S}_{2k,r}(e \to e) \\ \mathcal{S}_{2k,r}(e \to e) \\ \mathcal{S}_{2k,r}(h \to h) \\ \mathcal{S}_{2k,r}(h \to h) \end{vmatrix} \otimes I_m \right) \tag{36}$$

Base cases:

$$\begin{aligned}
B_{2m}^e &= I_m \oplus D_m, & B_{2m}^{e^{1/2}} &= I_m \oplus -D_m, \\
B_{2m}^h &= I_{2\lceil m/2 \rceil} \oplus (I_{\lfloor m/2 \rfloor} \otimes J_2), & B_{2m}^{h^{1/2}} &= I_{2\lceil m/2 \rceil} \oplus (I_{\lfloor m/2 \rfloor} \otimes J_2), \\
\mathrm{RDFT}_2 &= F_2, & \mathrm{DHT}_2 &= F_2, \\
\mathrm{RDFT\text{-}2}_2 &= F_2, & \mathrm{DHT\text{-}2}_2 &= F_2, \\
\mathrm{RDFT\text{-}3}_2 &= D_2 = \mathrm{diag}(1,-1), & \mathrm{DHT\text{-}3}_2 &= F_2, \\
\mathrm{RDFT\text{-}4}_2 &= \tfrac{\sqrt{2}}{2} \begin{bmatrix} 1 & -1 \\ -1 & -1 \end{bmatrix}, & \mathrm{DHT\text{-}4}_2 &= \sqrt{2} I_2, \\
\mathrm{RDFT\text{-}3}_4 &= \mathrm{diag}(1,-1,1,1)(F_2 \otimes I_2)(I_2 \oplus R_{1/8}) L_2^4, & \mathrm{DHT\text{-}3}_4 &= (I_2 \oplus J_2)(F_2 \otimes I_2)(F_2 \oplus \sqrt{2} D_2) L_2^4, \\
\mathrm{RDFT\text{-}4}_4 &= (D_2 \oplus D_2 J_2)(F_2 \otimes I_2)(R_{1/16} \oplus R_{3/16}) L_2^4, & \mathrm{DHT\text{-}4}_4 &= \mathrm{diag}(1,1,1,-1)(F_2 \otimes I_2)(R_{-1/16} \oplus R_{1/16}) L_2^4, \\
\mathcal{S}_{4,r}(e \to e) &= (F_2 \otimes I_2)(I_2 \oplus R_{-r/2}) L_2^4, & \mathcal{S}_{4,r}(h \to h) &= D_4 (F_2 \otimes I_2)(I_2 \oplus R_{r/2}) D_4 L_2^4, \\
\mathcal{S}_{4,r}(e \to e^{1/2}) &= (I_2 \oplus D_2 J_2)(F_2 \otimes I_2)(R_{-r/4} \oplus R_{-3r/4}) L_2^4, & \mathcal{S}_{4,r}(h \to h^{1/2}) &= (I_2 \oplus -D_2 J_2)(F_2 \otimes I_2)(R_{r/4} \oplus R_{3r/4}) D_4 L_2^4.
\end{aligned}$$

Proof: By the CRT, ϕ_{2n} is a homomorphism of algebras. It remains to show that it is also bijective. Since $\mathbb{R}[x]/p_{2n,r}(x)$ and $\mathbb{C}[x]/(x^n - w_r)$ have the same dimension $2n$ as $\mathbb{R}$-vector spaces, it suffices to show that ϕ_{2n} is injective.

$\phi_{2n}(s(x)) = 0$ implies that $x^n - w_r$ divides $s(x)$. Since s is real, $p_{2n,r}$ divides s and hence $s(x) \equiv 0 \bmod p_{2n,r}(x)$ as desired. ∎

Using Lemma 2 we establish the diagram in Fig. 3, which expresses that

$$\mathcal{S}_{2n,r}(b \to f) = \left(\bigoplus_{0 \le i < n} \phi_2^{-1} \right) \left[\overline{\mathrm{DFT}(r)} \text{ or } \overline{\mathrm{DFT\text{-}2}(r)} \right] \phi_{2n}.$$

The choice of DFT depends on f; the exact equations are computed next.

First, we note that this diagram represents *real* linear mappings between *real* algebras. Hence $\overline{\mathrm{DFT}(r)}$ or $\overline{\mathrm{DFT\text{-}2}(r)}$ in this diagram are real transforms that are obtained from $\mathrm{DFT}(r)$ or $\mathrm{DFT\text{-}2}(r)$ (defined later) as explained next.

$$\begin{array}{ccc}
\mathbb{R}[x]/p_{2n,r} & \xrightarrow{\ \phi_{2n}\ } & \mathbb{C}[x]/(x^n - w_r) \\
\Big\downarrow {\scriptstyle \mathcal{S}_{2n,r}(b \mapsto f)} & & \Big\downarrow {\scriptstyle \overline{\mathrm{DFT}(r)} \text{ or } \overline{\mathrm{DFT\text{-}2}(r)}} \\
\bigoplus_{0 \le i < n} \mathbb{R}[x]/p_{2n,(i+r)/n} & \xleftarrow{\ \bigoplus \phi_2^{-1}\ } & \bigoplus_{0 \le i < n} \mathbb{C}[x]/(x - w_{(i+r)/n})
\end{array}$$

Fig. 3. Translating skew real DFTs into skew complex DFTs.

Complex transforms, when performed by a computer, operate on real data organized, for example, using the interleaved complex format (alternating real and imaginary parts of the complex entries). Since the complex multiplication $(a+ib)(x+iy)$ is equivalent to the real multiplication $\left[\begin{smallmatrix} a & -b \\ b & a \end{smallmatrix}\right] \left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]$, every

TABLE VII

BRUUN-COOLEY-TUKEY-TYPE ALGORITHMS (TYPE $b \to s \to f$) FOR REAL DFTS AND BASE CASES.

Algorithms:

$$\begin{vmatrix} \mathrm{RDFT}_n \\ \mathrm{RDFT\text{-}2}_n \\ \mathrm{DHT}_n \\ \mathrm{DHT\text{-}2}_n \\ \mathrm{BRDFT}_n \end{vmatrix} = P_m^n \cdot \left(\begin{vmatrix} \mathrm{RDFT}_m \\ \mathrm{RDFT\text{-}2}_m \\ \mathrm{DHT}_m \\ \mathrm{DHT\text{-}2}_m \\ \mathrm{BRDFT}_m \end{vmatrix} \oplus \begin{vmatrix} \mathrm{RDFT\text{-}3}_m \\ \mathrm{RDFT\text{-}4}_m \\ \mathrm{DHT\text{-}3}_m \\ \mathrm{DHT\text{-}4}_m \\ \mathrm{BRDFT\text{-}3}_m \end{vmatrix} \oplus \left(I_{k/2-1} \otimes_i \begin{vmatrix} B^e_{2m} & \mathcal{S}_{2m,(i+1)/k}(s \to e) \\ B^{e^{1/2}}_{2m} & \mathcal{S}_{2m,(i+1)/k}(s \to e^{1/2}) \\ B^h_{2m} & \mathcal{S}_{2m,(i+1)/k}(s \to h) \\ B^{h^{1/2}}_{2m} & \mathcal{S}_{2m,(i+1)/k}(s \to h^{1/2}) \\ B^s_{2m} & \mathcal{S}_{2m,(i+1)/k}(s \to s) \end{vmatrix} \right) \right) \cdot (\mathrm{BRDFT}_k \otimes I_m), \quad k \text{ even} \tag{37}$$

$$\begin{vmatrix} \mathrm{RDFT}_n \\ \mathrm{RDFT\text{-}2}_n \\ \mathrm{DHT}_n \\ \mathrm{DHT\text{-}2}_n \\ \mathrm{BRDFT}_n \end{vmatrix} = P_m^n \cdot \left(\begin{vmatrix} \mathrm{RDFT}_m \\ \mathrm{RDFT\text{-}2}_m \\ \mathrm{DHT}_m \\ \mathrm{DHT\text{-}2}_m \\ \mathrm{BRDFT}_m \end{vmatrix} \oplus \left(I_{\lfloor k/2 \rfloor} \otimes_i \begin{vmatrix} B^e_{2m} & \mathcal{S}_{2m,(i+1)/k}(s \to e) \\ B^{e^{1/2}}_{2m} & \mathcal{S}_{2m,(i+1)/k}(s \to e^{1/2}) \\ B^h_{2m} & \mathcal{S}_{2m,(i+1)/k}(s \to h) \\ B^{h^{1/2}}_{2m} & \mathcal{S}_{2m,(i+1)/k}(s \to h^{1/2}) \\ B^s_{2m} & \mathcal{S}_{2m,(i+1)/k}(s \to s) \end{vmatrix} \right) \right) \cdot \quad (\mathrm{BRDFT}_k \otimes I_m), \quad k \text{ odd} \tag{38}$$

