A Survey of Parallel Algorithms for Sparse Algebraic Interpolation

Thorsten Werther
Dept. of Computer Science
University of Bonn

August 1988

Abstract

This paper gives a survey of some recent results on the parallel complexity of the sparse black box interpolation problem for multivariate polynomials and rational functions over arbitrary fields. In this setting, rather than the degree of the polynomial, the number of terms is of importance. Given a multivariate polynomial (or rational function) over an arbitrary field, as a black box (input oracle), and an information about its sparsity (a bound on the number of non-zero coefficients) we have to determine the complexity of reconstructing the polynomial. Some of the recent NC-algorithms and implementations using the computer-algebra system Scratchpad II are presented.

This paper is a revised version of a thesis for a Master's Degree in Computer Science, University of Bonn, August 1988.

Contents

1 Introduction 1

2 Efficient Algorithms for the Permanent Problem 3

2.1 The Decision Problem 3

2.2 The Construction Problem 6

2.3 The Enumeration Problem 6

3 Polynomial Interpolation over Fields of Characteristic Zero 8

3.1 The First Deterministic Polynomial Algorithm 8

3.2 Development of an NC-Algorithm 11

4 Polynomial Interpolation over Finite Fields 15

4.1 Bounds on the Number of Evaluation Points 15

4.1.1 Upper Bounds 16

4.1.2 Lower Bounds 21

4.2 Efficient Algorithms with few Evaluation Points 24

4.3 Development of the first NC-Algorithm 29

4.3.1 The Zero Test 30

4.3.2 The Enumeration Technique 32

4.3.3 The Basis Step 33

4.3.4 The Recursion Step 34

4.3.5 The NC-Interpolation Algorithm for Sparse Polynomials 35

4.3.6 Analysis of the Algorithm 37

5 NC-Interpolation of Rational Functions 39

5.1 The Basis Step 40

5.2 Recursion Step 42

5.3 The Rational NC-Interpolation Algorithm 45

5.4 Analysis of the Algorithm 47

5.5 Handling pathological situations 48

6 Conclusions **49**

A Implementation in Scratchpad II **50**

A.1 The Computer Algebra System Scratchpad II 50

A.2 The Interpolation Algorithm [CGK 87] 51

A.3 The NC-Interpolation Algorithm [GKS 88] 57

A.4 The rational NC-Interpolation Algorithm [GK 88] 63

1 Introduction

The problem of interpolating polynomials reflects an important topic in the history of mathematics. The classical interpolation problem states as follows: Given $n+1$ function values $y(x_k) = y_k$, an interpolation formula approximates the function $y(x)$ by a suitable function $Y(x) \equiv Y(x, \alpha_0, \ldots, \alpha_n)$ from a given class of functions (e.g. polynomials, rational functions) depending on $n+1$ parameters α_j chosen so that $Y(x_k) = y(x_k)$ for the given set of $n+1$ argument values x_k.

The interpolation formulae of Newton, Lagrange and Hermite for univariate polynomials of fixed degree over fields of characteristic zero laid foundations for the whole area of numeric analysis. However, polynomials are not well suited for approximating smooth functions since they tend to oscillate between interpolation points. Rational functions belong to a more adaptable function class (one aspect in rational interpolation is the construction of algorithms to speed up convergence of certain series, e.g. Richardson's method, extrapolation algorithms). Further generalizations of the classical polynomial interpolation include the study of other function classes (trigonometric functions for Fourier Analysis, Spline interpolation for manipulation of ordinary and partial differential equations) as well as interpolation over finite fields and multivariate interpolation.

A more recent motivation for the study of algebraic interpolation problems is the question of specifying appropriate data structures to store polynomials efficiently. The sparse representation of a polynomial, i.e. representing a polynomial by a list of non-zero coefficients and the corresponding exponents, proved to be very successful. Another possibility is given by algebraic straight-line programs (cf. [Ka 85]) representing an abstract operational command-sequence to describe a polynomial. The length of the derivation-sequence corresponds to the time complexity of the straight-line program; the space complexity is given by the length of the encoding of the straight-line program.

In this context, conversion algorithms from one representation into another one are of special interest. Here, not the degree of the polynomial is of importance, but rather the number of non-zero coefficients.

This motivates the problem of sparse black box interpolation. In this scenario we are given a polynomial as a black box (input oracle), and an information about its sparsity (a bound on the number of non-zero coefficients). For any evaluation point as input, the black box outputs the value of the polynomial at this point. Given this, we have to determine an efficient interpolation algorithm reconstructing the sparse representation of the polynomial.

The model of black box is independent of the representation of the polynomial. One may think of the black box as a straight-line program, the solution of the sparse black box interpolation problem serves as a conversion algorithm into the sparse representation.

This paper studies the parallel complexity of the black box interpolation for multivariate polynomials and rational functions. The reader is referred to [Co 85], [KR 88] for the basic models of parallel computation.

In case of fields of characteristic 0 we say the black box interpolation problem for arbitrary t-sparse n-multivariate polynomials and rational functions is in NC^k if there exists a class of uniform $(ntd)^{O(1)}$-size and $O(\log^k(ntd))$-depth boolean circuits with oracle nodes (returning values of the black box) computing the sparse representation. Here, d is a bound on the degree

of the polynomial or rational function.

For the finite field GF(q) the black box interpolation problem for t-sparse n-multivariate polynomials is in NC^k if there exists a class of uniform $(ntq)^{O(1)}$-size and $O(\log^k(ntq))$-depth boolean circuits with oracle nodes (returning values of the black box) computing the sparse representation.

If the computation of function values is in NC, then the explicit interpolation problem lies in NC provided the black box interpolation problem is in NC.

This paper is a revised version of my thesis for a Master's Degree in Computer Science, University of Bonn, August 1988. It is organized as follows.

In Chapter 2, we present the work of Grigoriev and Karpinski [GK 87] on finding perfect matchings in a bipartite graph with polynomially bounded permanent, which represents a crucial step towards the construction of deterministic interpolation algorithms.

Chapter 3 studies the problem of deterministic polynomial interpolation over fields of characteristic zero. The techniques developed by Grigoriev and Karpinski given in Chapter 2 are extended by Ben-Or and Tiwari and lead to a deterministic polynomial algorithm [BT 88].

The more complex problem of interpolation over finite fields gains a definite relevance in the construction of efficient algorithms for algebraic problems (factorization of polynomials, computation of primitive elements, computation of discrete logarithms, cf. [Ga 83], [Ga 84], [Ka 85]) as well as for applications in coding theory and cryptography (cf. [LN 86]). Chapter 4 gives bounds on the number of evaluation points. We also present deterministic polynomial interpolation algorithms.

It has been assumed that the problem of interpolating rational functions is only solvable non-deterministically in polynomial time. The algorithms developed in the previous chapters result in an efficient probabilistic solution. However, the methods introduced prove themselves strong enough to derive even a deterministic polynomial algorithm for rational interpolation (over fields of characteristic zero). This is pointed out in Chapter 5.

Finally, implementations of the interpolation algorithms presented in this paper are introduced in appendix A. For this purpose the computer-algebra system Scratchpad II is used. The implementations were designed at the Scientific Center of IBM Germany in Heidelberg.

2 Efficient Algorithms for the Permanent Problem

A crucial step for the construction of efficient NC-interpolation algorithms is supplied by the work of Grigoriev and Karpinski [GK 87] on finding matchings for bipartite graphs. Because of the equivalence to the perfect matching problem of bipartite graphs the permanent forms the core of the solution of many interesting problems concerning parallel computation theory (e.g. Maximum Flow Problem, Two-Processor Scheduling Problem).

Definition 2.1 (Permanent)
The *permanent* $\mathrm{perm}(A)$ of a matrix $A \in (n \times n, I\!K)$ is defined by

$$\mathrm{perm}(A) := \sum_{\sigma \in S_n} \prod_{j=1}^{n} a_{j,\sigma(j)}. \tag{1}$$

A permutation $\sigma \in S_n$ is called *non-vanishing* if σ contributes a non-zero term to $\mathrm{perm}(A)$.

Let $A \in (n \times n, \{0,1\})$ be the adjacency matrix of a bipartite graph G. Any non-vanishing permutation corresponds uniquely to a perfect matching of G. Hence, the number of perfect matchings of G is given by $\mathrm{perm}(A)$.

The problem of computing the permanent is threefolded:

Decision problem: The computation of the logical permanent:

$$\mathrm{perm}(A) := \bigvee_{\sigma \in S_n} \bigwedge_{j=1}^{n} a_{j,\sigma(j)}$$

(does G have a perfect matching).

Construction problem: Construction of a non-vanishing permutation (construction of arbitrary perfect matchings of G).

Enumeration problem: Enumeration of all non-vanishing permutations (enumeration of all perfect matchings of G).

The solution of these three problems in the special case of polynomially bounded permanent ($\mathrm{perm}(A) \leq cn^k$) is carried out using techniques from [GK 87]. The adaptability of these techniques for the interpolation is of special interest.

2.1 The Decision Problem

Let $A \in (n \times n, I\!K)$. We design an efficient zero test for $\mathrm{perm}(A)$.

Let l be the number of ones occurring in A. It holds $0 \leq l \leq n^2$. Let A_z be the matrix resulting from A by substituting the ones in A by pairwise distinct variables z_i.

$$(A_z)_{i,j} := \begin{cases} z_k & \text{if } a_{i,j} \text{ is the } k\text{-th one occurring in } A \\ 0 & \text{if } a_{i,j} = 0 \end{cases} \tag{2}$$

Let ρ be a polynomial in the variables $z_1, \ldots, z_l$ defined by $\rho(z_1, \ldots, z_l) = \det(A_z)$. ρ has coefficients from $\{-1, 0, 1\}$. It holds:

$$\begin{aligned}
\rho(z) &= \sum_{\sigma \in S_n} (-1)^\sigma \cdot \prod_{j=1}^{n} (A_z)_{j,\sigma(j)} \\
&= \sum_{\substack{\sigma \in S_n \\ (-1)^\sigma = 1}} \underbrace{\prod_{j=1}^{n} (A_z)_{j,\sigma(j)}}_{=t'_i} - \sum_{\substack{\sigma \in S_n \\ (-1)^\sigma = -1}} \underbrace{\prod_{j=1}^{n} (A_z)_{j,\sigma(j)}}_{=t''_j} \\
&= \sum_i t'_i - \sum_j t''_j .
\end{aligned}$$

The t'_i, t''_j are the monomials with non-zero coefficients occurring in ρ. They correspond to the pairwise distinct non-vanishing permutations of A. Concerning matrices over $\{0, 1\}$, the number of monomials corresponds to perm(A).

Lemma 2.2 $\qquad \text{perm}\,(A) = 0 \iff \rho \equiv 0$

PROOF:

$$\begin{aligned}
\text{perm}\,(A) = 0 &\iff \forall \sigma \in S_n : \prod_{j=1}^{n} a_{j,\sigma(j)} = 0 \\
&\iff \forall i : t'_i \equiv 0 \quad \wedge \quad \forall j : t''_j \equiv 0 \\
&\iff \rho \equiv 0
\end{aligned}$$

□

By means of Lemma 2.2, the decision problem is reduced to the problem of testing ρ to zero. This fits in our scenario of black box interpolation. ρ is defined as the determinant of A_z, hence a black box evaluation of ρ corresponds to computing $\det(A_z)$ for some given argument. There exist several efficient NC-algorithms for computing the determinant of an $n \times n$ matrix with $n^{O(1)}$-bit entries, e.g. the algorithm proposed in [BGH 82] takes $O(\log^2 n)$ boolean parallel time and $O(n^{4.5})$ boolean processors.

In order to test ρ to zero, we have to choose a small number of evaluation points such that the information extracted from these function values distinguishes ρ from the zero-polynomial. Grigoriev and Karpinski solve this problem by evaluating the determinant of the matrix A_z at points $(p_1^m, \ldots, p_l^m)$ with pairwise distinct primes p_i, and using the uniqueness of the prime factorization. Let

$$A_m := (a_{i,j}^m) = A_z|_{z=(p_1^m, \ldots, p_l^m)}$$

and

$$\begin{aligned}
\rho_m &:= \rho(p_1^m, \ldots, p_l^m) \\
&= \sum_i t'_i(p_1^m, \ldots, p_l^m) - \sum_j t''_j(p_1^m, \ldots, p_l^m) \\
&= \sum_i \underbrace{t'_i(p_1, \ldots, p_l)^m}_{x_i} - \sum_j \underbrace{t''_j(p_1, \ldots, p_l)^m}_{y_j} \\
&= \sum_i x_i^m - \sum_j y_j^m . \qquad (3)
\end{aligned}$$

The evaluation at prime powers yields a separation of different monomials of ρ. Therefore $\{x_i\}$ and $\{y_j\}$ are unique prime codings of pairwise distinct permutations; so all $x_i \neq 0, y_j \neq 0$ are pairwise distinct. We will use this technique also for interpolation.

Theorem 2.3 Let $A \in (n \times n, \{0,1\})$ and let ρ be defined as above. Then

$$\rho \equiv 0 \iff \forall\, 1 \leq m \leq \text{perm}\,(A) : \rho_m = 0$$

PROOF: The 'if' part of the statement is trivial. We have to prove the 'only if' part. x_i, y_j are unique codings of the monomials of ρ, therefore the following equivalences hold:

$$\begin{aligned} \rho \equiv 0 &\iff \forall i : t_i' \equiv 0 \quad \wedge \quad \forall j : t_j'' \equiv 0 \\ &\iff \forall i : x_i = 0 \quad \wedge \quad \forall j : y_j = 0 \\ &\iff \{x_i\} = \{y_j\}, \end{aligned}$$

because all non-zero x_i, y_j are pairwise distinct. We have to prove:

$$\forall\, 1 \leq m \leq \text{perm}\,(A) : \rho_m = 0 \implies \{x_i\} = \{y_j\}.$$

Let c_i (d_i) be the i-th elementary symmetric polynomial, evaluated at $x_1, \ldots, x_{\text{perm}(A)}$ (evaluated at $y_1, \ldots, y_{\text{perm}(A)}$). Let

$$s_m = \sum_{i=1}^{\text{perm}(A)} x_i^m \qquad \text{and} \qquad r_m = \sum_{j=1}^{\text{perm}(A)} y_j^m.$$

Equation (3) implies that $s_m = r_m$ for $1 \leq m \leq \text{perm}\,(A)$. The *Newton* formula (cf. [LN 86]) yields the following recurrence equations for s_m and r_m:

$$\begin{aligned} s_1 &= c_1 \\ s_2 &= c_1 s_1 - 2c_2 \\ &\vdots \\ s_{\text{perm}(A)} &= \sum_{i=1}^{\text{perm}(A)-1} (-1)^{i-1} c_i s_{\text{perm}(A)-i} + (-1)^{\text{perm}(A)-1} \text{perm}(A) c_{\text{perm}(A)} \end{aligned}$$

and similarly for r_m:

$$r_m = \sum_{i=1}^{m-1} (-1)^{i-1} d_i r_{m-i} + (-1)^{m-1} m d_m.$$

Since $r_m = s_m$ we conclude $c_m = d_m$ for $1 \leq m \leq \text{perm}\,(A)$; therefore there exists a permutation σ with $x_i = y_{\sigma(i)}$, so $\{x_i\} = \{y_j\}$. □

As an immediate consequence of Theorem 2.3 we obtain:

Theorem 2.4 Let $A \in (n \times n, \{0,1\})$ with polynomially bounded permanent, i.e. $\text{perm}\,(A) \leq cn^k$ for some constants c, k. Then the decision problem for A is in NC^2. The algorithm takes $O(\log^2 n)$ parallel time and $O(n^{k+4.5} \log n)$ processors.

PROOF: Let A_m be defined as above. By Theorem 2.3 it suffices to calculate the determinants of these matrices for $1 \leq m \leq cn^k$. For that purpose we need at most the first n^2 primes. By the prime theorem the n^2-th prime is bounded by $O(n^2 \log n)$. Therefore the greatest entry $p_{n^2}^{cn^k}$ can be represented by $O(n^k \log n)$ bits. Hence, the determinants can be computed in parallel by the NC^2-algorithm from [BGH 82] mentioned above. We have to evaluate at most $O(n^k)$ determinants, so our statement holds. □

2.2 The Construction Problem

The efficient zero test for polynomially bounded permanents forms the core for the solution of the construction problem.

Let $A \in (n \times n, \{0,1\})$ with polynomially bounded permanent. Recursive parallel splitting of A using the divide-and-conquer method and applications of the zero test put the status of the construction problem in NC^3.

Theorem 2.5 Let $A = (a_{i,j}) \in (n \times n, \{0,1\})$ with $\text{perm}(A) \leq cn^k$. The construction of a non-vanishing permutation of A lies in NC^3.

PROOF: Let $A_{i,j}$ be the matrix obtained from A by canceling the i-th row and j-th column. Using Theorem 2.3 we test in parallel for each $a_{i,j} = 1$, whether $\text{perm}(A) \neq 0$ (whether there is a non-vanishing permutation σ with $\sigma(i) = j$). We call an element $a_{i,j}$ with this property a *generator*.
If there is exactly one generator $a_{i_0,j}$ for each row i_0 of A, then a unique permutation is given. Otherwise the row i_0 consists of at least of two generators a_{i_0,j_1} and a_{i_0,j_2}. Hence the non-vanishing permutation σ is not unique at point i_0. However, either A_{i_0,j_1} or A_{i_0,j_2} contains at most half of the non-vanishing permutations. The procedure is applied in parallel to the matrices A_{i_0,j_1} and A_{i_0,j_2}.
While the number of non-vanishing permutations is bounded by cn^k, after at most $t \leq \log(cn^k) = O(\log n)$ recursion steps we find a unique non-vanishing partial permutation, which can be composed to a unique non-vanishing permutation.
The zero test is in NC^2; hence the proposed construction lies in NC^3. □

2.3 The Enumeration Problem

Once again recursive applications of the zero test lead to a solution of the enumeration problem. However, a new logarithmic enumeration technique developed by Grigoriev and Karpinski is of crucial importance. As described in Chapter 4.3 and 5, this enumeration technique is also used for interpolation of polynomials in finite fields and of rational functions yielding NC-algorithms for those problems, too.

A set $\{a_{i_1,j_1}, \ldots, a_{i_r,j_r}\}$ of entries in A is called a partial solution of a non-vanishing permutation of A, if there exists a non-vanishing permutation σ, such that the elements of $\{a_{i_1,j_1}, \ldots, a_{i_r,j_r}\}$ occur in σ:

$$\exists\, \sigma \in S_n, \sigma \text{ non-vanishing} : \ \sigma(i_1) = j_1, \ldots, \sigma(i_r) = j_r.$$

Let $\mathcal{S} = \{a_{i_1,j_1}, \ldots, a_{i_r,j_r}\}$ with $a_{i_s,j_s} \neq 0$ for $s = 1, \ldots, r$, pairwise distinct indices $i_1, \ldots, i_r$ and pairwise distinct indices $j_1, \ldots, j_r$. Theorem 2.3 supplies an efficient test whether $\mathcal{S}$ represents a partial solution:
Let $A^{(i_1,\ldots,i_r)}_{(j_1,\ldots,j_r)}$ be the matrix, obtained from A by canceling the rows $i_1, \ldots, i_r$ and the columns $j_1, \ldots, j_r$. Then $\mathcal{S}$ is a partial solution if there exists a non-vanishing permutation for $A^{(i_1,\ldots,i_r)}_{(j_1,\ldots,j_r)}$.

Theorem 2.6 Let $A = (a_{i,j}) \in (n \times n, \{0,1\})$ with $\text{perm}(A) \leq cn^k$. Then there exists an NC^3-algorithm for determining all non-vanishing permutations of A. The algorithm requires $O(\log^3 n)$ parallel time and $O(n^{3k+5.5} \log n)$ processors.

PROOF: Assume $n = 2^m$ for simplicity of notation. For fixed α with $1 \leq \alpha \leq m+1$ we distribute the rows of A among $2^{m+1-\alpha}$ blocks consisting of $2^{\alpha-1}$ successive rows. The blocks are numbered from 0 to $2^{m+1-\alpha}-1$ and are denoted by $B_{\alpha,\beta}$ for $0 \leq \beta < 2^{m+1-\alpha}$. $B_{\alpha,\beta}$ consists of the rows with indices $\beta 2^{\alpha-1}+1, \ldots, \beta 2^{\alpha-1}+2^{\alpha-1}$. Let $S_{\alpha,\beta}$ be the set of partial solutions corresponding to the rows of block $B_{\alpha,\beta}$:

$$S_{\alpha,\beta} = \{(j_1, \ldots, j_{2^{\alpha-1}}) \mid a_{\beta \cdot 2^{\alpha-1}+1,j_1}, \ldots, a_{\beta \cdot 2^{\alpha-1}+2^{\alpha-1},j_{2^{\alpha-1}}} \text{ is a partial solution}\}$$

with $1 \leq \alpha \leq m+1$ and $0 \leq \beta < 2^{m+1-\alpha}$.

For $0 \leq \beta < n$, $B_{1,\beta}$ consists of the row with index $\beta+1$. Then

$$S_{1,\beta} = \{j \mid a_{\beta+1,j} \text{ is a generator}\}.$$

The sets $S_{1,\beta}$ form a basis for recursive construction of larger partial solutions. $S_{1,\beta}$ can be computed in parallel for β, testing for each $1 \leq j \leq n$, whether $a_{\beta+1,j}$ is a generator (cf. Theorem 2.5). This basic step can be performed by n^2 parallel applications of the zero test given by Theorem 2.3.

With $\alpha = m+1$ ($\beta = 0$) we obtain the set of non-vanishing permutations of A:

$$\begin{aligned} S_{m+1,0} &= \{(j_1, \ldots, j_n) \mid a_{1,j_1}, \ldots a_{n,j_n} \text{ is a general solution}\} \\ &\cong \{\sigma \in S_n \mid \sigma \text{ is a non-vanishing permutation of } A\}. \end{aligned}$$

Starting with $S_{1,\beta}$, we determine $S_{\alpha,\beta}$ recursively for $\alpha = 2, \ldots, m+1$. Construct the set $\bar{S}_{\alpha+1,\beta}$ from $S_{\alpha,2\beta}$ and $S_{\alpha,2\beta+1}$:

$$\bar{S}_{\alpha+1,\beta} = \{u, v \mid u \in S_{\alpha,2\beta},\ v \in S_{\alpha,2\beta+1}\}.$$

$\bar{S}_{\alpha+1,\beta}$ consists of potential candidates for $S_{\alpha+1,\beta}$. We have to eliminate those elements from $\bar{S}_{\alpha+1,\beta}$ which do not represent a partial solution. Then

$$S_{\alpha+1,\beta} = \{w \in \bar{S}_{\alpha+1,\beta} \mid w \text{ is a partial solution}\}$$

(cf. Fig. 1 in Section 4.3).

Within $m = O(\log n)$ recursion steps, the set of non-vanishing permutations $S_{m+1,0}$ is constructed.

$|S_{\alpha+1,\beta}|$ is bounded by $\operatorname{perm}(A) \leq cn^k$; hence, $|\bar{S}_{\alpha+1,\beta}|$ is bounded by $c^2 n^{2k}$ for any β. At most n sets of potential candidates are to be tested in each recursion step; therefore the zero test of the corresponding partial matrix is performed at most $O(n^{2k+1})$ times. As stated in Theorem 2.4 the zero test requires $O(\log^2 n)$ parallel time and $O(n^{k+4.5} \log n)$ processors; so the enumeration algorithm takes $O(\log^3 n)$ parallel time and $O(n^{3k+5.5} \log n)$ processors. □

Obviously, the number of all non-vanishing permutations is determined by enumerating them. Concerning any matrix with entries from $\{0,1\}$, this number corresponds to its permanent.

Corollary 2.7 Let $A \in (n \times n, \{0,1\})$ with $\operatorname{perm}(A) \leq cn^k$. The computation of $\operatorname{perm}(A)$ lies in NC^3.

