Foundations of Computer Algebra: Fast Polynomial Multiplication

Marc Moreno Maza
University of Western Ontario
CS 424 / CS 556 – Winter 2006

January 7, 2008

1 Fast Polynomial Multiplication based on the Discrete Fourier Transform

In the whole section we consider a commutative ring R with units. Let us consider two univariate polynomials $f, g \in R[x]$ of degree less than n. Then their product has degree less than $2n-1$. Assume that we are given a finite subset P of R with $2\,n-1$ elements such that $f(r)$ and $g(r)$ are known for every $r \in R$. Then the values $f(r)\,g(r)$ define the polynomial $f\,g$ completely. Indeed, by Lagrange interpolation we construct the polynomial $f\,g$ (which has at most $2n-1$ coefficients) from the $2\,n-1$ values $f(r)\,g(r)$. Observe that the cost of defining $f\,g$ by the values $f(r)\,g(r)$ is $2\,n-1$, which is cheap!

However if we need information about $f\,g$ such as its leading coefficient or its number of terms we need to compute the coefficients of $f\,g$. The cost of this construction is in $\mathcal{O}(n^2)$.

The underlying idea of the fast polynomial multiplication based on the discrete Fourier transform is the following. Assume that there exists a subset P of R with $2\,n-1$ elements such that

- evaluating f and g at every $r \in P$ can be done at *nearly linear time cost* (such as $\mathcal{O}(n\log(n))$),
- interpolating $f(r)\,g(r)$ for $r \in P$ can be done at *nearly linear time cost*.

Then the multiplication of f by g, represented by their coefficients, can be done in $\mathcal{O}(n\log(n))$.

Such sets P do not always exist. This depends on the ring R and the degrees of the polynomials f and g. However many finite fields possess such sets for small enough degrees of f and g. For the arbitrary ring R, it can be shown that the multiplication of f nad g, represented by their coefficients, can be done in $\mathcal{O}(n\log(n)\log(\log(n)))$.

1.1 Primitive roots of unity

Let us recall that an element $a \in R$ is a zero divisor if $a \neq 0$ and there exists $b \in R$ such that $a\,b = 0$ and $b \neq 0$.

Let us recall also that because the ring R has a unit 1 then there is a map

$$\begin{array}{ccc} \mathbb{Z} & \longmapsto & R \\ n & \longmapsto & n\ 1_R \end{array} \tag{1}$$

which allows to talk of an integer as an element of R. However a nonzero integer n may become 0_R. For instance every multiple of p becomes 0 in $\mathbb{Z}/p\mathbb{Z}$.

Observe also the ring R may contain nonzero elements that are neither zero divisor, nor units. For instance in the ring of square matrices of order 2 with integer coefficients the matrix $\begin{pmatrix} 1 & 2 \\ 0 & 3 \end{pmatrix}$ is nonsingular but has no inverse (right or left). But this is not a good example since we limit ourselves to commutative rings. So consider instead the following example.

Example 1 Let $R = \mathbb{Z}/6\mathbb{Z}$ and let $U = R[x]$ be the ring of univariate polynomials over R. The ring U has units (for instance the constant polynomial 5), it has zero divisors (for instance the polynomial $2x+4$ since $3(2x+4) = 0$) and also elements like $x + 1$ which is not a unit nor a zero divisor.

Why do we need to take such exotic examples? Because among the rings you are familiar with, this is the simplest example of ring having zero divisors, units and nonzero elements that are not zero divisors or units. Indeed our usual rings are $\mathbf{k}[x]$ where $\mathbf{k}$ is a field and the residue class ring $\mathbb{Z}/m\mathbb{Z}$ (with $m > 1$ not necessarily prime). The ring $\mathbf{k}[x]$ has no zero divisors whereas $\mathbb{Z}/m\mathbb{Z}$ has zero divisors iff m is not prime. But for every non prime m the ring $\mathbb{Z}/m\mathbb{Z}$ consists only of zero, units and zero divisors. This follows from the proposition below. This is why we were led to the $\mathbb{Z}/6\mathbb{Z}[x]$ example.

Proposition 1 *Let E be an Euclidean domain and m an element of E with $m \neq 0$ and $m \neq 1$. Consider the residue class ring $R = E/\langle m \rangle$ where $\langle m \rangle$ denotes the ideal generated by m. Then every element of R is either zero, a zero divisor or a unit.*

PROOF $\triangleright$ Studying the elements of R (for deciding if they are null, zero divisors or units) can be done by studying the $a \mod m$ for every $a \in E$. So let a be in E. If a is a multiple of m then $a \equiv 0 \mod m$. Assume from now on that a is not a a multiple of m. Consider g the gcd of a and m together with the Bezout relation

$$u\,a + v\,m \;=\; g \tag{2}$$

Observe that $u \mod m$ cannot be zero (otherwise $g \mod m$ would be zero and m would divide g and thus a). If $g = 1$ then $u\,a \equiv 1 \mod m$ and this shows $a \mod m$ is a unit of R. The same result holds if g is a unit. Assume from now on that g is not a unit. Since g divides a and m let a' and m' be such that

$$a \;=\; g\,a' \quad \text{and} \quad m \;=\; g\,m' \tag{3}$$

Observe that $m' \mod m$ is not zero (otheriwse m' would write $m' = m''m$ leading to the contradiction $g\,m'' = 1$). We have

$$m'\,a \;=\; m'\,g\,a' \;=\; m\,a' \equiv 0 \mod m \tag{4}$$

This shows that $a \mod m$ is a zero divisor of R. Yes, this was that simple! $\triangleleft$

Definition 1 *Let n be a positive integer and $\omega \in R$.*

1. *ω is a* n-th root of unity *if $\omega^n = 1$.*
2. *ω is a* primitive n-th root of unity *if*
   (a) *$\omega^n = 1$,*
   (b) *n is a unit in R,*
   (c) *for every prime divisor t of n the element $\omega^{n/t} - 1$ is neither zero nor a zero divisor.*

Remark 1 When R is an integral domain the last condition of Definition 1 becomes: for every prime divisor t of n the element $\omega^{n/t} \neq 1$. Observe also that a n-th root of unity is necessarily a unit.

Example 2 Consider that R is the field $\mathbb{C}$ of complex numbers. The number $\omega = e^{2\,\imath\pi/8}$ is a a primitive 8-th root of unity.

Example 3 In $R = \mathbb{Z}/8\mathbb{Z}$ we have $3^2 \equiv 1$. However 3 is not a primitive 2-th of unity, since $n = 2$ is not a unit in R.

Example 4 In $R = \mathbb{Z}/17\mathbb{Z}$ we have the following computation in AXIOM

```
(1) -> R := PF(17)

   (1)  PrimeField 17
                                                        Type: Domain
(2) -> w: R := 3

   (2)  3
                                                   Type: PrimeField 17
(3) -> [w^i for i in 0..16]

   (3)  [1,3,9,10,13,5,15,11,16,14,8,7,4,12,2,6,1]
                                              Type: List PrimeField 17
(4) -> u: R := 2

   (4)  2
                                                   Type: PrimeField 17
(5) -> [u^i for i in 0..16]

   (5)  [1,2,4,8,16,15,13,9,1,2,4,8,16,15,13,9,1]
                                              Type: List PrimeField 17
```

The first list shows that 3 is a primitive 16-th root of unity. However with $\omega = 2$ we have $\omega^8 - 1 = 0$ since $(2^4 - 1)(2^4 + 1) = 15 \times 17 \equiv 0$.