$$\begin{vmatrix} \mathrm{RDFT\text{-}3}_n \\ \mathrm{RDFT\text{-}4}_n \\ \mathrm{DHT\text{-}3}_n \\ \mathrm{DHT\text{-}4}_n \\ \mathrm{BRDFT\text{-}3}_n \end{vmatrix} = Q_m^n \cdot \left(I_k \otimes_i \begin{vmatrix} B^e_{2m} & \mathcal{S}_{2m,(i+1/2)/k}(s \to e) \\ B^{e^{1/2}}_{2m} & \mathcal{S}_{2m,(i+1/2)/k}(s \to e^{1/2}) \\ B^h_{2m} & \mathcal{S}_{2m,(i+1/2)/k}(s \to h) \\ B^{h^{1/2}}_{2m} & \mathcal{S}_{2m,(i+1/2)/k}(s \to h^{1/2}) \\ B^s_{2m} & \mathcal{S}_{2m,(i+1/2)/k}(s \to s) \end{vmatrix} \right) \cdot \quad (\mathrm{BRDFT\text{-}3}_k \otimes I_m), \quad k \text{ even} \tag{39}$$

$$\begin{vmatrix} \mathrm{RDFT\text{-}3}_n \\ \mathrm{RDFT\text{-}4}_n \\ \mathrm{DHT\text{-}3}_n \\ \mathrm{DHT\text{-}4}_n \\ \mathrm{BRDFT\text{-}3}_n \end{vmatrix} = Q_m^n \cdot \left(\left(I_{\lfloor k/2 \rfloor} \otimes_i \begin{vmatrix} B^e_{2m} & \mathcal{S}_{2m,(i+1/2)/k}(s \to e) \\ B^{e^{1/2}}_{2m} & \mathcal{S}_{2m,(i+1/2)/k}(s \to e^{1/2}) \\ B^h_{2m} & \mathcal{S}_{2m,(i+1/2)/k}(s \to h) \\ B^{h^{1/2}}_{2m} & \mathcal{S}_{2m,(i+1/2)/k}(s \to h^{1/2}) \\ B^s_{2m} & \mathcal{S}_{2m,(i+1/2)/k}(s \to s) \end{vmatrix} \right) \oplus \begin{vmatrix} \mathrm{RDFT\text{-}3}_m \\ \mathrm{RDFT\text{-}4}_m \\ \mathrm{DHT\text{-}3}_m \\ \mathrm{DHT\text{-}4}_m \\ \mathrm{BRDFT\text{-}3}_m \end{vmatrix} \right) \cdot \quad (\mathrm{BRDFT\text{-}3}_k \otimes I_m), \quad k \text{ odd} \tag{40}$$

$$\begin{vmatrix} \mathcal{S}_{2n,r}(s \to e) \\ \mathcal{S}_{2n,r}(s \to e^{1/2}) \\ \mathcal{S}_{2n,r}(s \to h) \\ \mathcal{S}_{2n,r}(s \to h^{1/2}) \\ \mathcal{S}_{2n,r}(s \to s) \end{vmatrix} = L_m^{2n} \cdot \left(I_k \otimes_i \begin{vmatrix} \mathcal{S}_{2m,(i+r)/k}(s \to e) \\ \mathcal{S}_{2m,(i+r)/k}(s \to e^{1/2}) \\ \mathcal{S}_{2m,(i+r)/k}(s \to h) \\ \mathcal{S}_{2m,(i+r)/k}(s \to h^{1/2}) \\ \mathcal{S}_{2m,(i+r)/k}(s \to s) \end{vmatrix} \right) \cdot \left(\mathcal{S}_{2k,r}(s \to s) \otimes I_m \right) \tag{41}$$

Base cases:

$$B^s_{2m} = I_{2m},$$

$$\mathrm{BRDFT}_2 = F_2, \qquad \mathrm{BRDFT\text{-}3}_2 = I_2,$$

$$\mathrm{BRDFT\text{-}3}_4 = (F_2 \otimes I_2) \begin{pmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & -\sqrt{2} \\ 0 & 0 & \sqrt{2} & 0 \end{pmatrix}, \qquad \mathcal{S}_{4,r}(s \to s) = (F_2 \otimes I_2) \begin{pmatrix} 0 & 1 & 0 & -1 \\ -1 & 0 & 1 & 0 \\ 2c_{r/2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 2c_{r/2} \end{pmatrix},$$

$$\mathcal{S}_{4,r}(s \to e) = (F_2 \otimes I_2) \left(\begin{bmatrix} c_r & 1 \\ s_r & 0 \end{bmatrix} \oplus \mathrm{diag}(c_{r/2}, s_{r/2}) F_2 \right) L^4_2, \quad \mathcal{S}_{4,r}(s \to e^{1/2}) = (I_2 \oplus D_2 J_2)(F_2 \otimes I_2) \left(\begin{bmatrix} c_{3r/4} & c_{r/4} \\ s_{3r/4} & s_{-r/4} \end{bmatrix} \oplus \begin{bmatrix} c_{r/4} & c_{3r/4} \\ s_{r/4} & s_{-3r/4} \end{bmatrix} \right) L^4_2,$$

$$\mathcal{S}_{4,r}(s \to h) = (F_2 \otimes I_2) \left(\begin{bmatrix} \mathsf{cms}_r & 1 \\ \mathsf{cas}_r & 1 \end{bmatrix} \oplus \begin{bmatrix} \mathsf{cms}_{r/2} & \mathsf{cas}_{r/2} \\ \mathsf{cas}_{r/2} & \mathsf{cms}_{r/2} \end{bmatrix} \right) L^4_2, \quad \mathcal{S}_{4,r}(s \to h^{1/2}) = (D_2 \oplus J_2)(F_2 \otimes I_2) \left(\begin{bmatrix} \mathsf{cms}_{3r/4} & \mathsf{cas}_{r/4} \\ \mathsf{cas}_{3r/4} & \mathsf{cms}_{r/4} \end{bmatrix} \oplus \begin{bmatrix} \mathsf{cms}_{r/4} & \mathsf{cas}_{3r/4} \\ \mathsf{cas}_{r/4} & \mathsf{cms}_{3r/4} \end{bmatrix} \right) L^4_2.$$

TABLE VIII

CONVERSION FROM SKEW REAL DFTS TO SKEW DFTS USING LEMMA 2.

Skew RDFTs occurring in natural Cooley-Tukey type algorithms (Table VI):

$$\mathcal{S}_{2n,r}(e \to e) = \overline{\mathrm{DFT}_n \, \mathrm{diag}_i \, w_{ri/n}} L^{2n}_n, \qquad \mathcal{S}_{2n,r}(e \to e^{1/2}) = \overline{\mathrm{DFT\text{-}2}_n \, \mathrm{diag}_i \, w_{r(i+1/2)/n}} L^{2n}_n, \tag{43}$$

$$\mathcal{S}_{2n,r}(h \to h) = \overline{\mathrm{DFT}_n \, \mathrm{diag}_i \, w_{ri/n}} L^{2n}_n, \qquad \mathcal{S}_{2n,r}(h \to h^{1/2}) = D_{2n} \overline{\mathrm{DFT\text{-}2}_n \, \mathrm{diag}_i \, w_{r(i+1/2)/n}} L^{2n}_n. \tag{44}$$

Skew RDFTs occurring in Bruun-Cooley-Tukey type algorithms (Table VII):

$$\mathcal{S}_{2n,r}(s \to e) = \overline{\mathrm{DFT}_n} \left(I_n \otimes_i \begin{bmatrix} c_{ri/n-r} & c_{ri/n} \\ -s_{ri/n-r} & -s_{ri/n} \end{bmatrix} \right) L^{2n}_n, \quad \mathcal{S}_{2n,r}(s \to e^{1/2}) = \overline{\mathrm{DFT\text{-}2}_n} \left(I_n \otimes_i \begin{bmatrix} c_{r(i+1/2)/n-r} & c_{r(i+1/2)/n} \\ -s_{r(i+1/2)/n-r} & -s_{r(i+1/2)/n} \end{bmatrix} \right) L^{2n}_n, \tag{45}$$

$$\mathcal{S}_{2n,r}(s \to h) = \overline{\mathrm{DFT}_n} \left(I_n \otimes_i \begin{bmatrix} \mathsf{cas}_{ri/n-r} & \mathsf{cas}_{ri/n} \\ \mathsf{cms}_{ri/n-r} & \mathsf{cms}_{ri/n} \end{bmatrix} \right) L^{2n}_n, \quad \mathcal{S}_{2n,r}(s \to h^{1/2}) = D_{2n} \overline{\mathrm{DFT\text{-}2}_n} \left(I_n \otimes_i \begin{bmatrix} \mathsf{cas}_{r(i+1/2)/n-r} & \mathsf{cas}_{r(i+1/2)/n} \\ \mathsf{cms}_{r(i+1/2)/n-r} & \mathsf{cms}_{r(i+1/2)/n} \end{bmatrix} \right) L^{2n}_n. \tag{46}$$

complex matrix-vector multiplication $Mx \in \mathbb{C}^n$ is equivalent to $\overline{M}x' \in \mathbb{R}^{2n}$, where we define $\overline{M}$ by replacing every entry $a+ib$ of M with $\left[\begin{smallmatrix} a & -b \\ b & a \end{smallmatrix}\right]$, and x' is x in the interleaved complex format.