3 Polynomial Interpolation over Fields of Characteristic Zero

3.1 The First Deterministic Polynomial Algorithm

The reduction of the number of requests to the black box was the primary target of effort in oder to develop efficient interpolation algorithms.

The first starting-point is the probabilistic interpolation algorithm of Zippel [Zi 79] for sparse polynomials in the variables $x_1, \ldots, x_n$. The algorithm works in n steps. At the i-th step, the polynomial is considered as a polynomial in the variables $x_1, \ldots, x_i$ with coefficients corresponding to polynomials in $x_{i+1}, \ldots, x_n$. Partial monomials in $x_1, \ldots, x_i$ with non-zero coefficients are determined by evaluating the polynomial at randomly selected points. Schwartz [Sc 80] shows that the probability of hitting a root of a non-zero polynomial is small. Linear systems of equations for the coefficients of partial monomials are derived which are solvable with high probability.

Tiwari [Ti 87] succeeded in designing a deterministic algorithm from [Zi 79] using techniques presented by Grigoriev and Karpinski [GK 87]. Instead of choosing the evaluation points at random, monomials are separated by evaluating the polynomial for prime powers. The resulting linear systems of equations are solvable in any case.

Theorem 3.1 Let $I\!K$ be an arbitrary field of characteristic zero and $f \in I\!K[x_1, \ldots, x_n]$ a t-sparse polynomial with $\deg_{x_i}(f) < d$. f is identical to zero iff f vanishes at the points $(p_1^k, \ldots, p_n^k)$ for $0 \leq k < t$, where the p_j's are pairwise distinct primes.

PROOF: Let

$$f(x_1, \ldots, x_n) = \sum_{i=1}^{t} c_i \cdot t_i(x_1, \ldots, x_n)$$

where the t_i's are the monomials occurring in f with coefficients $c_i \in I\!K$. Let μ_k be the value of f at the point $(p_1^k, \ldots, p_n^k)$ and let $\Omega_i = t_i(p_1, \ldots, p_n)$. Due to the uniqueness of prime factorization (cf. [GK 87])

$$\Omega_i \neq \Omega_j \quad \text{for} \quad i \neq j. \tag{4}$$

We have to prove: $f \equiv 0 \iff \mu_k = 0$ for all $0 \leq k < t$. It holds:

$$\mu_k = f(p_1^k, \ldots, p_n^k) = \sum_{i=1}^{t} c_i t_i(p_1^k, \ldots, p_n^k) = \sum_{i=1}^{t} c_i t_i(p_1, \ldots, p_n)^k = \sum_{i=1}^{t} c_i \Omega_i^k.$$

We obtain the following linear system of equations:

$$\underbrace{\left(\Omega_i^k\right)_{\substack{0 \leq k < t \\ 1 \leq i \leq t}}}_{=: \mathcal{V}} \cdot (c_i)_{1 \leq i \leq t} = (f_i)_{0 \leq k < t} \tag{5}$$

By (4), $\mathcal{V}$ is a non-singular Vandermonde matrix and the linear system of equations has a unique solution. Hence

$$f \equiv 0 \iff c_i = 0 \text{ for all } 1 \leq i \leq t \iff \mu_k = 0 \text{ for all } 0 \leq k < t.$$

□

This zero test is to be used for the interpolation of t-sparse polynomials. We determine recursively all occurring partial monomials in the variables $x_1, \ldots, x_{i+1}$ where the partial monomials in the variables $x_1, \ldots, x_i$ and the exponents of x_{i+1} are known.

Theorem 3.2 Let $f \in \mathbb{K}[x_1, \ldots, x_n]$ be a t-sparse polynomial with $\deg_{x_i}(f) < d$. Then $O(ndt^2)$ requests to the black box are sufficient to reconstruct f in $O(nd^2t^3)$ (sequential) time.

PROOF: First, for every variable x_i, we determine the set S_i of exponents of x_i occurring in f. Rewrite f according to powers of x_i:

$$f(x_1, \ldots, x_n) = \sum_{j=0}^{d-1} x_i^j \cdot p_j^{(i)}(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n).$$

Then

$$S_i = \{x_i^j \mid p_j^{(i)} \not\equiv 0\}.$$

Since f is t-sparse, we conclude that $p_j^{(i)}$ is t-sparse. In order to apply the zero test to $P_j^{(i)}$, we have to determine the values $\mu_{j,k}^{(i)} = p_j^{(i)}(p_1^k, \ldots, p_{n-1}^k)$ with $0 \le k < t$. Let $\nu_{l,k}^{(i)}$ be defined by

$$\nu_{l,k}^{(i)} = f(p_1^k, \ldots, p_{i-1}^k, l, p_i^k, \ldots, p_{n-1}^k) \quad \text{for } 1 \le l \le d.$$

Then

$$\nu_{l,k}^{(i)} = \sum_{j=0}^{d-1} l^j \cdot p_j^{(i)}(p_1^k, \ldots, p_{n-1}^k) = \sum_{j=0}^{d-1} l^j \cdot \mu_{j,k}^{(i)} \quad \text{for } 1 \le l \le d$$

yielding the linear system of equations:

$$\begin{pmatrix} \nu_{1,k}^{(i)} \\ \vdots \\ \nu_{d,k}^{(i)} \end{pmatrix} = \underbrace{\begin{pmatrix} 1^0 & \ldots & 1^{d-1} \\ \vdots & & \vdots \\ d^0 & \ldots & d^{d-1} \end{pmatrix}}_{=: C} \cdot \begin{pmatrix} \mu_{0,k}^{(i)} \\ \vdots \\ \mu_{d-1,k}^{(i)} \end{pmatrix}.$$

C is a non-singular Vandermonde matrix, therefore the linear system of equations can be solved for any k and any i and we obtain all necessary values. We need $n \cdot d \cdot t$ requests to the black box to determine S_i for $1 \le i \le n$.

Let $\bar{S}_i$ be the set of the partial monomials occurring in f in the variables $x_1, \ldots, x_i$. It holds:

$$\bar{S}_1 = S_1.$$

We determine $\bar{S}_i$ from $\bar{S}_{i-1}$ and S_i for $i = 2, \ldots n$. Let $H_i = \bar{S}_{i-1} \times S_i$, i.e.

$$H_i = \{\alpha \cdot \beta \mid \alpha \in \bar{S}_{i-1}, \beta \in S_i\} =: \{t_{i,j}(x_1, \ldots, x_i)\}$$

with monomials $t_{i,j}$. The set H_i consists of potential candidates for $\bar{S}_i$. Since f is t-sparse, it holds $|\bar{S}_{i-1}| \le t$ and because of $\deg_{x_i}(f) < d$, it holds $|S_{i-1}| < d$. Therefore, the number of elements in H_i is bounded by

$$h_i := |H_i| \le t \cdot d$$

Rewrite f according to the elements in H_i:

$$f(x_1, \ldots, x_n) = \sum_{j=1}^{h_i} t_{i,j}(x_1, \ldots, x_i) \cdot p_{i,j}(x_{i+1}, \ldots, x_n).$$

Then

$$\bar{S}_i = \{t_{i,j} \mid p_{i,j} \not\equiv 0\}.$$

Furthermore the polynomials $p_{i,j}$ are t-sparse because f is t-sparse.

Like the procedure above, we eliminate those $t_{i,j}$ from H_i with $p_{i,j} \equiv 0$ with the help of the zero test given by Theorem 3.1. However, there is no black box for the polynomials $p_{i,j}$ explicitly given. They are to be constructed from the given black box for f.

Testing $p_{i,j}$ to zero for $j = 1, \ldots, h_i$ requires the values

$$\mu_{j,k}^{(i)} = p_{i,j}(p_1^k, \ldots, p_{n-i}^k) \quad \text{for } 0 \leq k < t.$$

Let

$$\nu_{l,k}^{(i)} = f(p_1^l, \ldots, p_i^l, p_1^k, \ldots, p_{n-i}^k) \quad \text{for } 0 \leq l < h_i.$$

Then

$$\begin{aligned} \nu_{l,k}^{(i)} &= \sum_{j=1}^{h_i} t_{i,j}(p_1^l, \ldots, p_i^l) \cdot p_{i,j}(p_1^k, \ldots, p_{n-i}^k) \\ &= \sum_{j=1}^{h_i} t_{i,j}(p_1, \ldots, p_i)^l \cdot \mu_{j,k}^{(i)} \quad \text{for } 0 \leq l < h_i. \end{aligned}$$

We obtain the following linear system of equations

$$\begin{pmatrix} \nu_{0,k}^{(i)} \\ \vdots \\ \nu_{h_i-1,k}^{(i)} \end{pmatrix} = \underbrace{\begin{pmatrix} t_{i,1}^0 & \cdots & t_{i,h_i}^0 \\ \vdots & & \vdots \\ t_{i,1}^{h_i-1} & \cdots & t_{i,h_i}^{h_i-1} \end{pmatrix}}_{=: \mathcal{T}} \cdot \begin{pmatrix} \mu_{1,k}^{(i)} \\ \vdots \\ \mu_{h_i,k}^{(i)} \end{pmatrix}$$

By (4), $\mathcal{T}$ is a non-singular Vandermonde matrix. Therefore the linear system of equations is solvable for any k and yields the required values.

For any H_i, $i = 2, \ldots, n$ we have to query the black box for values $\nu_{l,k}^{(i)}$ with $0 \leq l < h_i$ ($h_i \leq td$). If $i = n$, it suffices to execute the step for $k = 0$ because the coefficients c_i correspond to the coefficient polynomials.

All in all, the number of requests to the black box is ndt for the computation of the S_i, $(n-2)dt^2$ for the computation of the H_i for $i = 2, \ldots, n-1$ and dt for the computation of the c_i yielding a total number of $O(ndt^2)$ requests.

We have to solve $O(nt)$ linear systems of equations of dimension dt. Using Berlekamp-Massey algorithm ([Bl 83]), the interpolation algorithm takes $O(nd^2t^3)$ (sequential) time. □

Remark: For $i = 2, \ldots, n$ the set $\bar{S}_i$ is determined sequentially from $\bar{S}_{i-1}$ and S_i. Therefore the time complexity of a parallel implementation is at least polynomial in n. Hence, the enumeration technique used here is not applicative to obtain NC-algorithms. The logarithmic enumeration techniques used for the enumeration of all perfect matchings of a bipartite graph in Section 2.3 are used for interpolation by Singer, Grigoriev and Karpinski and lead to NC-algorithms even for interpolation over finite field (cf. Chapter 4.3).

3.2 Development of an NC-Algorithm

We present new techniques of Ben-Or and Tiwari ([BT 88]) minimizing the number of requests to the black box. These techniques are similar to those used by Clausen, Grabmeier and Karpinski (cf. Chapter 4.2). However, in contrast to [CGK 87] we obtain a first NC-interpolation algorithm because of the simpler arithmetic in fields of characteristic zero.

Theorem 3.3 Let $I\!K$ be an arbitrary field of characteristic zero and let $f \in I\!K[x_1, \ldots, x_n]$ be a t-sparse polynomial with $\deg(f) = d$. Then f can be reconstructed deterministically by $2t$ queries to the black box in $O(t^3 dn \log n)$ (sequential) time.

Let

$$f(x_1, \ldots, x_n) = \sum_{i=1}^{t} a_i \cdot t_i(x_1, \ldots, x_n)$$

with monomials $t_i(x_1, \ldots, x_n) = x_1^{\alpha_{i,1}}, \ldots, x_n^{\alpha_{i,n}}$, where $\alpha_{i,j} \in I\!N_0, a_i \in I\!K$. Let k be the (à priori unknown) exact number of monomials occurring in f.
Let p_j be the j-th prime. The black box is evaluated at the points

$$\nu_i = (p_1^i, \ldots, p_n^i)$$

with $0 \leq i < 2t$. Let $m_i = t_i(\nu_1)$. By this choice of evaluation points, the monomials of f are separated (cf. [GK 87]), i.e. $m_i \neq m_j$, for $i \neq j$.

The reconstruction of f involves two major computational steps: (1) determining the exponents $\alpha_{i,j}$ for $1 \leq j \leq n$ and $1 \leq i \leq k$ and (2) determining the coefficients a_i for $1 \leq i \leq k$.

If all the exponents $\alpha_{i,j}$ are known and therefore k and the monomials t_i occurring in f for $1 \leq i \leq k$ are determined, the coefficients c_i can be computed as follows. Let $v_r = f(\nu_r)$, then

$$v_r = \sum_{i=1}^{k} a_i \cdot t_i(\nu_r) = \sum_{i=1}^{k} a_i \cdot m_i^r \quad \text{where } 0 \leq r < k.$$

We obtain the linear system of equations

$$\begin{pmatrix} v_0 \\ \vdots \\ v_{k-1} \end{pmatrix} = \underbrace{\begin{pmatrix} m_1^0 & \ldots & m_k^0 \\ \vdots & & \vdots \\ m_1^{k-1} & \ldots & m_k^{k-1} \end{pmatrix}}_{=: \mathcal{M}} \cdot \begin{pmatrix} a_1 \\ \vdots \\ a_k \end{pmatrix}.$$

Since $m_i \neq m_j$ for $i \neq j$, $\mathcal{M}$ is a non-singular Vandermonde matrix. Hence the given linear system of equations has a unique solution.

In the following, it remains to handle the first step, namely the determination of the exponents $\alpha_{i,j}$. This is based on a technique for decoding BCH-codes ([Bl 83],[LN 86]).

First we assume that t corresponds to the exact number k of monomials. The general case $k \leq t$ is reduced to this case by determining k.

In order to determine the exponents $\alpha_{i,j}$, we determine m_i for $1 \leq i \leq t$ and factor it into prime powers. The factorization of m_i is simple because the prime factors of m_i are just $p_1, \ldots, p_n$.

This fast factorization points out the difference of the complexity compared to the analogous algorithm [CGK 87] for finite fields. There the computation of discrete logarithms is required (for which no efficient deterministic algorithm is known).

The core of the problem is the computation of the m_i's. We define a polynomial $\Lambda \in \mathbb{Z}(z)$, whose roots are just the m_i's:

$$\Lambda(z) := \prod_{i=1}^{t}(z - m_i) = \sum_{i=0}^{t} \lambda_i z^i, \quad \lambda_t = 1.$$

In the following we derive linear systems of equations to obtain the coefficients λ_i (thereby the polynomial Λ is completely determined) and then determine the zeros of Λ.

Since $\Lambda(m_i) = 0$, $a_i \cdot m_i^l \cdot \Lambda(m_i) = 0$ for arbitrary l. Then:

$$\begin{aligned} 0 &= a_i \cdot (\lambda_0 m_i^l + \lambda_1 m_i^{l+1} + \cdots \lambda_t m_i^{l+t}) \\ &= \sum_{i=1}^{t} a_i \cdot (\lambda_0 m_i^l + \lambda_1 m_i^{l+1} + \cdots + \lambda_t m_i^{l+t}) \\ &= \lambda_0 \sum_{i=1}^{t} a_i m_i^l + \lambda_1 \sum_{i=1}^{t} a_i m_i^{l+1} + \cdots + \lambda_t \sum_{i=1}^{t} a_i m_i^{l+t} \\ &= \lambda_0 v_l + \lambda_1 v_{l+1} + \cdots + \lambda_t v_{l+t}. \end{aligned}$$

With $\lambda_t = 1$, we conclude $-v_{l+t} = \lambda_0 v_l + \lambda_1 v_{l+1} + \cdots + \lambda_t v_{l+t-1}$ yielding the following linear system of equations:

$$\underbrace{(v_{i+j})_{0 \le i,j < t}}_{=: \mathcal{V}} \cdot (\lambda_i)_{0 \le i < t} = (-v_{t+l})_{0 \le l < t} \cdot \tag{6}$$

Consider the matrix $\mathcal{V}$ ($\mathcal{V}$ is a *Toeplitz*-Matrix):

$$v_{i+j} = \sum_{r=1}^{t} a_r \cdot m_r^{i+j} = \sum_{r=1}^{t} m_r^i \cdot a_r \cdot m_r^j$$

$$\Longrightarrow (v_{i+j})_{0 \le i,j < t} = \left(m_r^i\right)^t_{\substack{0 \le i < t \\ 0 \le r < t}} \cdot D_A \cdot \left(m_r^j\right)_{\substack{0 \le j < t \\ 0 \le r < t}}$$

where $D_A = \operatorname{diag}((a_i)_{0 \le i < t})$.

Since $a_i \neq 0$ for $0 \le i < t$, the rank of $\mathcal{V}$ equals t. Therefore $\mathcal{V}$ is non-singular and the linear system of equations (6) has a unique solution λ_i, $0 \le i < t$ (the coefficients of the polynomial Λ). We used $2t$ evaluations of f to obtain the values $v_0, \ldots, v_{2t-1}$.

In the general case $k < t$, we have to determine k. For this purpose let $\mathcal{V}$ be defied as above and $\mathcal{V}_l$ the $(l \times l)$-matrix obtained from the first l rows and the first l columns of $\mathcal{V}$. For $k < t$, the a_i are not necessaryly non-zero, as a consequence the corresponding matrix D_A has not full rank. However, the following lemma holds:

Lemma 3.4

$$\det(\mathcal{V}_l) = \begin{cases} \displaystyle\sum_{\substack{S \subset \{1, \ldots, k\} \\ |S| = l}} \left(\prod_{i \in S} a_i \cdot \prod_{\substack{i,j \in S \\ i > j}} (m_i - m_j)^2 \right) & \text{if } l \le k \\ 0 & \text{if } l > k \end{cases}$$

Corollary 3.5 Let k be the number of non-zero coefficients of f and $k \leq t$. Then

$$k = \max\{j \leq t \mid \mathcal{V}_j \text{ non-singular}\}.$$

By Corollary 3.5 we obtain k by computing the rank of $\mathcal{V}$:

$$k = \text{rank}(\mathcal{V}).$$

In summary, we have the following algorithm for the interpolation of a t-sparse polynomial $f \in I\!K[x_1, \ldots, x_n]$. The correctness of the algorithm follows from above.

Interpolation Algorithm [BT 88]

Input: Black box for $f \in I\!K[x_1, \ldots, x_n]$, t-sparse.

Output: The monomials occurring in f and the corresponding coefficients.

Step 1: Query the black box at points $\nu_i = p_1^i, \ldots, p_n^i$. Let $v_i = f(\nu_i)$, $0 \leq i < 2t$.

Step 2: Let $\mathcal{V} = (v_{i+j})_{0 \leq i,j < t}$. Determine $k = \text{rank}(\mathcal{V})$. Let $\bar{\mathcal{V}} = (v_{i+j})_{0 \leq i,j < k}$.

Step 3: Let $s := (s_0, \ldots, s_{k-1})^t$ where $s_i := v_{i+k}$. Solve the linear system of equations

$$\bar{\mathcal{V}} \cdot \lambda = -s, \qquad \lambda := (\lambda_0, \ldots, \lambda_{k-1})^t.$$

Step 4: Determine the roots $m_1, \ldots, m_k$ of the polynomial

$$\Lambda(z) = z^k + \sum_{i=0}^{k-1} \lambda_i z^i.$$

Step 5: Compute the prime factorization of

$$m_i = p_1^{\alpha_{i,1}} \cdot \ldots \cdot p_n^{\alpha_{i,n}} \qquad \text{for } 1 \leq i \leq k.$$

Step 6: Let $M = \left(m_j^i\right)_{\substack{0 \leq i < k \\ 1 \leq j \leq k}}$ and $v = (v_0, \ldots, v_{k-1})^t$. Solve the linear system of equations

$$M \cdot a = v, \qquad a := (a_1, \ldots, a_k)^t.$$

Step 7: Output: $(a_i, (\alpha_{i,1}, \ldots, \alpha_{i,n}))_{i=1,\ldots,k}$.

It remains to settle the determination of the roots of Λ in Step 4. It holds:

$$\Lambda(z) = \prod_{i=1}^{k}(z - m_i) \in \mathbb{Z}[z], \text{ therefore}$$

$$\deg(\Lambda) = k \leq t, \text{ and}$$

$$m_i = p_1^{\alpha_{i,1}} \cdot \ldots \cdot p_n^{\alpha_{i,n}} \leq (p_1 \cdot \ldots \cdot p_n)^d.$$

By the prime theorem we have $p_n = O(n \log n)$, furthermore the product of all primes less than l is at most $O(2^l)$, hence

$$m_i \leq 2^{O(nd \log n)}.$$

With this upper bound for the absolute value of the roots of Λ the algorithm of Loos ([Lo 83]) determines the set of roots in $O(t^3 dn \log n)$ time.

Step 2 and 3 can be performed in $O(t^2)$ time using Berlekamp-Massey algorithm ([Bl 83]). The same complexity bound is valid for Step 6 applying Zippel's algorithm for the inversion of non-singular Vandermonde matrices ([Zi 88]). This completes the proof of Theorem 3.3. □

The given algorithm is non-adaptive, because the choice of the evaluation points is fixed; they are not determined successively depending upon the values attained at previous evaluations. When restricted to non-adaptive algorithms the given algorithm is optimal concerning the number of evaluation points:

Theorem 3.6 Any non-adaptive algorithm reconstructing t-sparse polynomials in n variables requires at least $2t$ evaluations of the polynomial.

PROOF: We consider the univariate case. Assume there is a non-adaptive algorithm $\mathcal{A}$ using only $l < 2t$ evaluations. Suppose the evaluation points are $\nu_1, \ldots, \nu_l$. Consider the univariate polynomial p:

$$p(x) = \prod_{i=1}^{l}(x - \nu_i) = \sum_{i=0}^{l} a_i x^i.$$

p has at most $l + 1 < 2t + 1$ non-zero coefficients. Let

$$p_1(x) = \sum_{i=0}^{\lfloor \frac{l}{2} \rfloor} a_i x^i \quad \text{and} \quad p_2(x) = -\sum_{i=\lfloor \frac{l}{2} \rfloor + 1}^{l} a_i x^i.$$

p_1, p_2 have at most $\lfloor \frac{l}{2} \rfloor + 1$ non-zero coefficients. Since $l + 1 < 2t + 1$, p_1 and p_2 are t-sparse. Furthermore $p(x) = p_1(x) - p_2(x)$, hence $0 = p(\nu_i) = p_1(\nu_i) - p_2(\nu_i)$. Therefore p_1 and p_2 coincide at the points $\nu_i, i = 1, \ldots, l$. Hence, the algorithm $\mathcal{A}$ cannot distinguish p_1 and p_2. □

It should be pointed out that the algorithm [BT 88] does not need the à-priori knowledge of d, d influences only the complexity of the algorithm. The problem of interpolating a t-sparse polynomial p in deterministic polynomial time without the à-priori knowledge of t is not solvable because it can be proved that it is impossible to obtain an upper bound t for the number of occurring monomials in polynomial time. However, if it is guaranteed that all coefficients of p are positive then, the algorithm [BT 88] yields a reconstruction of p in polynomial time.

Lemma 3.7 Let a t-sparse polynomial $f \in I\!K[x_1, \ldots, x_n]$ with $\deg(f) < d$ and positive coefficients be given by a black box. Suppose that t and d are unknown. Then there exists a polynomial (in n, t, d) deterministic algorithm reconstructing f.

PROOF: Apply the algorithm [BT 88] for $t = 1, 2, \ldots$. The stopping rule is given by Lemma 3.4 since if all coefficients a_i are positive, then $\det(V_l) \neq 0$ for $l \leq k$. Hence $\det(\mathcal{V}_l) \neq 0$ and $\det(\mathcal{V}_{l+1}) = 0$ implies $t = l$. □

However, the stopping rule is not correct in general:

Lemma 3.8 If the rank of $(\mathcal{V}_l) = k$, then the number of the monomials occurring in f is either exactly k or at least $2l - k$.

4 Polynomial Interpolation over Finite Fields

4.1 Bounds on the Number of Evaluation Points

An important aspect for the reconstruction of polynomials and for the test of polynomials to zero, is the necessary number of requests to the black box.

The main reason making interpolation over finite fields $\mathrm{GF}(q)$ harder than interpolation over fields of characteristic zero, is the limited number of elements in $\mathrm{GF}(q)$. Thus, the information concluded from a function value in $\mathrm{GF}(q)$ is worth less concerning the reconstruction of the polynomial. This yields a large number of evaluation points required for interpolation.