Lemma 1 *Let $1 < \ell < n$ be integers and let ω be a primitive n-th root of unity. Then we have*

1. *$\omega^\ell - 1$ is neither zero nor a zero divisor in R,*
2. *$\Sigma_{0 \le j < n}\, \omega^{j\,\ell} = 0$.*

PROOF $\triangleright$ It relies on the formula

$$(c-1)\ \Sigma_{0\leq j<m}\ c^j \quad = \quad c^m - 1 \tag{5}$$

which holds for every $c \in R$ and every positive integer m.

Let us prove the first statement of the lemma. Let g be the gcd of ℓ and n. Let $u, v \in \mathbb{Z}$ be such that

$$u\,\ell + v\,n \ = \ g \tag{6}$$

Since $\ell < n$ we have $1 \leq g < n$. Hence we can cancel a prime factor t of n such that

$$g \ \mid \ (n/t) \tag{7}$$

Let

$$c \ = \ \omega^g \ \text{ and } \ m = n/(tg) \tag{8}$$

in Relation (5) leading to

$$(\omega^g - 1)\ a \quad = \quad (\omega^{n/t} - 1) \tag{9}$$

for some $a \in R$. Hence if $(\omega^g - 1)$ would be zero or a zero divisor then so would be $(\omega^{n/t} - 1)$ which is false. Now applying Relation (5) with $c = \omega^\ell$ and $m = u$ implies that $(\omega^\ell - 1)$ divides $(\omega^{u\,\ell} - 1)$. But with Relation (6) we obtain

$$(\omega^{u\,\ell} - 1) \ = \ (\omega^{u\,\ell}\,\omega^{v\,n} - 1) \ = \ (\omega^g - 1) \tag{10}$$

Hence

$$(\omega^\ell - 1) \ \mid \ (\omega^g - 1) \tag{11}$$

Therefore $(\omega^\ell - 1)$ cannot be zero or a zero divisor.

Now let us prove the second statement of the lemma. By applying Relation (5) with $c = \omega^\ell$ and $m = n$ we have

$$(\omega^\ell - 1)\ \Sigma_{0\leq j<n}\ \omega^{\ell\,j} \quad = \quad \omega^{\ell\,n} - 1 \quad = 0 \tag{12}$$

Since $(\omega^\ell - 1)$ is neither zero nor a zero divisor we otain the desired formula. $\triangleleft$

1.2 The Discrete Fourier Transform

Let n be a positive integer and $\omega \in R$ be a primitive n-th root of unity. In what follows we identify every univariate polynomial

$$f \;=\; \Sigma_{0 \leq i < n}\; f_i\, x^i \;\in R[x] \tag{13}$$

of degree less than n with its coefficient vector $(f_0, \ldots, f_{n-1}) \in R^n$.

Definition 2 *The R-linear map*

$$DFT_\omega \;:\; \left\{ \begin{array}{rcl} R^n & \longmapsto & R^n \\ f & \longmapsto & (f(1), f(\omega), f(\omega^2), \ldots, f(\omega^{n-1})) \end{array} \right. \tag{14}$$

which evaluates a polynomial at the powers of ω is called the Discrete Fourier Transform *(DFT).*

Proposition 2 *The R-linear map DFT_ω is an isomorphism.*

Proof $\triangleright$ Since the R-linear map DFT_ω is an endomorphism (the source and target spaces are the same) we only need to prove that DFT_ω is bijective. Observe that the Vandermonde matrix $VDM(1, \omega, \omega^2, \ldots, \omega^{n-1})$ is the matrix of the R-linear map DFT_ω. Then for proving that DFT_ω is bijective we need only to prove that $VDM(1, \omega, \omega^2, \ldots, \omega^{n-1})$ is invertible which holds iff the values $1, \omega, \omega^2, \ldots, \omega^{n-1}$ are pairwise different. A relation $\omega^i = \omega^j$ for $0 \leq i < j < n$ would imply $\omega^i\,(1 - \omega^{j-i}) = 0$. Since $(1 - \omega^{j-i})$ cannot be zero or a zero divisor then ω^i and thus ω must be zero. Then ω cannot be a root of unity. A contradiction. Therefore the values $1, \omega, \omega^2, \ldots, \omega^{n-1}$ are pairwise different and DFT_ω is an isomorphism. $\triangleleft$

Proposition 3 *Let V_ω denote the matrix of the isomorphism DFT_ω. Then ω^{-1} the inverse of ω is also a primitive n-th root of unity and we have*

$$V_\omega\, V_{\omega^{-1}} \;=\; nI_n \tag{15}$$

where I_n denotes the unit matrix of order n.

Proof $\triangleright$ Define $\omega' = \omega^{-1}$. Observe that $\omega' = \omega^{n-1}$. Thus ω' is a root of unity. We leave to the reader the proof that this a primitive n-th root of unity,

Let us consider the product of the matrix V_ω and $V_{\omega'}$. The element at row i and column k is

$$\begin{array}{rcl}(V_\omega\, V_{\omega'})_{ik} & = & \Sigma_{0\leq j<n}\ (V_\omega)_{ij}(V_{\omega'})_{jk} \\ & = & \Sigma_{0\leq j<n}\ \omega^{ij}\,(\omega')^{jk} \\ & = & \Sigma_{0\leq j<n}\ \omega^{ij}\,\omega^{-jk} \\ & = & \Sigma_{0\leq j<n}\ (\omega^{i-k})^j\end{array} \tag{16}$$

Observe that ω^{i-k} is either a power of ω or a power of its inverse. Thus, in any case this is a power of ω. If $i=k$ this power is 1 and $(V_\omega\, V_{\omega'})_{ik}$ is equal to n. If $i\neq k$ then the conclusion follows by applying the second statement of Lemma 1 which shows that $(V_\omega\, V_{\omega'})_{ik}=0$. $\lhd$

1.3 Convolution of polynomials

Let n be a positive integer and $\omega\in R$ be a primitive n-th root of unity.

Definition 3 *The* convolution *w.r.t. n of the polynomials $f=\Sigma_{0\leq i<n}\ f_i\,x^i$ and $g=\Sigma_{0\leq i<n}\ g_i\,x^i$ in $R[x]$ is the polynomial*

$$h\ =\ \Sigma_{0\leq k<n}\ h_k\,x^k \tag{17}$$

such that for every $k=0\cdots n-1$ the coefficient h_k is given by

$$h_k\ =\ \Sigma_{i+j\equiv k\mod n}\ \ f_i\,g_j \tag{18}$$

The polynomial h is also denoted by $f*_n g$ or simply by $f*g$ if not ambiguous.