Algebraically, if M is a polynomial transform for $\mathbb{C}[x]/p(x)$ viewed as a complex vector space with basis t_n and spectral bases (1), then $\overline{M}$ is the corresponding matrix with respect to the *real* vector space basis $(1, j, x, jx, \ldots, x^{n-1}, jx^{n-1})$ and spectral bases $(1, j)$.

With this information, we can compute the matrix representations of ϕ_{2n} and ϕ_2^{-1} in the above diagram with respect to the different choices of b and f.

In the case of $b \in \{e, h\}$ and spectral basis $f \in \{e, h\}$ we convert $\mathcal{S}_{2n,r}(b \to f)$ into $\overline{\mathrm{DFT}_n(r)}$, and the base changes consist of a permutation (L_n^{2n}) and in some cases extra sign changes. The results are shown in Table VIII, where $\mathrm{DFT}(r)$ is factored as

$$\begin{aligned}\mathrm{DFT}_n(r) &= \left[w_{(k+r)l/n}\right]_{0\le k,l<n} \\ &= \mathrm{DFT}_n \,\mathrm{diag}_{0\le l<n}(w_{rl/n}). \end{aligned} \tag{47}$$

Note, that incidentally $\mathcal{S}_{2n,r}(e \to e) = \mathcal{S}_{2n,r}(h \to h)$, because a simplification was possible to the $\mathcal{S}_{2n,r}(h \to h)$ factorization. This fact was (indirectly) discovered in [8], when the authors derived a split-radix Hartley transform algorithm with improved operation count.

In the case of $f \in \{e^{1/2}, h^{1/2}\}$, the corresponding DFT is type 2, DFT-2_n, whose skew equivalent takes the form

$$\begin{aligned}\text{DFT-2}_n(r) &= \left[w_{(k+r)(l+1/2)/n}\right]_{0\le k,l<n} \\ &= \text{DFT-2}_n \,\mathrm{diag}_{0\le l<n}(w_{r(l+1/2)/n}). \end{aligned} \tag{48}$$

The base changes are again arithmetic free.

Finally, in the case of $b = s$, ϕ_{2n} is no longer trivial, and takes the form

$$\phi_{2n} = \left(I_n \otimes \begin{bmatrix} c_r & 1 \\ s_r & 0 \end{bmatrix}\right) L_n^{2n},$$

but it can be fused with the block diagonal $\overline{\mathrm{diag}_l(w_{r(l+1/2)/n})}$ from (48) to save multiplications, leading to the final result shown in Table VIII.

Identities (43)–(44) from Table VIII for computing the skew real DFTs reduce the operations count compared to (36). Identities (45)–(46) do not reduce the operations count compared to (41), but can be used as base cases.

B. Regularized Algorithms

It is often more important for an algorithm to have a regular structure than a minimal operations count to enable parallelization, vectorization, or efficient hardware implementation. The real DFTs are inherently more irregular than the DFT since their spectral components have dimensions one or two as shown in (6)–(9) with the exception of (8).

In (6), this problem can be alleviated by not performing a complete real decomposition but leaving $(x-1)(x+1) = x^2 - 1$ "fused." This idea yields the "regularized" version of (6):

$$\mathbb{R}[x]/(x^n-1) \to \mathbb{R}[x]/(x^2-1) \oplus \bigoplus_{0<i<n/2} \mathbb{R}[x]/p_{2,i/n}. \tag{49}$$

There is no equivalent technique for odd n, i.e., for (7) and (9).

We will denote the transform associated with (49) as $\mathcal{R}_n(b \to f)$, where b is the chosen basis in the algebra, f is the chosen spectral basis, and n is necessarily even. Table IX shows the new transforms we define.

TABLE IX
REGULARIZED REAL DFTS.

Transform	Algebra	Basis b	Spectral basis f	Unified notation
URDFT_n	x^n-1	t	e	$\mathcal{R}_n(t \to e)$
UDHT_n	x^n-1	t	h	$\mathcal{R}_n(t \to h)$
UBRDFT_n	x^n-1	t	s	$\mathcal{R}_n(t \to s)$

To complete the real decomposition of (49), $\mathbb{R}[x]/(x^2-1)$ must be decomposed using a butterfly matrix, and thus

$$\mathcal{F}_n(b \to f) = (F_2 \oplus I_{n-2})\mathcal{R}_n(b \to f).$$

The idea is now to use $\mathcal{R}_k(t \to c)$, $c \in \{e, h, s\}$, instead of $\mathcal{F}_k(t \to c)$ in the first, coarse decomposition step. The consequence is that in (19) the first two summands stay fused. More precisely, the decomposition now takes the regularized form

$$\begin{aligned} &\mathbb{R}[x]/((x^m)^k - 1) \\ &\to \left[\mathbb{R}[x]/(x^{2m}-1)\right] \oplus \left[\bigoplus_{0<i<k/2} \mathbb{R}[x]/p_{2m,i/k}\right] && (50) \\ &\to \left[\mathbb{R}[x]/(x-1) \oplus \mathbb{R}[x]/(x+1) \oplus \bigoplus_{0<i<m} \mathbb{R}[x]/p_{2,i/2m}\right] \\ &\qquad \oplus \left[\bigoplus_{0<i<k/2} \bigoplus_{0<j<m} \mathbb{R}[x]/p_{2,(j+i/k)/m}\right] && (51) \\ &\to \mathbb{R}[x]/(x-1) \oplus \mathbb{R}[x]/(x+1) \oplus \bigoplus_{0<i<km/2} \mathbb{R}[x]/p_{2,i/km} && (52) \end{aligned}$$

In step (50) all summands now have the same dimension $2m$. Step (51) completely decomposes the polynomial algebras of dimension $2m$ over $\mathbb{R}$. A suitable permutation in (52) then reorders the one- and two-dimensional polynomial algebras into the required order.

The resulting algorithms are given in Table X. They are the regularized variants of the algorithms in Tables VI and VII for even n.

C. Arithmetic-free conversions

Using a variable change in the algebra. If n is odd, and we set $y = -x$, we have the following

$$\begin{aligned}\mathbb{R}[x]/(x^n+1) &= \mathbb{R}[-y]/((-y)^n+1) = \mathbb{R}[-y]/(-y^n+1) \\ &= \mathbb{R}[-y]/(y^n-1) \cong \mathbb{R}[y]/(y^n-1).\end{aligned}$$

The above implies that for odd n the transforms of type 3 can be converted to transforms of type 1 and vice versa, using simple base changes and a permutation of spectral

TABLE X
REGULARIZED ALGORITHMS.

Regularized version of (27) **in Table IV:**

$$\begin{vmatrix}\mathcal{F}_n(b\to f)\\ \mathcal{R}_n(b\to f)\end{vmatrix} = P_m^n \left(P'_{2m} \begin{vmatrix}\mathcal{F}_{2m}(t\to f)\\ \mathcal{R}_{2m}(t\to f)\end{vmatrix} \oplus I_{k/2-1}\otimes_i B^f_{2m}\mathcal{S}_{2m,(i+1)/k}(c\to f)\right)\left(\mathcal{R}_k(b\to c)\otimes I_m\right), \quad k \text{ even}, \tag{53}$$

$$P'_{2m} = \begin{cases}(L^m_2\otimes I_2), & m \text{ even},\\ (I_2\oplus(L^{m-1}_2\otimes I_2)), & m \text{ odd}.\end{cases} \qquad P \text{ is given in Table V}$$

Regularized version of (32) **in Table VI:**

$$\begin{vmatrix}\mathrm{RDFT}_n\\ \mathrm{RDFT\text{-}2}_n\\ \mathrm{URDFT}_n\\ \mathrm{DHT}_n\\ \mathrm{DHT\text{-}2}_n\\ \mathrm{UDHT}_n\end{vmatrix} = P^n_m \left(P'_{2m} \begin{vmatrix}\mathrm{RDFT}_{2m}\\ \mathrm{RDFT\text{-}2}_{2m}\\ \mathrm{URDFT}_{2m}\\ \mathrm{DHT}_{2m}\\ \mathrm{DHT\text{-}2}_{2m}\\ \mathrm{UDHT}_{2m}\end{vmatrix} \oplus \left(I_{k/2-1}\otimes_i \begin{vmatrix} B^e_{2m} & \mathcal{S}_{2m,(i+1)/k}(e\to e)\\ B^{e^{1/2}}_{2m} & \mathcal{S}_{2m,(i+1)/k}(e\to e^{1/2})\\ B^e_{2m} & \mathcal{S}_{2m,(i+1)/k}(e\to e)\\ B^h_{2m} & \mathcal{S}_{2m,(i+1)/k}(h\to h)\\ B^{h^{1/2}}_{2m} & \mathcal{S}_{2m,(i+1)/k}(h\to h^{1/2})\\ B^h_{2m} & \mathcal{S}_{2m,(i+1)/k}(h\to h)\end{vmatrix}\right)\right) \left(\begin{vmatrix}\mathrm{URDFT}_k\\ \mathrm{URDFT}_k\\ \mathrm{URDFT}_k\\ \mathrm{UDHT}_k\\ \mathrm{UDHT}_k\\ \mathrm{UDHT}_k\end{vmatrix} \otimes I_m \right), \quad k \text{ even}, \tag{54}$$