Assume we are enabled to use some finite extension $\mathrm{GF}(q^m)$ of $\mathrm{GF}(q)$. That is, we look at a polynomial in $\mathrm{GF}(q)[x_1,\ldots,x_n]$ as a polynomial in $\mathrm{GF}(q^m)[x_1,\ldots,x_n]$ and assume the black box to be capable of evaluating the polynomial in $\mathrm{GF}(q^m)$. Then, more information can be extracted from a function value in $\mathrm{GF}(q^m)$ yielding a smaller number of necessary evaluation points. However, we have to take into account that the arithmetic in $\mathrm{GF}(q^m)$ is harder than in $\mathrm{GF}(q)$; the larger the degree of the extension the more expensive the arithmetic.

To obtain an efficient reconstruction algorithm, we have to compromise according to the degree of the extension, such that the number of evaluation points is yet small and the arithmetic in the field extension can still be performed quickly.

This section gives upper and lower bounds on the number of evaluation points using techniques suggested in [CDGK 88].

The finite field $\mathrm{GF}(q)$ with $q = p^m$ is isomorphic to the splitting field of $x^q - x$ over $\mathrm{GF}(p)$. Therefore, the ring of polynomial functions over $\mathrm{GF}(q)$ in n variables is isomorphic to the polynomial ring $\mathrm{GF}(q)[x_1,\ldots,x_n]$ modulo the ideal generated by $(x_1^q - x_1,\ldots,x_n^q - x_n)$. An element in $\mathrm{GF}(q)[x_1,\ldots,x_n]$ can be identified with an element $f \in \mathrm{GF}(q)[x_1,\ldots,x_n]$ with $\deg_{x_i}(f) < q$. Therefore, we assume that the black box represents a polynomial $f \in \mathrm{GF}(q)[x_1,\ldots,x_n]$ with $\deg_{x_i}(f) < q$ for $1 \le i \le n$.

The general problem, where a black box for $f \in \mathrm{GF}(q)[x_1,\ldots,x_n]$ is given and we have to construct efficiently a black box for $f \bmod (x_1^q - x_1,\ldots,x_n^q - x_n)$, is non-trivial for the case when the black box is evaluated in some proper extension of $\mathrm{GF}(q)$. The problem of lifting the black box from $\mathrm{GF}(q)$ to $\mathrm{GF}(q^s)$ efficiently is still unsolved.

Let $\mathcal{P}_t^n(q)$ be the set of t-sparse polynomials from $\mathrm{GF}(q)[x_1,\ldots,x_n]$ with $\deg_{x_i}(f) < q$ for $1 \le i \le n$. We are interested in minimal test sets separating polynomials in $\mathcal{P}_t^n(q)$ and in minimal test sets distinguishing polynomials in $\mathcal{P}_t^n(q)$ from the zero-polynomial.

Let $\mathcal{P}$ be the set of polynomials in $\mathcal{P}_t^n(q)$ considered as mappings from $[\mathrm{GF}(q^m)]^n$ to $\mathrm{GF}(q^m)$. Let

$$\mathcal{B}_t^n(q,m) \quad := \quad \{\, \mathcal{B} \;\mid\; \mathcal{B} \subseteq \mathrm{GF}(q^m);\quad \forall f,g \in \mathcal{P} \quad \exists b \in \mathcal{B} \quad (f(b) \neq g(b)) \,\}$$

be the set of all test sets in $\mathrm{GF}(q^m)$ separating polynomials in $\mathcal{P}_t^n(q)$ and let

$$\mathcal{A}_t^n(q,m) \quad := \quad \{\, \mathcal{A} \;\mid\; \mathcal{A} \subseteq \mathrm{GF}(q^m);\quad \forall g \in \mathcal{P} \setminus \{0\} \quad \exists a \in \mathcal{A} \quad (g(a) \neq 0) \,\}$$

be the sets of all test sets in $\mathrm{GF}(q^m)$ distinguishing polynomials in $\mathcal{P}_t^n(q)$ from the zero-polynomial. These sets are closely related:

Lemma 4.1 $\mathcal{B}_t^n(q,m) = \mathcal{A}_{2t}^n(q,m)$

PROOF:

'$\supseteq$' Let $B \in \mathcal{B}_t^n(q,m)$ and $h \in \mathcal{P}_{2t}^n(q), h \not\equiv 0$. Then there are polynomials $f, g \in \mathcal{P}_t^n(q)$ so that $h = f - g$. While $h \not\equiv 0$ there exists some $b \in B$ with $f(b) \neq g(b)$ therefore $h(b) = f(b) - g(b) \neq 0$, too. Hence $B \in \mathcal{A}_{2t}^n(q,m)$.

'$\subseteq$' Let $A \in \mathcal{A}_{2t}^n(q,m)$ and $f, g \in \mathcal{P}_t^n(q)$ with $f \not\equiv g$. Then $h = f - g \in \mathcal{P}_{2t}^n(q)$ and $h \not\equiv 0$. Therefore there exists some $a \in A$ where $h(a) \neq 0$, thereby $f(a) \neq g(a)$. Hence $A \in \mathcal{B}_t^n(q,m)$. □

We construct upper and lower bounds for

$$c_t^n(q,m) \quad := \quad \min\{\, \#A \;\mid\; A \in \mathcal{A}_t^n(q,m) \,\},$$

the minimal number of evaluation points required to separate a polynomial in $\mathcal{P}_t^n(q)$ from the zero-polynomial. The corresponding bounds for $\mathcal{B}_t^n(q,m)$ are obtained with Lemma 4.1.

4.1.1 Upper Bounds

In Section 4.2 we present a zero test for a t-sparse polynomial $f \in \mathrm{GF}(q)[x_0, \ldots, x_{n-1}]$ with $\deg_{x_i}(f) < q$ for $0 \leq i < n$, using only a small number of evaluation points. This zero test works in the field extension $\mathrm{GF}(q^n)$ of $\mathrm{GF}(q)$ and requires the function values

$$f_i := f(\omega^{iq^0}, \omega^{iq^1}, \ldots, \omega^{iq^{n-1}}) \quad \text{for} \quad 0 \leq i < t, \quad q \not| i \quad \text{and} \quad f(0, \ldots, 0),$$

where ω is a primitive element in $\mathrm{GF}(q^m)$. We denote this set by A_t^n:

$$\begin{aligned} A_t^n \quad &:= \quad \{\, a_i = (a_{ij})_{0 \leq j < n} \in \mathrm{GF}(q^n)^n \;\mid\; a_{ij} = \omega^{iq^j}, \quad 0 \leq i < t, \quad q \not| i \,\} \\ &\cup \quad \{\, (0, \ldots, 0) \in \mathrm{GF}(q^n)^n \,\}. \end{aligned} \tag{7}$$

Theorem 4.12 implies $A_t^n \in \mathcal{A}_t^n(q,n)$ yielding an upper bound for $c_t^n(q,n)$:

$$c_t^n(q,n) \leq \#A_t^n = 1 + t - \left\lfloor \frac{t-1}{q} \right\rfloor. \tag{8}$$

However, we are also interested in upper bounds for $c_t^n(q,m)$ for $1 \leq m < n$.

As mentioned above we know sets in $\mathcal{A}_\tau^m(q,m)$, i.e. zero test sets in $\mathrm{GF}(q^m)$ for polynomials f_τ over $\mathrm{GF}(q)$ in m variables, f_τ τ-sparse, for arbitrary τ. By splitting the problem into the known zero test sets $A_\tau^m \in \mathcal{A}_\tau^m(q,m)$, we construct a zero test set $T_t^n(q,m) \in \mathcal{A}_t^n(q,m)$ for a polynomial $f \in \mathrm{GF}(q)[x_0, \ldots, x_{n-1}]$.

Let $f \in \mathrm{GF}(q)[x_0, \ldots, x_{n-1}]$, t-sparse and $n = n_1 + n_2$. We assume that zero test sets in $\mathrm{GF}(q^m)$ for polynomials in n_1, n_2 variables are known, i.e. $A_{t_1}^{n_1} \in \mathcal{A}_{t_1}^{n_1}(q,m)$, $A_{t_2}^{n_2} \in \mathcal{A}_{t_2}^{n_2}(q,m)$, for arbitrary t_1, t_2. From these sets we construct a zero test set $A_t^n \in \mathcal{A}_t^n(q,m)$.

Lemma 4.2 Let $n = n_1 + n_2$, $A_{t_1}^{n_1} \in \mathcal{A}_{t_1}^{n_1}(q,m)$ and $A_{t_2}^{n_2} \in \mathcal{A}_{t_2}^{n_2}(q,m)$ for all $t_1 \cdot t_2 \leq t$. Then

$$\bigcup_{t_1 \cdot t_2 \leq t} A_{t_1}^{n_1} \times A_{t_2}^{n_2} \in \mathcal{A}_t^n(q,m).$$

PROOF: Let $f \in \mathrm{GF}(q)[x_0, \ldots, x_{n-1}]$ be a t-sparse polynomial. We have to show:

$$f \not\equiv 0 \implies \exists\, (a^{(1)}, a^{(2)}) \in \bigcup_{t_1 \cdot t_2 \le t} A_{t_1}^{n_1} \times A_{t_2}^{n_2} : f(a^{(1)}, a^{(2)}) \neq 0.$$

Let $f \not\equiv 0$. Rewrite f as

$$f = \sum_{i=1}^{\tau_1} \beta_i(x_{n_1}, \ldots, x_{n-1}) \cdot x_0^{\alpha_{0,i}} \cdots x_{n_1-1}^{\alpha_{n_1-1,i}}$$

with coefficients $\beta_i \in \mathrm{GF}(q)[x_{n_1}, \ldots, x_{n-1}]$. Trivially, $\tau_1 < t$. Furthermore, there is some j, such that $\beta_j \not\equiv 0$ is τ_2-sparse with $\tau_2 \le \lfloor t/\tau_1 \rfloor$ (or $\tau_1 \tau_2 \le t$). Hence, there exists $a^{(2)} \in A_{\tau_2}^{n_2}$ with $\beta_j(a^{(2)}) \neq 0$. Let $f_1 = f(x_0, \ldots, x_{n_1-1}, a^{(2)})$. f_1 is τ_1-sparse and $f_1 \not\equiv 0$ since $\beta_j(a^{(2)}) \neq 0$. Hence, there exists $a^{(1)} \in A_{\tau_1}^{n_1}$ with $f_1(a^{(1)}) \neq 0$, i.e. $f(a^{(1)}, a^{(2)}) \neq 0$. □

Lemma 4.2 implies

Corollary 4.3 Let $\pi = (\pi_0, \ldots, \pi_{s-1})$ be an arbitrary partition of n ($\pi \models n$). Let $A_\tau^\pi \in \mathcal{A}_\tau^\pi(q, m)$ for $\tau \le t$. Then

$$\bigcup_{\tau_0 \cdot \tau_1 \cdot \ldots \cdot \tau_{s-1} \le t} A_{\tau_0}^{\pi_0} \times A_{\tau_1}^{\pi_1} \times \cdots \times A_{\tau_{s-1}}^{\pi_{s-1}} \in \mathcal{A}_t^n(q, m).$$

We use this corollary to construct a zero test set $T_t^n(q, m) \in \mathcal{A}_t^n(q, m)$. By Theorem 4.12 zero test sets $A_\tau^m \in \mathcal{A}_t^m(q, m)$ for arbitrary τ are known. Therefore we consider the partition

$$\pi = (\underbrace{m, \ldots, m, m_1}_{s = \lceil \frac{n}{m} \rceil \text{ components}}) \quad \text{where } m_1 \le m.$$

Then

$$\bar{T}_t^n(q, m) = \bigcup_{\tau_0 \cdot \tau_1 \cdot \ldots \cdot \tau_{s-1} \le t} A_{\tau_0}^m \times \cdots \times A_{\tau_{s-2}}^m \times A_{\tau_{s-1}}^{m_1} \in \mathcal{A}_t^n(q, m).$$

We choose $A_{\tau_{s-1}}^{m_1}$ to be the set $\{(a_0, \ldots, a_{m_1-1}) | (a_0, \ldots, a_{m_1-1}, a_{m_1} \ldots, a_{m-1}) \in A_{\tau_{s-1}}^m\}$ because a polynomial in m_1 variables can be interpreted as a polynomial in m variables.

Let ω be a primitive element in $\mathrm{GF}(q^m)$. Hence, by equation (7)

$$\begin{aligned} A_{\tau_\mu}^m \quad := \quad & \{\, a_i = (a_{ij})_{0 \le j < m} \in \mathrm{GF}(q^m)^m \mid a_{ij} = \omega^{i_\mu q^j},\ 0 \le i_\mu < \tau_\mu, \\ & \qquad i_\mu = \begin{cases} i & \text{if } q \nmid i \\ 0 & \text{otherwise} \end{cases} \,\} \\ \cup \quad & \{\, (0, \ldots, 0) \in \mathrm{GF}(q^n)^n \,\} \end{aligned} \tag{9}$$

To indicate $\bar{T}_t^n(q, m)$ we have to form all pairs of elements $a_\mu \in A_{\tau_\mu}^m$, $\mu = 1, \ldots, s$ for all combinations $\tau_0 \cdot \tau_1 \cdot \ldots \cdot \tau_{s-1} \le t$.

If $(0, \ldots, 0) \in A_{\tau_\mu}^m$ represents a component of an element in $\bar{T}_t^n(q, m)$, the corresponding τ_μ is at *least* 2 (since A_1^m does not contain $(0, \ldots, 0)$); if $a_\mu \in A_{\tau_\mu}^m \setminus \{0\}$ is chosen, i.e. $a_\mu = \omega^{i_\mu q^j}$, then τ_μ is at *least* $i_\mu + 1$, because $i_\mu < \tau_\mu$.

Based on this consideration, we can avoid the term $\tau_0 \cdot \tau_1 \cdot \ldots \cdot \tau_{s-1} \leq t$ in $\bar{T}_t^n(q,m)$, by constructing the set $T_t^n(q,m) \supseteq \bar{T}_t^n(q,m)$:

$$\begin{aligned} T_t^n(q,m) \;=\; \{\; & a = (a_\nu)_{0\leq j<n} \in \mathrm{GF}(q^m)^n \;: \\ & \nu = \mu \cdot m + j \quad (\ \mu \text{ is index of the component } A^m_{\tau_\mu}\ ) \\ & a_\nu = \epsilon_\mu \omega^{i_\mu q^j}, \quad (\ \text{by equation 9}\ ) \\ & \epsilon_\mu \in \{0,1\}, \qquad (\ \epsilon_\mu = 0 \text{ if } a_\mu = 0 \in A^m_{\tau_\mu}\ ) \\ & 2^{\#\{\mu:\epsilon_\mu=0\}} \cdot \prod_{\{\mu\ \epsilon_\mu\neq 0\}} (1+i_\mu) \leq t\ \}, \end{aligned}$$

where ω is a primitive element in $\mathrm{GF}(q^m)$. Since $T_t^n(q,m) \supseteq \bar{T}_t^n(q,m)$ (we dropped the requirement $q \nmid i$), it holds that $T_t^n(q,m) \in \mathcal{A}_t^n(q,m)$. We conclude

Theorem 4.4 Let $f \in \mathrm{GF}(q)[x_0,\ldots,x_{n-1}]$ be a t-sparse polynomial, $t \geq 2$, where $\deg_{x_i}(f) < q$ for every i. Then

$$f \equiv 0 \quad \Longleftrightarrow \quad f(a) = 0 \text{ for every } a \in T_t^n(q,m).$$

Estimating the number of elements in $T_t^n(q,m)$ yields upper bounds for $c_t^n(q,m)$.

For $m = 1$ we have:

$$\begin{aligned} T_t^n(q,1) \;=\; \{\; & a = (a_\mu)_{0\leq \mu<n} \in \mathrm{GF}(q)^n \;: \\ & a_\mu = \epsilon_\mu \omega^{i_\mu}, \quad 0 \leq i_\mu < t \\ & \epsilon_\mu \in \{0,1\}, \\ & 2^{\#\{\mu:\epsilon_\mu=0\}} \cdot \prod_{\{\mu\cdot\epsilon_\mu\neq 0\}} (1+i_\mu) \leq t\ \}. \end{aligned}$$

Let ω be a primitive element in $\mathrm{GF}(q)$, ω generates $\mathrm{GF}(q) \setminus \{0\}$.
Let $\mathrm{GF}(q) = \{a^{(0)},\ldots,a^{(q-1)}\}$ with $a^{(0)} = 0$ and $a^{(i+1)} = \omega^i$, $0 \leq i < q-2$. Hence

$$\begin{aligned} T_t^n(q,1) \;=\; \{\; & a = (a_\mu)_{0\leq j<n} \in \mathrm{GF}(q)^n \;: \\ & a_\mu = 0 \text{ or } a_\mu = \omega^i,\ 0 \leq i < q-1 \quad \text{where} \\ & 2^{\#\{\mu:a_\mu=0\}} \cdot \prod_{\{\mu\ a_\mu=\omega^i\}} (1+i) \leq t\ \}. \end{aligned}$$

Let κ_i be the number of occurrences of $a^{(i)} \in \mathrm{GF}(q)$ in $a = (a_0,\ldots,a_{n-1}) \in T_t^n(q,1)$.

$$\kappa_i := \#\{\ a_\mu \mid a = (a_\mu)_{0\leq\mu<n} : a_\mu = a^{(i)}\ \} \quad 1 \leq i < q.$$

It holds $\sum_{i=0}^{q-1} \kappa_i = n$; therefore $\kappa = (\kappa_0,\ldots,\kappa_{q-1})$ is a partition of n.

We calculate $\#T_t^n(q,1)$ from the number of possibilities to distribute κ_i elements $a^{(i)}$ among the n components of $a = (a_0,\ldots,a_{n-1}) \in T_t^n(q,1)$ where

$$\overbrace{2^{\#\{\mu\,:\,a_\mu=0\}}}^{=\kappa_0} \cdot \underbrace{\prod_{\{\mu\,:\,a_\mu=\omega^i\}} (1+i)}_{=\kappa_{i-1}} \leq t$$

$$\begin{aligned}
&\iff 2^{\kappa_0} \cdot \prod_{j=1}^{\kappa_1}(0+1) \cdot \prod_{j=1}^{\kappa_2}(1+1) \cdot \prod_{j=1}^{\kappa_3}(2+1) \cdot \ldots \cdot \prod_{j=1}^{\kappa_{q-1}}(q-2+1) \leq t \\
&\iff 2^{\kappa_0} \cdot 1^{\kappa_1} \cdot 2^{\kappa_2} \cdot 3^{\kappa_3} \cdot \ldots \cdot (q-1)^{\kappa_{q-1}} \leq t \\
&\iff 2^{\kappa_0+\kappa_2} \cdot 3^{\kappa_3} \cdot \ldots \cdot (q-1)^{\kappa_{q-1}} \leq t.
\end{aligned}$$

Let $\#T_t^n(q,1) =: \Phi(n,t,q)$. Then

$$\begin{aligned}
\Phi(n,t,q) &= \sum_{\substack{\kappa=(\kappa_0,\ldots,\kappa_{q-1})\models n \\ 2^{\kappa_0+\kappa_2}3^{\kappa_3}\ldots(q-1)^{\kappa_{q-1}}\leq t}} \binom{n}{\kappa_0} \cdot \binom{n-\kappa_0}{\kappa_1} \cdot \binom{n-\kappa_0-\kappa_1}{\kappa_2} \cdot \ldots \cdot \binom{\kappa_{q-1}}{\kappa_{q-1}} \\
&= \sum_{\substack{\kappa=(\kappa_0,\ldots,\kappa_{q-1})\models n \\ 2^{\kappa_0+\kappa_2}3^{\kappa_3}\ldots(q-1)^{\kappa_{q-1}}\leq t}} \frac{n!}{\kappa_0! \cdot \kappa_1! \cdot \ldots \cdot \kappa_{q-1}!} \\
&= \sum_{\substack{\kappa=(\kappa_0,\ldots,\kappa_{q-1})\models n \\ 2^{\kappa_0+\kappa_2}3^{\kappa_3}\ldots(q-1)^{\kappa_{q-1}}\leq t}} \binom{n}{\kappa}.
\end{aligned}$$

Similarly, we can estimate $T_t^n(q,m)$ for $2 \leq m < n$ to obtain an upper bound for $c_t^n(q,m)$.

An element $a \in T_t^n(q,m)$ consists of $\lceil \frac{n}{m} \rceil$ components a_μ with:

$$\begin{aligned}
&a_\mu = (a_{\mu r})_{0 \leq r < m \text{ resp. } m_1} : a_{\mu r} = \omega^{i_\mu q^r}, \quad \text{for a } i_\mu \text{ with } 0 \leq i_\mu < t,\ q \nmid i_\mu \\
\text{or} \quad &a_\mu = (0,\ldots,0) \in \mathrm{GF}(q^m)^m \text{ resp. } \mathrm{GF}(q^m)^{m_1}
\end{aligned}$$

Let ω be a primitive element in $\mathrm{GF}(q^m)$. ω generates $\mathrm{GF}(q^m) \setminus \{0\}$, therefore we can substitute $a_{\mu r}$ by $a_{\mu r} = \omega^{iq^r}$ with $0 \leq i < q^m - 1$. The component a_μ is determined by the choice of i. Let κ_i be defined as above by

$$\begin{aligned}
\kappa_{i+1} &:= \#\{\, a_i = (\omega^{iq^r})_{0 \leq r < m \text{ resp. } m_1} : \\
&\qquad a_i \text{ is component of } a \in T_t^n(q,m)\,\} \quad 0 \leq i < q^m - 1 \\
\text{and} \quad \kappa_0 &:= \#\{\, a^{(0)} = (0,\ldots,0) \in \mathrm{GF}(q^m)^m (\text{resp. } \mathrm{GF}(q^m)^{m_1}) : \\
&\qquad a^{(0)} \text{ is component of } a \in T_t^n(q,m)\,\}
\end{aligned}$$

with $\sum_{i=0}^{q-1} \kappa_i = \lceil \frac{n}{m} \rceil$, because $a \in T_t^n(q,m)$ consists of $\lceil \frac{n}{m} \rceil$ components a_i.

Then

$$\#T_t^n(q,m) \leq \sum_{\substack{\kappa=(\kappa_0,\ldots,\kappa_{q^m-1})\models \lceil \frac{n}{m} \rceil \\ 2^{\kappa_0+\kappa_2}3^{\kappa_3}\ldots(q^m-1)^{\kappa_{q^m-1}}\leq t}} \binom{\lceil \frac{n}{m} \rceil}{\kappa} = \Phi(\lceil \frac{n}{m} \rceil, k, q^m).$$

In summary, we have proven:

Corollary 4.5 Let

$$\Phi(n,k,q) := \sum_{\substack{\kappa=(\kappa_0,\ldots,\kappa_{q-1})\models n \\ 2^{\kappa_0+\kappa_2}3^{\kappa_3}\ldots(q-1)^{\kappa_{q-1}}\leq t}} \binom{n}{\kappa}.$$

Then

$$c_t^n(q,m) \leq \Phi(\lceil \frac{n}{m} \rceil, k, q^m).$$

In the following, two cases are studied where lower and upper bounds coincide.

Consider polynomials over GF(2), i.e. boolean expressions over $\{\wedge, \otimes, 0, 1\}$ (*RSE-expressions*).