Remark 2 Observe that the product of f by g is

$$p\ =\ \Sigma_{0\leq k<2n-1}\ \ p_k\,x^k \tag{19}$$

where for every $k=0\cdots 2n-2$ the coefficient p_k is given by

$$p_k\ =\ \Sigma_{i+j=k}\ \ f_i\,g_j \tag{20}$$

We can rearrange the polynomial p as follows.

$$\begin{array}{rcl}p & = & \Sigma_{0\leq k<n}\ (p_k\,x^k)\ \ +\ \ x^n\ \Sigma_{0\leq k<n-1}\ (p_{k+n}\,x^k) \\ & = & \Sigma_{0\leq k<n}\ (p_k\ +\ p_{k+n}\,x^n)\,x^k\end{array} \tag{21}$$

Therefore we have

$$f*g\ \equiv\ fg\mod x^n-1 \tag{22}$$

Example 5 Let $R = \mathbb{Z}/5\mathbb{Z}$, $n = 4$, $u \in R$ and consider the polynomials

$$f \;=\; x^3 + 1 \;\text{ and }\; g \;=\; 2x^3 + 3x^2 + x + 1. \tag{23}$$

Assume that we want to multiply f and g modulo $x^4 = u$. Then we have to arrange the product fg in a form $q(x^4 - u) + r$ where r has degree less than 4. We obtain

$$\begin{array}{rcl} f\,g & = & 2\,x^6 + 3\,x^5 + x^4 + 3\,x^3 + 3\,x^2 + x + 1 \\ & = & (2x^2 + 3x + 1)(x^4 - u) + 3x^3 + 3x^2 + x + 1 + (2x^2 + 3x + 1)u \\ & = & (2x^2 + 3x + 1)(x^4 - u) + 3x^3 + (3 + 2u)x^2 + (1 + 3u)x + (1 + u) \end{array} \tag{24}$$

For $u = 1$ we obtain

$$f \,*\, g \;=\; 3x^3 + 4x + 2 \tag{25}$$

Lemma 2 *For $f, g \in R[x]$ univariate polynomials of degree less than n we have*

$$DFT_\omega(f * g) \;=\; DFT_\omega(f) DFT_\omega(g) \tag{26}$$

where the product of the vectors $DFT_\omega(f)$ and $DFT_\omega(g)$ is computed componentwise.

Proof ▷ Since $f \,*\, g$ and $f\,g$ are equivalent modulo $x^n - 1$, there exists a polynomial $q \in R[x]$ such that

$$f \,*\, g \;=\; f\,g + q\,(x^n - 1) \tag{27}$$

Hence for $i = 0 \cdots n - 1$ we have

$$\begin{array}{rcl} (f \,*\, g)(\omega^i) & = & f(\omega^i)\, g(\omega^i) + q(\omega^i)(\omega^{i\,n} - 1) \\ & = & f(\omega^i)\, g(\omega^i) \end{array} \tag{28}$$

since $\omega^n = 1$. ◁

Remark 3 Consider the multi-point evaluation map

$$E_\omega \;:\; \left\{ \begin{array}{rcl} R[x] & \longmapsto & R^n \\ f & \longmapsto & (f(1), f(\omega), f(\omega^2), \ldots, f(\omega^{n-1})) \end{array} \right. \tag{29}$$

Its kernel is generated by $x^n - 1$. Since E_ω is surjective, it follows that

$$R[x]/\langle x^n - 1 \rangle \;\simeq\; R^n \tag{30}$$

(which is not a surprise!) and DFT_ω realizes this isomorphism.

If R is a field, then this a special case of the Chinese Remaindering Theorem where $m_i = x - \omega^i$ for $i = 0 \cdots n - 1$.

1.4 The Fast Fourier Transform

The Fast Fourier Transform computes the DFT quickly. This important algorithm for computer science (not only computer algebra, but also digital signal processing for instance) was (re)-discovered in 1965 by Cooley and Tukey.

Let n be a positive **even** integer, $\omega \in R$ be a primitive n-th root of unity and $f = \Sigma_{0 \leq i < n}\ f_i\, x^i$. In order to evaluate f at $1, \omega, \omega^2, \ldots, \omega^{n-1}$, we follow a *divide-and-conquer* strategy; more precisely, we consider the divisions with remainder of f by $x^{n/2} - 1$ and $x^{n/2} + 1$. So let q_0, q_1, r_0, r_1 be polynomials such that

$$f \ = \ q_0(x^{n/2} - 1) + r_0 \quad \text{with} \quad \left\{ \begin{array}{l} \deg(r_0) < n/2 \\ \deg(q_0) < n/2 \end{array} \right. \tag{31}$$

and

$$f \ = \ q_1(x^{n/2} + 1) + r_1 \quad \text{with} \quad \left\{ \begin{array}{l} \deg(r_1) < n/2 \\ \deg(q_1) < n/2 \end{array} \right. \tag{32}$$

The relations $\deg(q_0) < n/2$ and $\deg(q_1) < n/2$ hold because the polynomial f has degree less than n. Observe that the computation of (q_0, r_0) and (q_1, r_1) can be done very easily. Indeed, let $F_0, F_1 \in R[x]$ be such that

$$f \ = \ F_1\, x^{n/2} + F_0 \quad \text{with} \quad \left\{ \begin{array}{l} \deg(F_1) < n/2 \\ \deg(F_0) < n/2 \end{array} \right. \tag{33}$$

We have

$$f \ = \ F_1(x^{n/2} - 1) + F_0 + F_1 \quad \text{and} \quad f \ = \ F_1(x^{n/2} + 1) + F_0 - F_1 \tag{34}$$

Hence we obtain

$$r_0 \ = \ F_0 + F_1 \quad \text{and} \quad r_1 \ = \ F_0 - F_1 \tag{35}$$

Let i be an integer such that $0 \leq i < n/2$. By using Relation (31) with $x = \omega^{2\,i}$ we obtain

$$f(\omega^{2\,i}) \ = \ q_0(\omega^{2\,i})(\omega^{n\,i} - 1) + r_0(\omega^{2\,i}) \ = \ r_0(\omega^{2\,i}) \tag{36}$$

since $\omega^{n\,i} = 1$. Then, by using Relation (32) with $x = \omega^{2\,i+1}$ we obtain

$$f(\omega^{2\,i+1}) \ = \ q_1(\omega^{2\,i+1})(\omega^{n\,i}\,\omega^{n/2} + 1) + r_1(\omega^{2\,i+1}) \ = \ r_1(\omega^{2\,i+1}) \tag{37}$$

since $\omega^{n/2} = -1$. Indeed, this last equation follows from

$$0 \;=\; \omega^n - 1 \;=\; (\omega^{n/2} - 1)(\omega^{n/2} + 1) \tag{38}$$

and the fact that $\omega^{n/2} - 1$ is not zero nor a zero divisor. Therefore we have proved the following

Proposition 4 *Evaluating $f \in R[x]$ (with degree less than n) at $1, \omega^1, \ldots, \omega^{n-1}$ is equivalent to*

- *evaluate r_0 at the even powers $\omega^{2\,i}$ for $0 \le i < n/2$, and*
- *evaluate r_1 at the odd powers $\omega^{2\,i+1}$ for $0 \le i < n/2$.*

Since it is easy to show that ω^2 is a primitive $n/2$-th root of unity we can hope for a recursive algorithm. This algorithm would be easier if both r_0 and r_1 would be evaluated at the same points. So we define

$$r_1{}^*(\omega^{2\,i}) = r_1(\omega^{2\,i+1}). \tag{39}$$

Now we obtain the following algorithm.