Regularized version of (37) **in Table VII:**

$$\begin{vmatrix}\mathrm{RDFT}_n\\ \mathrm{RDFT\text{-}2}_n\\ \mathrm{DHT}_n\\ \mathrm{DHT\text{-}2}_n\\ \mathrm{UBRDFT}_n\end{vmatrix} = P^n_m \left(P'_{2m} \begin{vmatrix}\mathrm{RDFT}_{2m}\\ \mathrm{RDFT\text{-}2}_{2m}\\ \mathrm{DHT}_{2m}\\ \mathrm{DHT\text{-}2}_{2m}\\ \mathrm{UBRDFT}_{2m}\end{vmatrix} \oplus \left(I_{k/2-1}\otimes_i \begin{vmatrix} B^e_{2m} & \mathcal{S}_{2m,(i+1)/k}(s\to e)\\ B^{e^{1/2}}_{2m} & \mathcal{S}_{2m,(i+1)/k}(s\to e^{1/2})\\ B^h_{2m} & \mathcal{S}_{2m,(i+1)/k}(s\to h)\\ B^{h^{1/2}}_{2m} & \mathcal{S}_{2m,(i+1)/k}(s\to h^{1/2})\\ B^s_{2m} & \mathcal{S}_{2m,(i+1)/k}(s\to s)\end{vmatrix}\right)\right) (\mathrm{UBRDFT}_k\otimes I_m). \quad k \text{ even}. \tag{55}$$

Base cases:

$$\mathrm{URDFT}_4 = \mathrm{diag}(1,1,1,-1)(F_2\otimes I_2), \qquad \mathrm{UDHT}_4 = (I_2\oplus F_2)(F_2\otimes I_2), \qquad \mathrm{UBRDFT}_4 = (F_2\otimes I_2).$$

components. For the transforms in Table I(a), $b = t_n$, and we have:

$$\begin{aligned}\mathcal{G}_n(t\to f) &= P\cdot\mathcal{F}_n(t(-x)\to f(-x))\\ &= P\cdot\mathcal{F}_n(t\to f(-x))\cdot D_n\\ &= P\cdot B\cdot\mathcal{F}_n(t\to f)\cdot D_n.\end{aligned}$$

Above D_n is a diagonal matrix with ± 1s defined in (15), and is the base change from t_n to $t_n(-x)$, B is the base change from f to $f(-x)$ in each spectral component whose precise form depends on the choice of f, and P is the permutation of the spectral components, which does not depend on f.

To compute the permutation we check how the spectral components of $\mathbb{R}[x]/(x^n-1)$ are mapped under the substitution:

$$\begin{aligned}\mathbb{R}[x]/((-x)-1) &= \mathbb{R}[x]/(x+1),\\ \mathbb{R}[x]/p_{2n,r}(-x) &= \mathbb{R}[x]/x^2+2xc_r+1\\ &= \mathbb{R}[x]/x^2-2xc_{r+1/2}+1\\ &= \mathbb{R}[x]/p_{2n,r+1/2}(x) = \mathbb{R}[x]/p_{2n,1/2-r}.\end{aligned}$$

Since we assume the ordering by ascending r, and $r<1/2$, the above mapping implies that the order is simply reversed. The first one-dimensional spectral component becomes $\mathbb{R}[x]/(x+1)$ and is moved to the end. Thus we have:

$$\mathcal{G}_n(b\to f) = \begin{bmatrix} & J_{\lfloor n/2\rfloor}\otimes I_2\\ 1 & \end{bmatrix}\cdot B\cdot\mathcal{F}_n(b\to f)\cdot D_n. \tag{56}$$

We now instantiate (56) by computing B for $f\in\{e,h,e^{1/2},h^{1/2},s\}$. The result is shown in (57)–(59) in Table XI. In the equations we fuse B with the final permutation into a single matrix.

Using the basis properties. It is easily shown that for any polynomial q

$$\mathcal{F}_n(t\to b) = \mathcal{F}_n(qt\to qb).$$

For odd n, we can use this property to convert transforms of type 2 (with spectral bases $\{e^{1/2},h^{1/2}\} = x^{-1/2}\{e,h^*\}$; see Table I(c)) into transforms of type 1 (with spectral bases e and h).

The basic idea is to set $q = x^{(n+1)/2} = \pm x^{1/2}$ in $\mathbb{R}[x]/(x^n-1)$, which holds for odd n. Due to non-uniqueness of $x^{1/2}$ also in the spectral components $\mathbb{R}[x]/p_{2,r}$, we have $q\equiv\pm x^{1/2} \mod p_{2,r}$, with the sign depending on r.

Given a spectral basis $f = x^{-1/2}c$ (where $c\in\{e,h^*\}$) and $q = x^{(n+1)/2}$ we have:

$$\begin{aligned}\mathcal{F}_n(t\to x^{-1/2}c) &= \mathcal{F}_n(qt\to qx^{-1/2}c)\\ &= \mathcal{F}_n(x^{(n+1)/2}t\to\pm c) = \mathcal{F}_n(t\to\pm c)B.\end{aligned} \tag{62}$$

Above, by abuse of notation, we denote with $\pm c$ the modified basis c in which the sign of each spectral component depends on r. It is $+$ if $x^{(n+1)/2}\equiv+x^{1/2} \mod p_{2,r}$, and $-$ otherwise. B is the base change from the shifted basis $x^{(n+1)/2}t_n$ to t_n.

We instantiate (62) for $f=\{e^{1/2},h^{1/2}\}$ in (60)–(61) in Table (XI). These equations can be combined with (57)–(58), making it possible to convert any type of DHT or RDFT of odd size into any other type for free.

VI. ALGORITHM ANALYSIS AND DISCUSSION

Table XII gives the arithmetic cost for the algorithms shown in this paper. In the remainder, we first identify three

TABLE XI

ARITHMETIC FREE CONVERSIONS BETWEEN RDFTS FOR ODD n. THE PROPERTIES BELOW CAN BE COMBINED TO YIELD FREE CONVERSIONS BETWEEN ANY OF TYPE 1–4 TRANSFORMS OF SAME KIND.

Identities based on (56)**:**

$$\text{RDFT-3}_n = \begin{bmatrix} 1 & & \\ & & J_{\lfloor n/2 \rfloor} \otimes \text{diag}(1,-1) \end{bmatrix} \text{RDFT}_n\, D, \qquad \text{RDFT-4}_n = \begin{bmatrix} 1 & & \\ & & -J_{n-1} \end{bmatrix} \text{RDFT-2}_n\, D, \tag{57}$$

$$\text{DHT-3}_n = J_n\, \text{DHT}_n\, D, \qquad \text{DHT-4}_n = \begin{bmatrix} 1 & & \\ & & J_{\lfloor n/2 \rfloor} \otimes \text{diag}(1,-1) \end{bmatrix} \text{DHT-2}_n\, D, \tag{58}$$

$$\text{BRDFT-3}_n = \begin{bmatrix} -1 & & \\ & & J_{\lfloor n/2 \rfloor} \otimes \text{diag}(1,-1) \end{bmatrix} \text{BRDFT}_n D. \tag{59}$$

Identities based on (62)**:**

$$\text{DHT-2}_n = (I_1 \oplus -\text{diag}_{0 \le i < n-1}(-1)^{i+\lfloor \frac{i}{2} \rfloor})\, \text{DHT}_n \begin{bmatrix} I_{\lceil n/2 \rceil} \\ I_{\lfloor n/2 \rfloor} \end{bmatrix}, \tag{60}$$

$$\text{RDFT-2}_n = (I_1 \oplus \text{diag}_{0 \le i < n-1} -(-1)^{\lfloor \frac{i}{2} \rfloor})\, \text{RDFT}_n \begin{bmatrix} I_{\lceil n/2 \rceil} \\ I_{\lfloor n/2 \rfloor} \end{bmatrix}. \tag{61}$$

good choices among them. Note that the best choice strongly depends on the implementation platform. Then we review published real DFT algorithms and relate them to the algorithms in this paper where possible. Finally, we briefly discuss inverse real DFTs and convolution.

A. Good Choices of Algorithms

Many possible real DFT algorithms are provided in this paper. We identify three choices with desirable features discussed next.

The regularized natural algorithm uses (54) together with the natural skew transform algorithm (36) to obtain the simplest overall structure. For the 2-power size, the arithmetic cost is $\frac{5}{2} n \log_2 n + O(n)$. The recursive formulation can be easily converted to an iterative algorithm amenable to hardware implementation.

The regularized natural algorithm with improved skew transform algorithm uses (54) with (43)–(44) to improve the arithmetic cost. The minimal cost of $2n \log_2 n + O(n)$ is achieved for $m = n/4$ in (54), but the maximal cost remains the same.

This algorithm is the main choice for automatic RDFT program and library generation in Spiral [29], [52].

The regularized Bruun algorithm is obtained with (55) combined with the Bruun skew transform algorithm (41). This combination also yields a cost of $2n \log_2 n + O(n)$ independent of the radix (albeit with a different multiply/add balance), and at the same time can be easily unfolded into an iterative algorithm (as in option 1 above). In the iterative version, two times less multiplications and two times less multiplicative constants are needed due to simpler base case. However, it is less numerically stable than other choices [17], [24].