Corollary 4.6 Let $f \in \mathrm{GF}(2)[x_0, \ldots, x_{n-1}]$ be a t-sparse polynomial with $\deg_{x_i}(f) < 2$, $0 \le i < n$. Then

$$f \equiv 0 \quad \Longleftrightarrow \quad f(a) = 0 \text{ for every } a \in A_t = T_t^n(2,1).$$

A_t consists of all elements in $\mathrm{GF}(2)^n$ having at most $\lfloor \log_2 t \rfloor$ zero components. Then

$$c_t^n(2,1) \le \sum_{i=0}^{\lfloor \log_2 t \rfloor} \binom{n}{i}.$$

PROOF:

$$\begin{aligned} A_t &= \{\ a = (a_\mu)_{0 \le \mu < n} \in \mathrm{GF}(2)^n : \\ &\qquad a_\mu = 0 \text{ or } a_\mu = 1 \text{ with} \\ &\qquad 2^{\#\{\mu : a_\mu = 0\}} \cdot \prod_{\{\mu\ a_\mu = 1\}} 1 \le t\ \} \\ &= \{\ a = (a_\mu)_{0 \le \mu < n} \in \mathrm{GF}(2)^n : \\ &\qquad a_\mu = 0 \text{ or } a_\mu = 1 \text{ with} \\ &\qquad \#\{\mu : a_\mu = 0\} \le \lfloor \log_2 t \rfloor\ \}. \end{aligned}$$

Therefore A_t consists of all elements in $\mathrm{GF}(2)^n$ having at most $\lfloor \log_2 t \rfloor$ zero components. Furthermore

$$c_t^n(2,1) \le \Phi(n,k,2) = \sum_{\substack{\kappa = (\kappa_0, \kappa_1) \models n \\ 2^{\kappa_0} \le t}} \frac{n!}{\kappa_0! \cdot \kappa_1!} = \sum_{\kappa_0 = 0}^{\lfloor \log_2 t \rfloor} \binom{n}{\kappa_0}.$$

□

Next we consider binomials:

Corollary 4.7 Let ω be a primitive element in $\mathrm{GF}(q)$. The set

$$\{(1, \ldots, 1)\} \cup \{\ a \in \mathrm{GF}(q)^n : a_\nu = \begin{cases} 0 \text{ or } \omega & \text{for exactly one } \nu \\ 1 & \text{otherwise} \end{cases}\ \}$$

is a zero test set for a binomial. Then

$$c_2^n(q,1) \le \begin{cases} 1+n & \text{if } q = 2 \\ 1+2n & \text{if } q \ne 2 \end{cases}$$

PROOF:

$$\begin{aligned} T_2^n(q,2) &= \{\ a = (a_\mu)_{0 \le \mu < n} \in \mathrm{GF}(q)^n : \\ &\qquad a_\mu = 0 \text{ or } a_\mu = \omega^i,\ 0 \le i < q-1 \text{ with} \\ &\qquad \underbrace{2^{\#\{\mu : a_\mu = 0\}}}_{=\alpha} \cdot \underbrace{\prod_{\{\mu : a_\mu = 1\}} (1+i)}_{=\beta} \le 2\ \} \end{aligned}$$

Consider the possible values of α and β:

$\alpha = 1, \beta = 1$ No a_μ equals zero, i must be zero, hence $a_\mu = \omega^0 = 1$
$\Rightarrow a = (1, \ldots, 1)$.

$\alpha = 1, \beta = 2$ No a_μ equals zero, i must be zero, hence $a_\mu = \omega^1$
$\Rightarrow a = (1, \ldots, 1, \omega, 1, \ldots, 1)$.

$\alpha = 2, \beta = 1$ Exactly one a_μ equals zero, i must be zero, hence $a_\nu = \omega^0 = 1$ for $\mu \neq \nu$
$\Rightarrow a = (1, \ldots, 1, 0, 1, \ldots, 1)$.

For $q = 2$ we have $\omega = 1$. □

4.1.2 Lower Bounds

We want to develop lower bounds of $c_t^n(q, m)$, i.e. the minimal cardinality of a set of evaluation points in $\mathrm{GF}(q^m)^n$ separating t-sparse polynomials from the zero-polynomial.

The most important result with regard to efficient interpolation algorithms is the fact that in the case of $m = 1$ (interpolation in the ground field), it is *not* possible to perform a zero test using only a polynomial number of evaluation points.

A rough lower bound is obtained by the following observation:

Every t-sparse polynomial f can be rewritten as the sum of a $\lfloor \frac{t}{2} \rfloor$-sparse polynomial f_1 and a $\lceil \frac{t}{2} \rceil$-sparse polynomial f_2. If $f \not\equiv 0$, then there is an $a \in \mathrm{GF}(q^m)^n$ with $f_1(a) \neq f_2(a)$. Therefore, a zero test set $A \in \mathcal{A}_t^m(q, m)$ for a t-sparse polynomial has to contain elements enabling us to distinguish all pairs of pairwise different $\lfloor \frac{t}{2} \rfloor$-sparse polynomials f_1, f_2. Since the mapping $\#\mathcal{P}^n_{\lfloor \frac{t}{2} \rfloor} \to \mathrm{GF}(q^m)^{\#A}$ with $p \mapsto (p(a_1), \ldots, p(a_{\#A}))$ is injective,

$$\#\mathcal{P}^n_{\lfloor \frac{t}{2} \rfloor}(q) \leq \#\mathrm{GF}(q^m)^{\#A}. \tag{10}$$

We calculate the number of polynomials in $\mathcal{P}^n_{\lfloor \frac{t}{2} \rfloor}(q)$. There are q^n different monomials; the number of possibilities to form a polynomial with exactly i non-zero coefficients is

$$\underbrace{\binom{q^n}{i}}_{\text{choice of } i \text{ monomials}} \cdot \underbrace{(q-1)^i}_{i \text{ coefficients} \neq 0}.$$

Then

$$\#\mathcal{P}^n_{\lfloor \frac{t}{2} \rfloor}(q) = \sum_{i=0}^{\lfloor \frac{t}{2} \rfloor} \binom{q^n}{i} \cdot (q-1)^i.$$

Let A be a minimal zero test set, i.e, $\#A = c_t^n(q, m)$. Then

$$\#\mathrm{GF}(q^m)^{\#A} = (q^m)^{c_t^n(q,m)}.$$

We obtain a lower bound by (10):

$$\frac{1}{m} \cdot \log_q \left(\sum_{i=0}^{\lfloor \frac{t}{2} \rfloor} \binom{q^n}{i} \cdot (q-1)^i \right) \leq c_t^n(q, m).$$

In general, it is difficult to find better lower bounds for arbitrary m. Therefore we examine some special cases in the sequel.

First, we derive the result for $m = 1$ mentioned above.

Theorem 4.8 Let $A \in \mathcal{A}_t^n(q,1)$, i.e. A distinguishes f in GF(q) from the zero-polynomial. Let T be a subset of $\{0,\ldots,n-1\}$ with $\#T \leq \lfloor \log_2 t \rfloor$. For each T let $a^T = (a_0^T,\ldots,a_{n-1}^T)$ be an element in GF$(q)^n$ with $(a_i^T = 0 \Leftrightarrow i \in T)$. Then

$$\forall_{T,\,\#T \leq \lfloor \log_2 t \rfloor}: \quad a^T \in A.$$

Hence, A consists of at least $\sum_{i=0}^{\lfloor \log_2 t \rfloor} \binom{n}{i}$ elements and

$$\sum_{i=0}^{\lfloor \log_2 t \rfloor} \binom{n}{i} \leq c_t^n(q,1).$$

PROOF: For every T with $\#T \leq \lfloor \log_2 t \rfloor$ we define a polynomial p_T by:

$$p_T(x_0,\ldots,x_{n-1}) := \prod_{i \in T}(x_i^{q-1} - 1) \cdot \prod_{i \notin T} x_i$$

p_T has the following properties:

1. p_T is t-sparse:
   The number of non-zero coefficients of p_T corresponds to the number of coefficients of $\prod_{i \in T}(x_i^{q-1} - 1) = 2^T \leq 2^{\lfloor \log_2 t \rfloor} \leq t$.

2. $p_T(a) \neq 0 \Longleftrightarrow (a_i = 0 \Leftrightarrow i \in T)$
   (i.e. we can distinguish p_T from the zero-polynomial only at the point a^T):
   The elements in GF$(q)\backslash\{0\}$ form a cyclic group of order $q-1$. Therefore, $\forall a \in \mathrm{GF}(q)\backslash\{0\}$: $a^{q-1} = 1$. This implies that the first factor is non-zero iff $\forall i \in T$: $a_i = 0$. The second factor is non-zero iff $\forall i \in T$: $a_i \neq 0$.

Therefore for every $T \subseteq \{0,\ldots,n-1\}$, $\#T \leq \lfloor \log_2 t \rfloor$ A contains an element a^T, hence

$$\sum_{i=0}^{\lfloor \log_2 t \rfloor} \binom{n}{i} \leq \#A.$$

□

Combining Corollary 4.6 and Theorem 4.8 we derive $c_t^n(2,1)$:

Corollary 4.9

$$c_t^n(2,1) = \sum_{i=0}^{\lfloor \log_2 t \rfloor} \binom{n}{i}.$$

Likewise, the declared upper and lower bounds coincide for $n = m = 1$, i.e. univariate polynomials:

Theorem 4.10

$$c_t^1(q,1) = \begin{cases} \min\{t+1, q\}, & \text{for } t \geq 2 \\ 1, & \text{for } t = 1 \end{cases}.$$

PROOF: For $t = 1$ we have ($f \equiv 0 \Leftrightarrow f(1) = 0$). Let $t \geq 2$. As an upper bound we have by (8)

$$c_t^1(q,1) \leq 1 + t - \left\lfloor \frac{t-1}{q} \right\rfloor,$$

matching the stated formula, since univariate polynomials over GF(q) have at most q terms, therefore $t \leq q$.
We obtain a lower bound as follows: If $t = q$, then every mapping GF(q) $\rightarrow$ GF(q) is in $\mathcal{P}_q^1(q)$, hence all elements in GF(q) belong to the minimal zero test set. If $q > t \geq 2$ and $A \subseteq \mathrm{GF}(q), A \in \mathcal{A}_t^1(q,1)$ with $\#A = t$ then $0 \in A$, since $f = x^{q-1} - 1$ is t-sparse and vanishes for all elements in GF(q)$\setminus\{0\}$. Then $f = \prod_{a \in A \setminus \{0\}} (x - a)$ is a non-vanishing polynomial of degree at most $t-1$, and therefore t-sparse. $(x \cdot f)$ is also t-sparse, but vanishes as well for $a = 0$ as for $a \in A \setminus \{0\}$. Therefore A must consist of at least $t+1$ elements. □

Theorem 4.11 Let $q > 2$ and ω be a primitive element in GF(q). Let $A \in \mathcal{A}_2^n(q,1)$ be a zero test set for a t-sparse binomial $f = c_\alpha x^\alpha + c_\beta x^\beta$ with $\deg_{x_i}(f) < q$ for all i. Then A comprises n elements $a^{(\mu)} = (a_0^{(\mu)}, \ldots, a_{n-1}^{(\mu)})$ where

$$a_\nu^{(\mu)} = 0 \Longleftrightarrow \mu = \nu \tag{11}$$

and $n+1$ pairwise different elements $a^{(\mu)}$, $n \leq \mu \leq 2n$ with no zero component, i.e.

$$\exists 0 \leq b_\nu^{(\mu)} < q-1 \quad a_\nu^{(n+\mu)} = \omega^{b_\nu^{(\mu)}}, \quad 0 \leq \nu < n. \tag{12}$$

In particular $c_2^n(q,1) = 2n+1$.

PROOF: Let $A \in \mathcal{A}_2^n(q,1)$ and A_1 be a set of n elements satisfying (11) and $A_2 \subseteq A$ be the set of elements in A satisfying (12). We have to prove that A being a zero test set for binomials implies (1) $A_1 \subseteq A$ and (2) the number of elements in A_2 is at least $n+1$.

[Printing Error in Paper [0514] we define a polynomial p_ν enforcing that the elements $a^{(\mu)}$, $0 \leq \mu < n$ belong to A. Let p_μ defined by:

$$p_\mu := (x_\mu^q - 1) \cdot \prod_{\substack{\nu=0 \\ \nu \neq \mu}}^{\nu=n-1} x_\nu \in \mathcal{P}_2^n(q, 0 \leq \mu < n.$$

Consider the arguments $a = (a_0, \ldots, a_{n-1})$ where p_μ does not vanish:

(a) $a_\mu = 0$, because for all $r \in \mathrm{GF}(q) \setminus \{0\}$ it holds that $r^{q-1} = 1$.

(b) $a_\nu \neq 0$, $0 \leq \nu < n$, $\nu \neq \mu$, thereby the second factor of p_μ does not vanish.

Hence: $a_\nu = 0 \Longleftrightarrow \mu = \nu$ for $0 \leq \mu < n$. Therefore $A_1 \subseteq A$.

(2) Let $\tilde{n} := |A_2|$ be the number of elements in A_2. With (12) let $A_2 = \{a^{(n)}, \ldots, a^{(n+\tilde{n}-1)}\}$. We have to show that the assumption $\tilde{n} \leq n$, leads to a contradiction. Let a binomial f be defined by

$$f := x_0 \cdot \ldots \cdot x_{n-1} \cdot (x^\alpha - \omega^d).$$

Thereby $\alpha \in \{0,\ldots,q-2\}^n \setminus \{(0,\ldots,0)\}$ and $0 \le d \le q-2$ are defined such that $a^\alpha = \omega^d$ holds for all $a \in A_2$. The existence of α and d with this property follows from the following:
Consider for $\tilde{n} \le n$ the following system of equations in α:

$$\begin{aligned} a^\alpha := \prod_{\nu=0}^{n-1} a_\nu^{\alpha_\nu} = \omega^d \quad & \text{for all } a \in A_2 \\ \iff \prod_{\nu=0}^{n-1} \omega^{b_\nu^{(\mu)} \cdot \alpha_\nu} = \omega^d \quad & \text{for } 0 \le \mu < \tilde{n} \\ \iff \omega^{\sum_{\nu=0}^{n-1} b_\nu^{(\mu)} \cdot \alpha_\nu} = \omega^d \quad & \text{for } 0 \le \mu < \tilde{n}. \end{aligned} \tag{13}$$

Since ω is a primitive element in GF(q), there exists a unique representation $y = \omega^d$ where $0 \le d < q-1$ for each $y \in \mathrm{GF}(q) \setminus \{0\}$. (13) is equivalent to

$$\sum_{\nu=0}^{n-1} b_\nu^{(\mu)} \cdot \alpha_\nu = d \quad \text{for } 0 \le \mu < \tilde{n}, \quad \text{hence}$$

$$\underbrace{(b_\nu^{(\mu)})_{\substack{0 \le \mu < \tilde{n} \\ 0 \le \nu < n}}}_{=:B} \cdot (\alpha_\nu)_{0 \le \nu < n} = (d,\ldots,d)^t \qquad \text{in } \mathbb{Z}_{p-1}.$$

B defines a linear map $(\mathbb{Z}_{p-1})^n \to (\mathbb{Z}_{p-1})^{\tilde{n}}$. If B is not injective, then the kernel of B does not equal $\{0\}$, therefore there exists some non-trivial $\alpha \ne (0,\ldots,0)$ with $B \cdot \alpha = 0$. If B is injective, and therefore bijective (since $\tilde{n}$ equals n), there exists for any $d \ne 0$ a non-trivial solution $\alpha \ne (0,\ldots,0)$ with $B \cdot \alpha = (d,\ldots,d)^t$.

$f(a) = 0$ for all $a \in A$ because first if $a \in A \setminus A_2$, then there exists some $a_i = 0$, hence $f(a) = 0$ and second if $a \in A_2$, therefore $a^\alpha = \omega^d$, thereby the second factor of f vanishes. For $\tilde{n} \le n$ we have $A \notin \mathcal{A}_2^n(q,1)$, therefore a zero test set A in GF(q) for a binomial must contain at least $n+1$ elements satisfying (12). □

4.2 Efficient Algorithms with few Evaluation Points

In this section we present an interpolation algorithm for t-sparse polynomials developed by Clausen, Grabmeier and Karpinski [CGK 87]. This algorithm uses only a number of evaluation points linear in t. However, it works in the *large* field extension GF(q^n) of GF(q). This field extension is too large to obtain an algorithm lying in NC.

Grigoriev and Karpinski [GK 87] utilize the uniqueness of the prime factorization to separate monomials and evaluate polynomials at points $(p_0^i,\ldots,p_{n-1}^i)$, where the p_j's are pairwise distinct primes. Different monomials have different values at these points.
In [CGK 87] this idea is transferred to the case of finite fields (polynomials over GF(q) in n variables). We represent monomials by the q-adic expansion of their exponents and use the uniqueness of the discrete logarithm to the base of a primitive element ω in GF(q^n). Monomials are separated by the points $\omega^{iq^0}, \omega^{iq^1}, \ldots, \omega^{iq^{n-1}}$.

Theorem 4.12 Let $f \in \mathrm{GF}(q)[x_0,\ldots,x_{n-1}]$ be a t-sparse polynomial, $t \ge 2$ with $\deg_{x_j}(f) < q$ for $1 \le j \le n$ and let ω be a primitive element in GF(q^n). Let $f_i := f(\omega^{iq^0}, \omega^{iq^1}, \ldots, \omega^{iq^{n-1}})$ for $0 \le i \le t-1$ and $q \nmid i$ for $i > 0$. Then for $t > 2$,

$$f \equiv 0 \iff f(0,\ldots,0) = 0 \text{ and } f_i = 0 \text{ for } 0 \le i \le t-1 \text{ and } q \nmid i \text{ for } i > 0,$$

and for $t = 1$, $f \equiv 0 \Leftrightarrow f(1,\ldots,1) = 0$.

PROOF: The case $t = 1$ is trivial, therefore let $t \geq 2$. Let $\mathbf{q}^n := \{0,\ldots,q-1\}^{\{1,\ldots,n\}}$ be a set of multiindices. f is a linear combination of the q^n monomials $x^\alpha := x_1^{\alpha_1} \cdot \ldots \cdot x_n^{\alpha_n}$:

$$f = \sum_{\alpha \in \mathbf{q}^n} c_\alpha x^\alpha.$$

Let $f(0,\ldots,0) = c_{(0,\ldots,0)} = 0$, otherwise $f \not\equiv 0$.
Consider the mapping $\Omega : \mathbf{q}^n \setminus \{(0,\ldots,0)\} \to \mathrm{GF}(q^n) \setminus \{0\}$ defined by

$$\Omega(\alpha) := \Omega_\alpha = \prod_{\nu=0}^{n-1} \omega^{\alpha_\nu q^\nu} = \omega^{(\sum_{\nu=0}^{n-1} \alpha_\nu q^\nu)}.$$

Since ω is a primitive element in $\mathrm{GF}(q^n)$, the mapping Ω is bijective because of the uniqueness of the q-adic representation of elements in $\mathrm{GF}(q^n)$. Thereby different monomials are assigned different values in $\mathrm{GF}(q^n)$ by means of Ω.
f is t-sparse, hence there are at most t non-zero coefficients of f. Let $\mathrm{supp}(f) := \{\alpha : c_\alpha \neq 0\}$ be the support of f. Then $|\mathrm{supp}(f)| =: k \leq t$.
Let $A = \{\alpha^{(0)},\ldots,\alpha^{(k-1)}\}$ be any k-subset of $\mathbf{q}^n \setminus \{(0,\ldots,0)\}$ with $\mathrm{supp}(f) \subseteq A$. Since $c_{(0,\ldots,0)} = 0$, it holds:

$$f = \sum_{\alpha \in \mathbf{q}^n} c_\alpha x^\alpha = \sum_{\alpha \in A} c_\alpha x^\alpha$$

hence, we have for all $0 \leq i < k \leq t$

$$\begin{aligned} f_i &= f(\omega^{iq^0}, \omega^{iq^1}, \ldots, \omega^{iq^{n-1}}) \\ &= \sum_{\alpha \in A} c_\alpha \cdot \omega^{\alpha_0 iq^0} \cdot \ldots \cdot \omega^{\alpha_{n-1} iq^{n-1}} \\ &= \sum_{\alpha \in A} c_\alpha \cdot \omega^{(\sum_{\nu=0}^{n-1} \alpha_\nu q^\nu)\cdot i} \\ &= \sum_{\alpha \in A} c_\alpha \cdot \Omega_\alpha^i. \end{aligned}$$

We obtain the following linear system of equations:

$$\underbrace{\left(\Omega_\alpha^i\right)_{\substack{0 \leq i < k \\ \alpha \in A}}}_{=: \mathcal{V}} \cdot (c_\alpha)_{\alpha \in A} = (f_i)_{0 \leq i < k}. \tag{14}$$

Since Ω is bijective we have $\Omega_\alpha \neq \Omega_\beta$ for $\alpha \neq \beta$. Therefore $\mathcal{V}$ is a non-singular Vandermonde matrix, and the linear system of equations (14) has a unique solution:

$$f \equiv 0 \iff (c_\alpha)_{\alpha \in A} = 0 \iff (f_i)_{0 \leq i < k} = 0.$$

It remains to be proved that the values f_i for $q \not| i$ can be derived. Consider the Frobenius automorphism $\Phi(y) = y^q$ in $\mathrm{GF}(q^n)$. It holds $f_{j \cdot q} = (f_j)^q$ for all $i < q^n$, since

$$f_{j\cdot q} = \sum_{\alpha \in A} c_\alpha \cdot \Omega_\alpha^{j\cdot q} = \sum_{\alpha \in A} c_\alpha^q \cdot \Omega_\alpha^{j\cdot q} = \sum_{\alpha \in A} (c_\alpha \cdot \Omega_\alpha^j)^q = (\sum_{\alpha \in A} c_\alpha \cdot \Omega_\alpha^j)^q = (f_j)^q.$$

Let $i = j \cdot q$, i.e. $q | i$ and $q \not| j$. Then $f_i = (f_j)^q$, therefore we can compute the missing f_i. □

Theorem 4.13 Let $f \in \mathrm{GF}(q)[x_0, \ldots, x_{n-1}]$ be a t-sparse polynomial with $\deg_{x_j}(f) < q$ for $1 \leq j \leq n$ and let ω be a primitive element in $\mathrm{GF}(q^n)$. The function values $f_i := f(\omega^{iq^0}, \omega^{iq^1}, \ldots, \omega^{iq^{n-1}})$ for $0 \leq i < 2t$ satisfying $q \nmid i$ for $i > 0$ and the value $f(0, \ldots, 0)$ enable the reconstruction of f.