Algorithm 1

Input: $n = 2^k$, $f = \Sigma_{0 \le i < n}\ f_i\, x^i$, and the powers $1, \omega, \omega^2, \ldots, \omega^{n-1}$ of a primitive n-th root of unity $\omega \in R$.

Output: $DTF_\omega(f) = (f(1), f(\omega), f(\omega^2), \ldots, f(\omega^{n-1}))$.

if $n = 1$ **return** (f_0)
$r_0 := \Sigma_{0 \le j < n/2}\ (f_j + f_{j+n/2})x^j$
$r_1^* := \Sigma_{0 \le j < n/2}\ \omega^j\ (f_j - f_{j+n/2})x^j$
call the algorithm recursively to evaluate r_0 and r_1^*
at the $n/2$ first powers of ω^2
return$(r_0(1), r_1^*(1), r_0(\omega^2), r_1^*(\omega^2), \ldots, r_0(\omega^{n-2}), r_1^*(\omega^{n-2}))$

Theorem 1 *Let n be a power of 2 and $\omega \in R$ be a primitive n-th root of unity. Then Algorithm 1 computes $DTF_\omega(f)$ using*

- $n \log(n)$ *additions in* R,
- $(n/2) \log(n)$ *multiplications by powers of* ω.

leading in total to $3/2\, n \log(n)$ *ring operations.*

PROOF ▷ By induction on $k = \log_2(n)$. Let $S(n)$ and $T(n)$ be the number of additions and multiplications in R that the algorithms requires for an input of size n. If $k = 0$ the algorithm returns (f_0) whose costs is null thus we have $S(0) = 0$ and $T(0) = 0$ which satisfies the formula since $\log(n) = \log(1) = 0$. Assume $k > 0$. Just by looking at the algorithm we that

$$S(n) \;=\; 2\, S(n/2) + n \quad \text{and} \quad T(n) \;=\; 2\, T(n/2) + n/2 \tag{40}$$

leading to the result by plugging in the induction hypothesis. ◁

1.5 Fast Convolution and Multiplication

Algorithm 2

Input: $f, g \in R[x]$ with degree less than $n = 2^k$, a primitive n-th root of unity $\omega \in R$.

Output: $f * g \in R[x]$

compute $1, \omega, \omega^2, \ldots, \omega^{n-1}$
$\alpha := DTF_\omega(f)$
$\beta := DTF_\omega(g)$
$\gamma := \alpha\, \beta$
return $(DTF_\omega)^{-1}(\gamma) \;=\; 1/n\, DTF_{\omega^{-1}}(\gamma)$

Theorem 2 *Let* n *be a power of* 2 *and* $\omega \in R$ *a primitive* n*-th root of unity. Then convolution in* $R[x]/\langle x^n - 1\rangle$ *and multiplication in* $R[x]$ *of polynomials whose product has degree less than* n *can be performed using*

- $3\, n \log(n)$ *additions in* R,
- $3/2\, n \log(n) + n - 2$ *multiplications by a power of* ω,
- n *multiplications in* R,

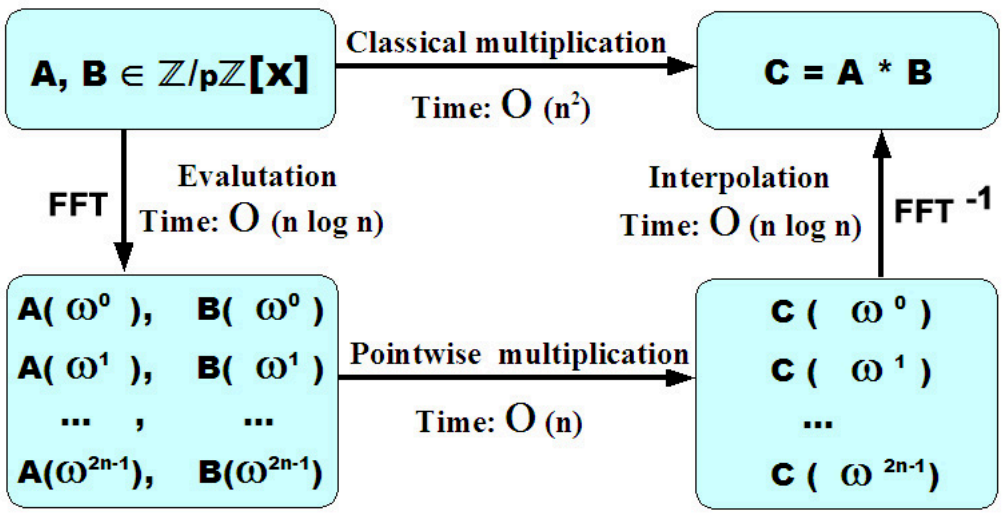

Figure 1: FFT-based univariate polynomial multiplication over $\mathbb{Z}/p\mathbb{Z}$

- *n divisions by n (as an element of R),*

leading to $9/2\, n \log(n) + \mathcal{O}(n)$ operations in R.

Remark 4 To multiply two arbitrary polynomials of degree less than $n \in \mathbb{N}$ we only need a primitive 2^k-th root of unity where

$$2^{k-1} \; < \; 2\, n \; \leq 2^k \tag{41}$$

Then we have decreased the cost of about $\mathcal{O}(n^2)$ of the classical algorithm to $\mathcal{O}(n \log(n))$.

2 Computing primitive n-th roots of unity

A preliminary step of the FFT-based multiplication is to find primitive n-th roots of unity. We explain in this section how to compute such numbers in $\mathbb{K} = \mathbb{Z}/p\mathbb{Z}$.

2.1 Some results from group theory

Definition 4 *Let $(G, .)$ be a (multiplicative) group with neutral element e. A nonempty subset $H \subseteq G$ is a* subgroup *of G if the following three statements hold*

- $e \in H$,
- *for every* $x, y \in H$ *we have* $x\,y \in H$,
- *for every* $x \in H$ *the inverse* x^{-1} *of* x *belongs to* H.

Theorem 3 *For every subgroup* H *of the additive abelian group* $(\mathbb{Z}, +)$ *there exists an element* $a \in \mathbb{Z}$ *such that* H *is the set of the multiples of* a*, that is* $H = a\mathbb{Z}$.