In its iterative radix-4 version (which can be obtained in exactly the same way as for standard real Cooley-Tukey FFT) it requires only 2 multiplications per stage, whereas the natural RDFT algorithm needs 4 multiplications per stage.

Note that algorithms with $\frac{17}{9} n \log_2(n) + O(n)$ operations exist [11] obtained from a properly chosen $2n \log_2(n) + O(n)$ algorithm by propagating and canceling constants without changing the computation structure. This method is orthogonal to ours, which identifies the different computation structures possible.

Also note that choices other than the above may be competitive. For example, the library FFTW [28] uses a non-regularized RDFT algorithm as explained in the following subsection.

B. Related Algorithms In The Literature

RDFT. The first RDFT algorithm was derived in 1968 by Bergland in [1]. Bergland's algorithm is an iterative equivalent of the combination of (54) with $m = 2$ and (36). The follow-up paper [2] gives a radix-8 version, which is the same algorithm with $m = 4$. The arithmetic cost of these algorithms is $\frac{5}{2} n \log_2 n + O(n)$, and does not depend on the split (i.e., the choice of factorization $n = km$). The tensor product formulation of the radix-4 variant of Bergland's algorithm is given in [32].

One can obtain the matrix form of the iterative RDFT algorithm for any radix, by fixing m, and successively applying (54) to the right-hand side of itself and using the property $(I \otimes AB) = (I \otimes A)(I \otimes B)$ to obtain the stages of the iterative algorithm operating on fuller length-n vectors.

To reduce the arithmetic cost, a "split-radix" version of the algorithm was proposed in [7], [23]. It reduces the cost to $2n \log_2 n + O(n)$. The algorithm in [7] is the combination of (32) for $k = 2$ and (43). The algorithm in [23] has no exact equivalent in this paper.

Bergland's original paper explains how one could obtain a general radix algorithm, but does not show any details. We found the fully specified general radix RDFT algorithm in other sources. Namely, from the source of the FFTW software library [28] and also in [9].

FFTW uses for composite RDFT sizes (32), (33), and (43). The occurring RDFT-3 transforms are not recursively expanded, but use automatically generated small size base case routines.

[10] gives a general-radix Cooley-Tukey type algorithm using a mixture of tensor product formulas and summations. Unfortunately, some matrices in the paper appear underspecified and are hard to reconstruct. The algorithm is a combination of (32), (43), and the seventh identity in Table II.

TABLE XII
ARITHMETIC COST ACHIEVABLE FOR THE REAL DFTS WITH THE ALGORITHMS IN THIS PAPER. WE OMIT THE $O(n)$ TERM.

Transform	Algorithm	Minimum # ops (adds + mults)	Maximum # ops (adds + mults)
RDFT-t$_n$	Natural	$(\frac{3}{2}n\log_2 n, \quad n\log_2 n)$	$(\frac{3}{2}n\log_2 n, \quad n\log_2 n)$
	Natural w/ improved skew	$(\frac{4}{3}n\log_2 n, \frac{2}{3}n\log_2 n)$	$(\frac{3}{2}n\log_2 n, \quad n\log_2 n)$
	Bruun	$(\frac{3}{2}n\log_2 n, \frac{1}{2}n\log_2 n)$	$(\frac{3}{2}n\log_2 n, \frac{1}{2}n\log_2 n)$
DHT-t$_n$	Natural	$(\frac{3}{2}n\log_2 n, \quad n\log_2 n)$	$(\frac{3}{2}n\log_2 n, \quad n\log_2 n)$
	Natural w/ improved skew	$(\frac{4}{3}n\log_2 n, \frac{2}{3}n\log_2 n)$	$(\frac{3}{2}n\log_2 n, \quad n\log_2 n)$
	Bruun	$(\frac{3}{2}n\log_2 n, \frac{1}{2}n\log_2 n)$	$(\frac{3}{2}n\log_2 n, \frac{1}{2}n\log_2 n)$

The generalized RDFTs (of type 2,3,4) are presented in [9], where the authors discuss their connection to cosine and sine transforms.

Our natural general radix algorithms from Table VI seem to be known only for RDFT. The natural regularized variant (54) is novel, and eliminates the need for RDFT-3 (or RDFT-4). As explained before, the arithmetic cost of these variants depends on the radix and is between $\frac{5}{2}n\log_2 n + O(n)$ and $2n\log_2 n + O(n)$.

Among other RDFT algorithms we find the so-called "quick" RDFT algorithm in [12], which uses DCT-1 and DST-1 using a similar technique as [4].

[16] also uses the CRT as this paper, but factors $x^n - 1$ over $\mathbb{Q}$, which leads to other, less structured algorithms for non-2-power sizes. Winograd also uses the CRT to derive optimal (with respect to the number of non-rational multiplications) DFT algorithms that, however, have an increased number of additions and irregular structure [53], [54]. In the language of this paper, the DFT is first decomposed by decomposing $\mathbb{C}[x]/(x^n - 1)$ over $\mathbb{Q}$, then the resulting blocks are considered as convolutions and decomposed further using a technique different from this paper. Heideman [13] provides the exact multiplicative complexity for a real DFT of two-power size gives concrete optimal (in this sense) real DFT algorithms in [13], [14], [55].

DHT. The DHT is often regarded as a transform different from the RDFT; hence there is a different set of papers discussing its fast algorithms.

In spirit closest to our work is [15], which derives DHT algorithms by projecting DFT algorithms using the theory of field extensions similar to work on ADFTs in [56]. The technique is not applicable to the RDFT and different from ours.

All of our general radix algorithms for DHTs of four types are novel. This includes the natural algorithms in Table VI, the Bruun type algorithms in Table VII, and the regularized variants of both in Table IX. However, we did find some special cases in the literature.

A split-radix DHT algorithm is given by [22] and [8]. The former has slightly suboptimal arithmetic cost and the latter improves the cost and is the combination of (32) with $k = 2$ and (44).

Reference [27] defines all types of DHTs (called W transforms) and gives algorithms for even sizes for all 4 types. Each transform is split into a DCT/DST pair of the corresponding type similar to [4].

The algorithms in [6] are the equivalents of our free conversions between DHTs of types 1,2,3,4 given in (58) and (61) (and combinations thereof). Interestingly, the same conversions (57), (60) and (59) between generalized RDFTs appear to be novel. A similar property also holds for complex DFTs, but we don't show the equations in this paper.

In [21] and [25] we find comparisons between DHT and RDFT algorithms. In this paper we answer the question of the true difference between DHT and RDFT algorithms based on the algebraic interpretation in [35]. We show that the algorithms are based on the same principle, have precisely the same dataflow structure, and only minor differences only exist in the small size base cases.

Bruun type algorithms for DHT and RDFT. The 1978 paper [5] by Bruun introduced the so-called Bruun FFT. The author uses the connection between the DFT and a filter bank with a very special set of filters, namely for $0 \le i < n$, $\{(x^n - 1)/(x - w_{i/n})\}$, $x = z^{-1}$. The special structure of the zeros of the filters is used to build the so-called *filter tree*, which computes the DFT. The filter tree is built by successively factoring the polynomial $x^n - 1$. One factorization leads to the familiar radix-2 Cooley-Tukey algorithm as also shown in [30], whereas another factorization (over the real numbers) leads to a new FFT, which exclusively uses multiplications by real constants, except for the final stage. Therefore, omitting the last stage yields an RDFT algorithm. The paper shows a decimation-in-time radix-2 algorithm only. The algorithm is the special case of (55) and (41) with $k = 2$ applied to the RDFT.

The same idea is explored further in [3]. The authors derive the decimation-in-frequency algorithm, but for the DHT. The algorithm is the special case of the *transpose* of (55) and (41) with $k = 2$ applied to the DHT. This is the only reference we found discussing a Bruun type algorithm for DHT.

Later, Murakami explained Bruun's algorithm in a more rigorous framework using the CRT and generalized the algorithm to arbitrary radix [18]–[20]. Murakami derives the algorithm for an arbitrary *even* size. His algorithm is (55) applied to RDFT or BRDFT combined with (41).

In this paper we showed that Cooley-Tukey type and Bruun type algorithms are actually instances of the same algorithm (e.g., Table IV) but with different choices of the intermediate basis c.