PROOF: We use the notation of Theorem 4.12. Suppose that $f(0, \ldots, 0) = c_{(0,\ldots,0)} = 0$, otherwise we reconstruct $f - f(0, \ldots, 0)$. Let

$$f = \sum_{\alpha \in \mathbf{q}^n \setminus \{(0,\ldots,0)\}} c_\alpha x^\alpha$$

and let $A = \{\alpha_0, \ldots, \alpha_l\}$ be any subset of $\mathbf{q}^n \setminus \{(0, \ldots, 0)\}$. Let $e_i(A), 0 \leq i \leq l$ be the i-th elementary symmetric polynomial in $|A|$ indeterminates evaluated at $(\Omega_\alpha)_{\alpha \in A}$ (this is well-defined, since $(0, \ldots, 0) \notin A$).

$$\begin{aligned}
e_0(A) &= 1 \\
e_1(A) &= \Omega_{\alpha_0} + \Omega_{\alpha_1} + \ldots + \Omega_{\alpha_l} \\
e_2(A) &= \Omega_{\alpha_0} \cdot \Omega_{\alpha_1} + \Omega_{\alpha_0} \cdot \Omega_{\alpha_2} + \ldots + \Omega_{\alpha_0} \cdot \Omega_{\alpha_l} + \\
&\quad \Omega_{\alpha_1} \cdot \Omega_{\alpha_2} + \ldots + \Omega_{\alpha_1} \cdot \Omega_{\alpha_l} + \ldots + \Omega_{\alpha_{l-1}} \cdot \Omega_{\alpha_l} \\
&\vdots \\
e_l(A) &= \Omega_{\alpha_0} \cdot \Omega_{\alpha_1} \cdot \ldots \cdot \Omega_{\alpha_l}.
\end{aligned}$$

Let

$$\Lambda(x) := \prod_{\beta \in A} (x - \Omega_\beta) \in \mathrm{GF}(q^n)[x].$$

By means of *Newton's Formula* (s. [LN 86]) we have:

$$\Lambda(x) = \sum_{j=0}^{|A|} \underbrace{(-1)^{|A|-j} \cdot e_{|A|-j}(A)}_{\lambda_j(A)} \cdot x^j.$$

The roots of this polynomial are just $(\Omega_\alpha)_{\alpha \in A}$. This yields the generalized Newton identities:

$$0 = \sum_{j=0}^{|A|} \lambda_j(A) \cdot \Omega_\alpha^j, \quad \alpha \in A.$$

Multiplying the equation corresponding to α by $c_\alpha \cdot \Omega_\alpha^i$ for fixed $i, 0 \leq i < q^n$ and summing over all $\alpha \in A$ results in:

$$\begin{aligned}
\sum_{\alpha \in A} 0 \cdot c_\alpha \cdot \Omega_\alpha^i &= \sum_{\alpha \in A} \sum_{j=0}^{|A|} \lambda_j(A) \cdot c_\alpha \cdot \Omega_\alpha^{j+i} \\
0 &= \sum_{j=0}^{|A|} \lambda_j(A) \cdot \sum_{\alpha \in A} c_\alpha \cdot \Omega_\alpha^{j+i} \\
0 &= \sum_{j=0}^{|A|} \lambda_j(A) \cdot f_{i+j}.
\end{aligned}$$

Since $\lambda_{|A|} = e_0(A) = 1$ it holds

$$\begin{aligned}\sum_{j=0}^{|A|-1} \lambda_j(A) \cdot f_{i+j} &= -\lambda_{|A|} \cdot f_{i+|A|} \\ &= -f_{i+|A|} \qquad \text{for all } 0 \le i < |A|.\end{aligned}$$

We obtain the following linear system of equations:

$$(f_{i+j})_{0\le i,j<|A|} \cdot (\lambda_j(A))_{0\le j<|A|} = \left(-f_{i+|A|}\right)_{0\le i<|A|}. \tag{15}$$

Consider the matrix $(f_{i+j})_{0\le i,j<|A|}$:

$$f_{i+j} = \sum_{\alpha\in A} c_\alpha \cdot \Omega_\alpha^{i+j} = \sum_{\alpha\in A} \Omega_\alpha^i \cdot c_\alpha \cdot \Omega_\alpha^j$$

$$\Longrightarrow (f_{i+j})_{0\le i,j<|A|} = \left(\Omega_\alpha^i\right)^t_{\substack{0\le i<|A| \\ \alpha\in A}} \cdot D_A \cdot \left(\Omega_\alpha^j\right)_{\substack{0\le i<|A| \\ \alpha\in A}}$$

where $D_A = \operatorname{diag}((c_\alpha)_{\alpha\in A})$.

Hence the rank of $(f_{i+j})_{0\le i,j<|A|}$ equals $\operatorname{rg}(D_A) = |\operatorname{supp}(f)| =: k \le t$.
Therefore we conclude that for $A = \operatorname{supp}(f)$ $(f_{i+j})_{0\le i,j<k}$ is non-singular.
Then the linear system of equations (15) has a unique solution and we obtain the coefficients

$$\lambda_j(\operatorname{supp}(f)),\ 0 \le j < k \quad \text{of the polynomial} \quad \Lambda(x) = \prod_{\beta\in\operatorname{supp}(f)} (x - \Omega_\beta).$$

Determining the roots of this polynomials supplies $(\Omega_\alpha)_{\alpha\in\operatorname{supp}(f)}$. Since Ω is bijective we can hereby determine $\operatorname{supp}(f)$.

The matrix $\mathcal{V}$ of the system of equations (14) is completely determined. We compute the coefficients $(c_\alpha)_{\alpha\in\operatorname{supp}(f)}$ of f. Hereby f is completely reconstructed.

As shown in the proof of Theorem 4.12, we can conclude that the values f_i where $q \mid i$ can be computed from the known values f_j where $q \nmid j$ by means of the properties of the Frobenius automorphism. Altogether we need $1 + t - \lfloor\frac{2t-1}{q}\rfloor$ evaluations. □

Theorem 4.12 and Theorem 4.13 yield the following algorithm reconstructing a t-sparse polynomial $f \in \mathrm{GF}(q)[x_0, \ldots, x_{n-1}]$ with $\deg_{x_j}(f) < q$ for all j.

Interpolation Algorithm [CGK 87]

Input: A black box for f.

Step 1: Take a primitive element ω in $\mathrm{GF}(q^n)$.

Step 2: Query the black box for the $1 + t - \lfloor\frac{2t-1}{q}\rfloor$ values $f(0, \ldots, 0)$ and
$f_i := f(\omega^{iq^0}, \omega^{iq^1}, \ldots, \omega^{iq^{n-1}})$ for $0 \le i < 2t$ satisfying $q \nmid i$ for $i > 0$.
If the constant term $f(0, \ldots, 0)$ is non-zero interpolate $f - f(0, \ldots, 0)$.

Step 3: For all $0 \le i < 2t$ which satisfy $i = q^s \cdot i_0, 1 \le s, s$ maximal, calculate the missing $f_i = f_{i_0}^{q^s}$.

Step 4: Determine $k = \mathrm{rg}((f_{i+j})_{0\le i,j<t})$.

Step 5: Solve the linear system of equations

$$(f_{i+j})_{0\le i,j<k} \cdot (\lambda_j)_{0\le j<k} = (-f_{i+k})_{0\le i<k}.$$

Step 6: Determine the roots of the polynomial

$$\Lambda(x) = x^k + \sum_{j=0}^{k-1} \lambda_j \cdot x^j.$$

We obtain $(\Omega_\alpha)_{\alpha\in \mathrm{supp}(f)}$.

Step 7: Calculate the q-adic expansion of the exponents of Ω_α with respect to ω. This yields $\mathrm{supp}(f) := A$.

Step 8: Solve the linear system of equations

$$\left(\Omega_\alpha^i\right)_{\substack{0\le i<k \\ \alpha\in A}} \cdot (c_\alpha)_{\alpha\in A} = (f_i)_{0\le i<k}$$

to obtain $(c_\alpha)_{\alpha\in \mathrm{supp}(f)}$.

Output: $(c_\alpha, \alpha)_{\alpha\in \mathrm{supp}(f)}$.

Suppose a primitive element $\omega \in \mathrm{GF}(q^n)$ is given. By [Mu 86] we can perform the steps 4, 5, 7 and 8 with $O(t^{4.5})$ processors in $O(\log^2 t)$ parallel time. The same complexity holds for factoring the univariate polynomial in step 6 by means of the algorithm suggested in [Ga 84].

Theorem 4.14 The interpolation algorithm [CGK 87] is NC^2-reducible to the computation of discrete logarithms in $\mathrm{GF}(q^n)$.

Remark: The algorithm of Mulmuley [Mu 86] works for arbitrary matrices. The question arises whether we can make use of the special structure of the maintained Vandermonde matrices to gain a better complexity of this algorithm.
By *Cramer's rule* we have to compute determinants of Vandermonde matrices where one power is skipped. Let

$$C_n^{(k)} = \det \begin{pmatrix} t_0^0 & \dots & t_0^{k-1} & t_0^{k+1} & \dots & t_0^{n+1} \\ \vdots & & \vdots & \vdots & & \vdots \\ t_n^0 & \dots & t_n^{k-1} & t_n^{k+1} & \dots & t_n^{n+1} \end{pmatrix} \quad \text{for } 0 \le k \le n+1,\ n \in \mathbb{N}_0.$$

Then

$$C_n^{(k)} = \sigma_n^{(k)} \cdot \prod_{i>j} (t_i - t_j) \quad \text{for } 0 \le k \le n+1,\ n \in \mathbb{N}_0$$

where $\sigma_n^{(k)}$ is the k-th elementary symmetric polynomial in the variables $t_0, \dots, t_n$. If we can compute $\sigma_n^{(k)}$ efficiently, we could reduce the effort of the algorithm. It holds

$$\sigma_n^{(k)} = t_n \cdot \sigma_{n-1}^{(k)} + \sigma_{n-1}^{(k-1)},$$

however, from this recursion formula, similar to that for binomial coefficients, an efficient computation is not quite obvious since we need n recursion steps.

Theorem 4.15 For univariate polynomials the interpolation algorithm [CGK 87] is optimal concerning the number of evaluation points.

PROOF: We have to consider the case $n = 1$ and $2t < q$. Let A be an arbitrary subset of GF(q) with at most $2t$ elements. Then $0 \not\equiv h := \prod_{a \in A}(x - a) \in \mathrm{GF}(q)[x]$ is a polynomial of degree at most $2t < q$ and with at most $2t$ monomials. Therefore h has a representation $f - g$ where f, g are t-sparse. Since h vanishes in A, f and g coincide in A. Hence A is not suitable to reconstruct t-sparse polynomials. □

4.3 Development of the first NC-Algorithm

In this section the first NC-algorithm for the interpolation of sparse polynomials over finite fields developed by Grigoriev, Karpinski and Singer [GKS 88] is introduced.

The algorithm [CGK 87] (cf. Section 4.2) requires the computation of discrete logarithms in GF(q^n). At present, no efficient algorithms for this purpose are known. In order to develop an NC-algorithm, we determine an extension GF(q^s) of GF(q), s as small as possible, such that arithmetic in GF(q^s) lies in NC and a polynomially number of function values in GF(q^s) provides enough information to reconstruct the polynomial.

By Theorem 4.8 the number of required evaluation points over the ground field GF(q) is not polynomially bounded. Therefore there is no polynomial interpolation algorithm working in GF(q). In [GKS 88] it is shown that a slight extension of logarithmic degree suffices to reconstruct the polynomial using only a polynomial number of evaluation points. Since NC-interpolation in GF(q) is not possible, this slight field extension is in a sense the smallest extension to carry out efficient interpolation.

Theorem 4.16 Let $f \in \mathrm{GF}(q)[x_1, \ldots, x_n]$ be a t-sparse polynomial for arbitrary q. Then there exists a deterministic parallel algorithm (NC^3) to interpolate f over the *slight* field extension $\mathrm{GF}(q^{\lceil 4 \log_q(nt)+3 \rceil})$ by means of $(O(qn^2t^5))$ evaluations. The algorithm takes $O(\log^3(ntq))$ parallel time and $O(n^2t^6q^3 \log^{5.5}(ntq) + q^{2.5} \log^2 q)$ processors.

One of the major algorithmic aspects is the problem of how to determine evaluation points so that distinct monomials are separated. In Section 4.2 this problem is solved by means of the uniqueness of the q-adic expansion and the uniqueness of discrete logarithm, at the cost of going to the field GF(q^n). A small number of evaluation points is needed, however, there are no effective deterministic procedures known even for finding primitive elements.
The idea suggested in [GKS 88] is to permit a larger number of evaluation points where not all of the evaluation points actually separate distinct monomials. However, they are constructed such that enough of them have this property in order to reconstruct the polynomial efficiently. To guarantee this property we make use of *Cauchy* matrices.

Definition 4.17 (Cauchy matrix)
Let x_i, y_i for $1 \le i, j \le N$ be fixed values. An $(N \times N)$-matrix $C = (c_{ij})_{1 \le i,j \le N}$ is called a *Cauchy matrix*, if

$$c_{ij} = \frac{1}{x_i + y_j} \qquad \text{for all } 1 \le i, j \le N.$$

Lemma 4.18 Let C be a Cauchy matrix. Then:

$$\det C = \frac{\prod_{1\le i<j\le N}(x_j - x_i)\cdot(y_j - y_i)}{\prod_{1\le i,j\le N}(x_i + y_i)}$$

A similar formula holds for any non-vanishing minor of C.

In the sequel we use a Cauchy matrix C defined by $c_{ij} := \frac{1}{i+j} \bmod p$ for $1 \le i, j \le N$ where p is a prime. If $2N < p$, then by Lemma 4.18 C and any non-vanishing minor of C are non-singular.

The algorithm [GKS 88] involves two major computational steps: (1) efficient zero test for polynomials in $\mathrm{GF}(q)[x_1, \ldots, x_n]$ over the *slight* field extension $\mathrm{GF}(q^{\lceil 2\log_q(nt)+3\rceil})$, and (2) solution of the interpolation problem by means of inductive enumeration of partial solutions for partial monomials and coefficients obtained by recursive application of the zero test mentioned above.

4.3.1 The Zero Test

We assume that the black box is capable of evaluating $f \in \mathrm{GF}(q)[x_1, \ldots, x_n]$ in an arbitrary field extension $\mathrm{GF}(q^s)$. We present a zero test for f working in (the slight extension) $\mathrm{GF}(q^{O(\log(nt))})$.

Zero Test [GKS 88]

Input: Black box for $f \in \mathrm{GF}(q)[x_1, \ldots, x_n]$, t-sparse.

Output: $\begin{cases} \text{Yes} & \text{if } f \equiv 0 \\ \text{No} & \text{if } f \not\equiv 0 \end{cases}$

Step 1: Determine a minimal s satisfying $q^s - 1 > 4qn \cdot (n-1) \cdot \binom{t}{2}$.

Step 2: Construct the field $\mathrm{GF}(q^s)$ and a primitive element ω in $\mathrm{GF}(q^s)$ with the help of the Berlekamp algorithm [Be 70].

Step 3: Let $N = \lceil \frac{q^s-1}{4nq} \rceil$. Use the sieve of Erastosthenes to find a prime p with $2N < p \le 4N$.

Step 4: Compute $c_{ij} := \frac{1}{i+j} \bmod p$ for $1 \le i, j \le N$ by means of the Euclidean algorithm.
Let $C = (c_{ij})_{1\le i,j\le N}$

Step 5: Denote by $\bar{C} = (\bar{c}_{ij})$ an arbitrary $(N \times n)$-submatrix of C.

Step 6: Query the black box in $\mathrm{GF}(q^s)$ for the points

$$\omega^{l\cdot\bar{c}_i} = (\omega^{l\cdot\bar{c}_{i1}}, \omega^{l\cdot\bar{c}_{i2}}, \ldots, \omega^{l\cdot\bar{c}_{in}}) \quad \text{for } i = 1, \ldots, N,\ l = 0, \ldots, t-1$$

and for the zero-point $(0, \ldots, 0)$. If all evaluations are zero then $f \equiv 0$.

Theorem 4.19 The zero test [GKS 88] is correct.

PROOF: Let $f(0,\ldots,0) = c_{(0,\ldots,0)} = 0$ otherwise $f \not\equiv 0$. We have to prove the following:

$$f \not\equiv 0 \iff \exists 0 \le l < t, 1 \le i \le N : f(\omega^{l \cdot \bar{c}_i}) \neq 0.$$

Let $\mathbf{q}^n$ be defined as in the proof of Theorem 4.12. f is a linear combination of the $q^n - 1$ monomials $x^\alpha := x_1^{\alpha_1} \cdot \ldots \cdot x_n^{\alpha_n}$:

$$f = \sum_{\alpha \in \mathbf{q}^n \setminus \{(0,\ldots,0)\}} c_\alpha x^\alpha$$

Let $\alpha' = (\alpha'_1,\ldots,\alpha'_n)$, $\alpha'' = (\alpha''_1,\ldots,\alpha''_n)$ and $\bar{c}_i = (\bar{c}_{i_1},\ldots,\bar{c}_{i_n})$. First, we want to prove the property of the evaluation points mentioned above, i.e. we have to show that there is an evaluation point separating a fixed pair of monomials $x^{\alpha'}$ and $x^{\alpha''}$ where $\alpha' \neq \alpha''$. That means that there exists some row $\bar{c}_i$ of the matrix $\bar{C}$ so that the monomials $x^{\alpha'}$ and $x^{\alpha''}$ take distinct values at the point $\omega^{\bar{c}_i}$. Let

$$\Omega_{\alpha,i} := x^\alpha|_{\omega^{\bar{c}_i}} = \omega^{(\sum_{j=1}^{n} \alpha_j \cdot \bar{c}_{i_j})}.$$

We have to show, that $\exists 1 \le i \le N : \Omega_{\alpha',i} \neq \Omega_{\alpha'',i}$ for any $\alpha' \neq \alpha''$. A row $\bar{c}_i$ of the matrix $\bar{C}$ is called *bad* for $\alpha' \neq \alpha''$, if $\bar{c}_i$ does not separate the monomials $x^{\alpha'}$ and $x^{\alpha''}$, i.e. if $\Omega_{\alpha',i} = \Omega_{\alpha'',i}$, that is $\omega^{\bar{c}_i \cdot \alpha'} = \omega^{\bar{c}_i \cdot \alpha''}$. ω is a primitive element in $\mathrm{GF}(q^s)$, hence ω generates the cyclic group $\mathrm{GF}(q^s) \setminus \{0\}$ of order $q^s - 1$. Then

$$\begin{aligned}
& \omega^{\bar{c}_i \cdot \alpha'} = \omega^{\bar{c}_i \cdot \alpha''} \\
\iff \quad & \bar{c}_i \cdot \alpha' \equiv \bar{c}_i \cdot \alpha'' \pmod{q^s - 1} \\
\iff \quad & \sum_{j=1}^{n} \bar{c}_{i_j} \cdot \alpha'_j \equiv \sum_{j=1}^{n} \bar{c}_{i_j} \cdot \alpha''_j \pmod{q^s - 1} \\
\iff \quad & \sum_{j=1}^{n} (\alpha'_j - \alpha''_j) \cdot \bar{c}_{i_j} \equiv 0 \pmod{q^s - 1}. \qquad (16)
\end{aligned}$$

It holds $0 \le \alpha'_j, \alpha''_j \le q-1$, therefore we have $|\alpha'_j - \alpha''_j| \le q-1$ and also by step 4 $|\bar{c}_{i_j}| < p \le 4N$. Hence

$$|\sum_{j=1}^{n} (\alpha'_j - \alpha''_j) \cdot \bar{c}_{i_j}| \le \sum_{j=1}^{n} |\alpha'_j - \alpha''_j| \cdot |\bar{c}_{i_j}| \le n \cdot (q-1) \cdot 4N. \qquad (17)$$

By step 3 $N = \frac{\lceil q^s - 1 \rceil}{4nq}$, therefore we continue estimating the inequality (17)

$$|\sum_{j=1}^{n} (\alpha'_j - \alpha''_j) \cdot \bar{c}_{i_j}| \le q^s - 1.$$

This yields a characterization of bad $\bar{c}_i$'s:

$$\bar{c}_i \text{ bad} \iff \sum_{j=1}^{n} (\alpha'_j - \alpha''_j) \cdot \bar{c}_{i_j} = 0 \qquad (18)$$

Let $x^{\alpha'}, x^{\alpha''}$ be a fixed pair of monomials. Assume that there are n distinct vectors $\bar{c}_i^{(1)},\ldots,\bar{c}_i^{(n)}$ which are bad for $x^{\alpha'}, x^{\alpha''}$. Then, by (18), the submatrix C built from $\bar{c}_i^{(1)},\ldots,\bar{c}_i^{(n)}$ would be

singular, because:

$$\underbrace{\begin{pmatrix} \bar{c}_{i,1}^{(1)} & \dots & \bar{c}_{i,1}^{(1)} \\ \vdots & & \vdots \\ \bar{c}_{i,1}^{(n)} & \dots & \bar{c}_{i,1}^{(n)} \end{pmatrix}}_{=: C} \cdot \begin{pmatrix} \alpha_1' - \alpha_1'' \\ \vdots \\ \alpha_n' - \alpha_n'' \end{pmatrix} = 0.$$

Thereby the kernel of C would not equal $\{0\}$ and C would be singular. But this is a contradiction to Lemma 4.18. Hence, we are guaranteed, that for a fixed pair of monomials $x^{\alpha'}, x^{\alpha''}$ there are at most $(n-1)$ bad vectors.

While f is t-sparse there are at most $\binom{t}{2}$ distinct pairs of monomials, therefore there are at most $(n-1)\cdot\binom{t}{2}$ bad vectors for arbitrary $\alpha' \neq \alpha''$. Since

$$(n-1)\cdot\binom{t}{2} = \frac{4nq\cdot(n-1)\cdot\binom{t}{2}}{4nq} < \frac{\lceil q^s - 1\rceil}{4nq} = N$$

we are guaranteed that there exists an $i_0, 1 \leq i_0 \leq N$ satisfying $\Omega_{\alpha',i_0} \neq \Omega_{\alpha'',i_0}$ for any $\alpha' \neq \alpha''$. Hence c_{i_0} separates all monomials.

Let $\Omega_\alpha = \Omega_{\alpha,i_0}$ and $f_l := f(\omega^{l\cdot\bar{c}_{i_0}})$. Then for all $0 \leq l < |\mathrm{supp}(f)| < t$ we have

$$\begin{aligned} f_l &= \sum_{\alpha\in\mathrm{supp}(f)} c_\alpha \cdot x^\alpha|_{\omega^{l\,\bar{c}_{i_0}}} \\ &= \sum_{\alpha\in\mathrm{supp}(f)} c_\alpha \cdot (\omega^{\bar{c}_{i_0}\cdot\alpha})^l \\ &= \sum_{\alpha\in\mathrm{supp}(f)} c_\alpha \cdot \Omega_\alpha^l. \end{aligned}$$

We obtain the following linear system of equations:

$$\underbrace{\left(\Omega_\alpha^l\right)_{\substack{0\leq l<|\mathrm{supp}(f)| \\ \alpha\in\mathrm{supp}(f)}}}_{=: \mathcal{V}} \cdot (c_\alpha)_{\alpha\in\mathrm{supp}(f)} = (f_l)_{0\leq l<|\mathrm{supp}(f)|}. \tag{19}$$

$\mathcal{V}$ is a non-singular Vandermonde matrix (because $\alpha \neq (0,\dots,0)$), and the linear system of equations (19) has a unique solution. Hence:

$$f \equiv 0 \iff (c_\alpha)_{\alpha\in\mathrm{supp}(f)} = 0 \iff (f_l)_{0\leq l<|\mathrm{supp}(f)|} = 0$$

This concludes the proof. □

4.3.2 The Enumeration Technique

In the following, we intend to determine partial solutions of the interpolation problem by reduction to the zero test. We solve the prime interpolation problem by the efficient composition of the partial solutions (cf. Section 2.3). Assume $n = 2^m$ for simplicity of notation.

The partial solutions are of the following form:

$$S_{\alpha,\beta} = \{(k_1,\dots,k_{2^{\alpha-1}}) \mid x_{\beta\cdot 2^{\alpha-1}+1}^{k_1} \cdot \ldots \cdot x_{\beta\cdot 2^{\alpha-1}+2^{\alpha-1}}^{k_{2^{\alpha-1}}} \text{ occurs in some monomial of } f\}$$

with $1 \leq \alpha \leq m+1$ and $0 \leq \beta < 2^{m+1-\alpha}$.
For $\alpha = 1$ and $0 \leq \beta < n$ we have:

$$\begin{aligned} S_{\alpha,\beta} &= \{k_1 \mid x_{\beta+1}^{k_1} \text{ occurs in some monomial of } f\} \\ &= \text{set of exponents of } x_{\beta+1} \text{ occurring in } f. \end{aligned}$$

$\alpha = m+1$, i.e. $\beta = 0$ yields the solution of the interpolation problem:

$$S_{m+1,0} = \{(k_1, \ldots, k_n) \mid x_1^{k_1} \cdot \ldots \cdot x_n^{k_n} \text{ occurs in some monomial of } f\}.$$

Given the sets $S_{1,\beta}$ we want to construct the sets $S_{\alpha,\beta}$ for $\alpha = 1, \ldots, m+1$ recursively.