Theorem 4 *Let* G *be a multiplicative group with neutral element* e*. Let* $x \in G$ *an element and* $\mathsf{gr}(x)$ *the subgroup of* H *consisting of all powers of* x *(including* $e = x^0$ *and* x^{-1} *the inverse of* x*). Let* $\Theta(x)$ *be the order of* $\mathsf{gr}(x)$*, that is the cardinality of* $\mathsf{gr}(x)$*. Then two cases arise*

- *either* $\mathsf{gr}(x)$ *is infinite and then the powers of* x *are pairwise different and thus* H *is isomorphic to* $\mathbb{Z}$.
- *or* $\mathsf{gr}(x)$ *is finite and we have the following properties*
  1. $\Theta(x)$ *is the smallest integer* n *such that* $x^n = e$,
  2. $x^m = x^{m'}$ *iff* $m \equiv m' \mod \Theta(x)$,
  3. H *is isomorphic to* $(\mathbb{Z}/n\mathbb{Z}) \setminus \{0\}$ *where* $n = \Theta(x)$,
  4. $H = \{e, x, x^2, \ldots, x^{n-1}\}$ *where* $n = \Theta(x)$.

Theorem 5 (Lagrange) *For every subgroup* H *of the* **finite** *group* G*, the order (that is the cardinality) of* H *divides that of* G.

2.2 Primitive n-th roots of unity of finite fields

Theorem 6 *For* $n, p > 1$*, the finite field* $\mathbb{Z}/p\mathbb{Z}$ *has a primitive* n*-th root of unity if and only if* n *divides* $p - 1$.

Proof ▷ If ω is a a primitive n-th root of unity in $\mathbb{Z}/p\mathbb{Z}$ then the set

$$\Omega \;=\; \{1, \omega, \ldots, \omega^{n-1}\} \tag{42}$$

forms a cyclic subgroup H of the multiplicative group G_{p-1} of $\mathbb{Z}/p\mathbb{Z}$. By vertue of Lagrange's theorem (Theorem 5) the cardinality of H divides that of G_{p-1}. Since G_{p-1} has $p - 1$ elements, n divides $p - 1$.

Conversly, it is known from finite field theory that G_{p-1} is a cyclic group (even if p is a power of a prime rather than a prime). Let α be a generator of this group, that is

$$G_{p-1} \;=\; \{1, \alpha, \alpha^2, \ldots, \alpha^{p-2}\} \tag{43}$$

Recall that $\alpha^{p-1} = 1$ from the little Fermat's theorem. Let $n > 1$ be an integer dividing $p-1$ and define

$$\omega \;=\; \alpha^{(p-1)/n}. \tag{44}$$

Then we have

$$\omega^n \;=\; 1. \tag{45}$$

For all $0 < k < n$ we have $k\frac{p-1}{n} < p-1$ so we have

$$\omega^k = \alpha^{k(p-1)/n} \neq 1. \tag{46}$$

This shows that ω is a primitive n-th root of unity. $\lhd$

Example 6 Since $8 \mid (41-1)$ in $\mathbb{Z}/41\mathbb{Z}$ we have primitive 8-th root of unity. In fact the element $14 \in \mathbb{Z}/41\mathbb{Z}$ is such a root of unity.

Remark 5 Theorem 6 gives a necessary and sufficient condition for the existence of primitive n-th roots of unity in $\mathbb{Z}/p\mathbb{Z}$. But this does not give an algorithm to construct them.

We could use brute force. Given $\mathbb{Z}/p\mathbb{Z}$, for every n that we are interested in, for every $g \in G_{p-1}$ try if the following both statements hold:

- $g^n = 1$,
- for every prime divisor t of n we have $g^{n/t} \neq 1$.

However we can reduce the complexity of this search by computing a generator of G_{p-1} by means of Theorem 7 and then applying Relation (44).

Theorem 7 *An element α of the multiplicative subgroup G_{p-1} of the prime finite field $\mathbb{Z}/p\mathbb{Z}$ is a generator of G_{p-1} iff for every prime factor t of $p-1$ we have*

$$\alpha^{(p-1)/t} \;\neq\; 1 \tag{47}$$

PROOF ▷ Assume that the condition holds and that α is not a generator of G_{p-1}. Then there exists an integer n such that

$$1 < n < p-1 \quad \text{and} \quad \alpha^n = 1. \tag{48}$$

Let us choose n minimum with this property. Then $H = \{1, \alpha, \alpha^2, \ldots, \alpha^{n-1}\}$ is a subgroup of G. By virtue of Lagrange's theorem, the integer n (the order of H) must divide $p-1$. If n is not a prime then there exists a prime t dividing n. Define

$$\beta \;=\; \alpha^{n/t}. \tag{49}$$

Then $H' = \{1, \beta, \beta^2, \ldots, \beta^{t-1}\}$ is a subgroup of G contradicting the choice of n. Therefore n must be a prime, which contradicts the assumption that the condition holds. ◁

Remark 6 For the purpose of multiplying polynomials with coefficients in $\mathbb{Z}/p\mathbb{Z}$ we are interested in primes p and integers n of the form $n = 2^k$ such that $\mathbb{Z}/p\mathbb{Z}$ possesses a primitive n-th root of unity. By Theorem 6, the field $\mathbb{Z}/p\mathbb{Z}$ has such a root if there exists $q \in \mathbb{Z}$ such that

$$p \;=\; q\,2^k + 1 \tag{50}$$

Such primes are called *Fourier primes*.

But not all numbers of the form $q\,2^k + 1$ are prime (consider $q = k = 2$). So how frequent are the primes among the numbers of the form $q\,2^k + 1$ (for a given k). The answer is giving by the following theorem.

Theorem 8 *Let a and b be two relatively prime integers. Then the number of primes*

- *in the arithmetic progression $a\,k + b$ for $k = 1, 2, \ldots$*
- *and not greater than a given integer x*

is approximatively

$$\frac{x}{\phi(a)\log(x)} \tag{51}$$

where $\phi(a)$ is the Euler function at a (the number of integers less than and relatively prime to a).

Remark 7 We apply the previous formula with $a = 2^k$. All odd numbers less than 2^k are relatively prime to 2^k. (The even numbers are not relatively prime to 2^k.) Thus $\phi(2^k) = 2^{k-1}$. Therefore there are approximatively

$$\frac{x}{2^{k-1}\log(x)} \tag{52}$$

Fourier primes less than a given integer x.

Let $x = 2^{31}$, which represents the usual size required for single precision integers. For $k = 20$, there are approximatively 130 Fourier primes

- of the form $2^k\, q + 1$,
- and in the range $2^{20} \cdots 2^{31}$.

This shows that there are 130 prime finite fields $\mathbb{Z}/p\mathbb{Z}$ such that

- p fits in a machine integer (for a 32-bit processor)
- and allowing multiplication of polynomials in $\mathbb{Z}/p\mathbb{Z}[x]$ whose product is of degree less than $2^{20} = 1048576$.

So, it looks like we have many possible primes for doing **fast** modular computations with polynomials. However primes p such that p^2 fits in a machine integer are preferable in order to avoid the use of two machine words for multiplying two elements of $\mathbb{Z}/p\mathbb{Z}$.