C. Inverse Transforms and Convolutions

Inverse transforms. So far in this paper we showed the algorithms, given as matrix factorizations, for the forward transforms. Algorithms for the inverse transforms are readily obtained in two different ways: by formally inverting the matrix factorizations, or by formally transposing the matrix formulas and using the orthogonality properties of the transforms. Both methods use the following inversion and transposition identities that can be recursively applied to any algorithm in this paper:

$$\begin{aligned}
(A \cdot B)^{-1} &= B^{-1} \cdot A^{-1}, & (A \cdot B)^{\top} &= B^{\top} \cdot A^{\top},\\
(A \otimes B)^{-1} &= (A^{-1} \otimes B^{-1}), & (A \otimes B)^{\top} &= (A^{\top} \otimes B^{\top}),\\
(I \otimes_i A_i)^{-1} &= (I \otimes_i A_i^{-1}), & (I \otimes_i A_i)^{\top} &= (I \otimes_i A_i^{\top}),\\
(A \oplus B)^{-1} &= (A^{-1} \oplus B^{-1}), & (A \oplus B)^{\top} &= (A^{\top} \oplus B^{\top}),\\
(J_n)^{-1} &= J_n^{\top} = J_n,\\
(L^{km}_m)^{-1} &= (L^{km}_m)^{\top} = L^{km}_k,\\
(R_a)^{-1} &= R_a^{\top} = R_{-a}.
\end{aligned}$$

Inversion of an algorithm is straightforward using this method. Transposition requires the use of the following explicit formulas for the inverse transforms (shown for even n):

$$\mathrm{RDFT}_n^{-1} = 1/n \cdot \mathrm{RDFT}_n^{\top}(I_2 \oplus 2I_{n-2}) \quad (63)$$

$$\text{RDFT-2}_n^{-1} = 1/n \cdot \text{RDFT-2}_n^{\top}(I_2 \oplus 2I_{n-2}) \quad (64)$$

$$\text{RDFT-3}_n^{-1} = 1/n \cdot \text{RDFT-3}_n^{\top}(2I_n) \quad (65)$$

$$\text{RDFT-4}_n^{-1} = 1/n \cdot \text{RDFT-4}_n^{\top}(2I_n) \quad (66)$$

$$\mathrm{DHT}_n^{-1} = 1/n \cdot \mathrm{DHT}_n^{\top} = 1/n \cdot U_n\,\mathrm{DHT}_n\,U_n, \quad (67)$$

$$\text{DHT-2}_n^{-1} = 1/n \cdot \text{DHT-2}_n^{\top} = 1/n \cdot V_n\,\text{DHT-3}_n\,U_n, \quad (68)$$

$$\text{DHT-3}_n^{-1} = 1/n \cdot \text{DHT-3}_n^{\top} = 1/n \cdot U_n\,\text{DHT-2}_n\,V_n, \quad (69)$$

$$\text{DHT-4}_n^{-1} = 1/n \cdot \text{DHT-4}_n^{\top} = 1/n \cdot V_n\,\text{DHT-4}_n\,V_n. \quad (70)$$

Note that often the scaling factor $1/n$ is omitted in actual implementations. The permutations U and V appear in DHT inverses due to our use of non-standard ordering, as explained in Section II and (13) and (14).

Since natural algorithms contain orthogonal building blocks only, inverting or transposing the algorithm will lead to almost the same result. However, Bruun type algorithms have non-orthogonal building blocks. Therefore, we can obtain *two* different inverse transform algorithm variants, first by inverting and second by transposing. Similarly, we can obtain another variant of the forward transform, by inverting and then transposing, as was already done in [36] to obtain new variants of the DCT/DST algorithms.

We also note that up to a permutation, DHT and DHT-4 are self-inverse, and the DHT-2/DHT-3 pair are mutual inverses. This means that inverting or transposing a DHT algorithm leads to a new DHT algorithm variant, and for the Bruun type algorithms inverting and then transposing leads to a third variant.

Linear and circular convolution The circular convolution of real vectors, which corresponds to multiplication in $\mathbb{R}[x]/(x^n - 1)$, can be computed using any of the $\mathcal{F}_n(t \to f)$ (and their inverse) transforms. This includes RDFT, RDFT-2, DHT, DHT-2, BRDFT, and BRDFT-2.

Skew circular convolution, which corresponds to multiplication in $\mathbb{R}[x]/(x^n + 1)$, can be computed using any of the $\mathcal{G}_n(f \to f)$ (and their inverse) transforms. This includes RDFT-3, RDFT-4, DHT-3, DHT-4, and BRDFT-3.

Computing linear convolution is usually done by partitioning it into blocks and embedding each block in a larger circular or skew circular convolution. Thus, linear convolution can be computed using any of the transforms from Table I(a). Apart from the common choice of type-1 transforms, using type-3 transforms is also a good choice, since they possess a more regular structure as can be seen from (29). Another interesting case is reported in [26]. If RDFT-3 is used together with a conversion to complex DFT using (43) we obtain a *real* convolution algorithm, which uses a *complex* DFT of half the size. With BRDFT or BRDFT-3 one can use the cheaper Bruun butterflies $\mathcal{S}_{4,*}(s \to s)$ (shown in Table VII). The use of BRDFT for computing the linear convolution was proposed in [19], [20].

VII. Conclusion

This paper, together with [36], shows that using the framework of polynomial algebras general-radix algorithms for all 1-D trigonometric transforms can be obtained using only one generic method. Since this method also yields the Cooley-Tukey FFT, we refer to all these algorithms as "Cooley-Tukey type." We believe that these papers are a big step towards consolidating the area of transform algorithms. Namely, the polynomial algebra framework enables the concise derivation, classification, and structural representation of the obtained algorithms. Further, it also explains the many relationships that hold between transforms. However, some work is still left to algebraically capture the full class of existing algorithms for trigonometric transforms.

The appearance of polynomial algebras in signal processing is not coincidental: in the algebraic signal processing theory (ASP) started in [35] we explain that they provide the structure for finite, shift-invariant signal processing and explain the exact forms they take for the commonly used 1-D transforms. Hence, in ASP the theory of transform algorithms (as shown in this paper) becomes a natural part of signal processing theory itself.

Appendix

Transform matrices. Table XIII shows the exact forms of the DFTs and real DFTs as used in this paper. We remind the reader that there is a degree of freedom in choosing the order of the output as explained in Section II-B.

Computing real DFT via half-size complex DFT. Two RDFTs of size n can be computed with a single complex DFT of size n, this property can be further used to compute one RDFT of size $2n$ using one complex DFT of size n [33].

We now derive these two properties using our matrix framework. First, from Table II,

$$\begin{aligned}
\mathrm{RDFT}_n &= \left(I_2 \oplus \left(I_{n/2-1} \otimes \tfrac{1}{2}\begin{bmatrix} 1 & 1 \\ -j & j \end{bmatrix}\right)\right) U_n^{-1}\,\mathrm{DFT}_n\\
&= Y_n\,\mathrm{DFT}_n\,.
\end{aligned}$$

TABLE XIII
COMPLEX AND REAL DFTS.

In the table 1_n, $1'_n$ are $1 \times n$ matrices, $1''_n$ is a $2 \times n$ matrix, and f is a quotient of sines, as defined below. All transform matrices are $n \times n$, and $0 \le \ell < n$. For the complex DFTs, $0 \le k < n$, for the real DFTs (in each case the two rows are for even and odd n, respectively), $0 \le k < \lfloor (n-1)/2 \rfloor$.

$$1_n = \begin{bmatrix} 1 & 1 & \dots & 1 & 1 \end{bmatrix}, \qquad 1'_n = \begin{bmatrix} 1 & -1 & \dots & 1 & -1 \end{bmatrix}, \qquad 1''_n = \begin{bmatrix} 1_n^\top & 1'^\top_n \end{bmatrix}^\top, \qquad f_{a,b} = s_{ab}/s_a.$$