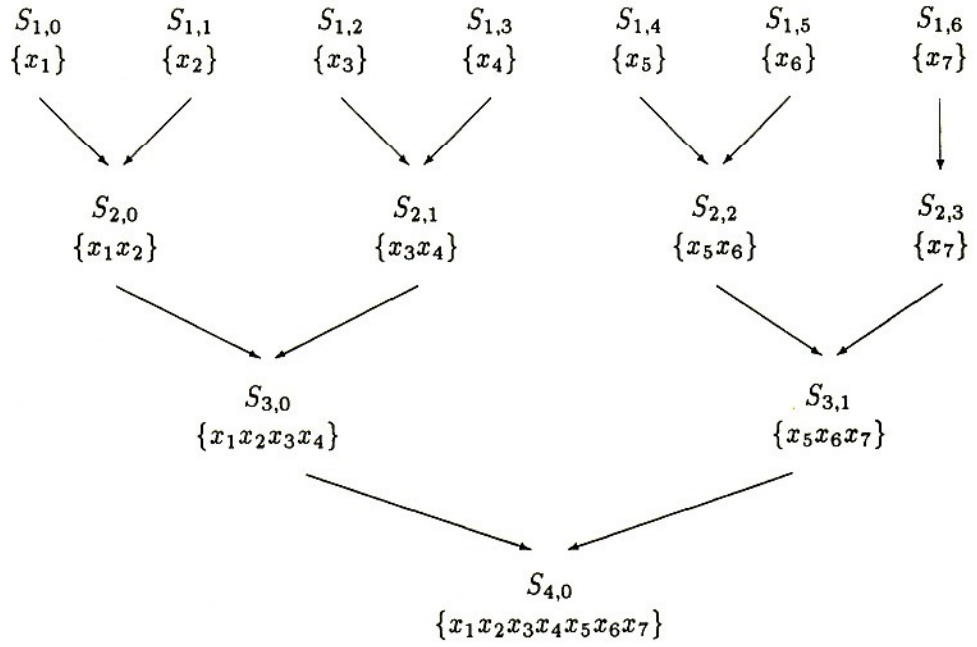

Figure 1: recursion scheme for $n = 7, m = \lceil \log_2 n \rceil = 3$

4.3.3 The Basis Step

We determine $S_{1,\beta}$ for $0 \leq \beta < n$, i.e. for all variables we determine the corresponding exponents occurring in f. Rewrite f according to powers of $x_{\beta+1}$:

$$f(x) = \sum_{l=0}^{q-1} x_{\beta+1}^l \cdot P_{l,\beta+1}(\underbrace{x_1, \ldots, x_\beta}_{=: x'}, \underbrace{x_{\beta+2}, \ldots, x_n}_{=: x''}),$$

where $P_{l,\beta+1} \in \mathrm{GF}(q)[x', x'']$. Then

$$S_{1,\beta} = \{l \mid P_{l,\beta+1} \not\equiv 0\}.$$

We intend to apply the zero test [GKS 88] to $P_{l,\beta+1}$. Therefore we have to construct a black box for $P_{l,\beta+1}$ from the given black box for f. Let $\mathrm{GF}(q) = \{a_1, \ldots, a_q\}$. Then for $1 \leq i \leq q$:

$$f(x', a_i, x'') = \sum_{l=0}^{q-1} a_i^l \cdot P_{l,\beta+1}(x', x'').$$

We obtain the following linear system of equations:

$$\underbrace{\begin{pmatrix} a_1^0 & \dots & a_1^{q-1} \\ \vdots & & \vdots \\ a_q^0 & \dots & a_q^{q-1} \end{pmatrix}}_{=:\mathcal{A}} \cdot \underbrace{\begin{pmatrix} P_{0,\beta+1}(x',x'') \\ \vdots \\ P_{q-1,\beta+1}(x',x'') \end{pmatrix}}_{=:\mathcal{P}_l} = \underbrace{\begin{pmatrix} f(x',a_1,x'') \\ \vdots \\ f(x',a_q,x'') \end{pmatrix}}_{=:\mathcal{F}}$$

$\mathcal{A}$ is a non-singular Vandermonde matrix, therefore the system of equations has a unique solution. With

$$\begin{aligned} P_{l,\beta+1}(x',x'') &= (0,\dots,0,1,0,\dots,0)\cdot\mathcal{P}_l \\ &= \underbrace{(0,\dots,0,1,0,\dots,0)}_{:=u_l}\cdot\mathcal{A}^{-1}\cdot\mathcal{F} \end{aligned}$$

we are given a black box for $P_{l,\beta+1}$, where u_l has 1 in the l-th position. □

4.3.4 The Recursion Step

Assume that for fixed $\alpha < m+1$ the sets $S_{\alpha,\beta}$ for $0 \le \beta < 2^{m+1-\alpha}$ are determined. We determine $S_{\alpha+1,\beta}$ from $S_{\alpha,2\beta}$ and $S_{\alpha,2\beta+1}$. Consider the construction of the partial solutions:

$$\begin{aligned} & \overbrace{x_{\beta\cdot 2^{\alpha+1}}^{k_1}\cdot\ldots\cdot x_{\beta\cdot 2^{\alpha+2^{\alpha-1}}}^{k_{2^{\alpha-1}}}\cdot x_{\beta\cdot 2^{\alpha+2^{\alpha-1}+1}}^{k_{2^{\alpha-1}+1}}\cdot\ldots\cdot x_{\beta\cdot 2^{\alpha+2^{\alpha}}}^{k_{2^{\alpha}}}}^{k\in S_{\alpha+1,\beta}} \\ = \; & \underbrace{x_{2\beta\cdot 2^{\alpha-1}+1}^{k_1}\cdot\ldots\cdot x_{2\beta\cdot 2^{\alpha-1}+2^{\alpha-1}}^{k_{2^{\alpha-1}}}}_{k\in S_{\alpha,2\beta}}\cdot\underbrace{x_{(2\beta+1)\cdot 2^{\alpha-1}+1}^{k'_1}\cdot\ldots\cdot x_{(2\beta+1)\cdot 2^{\alpha-1}+2^{\alpha-1}}^{k'_{2^{\alpha-1}}}}_{k'\in S_{\alpha,2\beta+1}}\cdot \end{aligned}$$

Let $S := S_{\alpha,2\beta}\times S_{\alpha,2\beta+1}$, i.e.

$$S = \{u\cdot v \mid u\in S_{\alpha,2\beta}, v\in S_{\alpha,2\beta+1}\}.$$

Obviously $S \supseteq S_{\alpha+1,\beta}$ and $S_{\alpha+1,\beta} = \{u\in S \mid u \text{ occurs in some monomial of } f\}$. Since f is t-sparse $|S_{\alpha,2\beta}| \le t$ and $|S_{\alpha,2\beta+1}| \le t$, hence $|S| \le t^2$.

The further proceedings are similar to the basis step. Rewrite f according to the elements in S, i.e. according to the corresponding polynomials in 2^α variables. Then we have to decide whether the corresponding coefficient polynomials are identical to zero. This is carried out by solving a system of equations with matrices whose entries are powers of values of elements in S, i.e. values of monomials in 2^α variables. In order to guarantee that the resulting systems of equations are solvable these values have to result in a non-singular matrix, i.e. we have to separate the monomials. This was easy for the basis step since monomials in one variable are separated by the elements in GF(q). For the recursion step we proceed similar to the zero test.

For this purpose determine a minimal s_1 satisfying $q^{s_1}-1 > 4\cdot q\cdot n\cdot(n-1)\cdot\binom{t^2}{2}$, i.e. $s_1 \le \lceil 4\log_q(nt)+3\rceil$. Construct the field GF(q^{s_1}) and a primitive element ω_1 in GF(q^{s_1}). Let $N_1 = \frac{\lceil q^{s_1}-1\rceil}{4nq}$, thereby $N_1 > (n-1)\cdot\binom{t^2}{2}$. Find a prime p_1 satisfying $2N_1 < p_1 \le 4N_1$. Construct the $(N_1\times N_1)$ Cauchy matrix $C=(c_{ij})$ with $c_{ij} := \frac{1}{i+j} \bmod p_1$ for $1\le i,j\le N_1$. Let V be an arbitrary $(N_1\times 2^\alpha)$-submatrix of C.

Then, as shown in Theorem 4.19, there exists some row v_0 of V such that the elements of S are separated at point $\omega_1^{v_0}$. Let $x^u = x_{\beta\cdot 2^\alpha+1}^{u_1} \cdot \ldots \cdot x_{\beta\cdot 2^\alpha+2^\alpha}^{u_{2^\alpha}}$ and $\Omega_u := x^u|_{\omega_1^{v_0}} = \omega_1^{u\cdot v_0}$ for $u \in S$. Then

$$u_1, u_2 \in S, u_1 \neq u_2 \Longrightarrow \Omega_{u_1} \neq \Omega_{u_2}. \tag{20}$$

Let $x' = x_1 \cdot \ldots \cdot x_{\beta\cdot 2^\alpha}$ and $x'' = x_{(\beta+1)\cdot 2^\alpha+1} \cdot \ldots \cdot x_n$. Rewrite f according to the elements of S:

$$f(x', x, x'') = \sum_{u\in S} x^u \cdot P_u(x', x''), \tag{21}$$

where $P_u \in \mathrm{GF}(q)[x', x'']$. Then $S_{\alpha+1,\beta} = \{u \mid P_u \not\equiv 0\}$.

Similar to the basis step we intend to apply the zero test [GKS 88] to P_u. Therefore we construct a black box for P_u from the given black box for f. We evaluate (21) at the points $x = \omega_1^{v_0\cdot l}$ for $0 \leq l < |S|$:

$$f(x', \omega_1^{v_{01}\cdot l}, \ldots, \omega_1^{v_{02^\alpha}\cdot l}, x'') = \sum_{u\in S} \Omega_u^l \cdot P_u(x', x'').$$

We obtain the following linear system of equations:

$$\underbrace{\left(\Omega_u^l\right)_{\substack{0\leq l<|S| \\ u\in S}}}_{=\mathcal{B}} \cdot \underbrace{\left(P_u(x', x'')\right)_{u\in S}}_{=:\mathcal{P}} = \underbrace{\left(f(x', \omega_1^{v_0\cdot l}, x'')\right)_{0\leq l<|S|}}_{=:\mathcal{F}} \tag{22}$$

By (20) $\mathcal{B}$ is a non-singular Vandermonde matrix, hence the system of equations (22) has a unique solution. With

$$\begin{aligned} P_u(x', x'') &= (0, \ldots, 0, 1, 0, \ldots, 0) \cdot \mathcal{P} \\ &= \underbrace{(0, \ldots, 0, 1, 0, \ldots, 0)}_{:= s_u} \cdot \mathcal{B}^{-1} \cdot \mathcal{F} \end{aligned}$$

we are given a black box for P_u, where s_u has 1 at the position corresponding to the position of u in S.

For $\alpha = m + 1$ all monomials occurring in f are determined. The corresponding coefficients are given by the solution of (19) □

4.3.5 The NC-Interpolation Algorithm for Sparse Polynomials

Interpolation Algorithm [GKS 88]

Input: Black box for $f \in \mathrm{GF}(q^s)[x_1, \ldots, x_n]$, t-sparse.

Output: Sparse Representation of f: $(c_u, u)_{u\in \mathrm{supp}(f)}$.

Step A: Execute Step 1–5 of the zero test and put the zero test set

$$\omega^{l\cdot \bar{c}_i} = (\omega^{l\cdot \bar{c}_{i,1}}, \omega^{l\cdot \bar{c}_{i,2}}, \ldots, \omega^{l\cdot \bar{c}_{i,n}}) \quad \text{for } 0 \leq i \leq N,\ 1 \leq l < t$$

at disposal.

Step B: For the purpose of the basis step compute the matrix

$$\mathcal{A}^{-1} = \left(a_k^j\right)_{\substack{a_k\in \mathrm{GF}(q) \\ 0\leq j<q}}^{-1}.$$

Step C: Execute the basis step in parallel for any $0 \leq \beta < n$.

Basis Step

Step 1: Query the black box with $s = \lceil \log_q(4qn(n-1)\binom{t}{2} + 2) \rceil$ for the values

$$f_{i,l}^{(k)} := f(\omega^{l \cdot \bar{c}_{i,1}}, \ldots, \omega^{l \cdot \bar{c}_{i,\beta}}, a_k, \omega^{l \cdot \bar{c}_{i,\beta+2}}, \ldots, \omega^{l \cdot \bar{c}_{i,n}})$$

for $0 \leq l < t$, $1 \leq i \leq N$, $1 \leq k \leq q$ and for the values

$$f_{0,t}^{(k)} := f(0, \ldots, 0, a_k, 0, \ldots, 0) \quad \text{for } 1 \leq k \leq q.$$

Step 2: Compute in parallel for $0 \leq l \leq t$, $0 \leq i < N$ the vectors

$$P_{i,l} := \mathcal{A}^{-1} \cdot \left(f_{i,l}^{(k)}\right)_{1 \leq k \leq q}.$$

Step 3: Let $S_{1,\beta} = \{\, k \mid \exists (i,l) \,:\, P_{i,l}^{(k)} \neq 0 \,\}$.

Step D: Prepare the recursion step:

Step 4: Determine a minimal s_1 satisfying $q^{s_1} - 1 > 4qn \cdot (n-1) \cdot \binom{t^2}{2}$, i.e. let $s_1 = \lceil \log_q(4qn(n-1)\binom{t^2}{2} + 2) \rceil$.

Step 5: Construct the field GF(q^{s_1}) and a primitive element ω_1 in GF(q^{s_1}) using the Berlekamp algorithm.

Step 6: Let $N_1 := \lceil \frac{q^{s_1}-1}{4nq} \rceil$. Use the sieve of Erastosthenes to find a prime p_1 with $2N_1 < p_1 \leq 4N_1$.

Step 7: Compute $c_{ij} := \frac{1}{i+j} \bmod p_1$ for $1 \leq i, j \leq N_1$ by means of the Euclidean algorithm. Let $C = (c_{i,j})_{1 \leq i,j \leq N_1}$.

Step 8: Denote by $V = (v_{i,j})$ an arbitrary $(N_1 \times n)$-submatrix of C.

Step 9: Put the test set

$$\omega_1^{l \cdot v_i} = (\omega_1^{l \cdot v_{i,1}}, \omega_1^{l \cdot v_{i,2}}, \ldots, \omega_1^{l \cdot v_{i,n}}) \quad \text{for } 0 \leq i \leq N,\ 0 \leq l < t$$

at disposal.

Step E: Let $m = \lceil \log_2 n \rceil$. Execute step F in sequential for $\alpha = 1, \ldots, m$.

Step F: Execute the recursion step in parallel for any $0 \leq \beta < 2^{m-\alpha}$.

Recursion Step

Step 10: Let $S := S_{\alpha,2\beta} \times S_{\alpha,2\beta+1}$

Step 11: Determine in parallel for $1 \leq j \leq N_1$ and $u \in S$ the sums

$$\sigma_{u,j} = \sum_{i=1}^{2^\alpha} u_i \cdot v_{j,i}.$$

Determine a row v_j of V such that the set $\{\sigma_{u,j} \mid u \in S\}$ consists of pairwise distinct elements. Compute $\Omega_u = \omega_1^{\sigma_{u,j}}$, $u \in S$.

Step 12: Compute the matrix

$$\mathcal{B}^{-1} = \left(\Omega_u^k\right)^{-1}_{\substack{0 \le k < |S| \\ u \in S}}.$$

Step 13: Query the black box with $s = s_1$ for the values

$$f_{i,l}^{(u)} := f(\omega^{l \cdot \bar{c}_{i,1}}, \ldots, \omega^{l \cdot \bar{c}_{i,\beta\, 2^\alpha}}, \omega_1^{k \cdot v_{j,1}}, \ldots, \omega_1^{k \cdot v_{j,2^\alpha}}, \omega^{l \cdot \bar{c}_{i,(\beta+1)\, 2^\alpha + 1}}, \ldots, \omega^{l \cdot \bar{c}_{i,n}})$$

for $0 \le l < t$, $1 \le i \le N$, $0 \le k < |S|$ and for the values

$$f_{0,t}^{(k)} := f(0, \ldots, 0, \omega_1^{k \cdot v_{j,1}}, \ldots, \omega_1^{k \cdot v_{j,2^\alpha}}, 0, \ldots, 0) \quad \text{for } 0 \le k < |S|.$$

Step 14: Compute in parallel for $0 \le l \le t$, $0 \le i < N$ the vectors

$$P_{i,l} = \mathcal{B}^{-1} \cdot \left(f_{i,l}^{(u)}\right)_{u \in S}.$$

Step 15: Let $S_{\alpha+1,\beta} = \{\, u \mid \exists (i,l) \,:\, P_{i,l}^{(u)} \neq 0 \,\}$.

Step G: Output $\mathrm{supp}(f) = S_{m+1,0}$. The coefficients of f results from the components of a vector $P_{i,l}$ of the last recursion step.

4.3.6 Analysis of the Algorithm

We start with estimating the order of s, s_1, N and N_1. We note that $s \le \lceil 3 + 2\log_q(nt) \rceil$, hence $s = O(\log_q(nt))$. From the definition of N we derive $N < qnt^2$ and similar for N_1, $N_1 < qnt^4$. Furthermore $|\mathrm{GF}(q^s)| \le Nnq$ and $|\mathrm{GF}(q^{s_1})| \le N_1 nq$.

Analysis of the zero test

In order to construct the field $\mathrm{GF}(q^s)$, we search for an irreducible polynomial $\Phi \in \mathrm{GF}(q)[z]$ of degree s. Testing reducibility for the q^{s+1} univariate polynomials of degree s using the algorithm of Berlekamp [Be 70] yields an irreducible Φ. This takes $q^{s+1}O(\log^{5.5}(Nnq)) = O(Nnq^2 \log^{5.5}(Nnq))$ processors and $O(\log^2(Nnq))$ parallel time. $\mathrm{GF}(q^s)$ is isomorphic to $\mathrm{GF}(q)[z]/\Phi$. The arithmetic in $\mathrm{GF}(q^s)$ is represented by the arithmetic for polynomial in $\mathrm{GF}(q)$ of degree s and modulo reduction according to Φ. If the polynomials are e.g. represented by their companion matrices, the factor for the arithmetic in $\mathrm{GF}(q^s)$ compared to $\mathrm{GF}(q)$ is at most $O(\log^{2.5}(ntq))$ processors and $O(\log(ntq))$ parallel time.

In order to find a primitive element ω in $\mathrm{GF}(q^s)$ we compute the prime factorization of $q^s - 1$. Let $q^s - 1 = \prod p_i^{\alpha_i}$. This takes $O((Nnq)^{0.75} \log(Nnq))$ processors and $O(\log(Nnq))$ parallel time using the sieve of Erastosthenes. Then we test for any $a \in \mathrm{GF}(q^s)$ whether the powers $a^{\frac{q^s-1}{p_i}}$ are distinct from one. In this case a is a primitive element of $\mathrm{GF}(q^s)$. Calculating the powers for all $a \in \mathrm{GF}(q^s)$ takes at most $O((Nnq)\log^{3.5}(Nnq))$ processors and $O(\log^2(Nnq))$ parallel time.

The next major step is the calculation of the evaluation points, that is we have to calculate Nnt powers of ω taking $O(Nnt \log^{2.5}(Nnq))$ processors and $O(\log^2(Nnqt))$ parallel time.

In summary, the zero test takes $O(Nnqt \log^{3.5}(Nnq))$ processors and $O(\log^2(Nnqt))$ parallel time.

Analysis of the basis step

The complexity is determined by the complexity of the zero test and the complexity for inverting the $(q \times q)$-matrix $A = (a_i^j)_{\substack{a_i \in GF(q) \\ 0 \le j < q}}$. This takes $O(q^{2.5} \log^2 q)$ processors and $O(\log^2 q)$ parallel time using [Mu 86]. Hence the complexity of the basic step is $O(Nnqt \log^{3.5}(Nnq) + q^{2.5} \log^2 q)$ processors and $O(\log^2(Nnqt))$ parallel time. We query the black box in parallel for tqN arguments.

Complexity of the preparation of the recursion steps

We have to construct the field $GF(q^{s_1})$ and to put the test set at disposal. As above this takes $O(N_1 nqt \log^{3.5}(N_1 nq))$ processors and $O(\log^2(N_1 nqt))$ parallel time.

Analysis of the recursions step

The recursion step is executed in parallel for $0 \le \beta < 2^{m-\alpha}$. In order to construct the black box for P_u, we have to compute the matrix B in (22). For this purpose we compute the $N_1 \cdot t^2$ sums $\sigma_{u,j} = \sum_{i=1}^{2^\alpha} u_i \cdot v_{j,i}$ for $u \in S$ and $1 \le j \le N_1$ and test whether these sums are pairwise distinct for a fixed j. Denote this row by v_0. With $\Omega_u = \omega_1^{\sigma_{u,0}}$ for $u \in S$ we obtain the $(t^2 \times t^2)$-matrix B. This takes $O(N_1 t^2 \log^{2.5}(N_1 nq))$ processors and $O(\log^2(N_1 nqt))$ parallel time. Inverting the matrix B takes $O(t^5 \log^{2.5}(N_1 nq))$ processors and $O(\log^2 t)$ parallel time. Step 13 queries the black box for $O(t^3 N)$ evaluations. In step 14 tN vectors of length $O(t^2)$ over $GF(q^{s_1})$ are computed in parallel. This takes $O(Nt^3 \log^2(N_1 nq))$ processors and $O(\log t)$ parallel time.

Hence the the complexity of one recursion step is estimated by $2^{m-\alpha}(O(N_1 t^2 \log^{2.5}(N_1 nq)) + O(t^5 \log^{2.5}(N_1 nq)))$ processors, $O(\log^2(N_1 nqt))$ parallel time and $2^{m-\alpha}(O(t^3 N))$ queries.

Complexity of the algorithm

Summing over $O(\log n)$ recursions and using the estimates of N and N_1 yields

- $O(t^6 n^2 q^3 \log^{5.5}(ntq) + q^{2.5} \log^2 q)$ processors,
- $O(\log^3(ntq))$ parallel time and
- $O(qn^2 t^5)$ queries

as claimed in Theorem 4.16. □

5 NC-Interpolation of Rational Functions

In this chapter we consider the problem of the rational interpolation. Let

$$f(x_1,\ldots,x_n) = \frac{P(x_1,\ldots,x_n)}{Q(x_1,\ldots,x_n)}$$

be a nonreducible representation of f, with $P,Q \in I\!K[x_1,\ldots,x_n]$, t-sparse with $\deg_{x_i}(P)$, $\deg_{x_i}(Q) < d$ and $(P,Q) = 1$.

While $(f \equiv 0) \iff (P \equiv 0)$ the test of f to zero can directly be reduced to the zero test for t-sparse polynomials of [GK 87] and [BT 88] (cf. Section 2.1, 3.2), i.e.

$$f \equiv 0 \iff f(p_1^l,\ldots,p_n^l) = 0 \quad \text{for all } 0 \le l < t$$

for pairwise distinct primes $p_1,\ldots,p_n$.

Similarly, we can easily solve the problem whether two rational functions $f_1 = \frac{P_1}{Q_1}$ and $f_2 = \frac{P_2}{Q_2}$ are identical. It holds:

$$f_1 \equiv f_2 \iff \underbrace{P_1 \cdot Q_2}_{t^2\text{-sparse}} \equiv \underbrace{P_2 \cdot Q_1}_{t^2\text{-sparse}} \iff \underbrace{P_1 \cdot Q_2 - P_2 \cdot Q_1}_{2t^2\text{-sparse}} \equiv 0.$$

Again, we can apply the zero test for $2t^2$-sparse polynomials:

$$f_1 \equiv f_2 \iff f_1(p_1^l,\ldots,p_n^l) = f_2(p_1^l,\ldots,p_n^l) \quad \text{for all } 0 \le l < 2t^2.$$

The problem of reconstructing rational functions is much harder to solve. Until recently it was unknown whether this problem is efficiently solvable, e.g. Ben-Or and Tiwari ([BT 88]) reduced the reconstruction to the determination of sparse vectors in the null space of a certain matrix, however, for general matrices this problem is known to be NP-complete.

Eventually, Grigoriev and Karpinski ([GK 88]) made the breakthrough. They show that the techniques introduced in the previous chapters for the separation of monomials ([GK 87], [BT 88]) and the technique for the enumeration of partial solutions ([GKS 88]) are sufficiently powerful to construct a deterministic NC-algorithm for interpolation of t-sparse rational functions.