In the *BasicMath* library, one of the parents of `libalgebra`, *nice primes* are organized in tables. So there is a category for *tables of primes*!

```
macro SI == SingleInteger;

PrimesTableCategory(s: SI): Category ==  with {
        sizeBound: () -> SI;
        tableSize: () -> SI;
        rank: SI -> Partial(SI);
        maxPrime: () -> SI;
        minPrime: () -> SI;
        previousPrime: SI -> SI;
        nextPrime: SI -> SI;
        primes: () -> Generator(SI);
        FourierDegree: SI -> SI;
```

```
        getPrimeOfFourierDegree: SI -> SI;
        nextPrimeOfFourierDegree: (SI, SI) -> SI;
        unitsGenerator: SI -> SI;
        primitiveRootofUnity: (SI, SI) -> SI;
}
```

where

- `PrimesTableCategory(s)` specifies the operations that we expect from a table of primes p less than 2^s. For obvious reasons, this category is restricted to primes that fit in a machine word.
- `sizeBound()` returns `s`.
- `tableSize()` returns the number of entries in the table.
- `rank(p)` returns tha rank of p in the table if it lies in.
- `FourierDegree(p)` returns the biggest integer r in the table such that 2^r divides $p-1$.
- `getPrimeOfFourierDegree(r)` returns the smallest prime p in the table such that `FourierDegree(p) >= r`, if any, otherwise 0 is returned.
- `nextPrimeOfFourierDegree(n,r)` returns the smallest prime p in the table such that `FourierDegree(p) = r` and $p > n$ hold, otherwise 0 is returned.
- `unitsGenerator(p)` assuming that p lies in the table returns a generator of the units group of the integers modulo p.
- `primitiveRootofUnity(p,n)` assuming that p lies in the table and assuming $0 < n \leq r$ where r is `FourierDegree(p)`, returns a primitive 2^n-root of unity of the units group of the integers modulo p.

```
macro SI == SingleInteger;

FourierPrimesTableCategory(r: SI, s: SI): Category ==  with {
        FourierDegree: () -> SI
        sizeBound: () -> SI
        tableSize: () -> SI
```

```
        rank: SI -> Partial(SI)
        maxPrime: () -> SI
        minPrime: () -> SI
        previousPrime: SI -> SI
        nextPrime: SI -> SI
        primes: () -> Generator(SI)
        unitsGenerator: SI -> SI
        primitiveRootofUnity: (SI, SI) -> SI
}
```

where`FourierPrimesTableCategory(r,s)` specifies the operations that we expect from the table of Fourier primes p less than 2^s and such that there exists an odd integer q satisfying $p - 1 = 2^r q$. Again we restrict to primes that fit in a machine word.

```
F6P15: FourierPrimesTableCategory(6,15) == add {
         local fd: SI == 6;
         local sb: SI == 15;
 ------------------------------------------------------------------------
 -- Table (6,15)                                                       --
 ------------------------------------------------------------------------

 -- Maximal size for a FFT: 2^6
 -- Number of 6-Fourier primes less than 2^15 is 58
MAXINDEX: SingleInteger == 58;
 -- These Fourier primes are:
local primeList: Array SingleInteger :=
   [193, 449, 577, 1217, 1601, 2113, 2753, 3137,
    4289, 4673, 4801, 5441, 5569, 5953, 6337, 6977,
    7489, 7873, 8513, 8641, 9281, 10177, 10433, 11329,
    11969, 12097, 13121, 13249, 13633, 14401, 14657, 15809,
    15937, 16193, 17729, 19009, 19777, 20161, 20929, 21313,
    21569, 22721, 23873, 24001, 25153, 25409, 25537, 25793,
    26177, 26561, 27073, 27329, 27457, 28097, 29633, 29761,
    30529, 32321];

 -- The corresponding units generators are:
```

```
local generatorList: Array SingleInteger :=
   [5, 3, 5, 3, 3, 5, 3, 3,
    3, 3, 7, 3, 13, 7, 10, 3,
    7, 5, 5, 17, 3, 7, 3, 7,
    3, 5, 7, 7, 5, 11, 3, 3,
    7, 5, 3, 23, 11, 13, 7, 5,
    3, 3, 3, 14, 10, 3, 10, 3,
    3, 3, 5, 3, 7, 3, 3, 17,
    13, 6];

        FourierDegree(): SI == fd;
        sizeBound(): SI == sb;
        tableSize(): SI == MAXINDEX;
.........................................
}



F7P15: FourierPrimesTableCategory(7,15) == add {
         local fd: SI == 7;
         local sb: SI == 15;
 ------------------------------------------------------------------------
 -- Table (7,15)                                                       --
 ------------------------------------------------------------------------

 -- Maximal size for a FFT: 2^7
 -- Number of 7-Fourier primes less than 2^15 is 29
MAXINDEX: SingleInteger == 29;
 -- These Fourier primes are:
local primeList: Array SingleInteger :=
   [641, 1153, 1409, 2689, 3457, 4481, 4993, 6529,
    7297, 9601, 9857, 10369, 11393, 12161, 13441, 13697,
    15233, 16001, 18049, 19073, 19841, 20353, 21121, 21377,
    26497, 28289, 29569, 30593, 31873];

 -- The corresponding units generators are:
local generatorList: Array SingleInteger :=
   [3, 5, 3, 19, 7, 3, 5, 7,
```

```
    5, 13, 5, 13, 3, 3, 11, 3,
    3, 3, 13, 3, 3, 5, 19, 3,
    5, 6, 17, 3, 11];

        FourierDegree(): SI == fd;
        sizeBound(): SI == sb;
        tableSize(): SI == MAXINDEX;
.....................................................


F8P15: FourierPrimesTableCategory(8,15) == add {
         local fd: SI == 8;
         local sb: SI == 15;
 ------------------------------------------------------------------------
 -- Table (8,15)                                                       --
 ------------------------------------------------------------------------

 -- Maximal size for a FFT: 2^8
 -- Number of 8-Fourier primes less than 2^15 is 12
MAXINDEX: SingleInteger == 12;
 -- These Fourier primes are:
local primeList: Array SingleInteger :=
   [257, 769, 3329, 7937, 9473, 14081, 14593, 22273,
    23297, 26881, 30977, 31489];

 -- The corresponding units generators are:
local generatorList: Array SingleInteger :=
   [3, 11, 3, 3, 3, 3, 5, 5,
    3, 11, 3, 7];

        FourierDegree(): SI == fd;
        sizeBound(): SI == sb;
        tableSize(): SI == MAXINDEX;
.....................................................


9P23: FourierPrimesTableCategory(9,23) == add {
         local fd: SI == 9;
```

```
        local sb: SI == 23;
 ------------------------------------------------------------------------
 -- Table (9,23)                                                        --
 ------------------------------------------------------------------------

 -- Maximal size for a FFT: 2^9
 -- Number of 9-Fourier primes less than 2^23 is 1092
       MAXINDEX: SingleInteger == 1092;
 -- These Fourier primes are:
       local primeList: Array SingleInteger :=
          [7681, 10753, 11777, 17921, 23041, 26113, 32257,
.....................................................

F10P24: FourierPrimesTableCategory(10,24) == add {
        local fd: SI == 10;
        local sb: SI == 24;
 ------------------------------------------------------------------------
 -- Table (10,24)                                                       --
 ------------------------------------------------------------------------

 -- Maximal size for a FFT: 2^10
 -- Number of 10-Fourier primes less than 2^24 is 1087
       MAXINDEX: SingleInteger == 1087;
 -- These Fourier primes are:
       local primeList: Array SingleInteger :=
          [13313, 15361, 19457, 25601,
.....................................................

F11P25: FourierPrimesTableCategory(11,25) == add {
        local fd: SI == 11;
        local sb: SI == 25;
 ------------------------------------------------------------------------
 -- Table (11,25)                                                       --
 ------------------------------------------------------------------------

 -- Maximal size for a FFT: 2^11
 -- Number of 11-Fourier primes less than 2^25 is 978
       MAXINDEX: SingleInteger == 978;
```

```
 -- These Fourier primes are:
        local primeList: Array SingleInteger :=
           [18433,
.....................................................

F12P26: FourierPrimesTableCategory(12,26) == add {
         local fd: SI == 12;
         local sb: SI == 26;
 ---------------------------------------------------------------------
 -- Table (12,26)                                                   --
 ---------------------------------------------------------------------

 -- Maximal size for a FFT: 2^12
 -- Number of 12-Fourier primes less than 2^26 is 972
        MAXINDEX: SingleInteger == 972;
 -- These Fourier primes are:
        local primeList: Array SingleInteger :=
           [12289,
.....................................................
```