$\mathrm{DFT}_n = \left[w_{k\ell/n}\right]$	$\mathrm{DFT\text{-}2}_n = \left[w_{k(\ell+\frac{1}{2})/n}\right]$	$\mathrm{DFT\text{-}3}_n = \left[w_{(k+\frac{1}{2})\ell/n}\right]$	$\mathrm{DFT\text{-}4}_n = \left[w_{(k+\frac{1}{2})(\ell+\frac{1}{2})/n}\right]$
$\mathrm{RDFT}_n = \begin{bmatrix} 1''_n \\ \begin{bmatrix} c_{k\ell/n} \\ -s_{k\ell/n} \end{bmatrix} \end{bmatrix}$	$\mathrm{RDFT\text{-}2}_n = \begin{bmatrix} 1''_n \\ \begin{bmatrix} c_{k(\ell+\frac{1}{2})/n} \\ -s_{k(\ell+\frac{1}{2})/n} \end{bmatrix} \end{bmatrix}$	$\mathrm{RDFT\text{-}3}_n = \begin{bmatrix} c_{(k+\frac{1}{2})\ell/n} \\ -s_{(k+\frac{1}{2})\ell/n} \end{bmatrix}$	$\mathrm{RDFT\text{-}4}_n = \begin{bmatrix} c_{(k+\frac{1}{2})(\ell+\frac{1}{2})/n} \\ -s_{(k+\frac{1}{2})(\ell+\frac{1}{2})/n} \end{bmatrix}$
$\mathrm{RDFT}_n = \begin{bmatrix} 1_n \\ \begin{bmatrix} c_{k\ell/n} \\ -s_{k\ell/n} \end{bmatrix} \end{bmatrix}$	$\mathrm{RDFT\text{-}2}_n = \begin{bmatrix} 1_n \\ \begin{bmatrix} c_{k(\ell+\frac{1}{2})/n} \\ -s_{k(\ell+\frac{1}{2})/n} \end{bmatrix} \end{bmatrix}$	$\mathrm{RDFT\text{-}3}_n = \begin{bmatrix} \begin{bmatrix} c_{(k+\frac{1}{2})\ell/n} \\ -s_{(k+\frac{1}{2})\ell/n} \end{bmatrix} \\ 1'_n \end{bmatrix}$	$\mathrm{RDFT\text{-}4}_n = \begin{bmatrix} \begin{bmatrix} c_{(k+\frac{1}{2})(\ell+\frac{1}{2})/n} \\ -s_{(k+\frac{1}{2})(\ell+\frac{1}{2})/n} \end{bmatrix} \\ 1'_n \end{bmatrix}$
$\mathrm{DHT}_n = \begin{bmatrix} 1''_n \\ \begin{bmatrix} cas_{k\ell/n} \\ cms_{k\ell/n} \end{bmatrix} \end{bmatrix}$	$\mathrm{DHT\text{-}2}_n = \begin{bmatrix} 1''_n \\ \begin{bmatrix} cas_{k(\ell+\frac{1}{2})/n} \\ -cms_{k(\ell+\frac{1}{2})/n} \end{bmatrix} \end{bmatrix}$	$\mathrm{DHT\text{-}3}_n = \begin{bmatrix} cas_{(k+\frac{1}{2})\ell/n} \\ cms_{(k+\frac{1}{2})\ell/n} \end{bmatrix}$	$\mathrm{DHT\text{-}4}_n = \begin{bmatrix} cas_{(k+\frac{1}{2})(\ell+\frac{1}{2})/n} \\ -cms_{(k+\frac{1}{2})(\ell+\frac{1}{2})/n} \end{bmatrix}$
$\mathrm{DHT}_n = \begin{bmatrix} 1_n \\ \begin{bmatrix} cas_{k\ell/n} \\ cms_{k\ell/n} \end{bmatrix} \end{bmatrix}$	$\mathrm{DHT\text{-}2}_n = \begin{bmatrix} 1_n \\ \begin{bmatrix} cas_{k(\ell+\frac{1}{2})/n} \\ -cms_{k(\ell+\frac{1}{2})/n} \end{bmatrix} \end{bmatrix}$	$\mathrm{DHT\text{-}3}_n = \begin{bmatrix} \begin{bmatrix} cas_{(k+\frac{1}{2})\ell/n} \\ cms_{(k+\frac{1}{2})\ell/n} \end{bmatrix} \\ 1'_n \end{bmatrix}$	$\mathrm{DHT\text{-}4}_n = \begin{bmatrix} \begin{bmatrix} cas_{(k+\frac{1}{2})(\ell+\frac{1}{2})/n} \\ -cms_{(k+\frac{1}{2})(\ell+\frac{1}{2})/n} \end{bmatrix} \\ 1'_n \end{bmatrix}$
$\mathrm{BRDFT}_n = \begin{bmatrix} 1''_n \\ \begin{bmatrix} -f_{k/n,\ell} \\ f_{k/n,\ell+1} \end{bmatrix} \end{bmatrix}$	$\mathrm{BRDFT\text{-}2}_n = \begin{bmatrix} 1''_n \\ \begin{bmatrix} -f_{k/n,\ell-\frac{1}{2}} \\ f_{k/n,\ell+\frac{1}{2}} \end{bmatrix} \end{bmatrix}$	$\mathrm{BRDFT\text{-}3}_n = \begin{bmatrix} -f_{(k+\frac{1}{2})/n,\ell} \\ f_{(k+\frac{1}{2})/n,\ell+1} \end{bmatrix}$	$\mathrm{BRDFT\text{-}4}_n = \begin{bmatrix} -f_{(k+\frac{1}{2})/n,\ell-\frac{1}{2}} \\ f_{(k+\frac{1}{2})/n,\ell+\frac{1}{2}} \end{bmatrix}$
$\mathrm{BRDFT}_n = \begin{bmatrix} 1_n \\ \begin{bmatrix} -f_{k/n,\ell} \\ f_{k/n,\ell+1} \end{bmatrix} \end{bmatrix}$	$\mathrm{BRDFT\text{-}2}_n = \begin{bmatrix} 1_n \\ \begin{bmatrix} -f_{k/n,\ell-\frac{1}{2}} \\ f_{k/n,\ell+\frac{1}{2}} \end{bmatrix} \end{bmatrix}$	$\mathrm{BRDFT\text{-}3}_n = \begin{bmatrix} \begin{bmatrix} -f_{(k+\frac{1}{2})/n,\ell} \\ f_{(k+\frac{1}{2})/n,\ell+1} \end{bmatrix} \\ 1'_n \end{bmatrix}$	$\mathrm{BRDFT\text{-}4}_n = \begin{bmatrix} \begin{bmatrix} -f_{(k+\frac{1}{2})/n,\ell-\frac{1}{2}} \\ f_{(k+\frac{1}{2})/n,\ell+\frac{1}{2}} \end{bmatrix} \\ 1'_n \end{bmatrix}$

The above is an equation that involves complex matrices, to obtain a purely real equation we apply $\overline{(\cdot)}$ (defined in Section V-A) to both sides, and get

$$\mathrm{RDFT}_n \otimes I_2 = \overline{Y_n} \cdot \overline{\mathrm{DFT}_n}, \tag{71}$$
$$\overline{Y_n} = \Big(I_4 \oplus \big(I_{n/2-1} \otimes \tfrac{1}{2} \begin{bmatrix} I_2 & I_2 \\ D_2 J_2 & -D_2 J_2 \end{bmatrix} \big) \Big) (U_n^{-1} \otimes I_2).$$

Note, that above we used the identity $\overline{A} = A \otimes I_2$ for a real matrix A. (71) expresses the fact that two RDFTs can be computed using one complex DFT (on the single interleaved sequence) and a post-processing step in the form of the matrix $\overline{Y_n}$. To obtain an algorithm for a single RDFT, we combine (71) and (32) with $k = n$ and $m = 2$ to get

$$\mathrm{RDFT}_{2n} = P_2^{2n} (\mathrm{RDFT}_2 \oplus \mathrm{RDFT\text{-}3}_2 \oplus \big(I_{n/2-1} \otimes_i \mathcal{S}_{4,(i+1)/n}(e \to e) \big)) \overline{Y_n \mathrm{DFT}_n}$$

and the final form after inserting the base cases from Table VI:

$$\mathrm{RDFT}_{2n} = P_2^{2n} \Big(F_2 \oplus D_2 \oplus \big(I_{n/2-1} \otimes_i (F_2 \otimes I_2)(I_2 \oplus R_{-(i+1)/2n}) L_2^4 \big) \Big) \cdot \overline{Y_n} \cdot \overline{\mathrm{DFT}_n}. \tag{72}$$

REFERENCES

[1] G. D. Bergland, "Numerical analysis: A fast Fourier transform algorithm for real-valued series," *Commun. ACM*, vol. 11, no. 10, pp. 703–710, 1968.

[2] ——, "A radix-eight fast Fourier transform subroutine for real-valued series," *IEEE Trans. on Audio and Electroacoustics*, vol. 17, no. 2, pp. 138–144, 1969.

[3] D. A. Bini and E. Bozzo, "Fast discrete transform by means of eigenpolynomials," *Computers & Mathematics (with Applications)*, vol. 26, no. 9, pp. 35–52, 1993.

[4] V. Britanak and K. R. Rao, "The fast generalized discrete Fourier transforms: A unified approach to the discrete sinusoidal transforms computation," *Signal Processing*, vol. 79, no. 2, pp. 135–150, 1999.

[5] G. Bruun, "z-transform DFT filters and FFTs," *IEEE Trans. ASSP*, vol. 26, no. 1, pp. 56–63, 1978.

[6] S. C. Chan and K. L. Ho, "Fast algorithms for computing the discrete W transforms," in *Proc. IEEE Region 10 Conference on Computer and Communication Systems (TENCON 90)*, vol. 1, 1990, pp. 183–185.

[7] P. Duhamel, "Implementation of "split-radix" FFT algorithms for complex, real, and real-symmetric data," *IEEE Trans. ASSP*, vol. 34, no. 2, pp. 285–295, 1986.

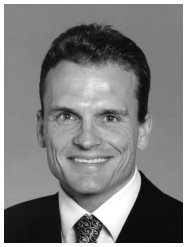
[8] P. Duhamel and M. Vetterli, "Improved Fourier and Hartley transform algorithms: Application to cyclic convolution of real data," *IEEE Trans. ASSP*, vol. 35, no. 6, pp. 818–824, 1987.

[9] O. K. Ersoy and N.-C. Hu, "A unified approach to the fast computation of all discrete trigonometric transforms," in *Proc. ICASSP*, vol. 12, 1987, pp. 1843–1846.

[10] N.-C. Hu and O. K. Ersoy, "Fast computation of real discrete Fourier transform for any number of data points," *IEEE Trans. on Circuits and Systems*, vol. 38, no. 11, pp. 1280–1292, 1991.