Theorem 5.1 Let $f \in \mathbb{Z}(x_1,\ldots,x_n)$ be a rational function with the representation $f = \frac{P}{Q}$, where $P,Q \in \mathbb{Z}[x_1,\ldots,x_n]$ are t-sparse polynomials with $\deg_{x_i}(P)$, $\deg_{x_i}(Q) < d$ for $1 \le i \le n$ and $(P,Q) = 1$.
Then there exists a deterministic parallel algorithm (NC^3) interpolating f. The algorithm takes $O(\log^3(ndt))$ parallel time and $O((ntd^{7.5} + n^3d^2t^{17.5})\log(ntd))$ processors.

The algorithm involves two major tools: (1) the efficient zero test for t-sparse polynomials ([GK 87], [BT 88]) and (2) the recursive composition of partial monomials used in Section 4.3.

By Theorem 3.1 a t-sparse polynomial P is identical to zero iff P vanishes at the points $(p_1^i,\ldots,p_n^i)$ for $0 \le i < t$, where the p_i's are pairwise distinct primes.

We determine partial solutions of the interpolation problem by reduction to the zero test. The partial solutions are composed to the total solution of the interpolation problem as in Section 4.3.
Assume that the number of variables is a power of 2 for simplicity of notation. Let $n = 2^m$ and

$$f = \frac{P}{Q}, \quad \text{with } P, Q \in \mathbb{Z}[x_1, \ldots, x_n],$$

where P, Q are t-sparse and relatively prime, with $\deg_{x_i}(P), \deg_{x_i}(Q) < d$ for all $1 \leq i \leq n$.

Define partial solutions for P and Q (cf. Section 4.3):

$$\begin{aligned} S^{(1)}_{\alpha,\beta} &= \{I = (i_1, \ldots, i_{2^{\alpha-1}}) \mid x^{i_1}_{\beta \cdot 2^{\alpha-1}+1} \cdot \ldots \cdot x^{i_{2^{\alpha-1}}}_{\beta \cdot 2^{\alpha-1}+2^{\alpha-1}} \text{ occurs in } P\} \quad \text{and} \\ S^{(2)}_{\alpha,\beta} &= \{J = (j_1, \ldots, j_{2^{\alpha-1}}) \mid x^{j_1}_{\beta \cdot 2^{\alpha-1}+1} \cdot \ldots \cdot x^{j_{2^{\alpha-1}}}_{\beta \cdot 2^{\alpha-1}+2^{\alpha-1}} \text{ occurs in } Q\} \end{aligned}$$

with $1 \leq \alpha \leq m+1$ and $0 \leq \beta < 2^{m+1-\alpha}$.

Starting with $S^{(1)}_{1,\beta}$ and $S^{(2)}_{1,\beta}$ we determine $S^{(1)}_{\alpha,\beta}$ and $S^{(2)}_{\alpha,\beta}$ recursively for $\alpha = 1, \ldots, m+1$. In addition, we have to guarantee that the partial solutions correspond to the nonreducible representation of f.

5.1 The Basis Step

For each variable we determine the set of exponents occurring in P and Q corresponding to the nonreducible representation of f, i.e. we determine in parallel the sets $S^{(1)}_{1,j-1}$ and $S^{(2)}_{1,j-1}$ for some fixed j with $0 \leq j < n$.

Rewrite P and Q according to powers of x_j. Let $x' = x_1, \ldots, x_{j-1}$ and $x'' = x_{j+1}, \ldots, x_n$.

$$f(x) = \frac{\sum_{i=0}^{d} x_j^i \cdot P_i^{(1)}(x', x'')}{\sum_{i=0}^{d} x_j^i \cdot Q_i^{(2)}(x', x'')} \quad \text{where } P_i^{(1)}, Q_i^{(2)} \in \mathbb{Z}[x', x'']. \tag{23}$$

We are interested in black boxes for $P_i^{(1)}$ and $Q_i^{(2)}$. However, substituting distinct values a_l for x_j and considering the system of equations derived from (23) does not lead in general to the nonreducible representation of f.
Therefore we consider the following representation of f for $0 \leq k_1, k_2 < d$ (this representation is perhaps invalid).

$$f(x) = \frac{\sum_{i=0}^{k_1} x_j^i \cdot P_i(x', x'')}{\sum_{i=0}^{k_2} x_j^i \cdot Q_i(x', x'')} \quad \text{with } P_i, Q_i \in \mathbb{Z}[x', x'']. \tag{24}$$

Among the valid representations we fix the one with the minimal k_1. This representation corresponds to the nonreducible representation of f: Suppose there are two different valid representations of f with k_1 and k_1' where $k_1 < k_1'$ and consider the unique factorization of the

numerator polynomial of both representations. Then k_1' cannot corresponds to the nonreducible representation because its factorization must contain an additional factor compared with the factorization of the representation with k_1.

Let $a_l \in \mathbb{Z}$, $0 \le l \le 2d$ be pairwise distinct numbers substituted in (24) for x_j then we have for all $0 \le l \le 2d$:

$$\sum_{i=0}^{k_1} a_l^i \cdot P_i(x', x'') = f(x', a_l, x'') \cdot \sum_{i=0}^{k_2} a_l^i \cdot Q_i(x', x''). \tag{25}$$

We show in proof of Lemma 5.2 that validity of (25) implies that (24) is a valid representation of f.

Interpreting $P_0, \ldots, P_{k_1}, Q_0, \ldots, Q_{k_2}$ as indeterminates yields the following system of equations. The coefficients are determined by means of the given black box for f.

$$\underbrace{\begin{pmatrix} a_0^0 & \ldots & a_0^{k_1} \\ \vdots & & \vdots \\ a_{2d}^0 & \ldots & a_{2d}^{k_1} \end{pmatrix}}_{=:\mathcal{A}'} \cdot \begin{pmatrix} P_0 \\ \vdots \\ P_{k_1} \end{pmatrix} = \begin{pmatrix} f_{a_0} & & 0 \\ & \ddots & \\ 0 & & f_{a_{2d}} \end{pmatrix} \cdot \underbrace{\begin{pmatrix} a_0^0 & \ldots & a_0^{k_2} \\ \vdots & & \vdots \\ a_{2d}^0 & \ldots & a_{2d}^{k_2} \end{pmatrix}}_{=:\mathcal{A}''} \cdot \begin{pmatrix} Q_0 \\ \vdots \\ Q_{k_2} \end{pmatrix} \tag{26}$$

where $f_{a_l} = f(x', a_l, x'')$ and P_i, Q_i corresponds to $P_i(x', x'')$, $Q_i(x', x'')$. Let

$$S_{1,j-1}^{(1)}(k_1) = \{i \mid P_i \not\equiv 0\} \quad \text{and} \quad S_{1,j-1}^{(2)}(k_2) = \{i \mid Q_i \not\equiv 0\}.$$

If $P_{k_1} \not\equiv 0, Q_{k_2} \not\equiv 0$. then the pair k_1, k_2 *fits* (there exists a representation $P/Q = f$ with $\deg_{x_j}(P) = k_1$, $\deg_{x_j}(Q) = k_2$).

Using the zero test supplied by Theorem 3.1, we test $P_{i_1}(x', x'')$, $Q_{i_2}(x', x'')$ to zero for $0 \le i_1 \le k_1$, $0 \le i_1 \le k_2$. We need to know whether P_{i_1}, Q_{i_2} vanish at the points $(p_1^i, \ldots, p_{n-1}^i)$ for $0 \le i < t$.

Consider the system of equations (26) for $(x', x'') = (p_1^i, \ldots, p_{n-1}^i)$ for $0 \le i < t$. Let $f_{l,i}$ denote $f_{a_l}(p_1^i, \ldots, p_{n-1}^i)$ and $P_{i_1,i}, Q_{i_2,i}$ denote $P_{i_1}(p_1^i, \ldots, p_{n-1}^i)$, $Q_{i_2}(p_1^i, \ldots, p_{n-1}^i)$.

$$\underbrace{\begin{pmatrix} a_0^0 & \ldots & a_0^{k_1} & -f_{0,i} \cdot a_0^0 & \ldots & -f_{0,i} \cdot a_0^{k_2} \\ \vdots & & \vdots & \vdots & & \vdots \\ a_{2d}^0 & \ldots & a_{2d}^{k_1} & -f_{2d,i} \cdot a_{2d}^0 & \ldots & -f_{2d,i} \cdot a_{2d}^{k_2} \end{pmatrix}}_{=:\mathcal{A}} \cdot \begin{pmatrix} P_{0,i} \\ \vdots \\ P_{k_1,i} \\ Q_{0,i} \\ \vdots \\ Q_{k_2,i} \end{pmatrix} = 0, \quad 0 \le i < t. \tag{27}$$

Let $a_{i,0}, a_{i,1}, \ldots, a_{i,k_1+k_2+1}$ be the rows of $\mathcal{A}$. Then we can represent (27) by:

$$c_0 \cdot a_{i,0} + c_1 \cdot a_{i,1} + \ldots + c_{k_1+k_2+1} \cdot a_{i,k_1+k_2+1} = 0, \tag{28}$$

where the coefficients c_i correspond to the evaluations of the coefficient polynomials P_{i_1}, Q_{i_2} at points $(p_1^i, \ldots, p_n^i)$. By means of the representation (28) the zero test for the coefficient polynomial with index s is equivalent to the test whether the vector $a_{i,s}$ for $0 \le i < t$ is linear independent from the vectors $a_{i,0}, \ldots, a_{i,s-1}, a_{i,s+1}, \ldots, a_{i,k_1+k_2+1}$.

This is tested by examining whether the rank of $\mathcal{A}$ changes when the column $a_{i,s}$ corresponding to s is canceled from $\mathcal{A}$. The computation of the rank can be done by means of the efficient NC-algorithm of Mulmuley ([Mu 86]).

Among all fitting pairs (k_1, k_2) where the corresponding representation of f by (24) is valid, we choose the minimal pair (e.g. the pair with the minimal k_1), and set

$$S^{(1)}_{1,j-1} = S^{(1)}_{1,j-1}(k_1) \quad \text{and} \quad S^{(2)}_{1,j-1} = S^{(2)}_{1,j-1}(k_2).$$

Lemma 5.2 The basis step is correct.

PROOF: Let k_1, k_2 be the minimal fitting pair. Consider the polynomials p_1, p_2 in the variables x_j over the polynomial ring $\mathbb{Z}[x', x'']$:

$$p_1(x_j) := (\sum_{i=0}^{k_1} x_j^i \cdot P_i) \cdot (\sum_{i=0}^{d} x_j^i \cdot Q_i^{(2)}) \quad \text{and} \quad p_2(x_j) := (\sum_{i=0}^{k_2} x_j^i \cdot Q_i) \cdot (\sum_{i=0}^{d} x_j^i \cdot P_i^{(1)}).$$

The degree of p_1, p_2 is at most $2d$ since $k_1, k_2 < d$. By (23) $P_i^{(1)}, Q_i^{(2)}$ correspond to the nonreducible representation of f. Furthermore P_i, Q_i satisfy the equation (24) for $x_j = a_l$, $0 \leq l \leq 2d$, hence:

$$\frac{\sum_{i=0}^{d} a_l^i \cdot P_i^{(1)}(x', x'')}{\sum_{i=0}^{d} a_l^i \cdot Q_i^{(2)}(x', x'')} = \frac{\sum_{i=0}^{k_1} a_l^i \cdot P_i(x', x'')}{\sum_{i=0}^{k_1} a_l^i \cdot Q_i(x', x'')}.$$

Thereby p_1 and p_2 coincide at $2d+1$ points, hence they are identical. Therefore

$$f(x) = \frac{\sum_{i=0}^{k_1} x_j^i \cdot P_i(x', x'')}{\sum_{i=0}^{k_1} x_j^i \cdot Q_i(x', x'')}$$

Since (k_1, k_2) is chosen as the minimal pair, $S^{(1)}_{1,j-1}(k_1)$, $S^{(2)}_{1,j-1}(k_2)$ correspond to the nonreducible representation of f. □

5.2 Recursion Step

Suppose that the sets $S^{(1)}_{\alpha,\beta}$ and $S^{(2)}_{\alpha,\beta}$ are determined for $0 \leq \beta < 2^{m+1-\alpha}$. For each $0 \leq \beta < 2^{m-\alpha}$ we construct in parallel the sets $S^{(1)}_{\alpha+1,\beta}$, $S^{(2)}_{\alpha+1,\beta}$ from $S^{(1)}_{\alpha,2\beta}$ and $S^{(1)}_{\alpha,2\beta+1}$, and from $S^{(2)}_{\alpha,2\beta}$ and $S^{(2)}_{\alpha,2\beta+1}$ (cf. Section 4.3).

Let $S^{(1)} := S^{(1)}_{\alpha,2\beta} \times S^{(1)}_{\alpha,2\beta+1}$ and $S^{(2)} := S^{(2)}_{\alpha,2\beta} \times S^{(2)}_{\alpha,2\beta+1}$,

$$S^{(1)} = \{u \cdot v \mid u \in S^{(1)}_{\alpha,2\beta}, v \in S^{(1)}_{\alpha,2\beta+1}\}$$
$$S^{(2)} = \{u \cdot v \mid u \in S^{(2)}_{\alpha,2\beta}, v \in S^{(2)}_{\alpha,2\beta+1}\}.$$

P and Q are t-sparse, therefore $|S^{(1)}|, |S^{(2)}| \leq t^2$. We have $S^{(1)}_{\alpha+1,\beta} \subseteq S^{(1)}$, $S^{(2)}_{\alpha+1,\beta} \subseteq S^{(2)}$, in the following we eliminate those multiindices $k \in S^{(1)}$, $k \in S^{(2)}$ whose corresponding partial monomial

$$\mathbf{x}^k = x^{k_1}_{\beta 2^\alpha+1} \cdot \ldots \cdot x^{k_{2^\alpha}}_{\beta 2^\alpha+2^\alpha}$$

does not occur in any monomial of P, Q.

Let $x' = x_1, \ldots, x_{\beta 2^\alpha}$ and $x'' = x_{(\beta+1)2^\alpha+1}, \ldots, x_n$. Rewrite P and Q according to the partial monomials in $S^{(1)}$, $S^{(2)}$. We obtain the nonreducible representation for f:

$$f(\mathbf{x}) = \frac{\sum\limits_{k^{(1)} \in S^{(1)}} \mathbf{x}^{k^{(1)}} \cdot \bar{P}_{k^{(1)}}(x', x'')}{\sum\limits_{k^{(2)} \in S^{(2)}} \mathbf{x}^{k^{(2)}} \cdot \bar{Q}_{k^{(2)}}(x', x'')} \quad \text{where } \bar{P}_{k^{(1)}}, \bar{Q}_{k^{(2)}} \in \mathbb{Z}[x', x'']. \tag{29}$$

Let $|k^{(1)}|$ for $k^{(1)} \in S^{(1)}$ be defined by $|k^{(1)}| := \sum\limits_{j=1}^{2^\alpha} k^{(1)}_j$ and $|k^{(2)}|$ for $k^{(2)} \in S^{(2)}$ similarly. It holds $0 \leq |k^{(1)}|, |k^{(2)}| < 2^\alpha d$, since $\deg_{x_i}(P) < d$, $\deg_{x_i}(Q) < d$.

In order to obtain the nonreducible representation for f we test in parallel similarly to the basis step whether for some pair k_1, k_2 the representation

$$f(\mathbf{x}) = \frac{\sum\limits_{|k^{(1)}| \leq k_1} \mathbf{x}^{k^{(1)}} \cdot P_{k^{(1)}}(x', x'')}{\sum\limits_{|k^{(2)}| \leq k_2} \mathbf{x}^{k^{(2)}} \cdot Q_{k^{(2)}}(x', x'')} \quad \text{where } P_{k^{(1)}}, Q_{k^{(2)}} \in \mathbb{Z}[x', x''] \tag{30}$$

is valid.

In the basis step we substitute pairwise distinct numbers $a_l \in \mathbb{Z}$, $0 \leq l \leq 2d$ for the variable x_j in order to guarantee that the matrices $\mathcal{A}'$ and $\mathcal{A}''$ of the resulting system of equations (26) have Vandermonde structure. Similarly, in the recursion step we have to separate the monomials in $S^{(1)}$, $S^{(2)}$ at evaluation points. Using the idea from [GK 87] we substitute powers of 2^α pairwise distinct primes $p^l_1, \ldots, p^l_{2^\alpha}$, $0 \leq l \leq 2t^3$ for $\mathbf{x}$. Let $\Omega_{k^{(1)}} = p_1^{k^{(1)}_1}, \ldots, p_{2^\alpha}^{k^{(1)}_{2^\alpha}}$ and let $\Omega_{k^{(2)}}$ be defined analogously.

Then by (30):

$$f_l = f(x', p^l_1, \ldots, p^l_{2^\alpha}, x'') = \frac{\sum\limits_{|k^{(1)}| \leq k_1} \Omega^l_{k^{(1)}} \cdot P_{k^{(1)}}}{\sum\limits_{|k^{(2)}| \leq k_2} \Omega^l_{k^{(2)}} \cdot Q_{k^{(2)}}} \quad \text{for all } 0 \leq l \leq 2t^3. \tag{31}$$

Again, solvability of (31) implies validity of (30). We obtain the system of equations:

$$\left(\Omega^l_{k^{(1)}}\right)_{\substack{0 \leq l \leq 2t^3 \\ |k^{(1)}| \leq k_1}} \cdot \left(P_{k^{(1)}}\right)_{|k^{(1)}| \leq k_1} = \begin{pmatrix} f_0 & & 0 \\ & \ddots & \\ 0 & & f_{2t^3} \end{pmatrix} \cdot \left(\Omega^l_{k^{(2)}}\right)_{\substack{0 \leq l \leq 2t^3 \\ |k^{(2)}| \leq k_2}} \cdot \left(Q_{k^{(2)}}\right)_{|k^{(2)}| \leq k_2} \tag{32}$$

and

$$\underbrace{\left(\left(\Omega^l_{k^{(1)}}\right)_{\substack{0\le l\le 2t^3\\ |k^{(1)}|\le k_1}} - \begin{pmatrix} f_0 & & 0\\ & \ddots & \\ 0 & & f_{2t^3}\end{pmatrix}\left(\Omega^l_{k^{(2)}}\right)_{\substack{0\le l\le 2t^3\\ |k^{(2)}|\le k_2}}\right)}_{=:\mathcal{B}} \cdot \begin{pmatrix} (P_{k^{(1)}})_{|k^{(1)}|\le k_1}\\ (Q_{k^{(2)}})_{|k^{(2)}|\le k_2}\end{pmatrix} = 0. \qquad (33)$$

Let

$$S^{(1)}_{\alpha+1,\beta}(k_1) = \{k^{(1)} \mid P_{k^{(1)}} \not\equiv 0\} \quad \text{and} \quad S^{(2)}_{\alpha+1,\beta}(k_2) = \{k^{(2)} \mid Q_{k^{(2)}} \not\equiv 0\}.$$

If $P_{k^{(1)}}, Q_{k^{(2)}} \not\equiv 0$ for some $|k^{(1)}| = k_1$, $|k^{(2)}| = k_2$ the pair k_1, k_2 *fits* (there exists a representation $P/Q = f$ with $\sum_{j=\beta 2^\alpha+1}^{(\beta+1)2^\alpha} \deg_{x_j}(P) = k_1$, $\sum_{j=\beta 2^\alpha+1}^{(\beta+1)2^\alpha} \deg_{x_j}(Q) = k_2$).

Consider the system of equations (33) for $(x', x'') = (p^l_1, \ldots, p^l_{n-2^\alpha})$ for $0 \le i < t$ and apply the zero test to $P_{k^{(1)}}$ and $Q_{k^{(2)}}$ supplied by Theorem 3.1. We determine those indices such that the rank of $\mathcal{B}$ changes when the column corresponding to the index is canceled from $\mathcal{B}$. The rank is computed using the NC-algorithm of Mulmuley ([Mu 86]).

Among all fitting pairs k_1, k_2 we choose the minimal one (e.g. the pair with minimal k_1) and set

$$S^{(1)}_{\alpha+1,\beta} = S^{(1)}_{\alpha+1,\beta}(k_1) \quad \text{and} \quad S^{(2)}_{\alpha+1,\beta} = S^{(2)}_{\alpha+1,\beta}(k_2).$$

Lemma 5.3 The recursion step is correct.

PROOF: Let k_1, k_2 be the minimal fitting pair. Then we have to show that equation (30) corresponds to the nonreducible representation (29) of f. P and Q are t-sparse, so are $\bar{P}_{k^{(1)}}$ and $\bar{Q}_{k^{(2)}}$.

Consider the polynomials p_1, p_2 in the variables $x_{\beta 2^\alpha+1} \cdot \ldots \cdot x_{\beta 2^\alpha+2^\alpha}$ over the polynomial ring $\mathbb{Z}[x', x'']$:

$$\begin{aligned} p_1(\mathbf{x}) &:= \Big(\sum_{|k^{(1)}|\le k_1} \mathbf{x}^{k^{(1)}} \cdot P_{k^{(1)}}\Big) \cdot \Big(\sum_{|k^{(2)}|\le d2^\alpha} \mathbf{x}^{k^{(2)}} \cdot \bar{Q}_{k^{(2)}}\Big),\\ p_2(\mathbf{x}) &:= \Big(\sum_{|k^{(2)}|\le k_2} \mathbf{x}^{k^{(2)}} \cdot Q_{k^{(2)}}\Big) \cdot \Big(\sum_{|k^{(1)}|\le d2^\alpha} \mathbf{x}^{k^{(1)}} \cdot \bar{P}_{k^{(1)}}\Big). \end{aligned}$$

Since P and Q are t-sparse the number of $\bar{P}_{k^{(1)}} \not\equiv 0$, $\bar{Q}_{k^{(2)}} \not\equiv 0$ is at most t. Furthermore $|S^{(1)}|, |S^{(2)}| \le t^2$, hence p_1 and p_2 have at most t^3 terms in the variables $x_{\beta 2^\alpha+1} \cdot \ldots \cdot x_{\beta 2^\alpha+2^\alpha}$. Let S be the set of the occurring multiindices. Rewrite $p := p_1 - p_2$ as

$$p = \sum_{k\in S} \mathbf{x}^k \cdot T_k \qquad \text{with } T_k \in \mathbb{Z}[x', x''].$$

$P_{k^{(1)}}, Q_{k^{(2)}}$ satisfy the equation (30) with $x_{\beta 2^\alpha+1} \cdot \ldots \cdot x_{\beta 2^\alpha+2^\alpha} = p^l_1, \ldots, p^l_{2^\alpha}$ for all $0 \le l \le 2t^3$. Then

$$\underbrace{\left(\Omega^l_k\right)_{\substack{0\le l\le 2t^3\\ k\in S}}}_{=:\mathcal{U}} \cdot \underbrace{(T_k)_{k\in S}}_{=:\mathcal{T}} = 0.$$

According to the construction, $\mathcal{U}$ is a non-singular Vandermonde matrix, hence $\mathcal{T} \equiv 0$. Therefore $p_1 \equiv p_2$ and

$$\frac{\sum\limits_{|k^{(1)}|\le d2^\alpha} \mathbf{x}^{k^{(1)}} \cdot \bar{P}_{k^{(1)}}(x', x'')}{\sum\limits_{|k^{(2)}|\le d2^\alpha} \mathbf{x}^{k^{(2)}} \cdot \bar{Q}_{k^{(2)}}(x', x'')} = \frac{\sum\limits_{|k^{(1)}|\le k_1} \mathbf{x}^{k^{(1)}} \cdot P_{k^{(1)}}(x', x'')}{\sum\limits_{|k^{(2)}|\le k_2} \mathbf{x}^{k^{(2)}} \cdot Q_{k^{(2)}}(x', x'')}.$$

This proves validity of (30).