2.3 A Probabilistic Approach

We explain in this section a simple probabilistic approach for computing primitive n-roots of unity in $\mathbb{Z}/p\mathbb{Z}$. Of course, we assume that n divides $p-1$. Hence, there exists an integer q such that $p = qn+1$ holds. According to little Fermat's theorem, for all $\alpha \in \mathbb{K}$, with $\alpha \neq 0$ and $\alpha \neq 1$, we have

$$\alpha^{p-1} = 1. \tag{53}$$

Therefore, we obtain

$$\alpha^{qn} = 1. \tag{54}$$

It follows that α^q is a candidate for being a primitive n-th root of unity. If α^q is not a primitive n-root root of unity then:

- the sequence of the α^{qk} for $k = 0, \ldots, n-1$ is periodic, with a period dividing n, and
- in particular $\alpha^{qn/2} = 1$ holds.

Since $\alpha^{qn/2}$ equals either 1 or -1, we have the following.

Proposition 5 *Let $p > 2$ be a prime number and let n, q be integers such that n is a power of 2 such that we have $p = nq + 1$. Let $\alpha \in \mathbb{Z}/p\mathbb{Z}$ different from 0 and 1. Then, α^q is a primitive n-th root of unity if and only if $\alpha^{qn/2} = -1$ holds.*

This proposition is very easy to implement and provides an efficient way to compute primitive roots of unity in practice. It avoids the construction of tables of primitive roots of unity, which was a usual approach as detailed above.

3 Efficient implementation

The theoretical complexity of Algorithm 1 is good but does not lead to a practical implementation of a fast multiplication of polynomials. According to [GCL92] it is not better than the classical algorithm for $n \leq 300$. The reasons are

- the recursive calls,
- the creation of intermediate polynomials.

We are going to get rid off these two difficulties leading to a practical algorithm better than the classical one for $n \geq 32$ (in ALDOR).

3.1 Towards an iterative algorithm for the DTF

We first change the intermediate polynomials. Let $n = 2^r$ be a power of 2, let $\omega \in R$ be a primitive n-th root of unity and $A(x) = \Sigma_{0 \leq i < n}\ a_i\, x^i$. We define

$$\begin{array}{rcl} A^{[0]}(x) & = & a_0 + a_2 x + a_4 x^2 + \cdots + a_{n-2} x^{n/2-1} \\ A^{[1]}(x) & = & a_1 + a_3 x + a_5 x^2 + \cdots + a_{n-1} x^{n/2-1} \end{array} \tag{55}$$

such that we have

$$A(x) \ = \ A^{[0]}(x^2) + x\, A^{[1]}(x^2). \tag{56}$$

Hence evaluating A at $1, \omega, \omega^2, \ldots, \omega^{n-1}$ is reduced to evaluate $A^{[0]}$ and $A^{[1]}$ at $1, \omega^2, \omega^4, \ldots, \omega^{2n-2}$. But this second sequence of powers of ω consists of two identical sequences of length $n/2$. Indeed, for every integer k we have

$$(\omega^{k+n/2})^2 \ = \ (\omega^k)^2 \tag{57}$$

Since $\omega^k(1-\omega^{n/2}) \neq 0$ holds, we obtain

$$\omega^{k+n/2} \;=\; -\omega^k \tag{58}$$

Hence we only need to evaluate $A^{[0]}$ and $A^{[1]}$ at

$$1, \omega^2, \omega^4, \ldots, \omega^{n-2}. \tag{59}$$

This provides us directly with the values of A at

$$1, \omega, \omega^2, \ldots, \omega^{n/2-1} \tag{60}$$

and the values of A at

$$\omega^{n/2}, \omega^{n/2+1}, \ldots, \omega^{n-1} \tag{61}$$

by using the fact these numbers are respectively equal to

$$-1, -\omega, -\omega^2, \ldots, -\omega^{n/2-1}. \tag{62}$$

To summarize, for $0 \leq i \leq n/2-1$ we have

$$\begin{array}{rcl} A(\omega^i) & = & A^{[0]}(\omega^{2i}) + \omega^i\, A^{[1]}(\omega^{2i}) \\ A(\omega^{i+n/2}) & = & A^{[0]}(\omega^{2i}) - \omega^i\, A^{[1]}(\omega^{2i}) \end{array} \tag{63}$$

Algorithm 3

Input: $n = 2^r$, $A = [a_0, a_1, \ldots, a^{n/2-1}, \ldots, a^{n-1}]$ an array for the coefficient of the polynomial, $\Omega = [1, \omega, \ldots, \omega^{n/2-1}, \ldots, \omega^{n-1}]$ an array of the powers of a primitive n-th root of unity ω.

Output: $DFT(n, A, \Omega)$ given as an array $y = [y_0, \ldots, y_{n-1}]$ such that y_i is $A(\omega^i)$, for $0 \leq i \leq n-1$.

$DFT(n, A, \Omega) ==$
  $step := \#(\Omega)/n$
  if $n = 1$ **return** $[f_0]$
  $A^{[0]} := [a_0, a_2, \ldots, a_{n-2}]$
  $A^{[1]} := [a_1, a_3, \ldots, a_{n-1}]$
  $y^{[0]} := DFT(n/2, A^{[0]}, \Omega)$
  $y^{[1]} := DFT(n/2, A^{[1]}, \Omega)$
  for $k := 0 \cdots n/2 - 1$ **repeat**
    $s :=$ step k
    $t := \Omega_s \;\; y_k^{[1]}$
    $y_k := y_k^{[0]} + t$
    $y_{k+n/2} := y_k^{[0]} - t$
  return y

The validity of Algorithm 3 follows from the two following observations.