[11] S. G. Johnson and M. Frigo, "A modified split-radix FFT with fewer arithmetic operations," *IEEE Trans. Signal Processing*, vol. 55, no. 1, pp. 111–119, 2007.
[12] H. Guo, G. A. Sitton, and C. S. Burrus, "The quick Fourier transform: An FFT based on symmetries," *IEEE Trans. on Signal Processing*, vol. 46, no. 2, pp. 335–341, 1998.
[13] M. T. Heideman and C. S. Burrus, "On the number of multiplications necessary to compute a length-2^n DFT," *IEEE Trans. ASSP*, vol. 34, no. 1, pp. 91–95, 1986.
[14] M. T. Heideman, *Multiplicative complexity, convolution, and the DFT* . Springer-Verlag, 1988.
[15] J. Hong, M. Vetterli, and P. Duhamel, "Basefield transforms with the convolution property," *Proceedings of the IEEE*, vol. 82, no. 3, pp. 400–412, 1994.
[16] J.-B. Martens, "Discrete Fourier transform algorithms for real valued sequences," *IEEE Trans. ASSP*, vol. 32, no. 2, pp. 390–396, 1984.
[17] S. Mittal, M. Z. A. Khan, and M. B. Srinivas, "A comparative study of different FFT architectures for software defined radio," in *SAMOS*, 2007, pp. 375–384.
[18] H. Murakami, "Prime-length real-valued polynomial residue division algorithms," *IEEE Trans. on Signal Processing*, vol. 50, no. 11, pp. 2777–2788, 2002.
[19] ——, "Real-valued decimation-in-time and decimation-in-frequency algorithms," *IEEE Trans. Circuits and Systems II: Analog and Digital Signal Processing*, vol. 41, no. 12, pp. 808–816, 1994.
[20] ——, "Real-valued fast discrete Fourier transform and cyclic convolution algorithms of highly composite even length," in *Proc. ICASSP*, vol. 3, 1996, pp. 1311–1314.
[21] M. Popović and D. Šević, "A new look at the comparison of the fast Hartley and Fourier transforms," *IEEE Trans. on Signal Processing*, vol. 42, no. 8, pp. 2178–2182, 1994.
[22] H. V. Sorensen, D. L. Jones, C. S. Burrus, and M. T. Heideman, "On computing the discrete Hartley transform," *IEEE Trans. ASSP*, vol. 33, no. 4, pp. 1231–1238, 1985.
[23] H. V. Sorensen, D. L. Jones, M. T. Heideman, and C. S. Burrus, "Real-valued fast Fourier transform algorithms," *IEEE Trans. ASSP*, vol. 35, no. 6, pp. 849–863, 1987.
[24] R. Storn, "Some results in fixed point error analysis of the Bruun-FTT algorithm," *IEEE Trans. on Signal Processing*, vol. 41, no. 7, pp. 2371–2375, 1993.
[25] P. R. Uniyal, "Transforming real-valued sequences: fast Fourier versus fast Hartley transform algorithms," *IEEE Trans. on Signal Processing*, vol. 42, no. 11, pp. 3249–3254, 1994.
[26] J. L. Vernet, "Real signals fast Fourier transform: Storage capacity and step number reduction by means of an odd discrete Fourier transform," *Proceedings of the IEEE*, vol. 59, no. 10, pp. 1531–1532, 1971.
[27] Z. Wang, "Fast algorithms for the discrete W transform and for the discrete Fourier transform," *IEEE Trans. ASSP*, vol. 32, no. 4, pp. 803–816, 1984.
[28] M. Frigo and S. G. Johnson, "The design and implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005, special issue on "Program Generation, Optimization.
[29] M. Püschel, J. M. F. Moura, J. Johnson, D. Padua, M. Veloso, B. W. Singer, J. Xiong, F. Franchetti, A. Gačić, Y. Voronenko, K. Chen, R. W. Johnson, and N. Rizzolo, "SPIRAL: Code generation for DSP transforms," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 232–275, 2005, special issue on "Program Generation, Optimization.
[30] H. J. Nussbaumer, *Fast Fourier Transformation and Convolution Algorithms*, 2nd ed. Springer, 1982.
[31] C. Van Loan, *Computational Framework of the Fast Fourier Transform*. SIAM, 1992.
[32] R. Tolimieri, M. An, and C. Lu, *Algorithms for Discrete Fourier Transforms and Convolution*, 2nd ed. Springer, 1997.
[33] W. H. Press, B. P. Flannery, T. S. A., and V. W. T., *Numerical Recipes in C: The Art of Scientific Computing*, 2nd ed. Cambridge University Press, 1992.
[34] Y. Voronenko and M. Püschel, "Algebraic derivation of general radix Cooley-Tukey algorithms for the real discrete Fourier transform," in *Proc. ICASSP*, vol. 3, 2006, pp. 876–879.
[35] M. Püschel and J. M. F. Moura, "Algebraic signal processing theory: Foundation and 1-D time," *IEEE Trans. on Signal Processing*, vol. 56, no. 8, pp. 3572–3585, 2008.
[36] ——, "Algebraic signal processing theory: Cooley-Tukey type algorithms for DCTs and DSTs," *IEEE Trans. on Signal Processing*, vol. 56, no. 4, pp. 1502–1521, 2008.
[37] ——, "Algebraic signal processing theory: 1-D space," *IEEE Trans. on Signal Processing*, vol. 56, no. 8, pp. 3586–3599, 2008.
[38] R. N. Bracewell, "The fast Hartley transform," *Proceedings of the IEEE*, vol. 72, 1984.
[39] I. J. Good, "The interaction algorithm and practical Fourier analysis," *Journal Royal Statist. Soc.*, vol. B20, pp. 361–375, 1958.
[40] M. Heideman, C. Burrus, and H. Johnson, "Prime factor FFT algorithms for real-valued series," in *Proc. ICASSP*, vol. 9, no. 1, 1984, pp. 492–495.
[41] C. M. Rader, "Discrete Fourier transforms when the number of data samples is prime," *Proceedings of the IEEE*, vol. 56, pp. 1107–1108, 1968.
[42] S. Chu and C. S. Burrus, "A prime factor FTT algorithm using distributed arithmetic," *IEEE Trans. ASSP*, vol. 30, no. 2, pp. 217–227, 1982.
[43] P. A. Fuhrman, *A Polynomial Approach to Linear Algebra*. New York: Springer Verlag, 1996.
[44] G. Bongiovanni, P. Corsini, and G. Frosini, "One-dimensional and two-dimensional generalized discrete Fourier transform," *IEEE Trans. ASSP*, vol. 24, no. 2, pp. 97–99, 1976.
[45] G. Bonnerot and M. Bellanger, "Odd-time odd-frequency discrete fourier transform for symmetric real-valued series," *Proceedings of the IEEE*, vol. 64, pp. 392–393, 1976.
[46] L. Auslander, E. Feig, and S. Winograd, "Abelian semi-simple algebras and algorithms for the discrete Fourier transform," *Advances in Applied Mathematics*, vol. 5, pp. 31–55, 1984.
[47] M. Püschel and J. M. F. Moura, "Algebraic signal processing theory," [Online]. Available: http://arxiv.org/abs/cs.IT/0612077.
[48] R. N. Bracewell, "Discrete Hartley transform," *J. Optical Society America*, vol. 73, no. 12, pp. 1832–1835, 1983.
[49] "Intel Integrated Performance Primitives 5.3." [Online]. Available: http://intel.com/software/products/ipp
[50] "FFTW 3.1.2." [Online]. Available: http://fftw.org
[51] M. Püschel, "Cooley-Tukey FFT like algorithms for the DCT," in *Proc. ICASSP*, vol. 2, 2003, pp. 501–504.
[52] Y. Voronenko, "Library generation for linear transforms," Ph.D. dissertation, Electrical and Computer Engineering, Carnegie Mellon University, 2008.
[53] S. Winograd, "On computing the discrete Fourier transform," *Mathematics of Computation*, vol. 32, pp. 175–199, 1978.
[54] ——, "On the multiplicative complexity of the discrete Fourier transform," *Advances in Mathematics*, vol. 32, pp. 83–117, 1979.
[55] P. Duhamel, "Algorithms meeting the lower bounds on the multiplicative complexity of length-2n DFTs and their connection with practical algorithms," *IEEE Trans. ASSP*, vol. 38, no. 9, pp. 1504–1511, 1990.
[56] T. Beth, *Verfahren der Schnellen Fouriertransformation [Fast Fourier Transform Methods]*. Teubner, 1984.

Yevgen Voronenko (S'03–M'08) is a Project Scientist at the Electrical and Computer Engineering Department at Carnegie Mellon University (CMU). He received a B.S. degree in Computer Science from Drexel University in 2003, and his Ph.D. from CMU in 2008, where he was awarded the A.G. Milnes outstanding Ph.D. dissertation award. His research interests include scientific computing, software engineering, programming languages, and compiler design.



Markus Püschel (M'99–SM'05) is an Associate Research Professor of Electrical and Computer Engineering at Carnegie Mellon University (CMU). He received his Diploma (M.Sc.) in Mathematics and his Doctorate (Ph.D.) in Computer Science, in 1995 and 1998, respectively, both from the University of Karlsruhe, Germany. From 1998-1999 he was a Postdoctoral Researcher at Mathematics and Computer Science, Drexel University, Philadelphia. Since 2000 he has been with CMU. He is an Associate Editor for the *IEEE Transactions on Signal Processing*, and was an Associate Editor for the *IEEE Signal Processing Letters*, a Guest Editor of the *Journal of Symbolic Computation*, and the *Proceedings of the IEEE*. He holds the title of Privatdozent of Applied Informatics at the Department of Computer Science, University of Technology, Vienna, Austria and was awarded (with J. Moura) the CMU College of Engineering Outstanding Research Award. His research interests include signal processing theory/software/hardware, scientific computing, compilers, applied mathematics and algebra.