- Formulas (63).
- If ω is a primitive 2^s-root of unity, with $s > 1$, then ω^2 is a primitive 2^{s-1}-root of unity.

Observe that the recursive calls of $DFT(n, A, \Omega)$ defines an ordering of the coefficients of A shown on Figure 2 for $n = 8$. Let us call this ordering the *DFT ordering* of A.

Observe that if the input array A is sorted w.r.t. this DFT ordering, one can describe easily the recursive calls with a binary tree. If this tree is traversed bottom-up, from left to right, we are led to **an iterative algorithm working in place!** For instance, for $n = 8$, assuming that the input array A is

$$| a_0 | a_4 | a_2 | a_6 | a_1 | a_5 | a_3 | a_7 |$$

We obtain the desired result in A using the following straigth-line program

1. $a := a_0$
2. $b := a_1$
3. $a_0 := a + b$
4. $a_1 := a - b$
5. $a := a_2$
6. $b := a_3$
7. $a_2 := a + b$
8. $a_3 := a - b$
9. $a := a_4$
10. $b := a_5$
11. $a_4 := a + b$
12. $a_5 := a - b$
13. $a := a_6$
14. $b := a_7$
15. $a_6 := a + b$
16. $a_7 := a - b$
17. $a := a_0$
18. $b := a_1$
19. $c := a_2$
20. $d := a_3$
21. $a_0 := a + c$
22. $a_2 := a - c$
23. $a_1 := b + \omega d$
24. $a_3 := b - \omega d$

25. $a := a_4$

26. $b := a_5$

27. $c := a_6$

28. $d := a_7$

29. $a_0 := a + c$

30. $a_2 := a - c$

31. $a_1 := b + \omega d$

32. $a_3 := b - \omega d$

33. $a := a_0$

34. $b := a_1$

35. $c := a_2$

36. $d := a_3$

37. $e := a_4$

38. $f := a_5$

39. $g := a_6$

40. $h := a_7$

41. $a_0 := a + e$

42. $a_1 := b + \omega f$

43. $a_2 := c + \omega^2 f$

44. $a_3 := d + \omega^3 h$

45. $a_4 := a + e$

46. $a_5 := b - \omega f$

47. $a_6 := c - \omega^2 f$

48. $a_7 := d - \omega^3 h$

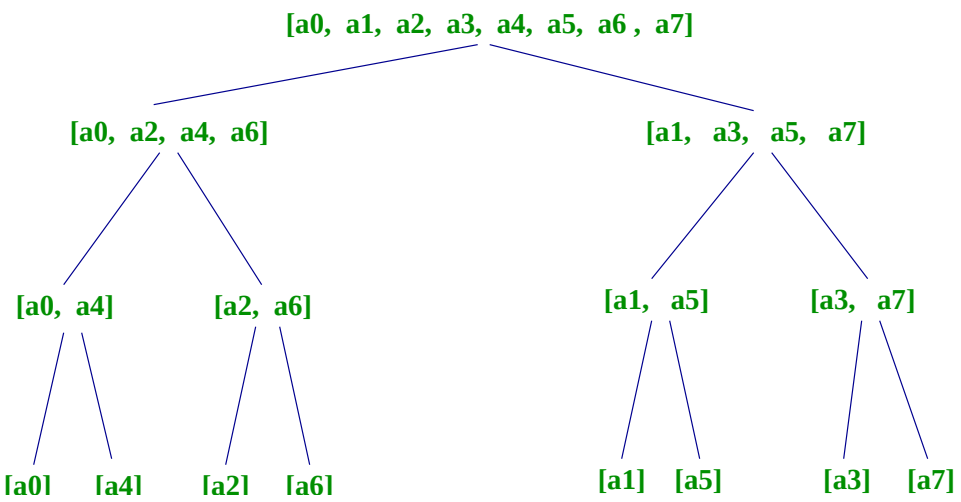

Figure 2: Recursive calls for $n = 8$.

Algorithm 4

Input: $n = 2^r$, A the array for the coefficient of the polynomial sorted by the *DFT ordering*. $\Omega = [1, \omega, \ldots, \omega^{n/2-1}, \ldots, \omega^{n-1}]$ an array of the powers of a primitive n-th root of unity ω.

Output: An array $y = [y_0, \ldots, y_{n-1}]$ such that y_i is $A(\omega^i)$ by calling $DFT(n, A, \Omega)$.

```
DFT(n, A, Ω) ==
  r := log_2(n)
  for s := 1 ··· r repeat
  # Traversing the tree, bottom-up.
    m := 2^s
    for k := 0 ··· n − 1 by m repeat
    # for each internal node from left to right
      for j = 0 ··· m/2 − 1 repeat
      # combine its two children
        t := Ω_jm  A[k + j + m/2]
        u := A[k + j]
        A[k + j] := u + t
        A[k + j + m/2] := u − t
  return A
```

Remark 8 To get the *DFT ordering* for A, that is $[a_0, a_4, a_2, a_6, a_1, a_5, a_3, a_7]$, observe that this ordering changes

$$000, 001, 010, 011, 100, 101, 011, 111 \tag{64}$$

to

$$000, 100, 010, 110, 001, 101, 110, 111 \tag{65}$$

Remark 9 The ring $\mathbb{Z}$ does not have any *interesting* roots of unity. In order to take advantage of the FFT-based multiplication one could use a modular approach as follows.

- Let f and g be two univariate polynomials in $\mathbb{Z}[x]$. We can assume that all their coefficients are positive. Otherwise, we can restrict to this case by setting

$$f \;=\; f^+ - f^- \quad \text{and} \quad g \;=\; g^+ - g^- \tag{66}$$

where f^+, f^-, g^+, g^- are polynomials with positive coefficients and writing

$$f\,g \;=\; f^+g^+ + f^-g^- - f^+g^- - f^-g^+ \tag{67}$$

and computing the products f^+g^+, f^-g^-, f^+g^- and f^-g^+ one after another.

- Let $||f||$ and $||g||$ be the max-norms of f and g. Let d be the degree of $f\,g$. For $k = 0 \cdots d$ a bound for the coefficient c_k of x^k in $f\,g$ is given by

$$|c_k| \;\leq\; (k+1)||f||||g|| \tag{68}$$

- Let M be the biggest value of these bounds. Let $p_1, \ldots, p_r$ be primes such
  - 2^d divides each $p_1 - 1, \ldots, p_r - 1$
  - the product of $p_1, \ldots, p_r$ is bigger than $2\,M$.
- Then we can compute $f\,g$ in each of the $\mathbb{Z}/p_i\mathbb{Z}[x]$ using the FFT-based multiplication.
- Finally we can recombine these modular product by applying the CRA to each coefficient c_k (of x^k) in $f\,g$.

References

[GCL92] K.O. Geddes, S.R. Czapor, and G. Labahn. *Algorithms for Computer Algebra*. Kluwer Academic Publishers, 1992.

[GG99] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.