Since (k_1, k_2) is chosen as the minimal fitting pair, $S^{(1)}_{\alpha+1,\beta}(k_1)$ and $S^{(2)}_{\alpha+1,\beta}(k_2)$ correspond to the nonreducible representation of f: Consider two distinct valid representations for f:

$$f(x', \mathbf{x}, x'') = \frac{\sum\limits_{\substack{k^{(1)}\in S^{(1)} \\ |k^{(1)}|\le k_1}} \mathbf{x}^{k^{(1)}} \cdot P_{k^{(1)}}(x', x'')}{\sum\limits_{\substack{k^{(2)}\in S^{(2)} \\ |k^{(2)}|\le k_2}} \mathbf{x}^{k^{(2)}} \cdot Q_{k^{(2)}}(x', x'')} = \frac{\sum\limits_{\substack{k^{(1)}\in S^{(1)} \\ |k^{(1)}|\le k'_1}} \mathbf{x}^{k^{(1)}} \cdot P'_{k^{(1)}}(x', x'')}{\sum\limits_{\substack{k^{(2)}\in S^{(2)} \\ |k^{(2)}|\le k'_2}} \mathbf{x}^{k^{(2)}} \cdot Q'_{k^{(2)}}(x', x'')}$$

and assume that $k_1 < k_2$. Hence there is some variable x_j from $\mathbf{x}$ occurring with a larger exponent in the second representation than in the first one. Therefore the corresponding factorization contains a factor which does not occur in the factorization of the first representation. Since the nonreducible representation of f is unique, we conclude that the second representation is reducible. We fix the minimal fitting pair (k_1, k_2) and are guaranteed that the largest occurring exponent of each variable from $(\mathbf{x})$ is less or equal than in all other fitting representation. So the minimal pair (k_1, k_2) corresponds to the nonreducible representation. □

5.3 The Rational NC-Interpolation Algorithm

Rational Interpolation Algorithm [GK 88]

$$f_{i,l} := f(p_1^i, \ldots, p_{j-1}^i, a_l, p_{j+1}^i, \ldots, p_n^i)$$

Input: Black box for a t-sparse rational function $f \in \mathbb{Z}[x_1, \ldots, x_n]$. Let $\frac{P}{Q}$ be the nonreducible representation of f. Then $\deg_{x_i}(P) < d, \deg_{x_i}(Q) < d$ for $1 \le i \le n$.

Output: Sparse representation of P, Q.

Step A: Let $m = \lceil \log n \rceil$. Use the sieve of Erastosthenes to find pairwise distinct primes $p_1, \ldots p_n$.

Step B: Execute the basis step in parallel for $1 \le j \le n$.

Basis Step

Step 1: Let $a_0, \ldots, a_{2d}$ be pairwise distinct numbers in $\mathbb{Z} \setminus \{0\}$.
Let $S^{(1)}_{1,j-1}(k_1), S^{(2)}_{1,j-1}(k_2) = \emptyset$ for any pair $0 \le k_1, k_2 < d$.

Step 2: Query the the black box for the values

$$f_{i,l} := f(p_1^i, \ldots, p_{j-1}^i, a_l, p_{j+1}^i, \ldots, p_n^i) \qquad 0 \le l \le 2d,\ 0 \le i < t.$$

Step 3: Execute Step 4-6 in parallel for each pair $0 \le k_1, k_2 < d$ and for $0 \le i < t$.

Step 4:

$$\mathcal{A}_{k_1,k_2} = \begin{pmatrix} a_0^0 & \ldots & a_0^{k_1} & -f_{i,0} \cdot a_0^0 & \ldots & -f_{i,0} \cdot a_0^{k_2} \\ \vdots & & \vdots & \vdots & & \vdots \\ a_{2d}^0 & \ldots & a_{2d}^{k_1} & -f_{i,2d} \cdot a_{2d}^0 & \ldots & -f_{i,2d} \cdot a_{2d}^{k_2} \end{pmatrix}. \tag{34}$$

Determine the rank of $\mathcal{A}_{k_1,k_2}$.

Step 5: Let $\mathcal{A}_{k_1,k_2}(r)$, $0 \le r \le k_1 + k_2 + 1$ be the matrix resulting by canceling the r-th column from $\mathcal{A}_{k_1,k_2}$. Determine in parallel the set R of those indices r satisfying $\mathrm{rg}(\mathcal{A}_{k_1,k_2}) > \mathrm{rg}(\mathcal{A}_{k_1,k_2}(r))$.

Step 6:

$$\begin{aligned} S_{1,j-1}^{(1)}(k_1) &= S_{1,j-1}^{(1)}(k_1) \cup \{r \in R \mid 0 \le r \le k_1\} \quad \text{and} \\ S_{1,j-1}^{(2)}(k_2) &= S_{1,j-1}^{(2)}(k_2) \cup \{r \in R \mid k_1 + 1 \le r \le k_1 + k_2 + 1\}. \end{aligned}$$

Step 7: Let $S_{1,j-1}^{(1)} = S_{1,j-1}^{(1)}(k_1)$ and $S_{1,j-1}^{(2)} = S_{1,j-1}^{(2)}(k_2)$ where k_1 is minimal and $k_1 \in S_{1,j-1}^{(1)}(k_1)$, $k_2 \in S_{1,j-1}^{(2)}(k_2)$.

Step C: Execute Step D for $\alpha = 1, \ldots, m$.

Step D: Execute the recursion step in parallel for $0 \le \beta < 2^{m-\alpha}$.

Recursion Step

Step 8: Let $S^{(1)} := S_{\alpha,2\beta}^{(1)} \times S_{\alpha,2\beta+1}^{(1)}$ and $S^{(2)} := S_{\alpha,2\beta}^{(2)} \times S_{\alpha,2\beta+1}^{(2)}$.
Let $S_{\alpha+1,\beta}^{(1)}(k_1), S_{\alpha+1,\beta}^{(2)}(k_2) = \emptyset$ for any pair $0 \le k_1, k_2 < 2^\alpha(d-1)$.

Step 9: Query the black box for the values

$$f_{i,l} := f(p_1^i, \ldots, p_{\beta 2^\alpha}^i, p_1^l, \ldots, p_{2^\alpha}^l, p_{(\beta+1)2^\alpha+1}^i, \ldots, p_n^i)$$

for $0 \le l \le 2t^3$ and $0 \le i < t$.
Let $\Omega_{k^{(1)}} = p_1^{k_1^{(1)}}, \ldots, p_{2^\alpha}^{k_{2^\alpha}^{(1)}}$ and $\Omega_{k^{(2)}} = p_1^{k_1^{(2)}}, \ldots, p_{2^\alpha}^{k_{2^\alpha}^{(2)}}$.

Step 10: Execute Step 11-12 in parallel for each pair $0 \le k_1, k_2 < 2^\alpha(d-1)$.

Step 11: Let $\{k^{(1)} \in S^{(1)} \mid |k^{(1)}| \le k_1\} = \{k_1^{(1)}, \ldots, k_{r_1}^{(1)}\}$ and $\{k^{(2)} \in S^{(2)} \mid |k^{(2)}| \le k_2\} = \{k_1^{(2)}, \ldots, k_{r_2}^{(2)}\}$.

Step 12: Execute Step 13-15 in parallel for $0 \le i < t$.

Step 13:

$$\mathcal{B}_{k_1,k_2} = \left(\left(\Omega_{k^{(1)}}^l\right)_{\substack{0 \le l \le 2t^3 \\ |k^{(1)}| \le k_1}} - \begin{pmatrix} f_{i,0} & & 0 \\ & \ddots & \\ 0 & & f_{i,2t^3} \end{pmatrix} \left(\Omega_{k^{(2)}}^l\right)_{\substack{0 \le l \le 2t^3 \\ |k^{(2)}| \le k_2}} \right). \tag{35}$$

Determine the rank of $\mathcal{B}_{k_1,k_2}$.

Step 14: Let $\mathcal{B}_{k_1,k_2}(r)$, $0 \le r \le r_1 + r_2$ the matrix obtained from $\mathcal{B}_{k_1,k_2}$ by canceling the r-th column. Determine in parallel the set R of indices r satisfying $\mathrm{rg}(\mathcal{B}_{k_1,k_2}) > \mathrm{rg}(\mathcal{B}_{k_1,k_2}(r))$.

Step 15:

$$S^{(1)}_{\alpha+1,\beta}(k_1) = S^{(1)}_{\alpha+1,\beta}(k_1) \cup \{r \in R \mid 0 \le r < r_1\} \quad \text{and}$$
$$S^{(2)}_{\alpha+1,\beta}(k_2) = S^{(2)}_{\alpha+1,\beta}(k_2) \cup \{r \in R \mid r_1 \le r < r_1 + r_2\}.$$

Step 16: Let $S^{(1)}_{\alpha+1,\beta} = S^{(1)}_{\alpha+1,\beta}(k_1)$ and $S^{(2)}_{\alpha+1,\beta} = S^{(2)}_{\alpha+1,\beta}(k_2)$ for k_1 minimal and $k_1 \in S^{(1)}_{\alpha+1,\beta}(k_1)$, $k_2 \in S^{(2)}_{\alpha+1,\beta}(k_2)$.

Step E: $\mathrm{supp}(P) = S^{(1)}_{m+1,0}$, $\mathrm{supp}(Q) = S^{(2)}_{m+1,0}$. The coefficients of P, Q are given by a non-trivial solution of the homogeneous system of equations (35) for $\alpha = m$ and the minimal pair k_1, k_2.

5.4 Analysis of the Algorithm

Basis Step: $O(ntd^{7.5}\log^2(ntd))$ processors, $O(\log^2 d)$ parallel time, since

- Step 2: $O(dtn\log(nt)\log d) = O(dtn\log(ntd))$ processors, $O(\log t)$ parallel time.
- Step 4: $O(d^{4.5})$ processors, $O(\log^2 d)$ parallel time ([Mu 86]).
- Step 5: $dO(d^{4.5})$ processors, $O(\log^2 d)$ parallel time ([Mu 86]).
- Step 3: $d^2t\cdot$ (number of processors for the Steps 4-6) $= O(td^{7.5})$ processors, $O(\log^2 d)$ parallel time.
- Step 7: By means of logarithmic search $O(\log d^2)$ parallel time.

Step B: $O(ntd^{7.5})\log(ntd)$ processors, $O(\log^2 d)$ parallel time.

Recursion Step: $O(n^2d^2t^{17.5}\log(ntd))$ processors, $O(\log^2(ndt))$ parallel time, since

- Step 9: similar to Step 2: $O(t^3n\log(ntd))$ processors, $O(\log t)$ parallel time.
- Step 13: $O((t^3)^{4.5}) = O(t^{13.5})$ processors, $O(\log^2 t)$ parallel time ([Mu 86]).
- Step 14: $O(t^3t^{13.5}) = O(t^{16.5})$ processors, $O(\log^2 t)$ parallel time ([Mu 86]).
- Step 10: $O(n^2d^2t^{13.5})$ processors, $O(\log^2 t)$ parallel time.
- Step 16: By means of logarithmic search: $O(\log(ndt))$ parallel time.

Step D: $O(n^3d^2t^{17.5}\log(ntd))$ processors, $O(\log^2(ndt))$ parallel time.

Step C: $O(\log n\log^2(ndt)) = O(\log^3(ndt))$ parallel time.

Totally: $O((ntd^{7.5} + n^3d^2t^{17.5})\log(ntd))$ processors, $O(\log^3(ndt))$ parallel time.

Hence we obtain the statement of Theorem 5.1. □

5.5 Handling pathological situations

Performing the basis and recursion step the algorithm looks in parallel over all pairs $0 \leq k_1, k_2 < nd$ and considers for every pair k_1, k_2 the linear system of equations

$$\sum_{|k^{(1)}| \leq k_1} \mathbf{p}^{lk^{(1)}} P_{k^{(1)}}(x', x'') = f(x', \mathbf{p}^l, x'') \cdot \sum_{|k^{(2)}| \leq k_2} \mathbf{p}^{lk^{(2)}} Q_{k^{(2)}}(x', x'') \tag{36}$$

for $0 \leq l \leq 2d$ (basis step) and $0 \leq l \leq 2t^3$ (recursion step) where $\mathbf{p}^{lk^{(*)}} = \bar{p}_1^{lk_1^{(*)}} \cdot \ldots \cdot \bar{p}_{2^\alpha}^{lk_{2^\alpha}^{(*)}}$ with pairwise different primes $\bar{p}_j$. We obtain black-boxes for $P_{k^{(1)}}$ and $Q_{k^{(2)}}$. Involving the zero test of Tiwari we test whether $P_{k^{(1)}} \not\equiv 0, Q_{k^{(2)}} \not\equiv 0$ by solving (36) for $(x', x'') = (p_1^i, \ldots, p_{n-2^\alpha}^i)$ for $0 \leq i < t$ where p_i are pairwise different primes and pairwise different from $\bar{p}_j$.

If the evaluation points coincide with roots of the numerator or denominator, this approach does not work. We denote these situations pathological (cf. [Sc 80]). In the following we show how to overcome these difficulties using an extended Vandermonde argument. A t-sparse polynomial p is identical to zero iff there are $i_1, \ldots i_t$ with $p(p_1^{i_j}, \ldots, p_n^{i_j}) = 0$ for $0 \leq j < t$.

We assume that the black-box for f indicates undefined function values, i.e. roots of the denominator by a special symbol, say "$\perp$".
First we check whether $f \not\equiv 0$ and $f \not\equiv \perp$. Query the black-box for the values

$$f_i = f(p_1^i, \ldots, p_n^i) \quad \text{for } 0 \leq i < 2t.$$

Then by the aid of the extended Vandermonde argument

$$f \not\equiv \perp \iff |\{i : f_i = \perp\}| < t \qquad \text{and}$$

$$f \not\equiv 0 \iff |\{i : f_i = 0\}| < t.$$

Hence we assume that $f \not\equiv 0$ and $f \not\equiv \perp$.

Let us return to the situation above. Let $f_{i,l} = f(x', \mathbf{p}^{il}, x'')$ where $(x', x'') = (p_1^i, \ldots, p_{n-2^\alpha}^i)$. We have to guarantee that all evaluations of f are different from 0 and $\perp$ by involving the extended Vandermonde argument. Since $f \not\equiv 0$ and $f \not\equiv \perp$ we conclude that among some indices $\bar{i}_0, \ldots, \bar{i}_{2t-1}$ there is some index $\bar{i}_k$ with $f_{\bar{i}_k,l} \neq 0, f_{\bar{i}_k,l} \neq \perp$. To preserve the extended Vandermonde argument for the zero-test of $P_{k^{(1)}}$ and $Q_{k^{(2)}}$ we choose $i_j = i + j \cdot t$ for $0 \leq j < 2t$.

Checking consistence for each l we can state that among the indices $i_0, \ldots, i_{4td}$ (basis step) and $i_0, \ldots, i_{4t^4}$ (recursion step) there is i_k with $f_{i_k,l} \neq 0, f_{i_k,l} \neq \perp$ for each l.

We can extent our argument to the t linear equation systems considered in order to perform the zero-test for $P_{k^{(1)}}$ and $Q_{k^{(2)}}$. We sum up to get a very rough upper bound for the number of required evaluation points.

- Basis step: $n(2dt^2 + 8d^2t)$
- Recursion step: $2^{m+1-\alpha}(2t^4 + 8t^7)$

6 Conclusions

Recently, Kaltofen and Yagati ([KY 88]) designed fast sequential algorithms for solving linear systems of equations with Vandermonde matrices and Toeplitz-matrices for fields of characteristic zero. The complexity of the algorithms introduced in Chapter 3 is reduced.

Unfortunately, these algorithms are not applicable to finite fields. Therefore it is still open how to improve the interpolation algorithms given in Chapter 4 by making use of the special structure of the apparent matrices.

Kaltofen and Trager ([KT 88]) succeed in designing efficient probabilistic algorithms for the reconstruction of t-sparse rational functions as well as a probabilistic polynomially factorization algorithm for sparse polynomials. To accomplish this they use the polynomial interpolation algorithm introduced in Chapter 3.2. The à priori knowledge of an upper bound d on the degree is not necessary. However, this does not prove that the interpolation of sparse rational functions lies in P.

This is achieved by the algorithm of Grigoriev and Karpinski presented in Chapter 5, given an upper bound on the degree. This puts the problem in P and in deterministic boolean NC.

The problem of rational interpolation is also connected in an interesting way to the seminal problem of Strassen [St 73] of computing the numerators and denominators of general functions given by straight-line programs. The algorithm of Grigoriev and Karpinski transforms deterministicly arbitrary sparse rational straight-line programs into equivalent programms where only one division is allowed at the end of a computation.

The algorithm lies in NC, however when implemented sequentially (cf. Appendix A.4) the algorithm is not practical due to the large number of processors. Consequently, the important practical problem arises to improve substantially the number of processors of the algorithm.

The most promising attempt is to make use of the structure of the apparent matrices in order to reduce the effort for computations of ranks and to develop a logarithmic method to determine linear independent columns.

A Implementation in Scratchpad II

A.1 The Computer Algebra System Scratchpad II

Scratchpad II is a language featuring parameterized abstract data types and generic operators particularly suited for computer algebra. It is currently developed at the IBM T.J. Watson Research Center.

The abstract data type concept of Scratchpad II is based on *categories*, allowing the algorithms to be written for algebraic objects at their natural level of abstraction. The essential entities in the Scratchpad II language are objects created and manipulated by functions. There are four kinds of objects in the language:

- Computational objects
- Functions
- Domains
- Categories

Objects which are members of domains are called computational objects. One can look at domains as abstract (algebraic) data types of computational objects. A domain consists of

- a set of generic operations,
- a set of functions, which implement the operations and
- a set of attributes, which designate properties of the operations.

Domains are organized in a hierarchy; for the representation of an abstract data type one can refer to the representation of existing domains.
The generic operations indicate the possible manipulations of members of the domains. These operations are specified by functions in the local, not accessible declarative part of the domain. By means of attributes the generic operations are assigned properties in order to classify a domain according to categories.
Domains are created by special functions (domain constructors).

A category is an abstraction of a class of types. It specifies those properties which some collection of domains have in common. A category determines a class of domains with common operations and attributes but which may well have different functions and representations. All categories have:

- a set of generic operations and
- a set of attributes.

A domain is a member of a category iff it supplies the generic operations on the one hand and these generic operations have the demanded attributes. Categories are created by special functions (category constructors). They can be organized in a hierarchy.

Example : Let `Set` be a category with the generic operation

```
□ = □ : ( $, $ ) -> Boolean .
```

Any domain containing a test to equality belongs to the category `Set` (The symbol `$` is used to determine "this domain", i.e. the one which has the categories).
By means of the category `Set` the category `SemiGroup` can be constructed:

```
SemiGroup : Category == with Set
    □*□ : ( $, $ ) -> $
    associative(□*□)
```

A domain belongs to the category `SemiGroup` iff it belongs to the category `Set` and a associative multiplication is defined for the members of the domain.
The category to which a domain belongs is thus not unique.

The facility for categories is unique to Scratchpad II. By means of this concept to it is possible to formulate polymorphic implementations of algebraic algorithms which is crucial to computer algebra.

Example : Consider the Euclidean algorithm for computing the greatest common divisor of two integers:

```
gcd(x,y) == if x=0 then y else gcd(y,remainder(x,y))
```

Although this algorithm was originally designed for integers, a closer examination shows that this algorithm can be applied for any integral domain which has an appropriate remainder function with the property (attribute), that a remainder sequence generated by any two elements. Thereby, the category `Euclideandomain` can be defined. Then

```
gcd(x:R,y:R) : R where R : Euclideandomain ==
        if x=0 then y else gcd(y,remainder(x,y))
```

is a specification of the Euclidean algorithm in the algebraic most general form and is defined for all domains belonging to the category `Euclideandomain`.

Scratchpad II contains an extensive library of algebraic algorithms. In addition to the polymorphic concept this language makes it possible to implement an algebraic algorithm quickly without the technical overhead which arises in usual programming languages in order to prepare the implementation of abstract algebraic data types and operations.

A.2 The Interpolation Algorithm [CGK 87]

First we present the implementation of the interpolation algorithm von Clausen, Grabmeier and Karpinski introduced in Section 4.2.

Let $f \in FF[x_1, \ldots, x_n]$ be a t-sparse polynomial over the finite field FF and let FFX be the extension field over FF of degree n. f is given by a black box which returns the evaluation of the polynomial f in the extension FFX for an argument from FFX^n. Thus, the black box is interpreted as a mapping (e.g. a straight-line program):

```
black : List(FFX) -> FFX
```

The necessary parameter are the basis field FF of type FiniteField, n and t of type NonNegativeInteger (NNI) and the black box black of type Mapping(FFX,List(FFX)). The sparse representation of the polynomial which is to be reconstructed is implemented as a list of coefficients and exponents.

```
SREP == List(Record(coeff : FF, alpha : List(NNI)))
```

If the black box evaluates the point zero to a non-zero value a then $f(x_1,\ldots,x_n) - a$ is interpolated, i.e we have to subtract the value a from the evaluations of the black box. Furthermore we use the fact that in this case the resulting polynomial is $(t-1)$-sparse.

The package CGK(FF,n) provides the function interpol computing the sparse representation of the polynomial with regard to the algorithm [CGK 87]:

```
)abbreviation package CGK InterpolationClausenGrabmeierKarpinski87

InterpolationClausenGrabmeierKarpinski87
 (FF : FiniteField , n : PositiveInteger) :
  public == private where

   FFX  ==> FiniteFieldExtension(FF,n)
   N    ==> PositiveInteger
   NNI  ==> NonNegativeInteger
   V    ==> Vector
   L    ==> List
   MREP ==> Record(coeff : FF, alpha : L(NNI))
   SREP ==> L(MREP)
   UP   ==> UnivariatePoly
   SM   ==> SquareMatrix
   SUP  ==> SparseUnivariatePolynomial(FF)

   public == with

    interpol   : (Mapping(FFX,L(FFX)),N) -> SREP
      ++ interpol(blackbox,t) generates the sparse representation
      ++ of the t-sparse polynomial in n variables over the
      ++ Galois-Field with q elements. The polynomial is given by
      ++ the function blackbox evaluating the polynomial in the
      ++ field extension of degree n

   private == add

    -- local functions

    zeros : UP(X,FFX) -> L(FFX)
      -- zeros(p) lists all roots of p using Horner evaluation
      -- to find a root, divides p by the linear factor and iterates

    disclogarithms : (FFX,L(FFX)) -> V(NNI)
      -- disclogarithms(omega,lffx) computes the discrete logarithms
      -- for all elements of the list lffx by comparing these elements
      -- with successive powers of omega

    getEvaluations : (Mapping(FFX,L(FFX)),FFX,NNI) -> V(FFX)
      -- evaluate the blackbox for the 2*t+1 evaluation points
      -- f(0,..,0) is stored in the last component of the result vector
```

```
pAdic : (NNI,NNI) -> L(NNI)
  -- pAdic(a,p) calculates the p-adic expansion of a


interpol(blackbox,tsparse) ==

  sparserep    : SREP    := nil$SREP
  coefficients : V(FFX)  := new(1,0$FFX)
  omega        : FFX     := primitiveElt()$FFX
  fvector      : V(FFX)  := getEvaluations(blackbox,omega,tsparse)
  q            : NNI     := size()$FF
  lambda       : V(FFX)
  bchpoly      : UP(X,FFX)
  b            : V(FFX)
  k            : NNI

  constflag : Boolean := false
  t : NNI := tsparse::NNI
  if fvector(2*t) ~= 0$FFX then
    constflag := true
    t := (t - 1)::NNI

  -- construct the matrix containing the evaluations
  fmatrixt     : SM(t,FFX)  := new(t,t,0$FFX)
  for i in 0..t-1 repeat
    for j in 0..t-1 repeat
      fmatrixt(i,j):=fvector(i+j)

  -- k is the exact number of monomials in the polynomial
  k := rank(fmatrixt)$SM(t,FFX)
  coefficients := new(k,0$FFX)
  lambda       := new(k,0$FFX)
  b            := new(k,0$FFX)
  if k = 0 then return sparserep    -- zero polynomial
  fmatrixk : SM(k,FFX) := new(k,k,0$FFX)
  for i in 0..k-1 repeat
    for j in 0..k-1 repeat
      fmatrixk(i,j):=fmatrixt(i,j)

  -- calculate the coefficients of the auxiliary polynomial
  for i in 0..k-1 repeat b(i):=-fvector(k+i)
  fmatrixk := (inverse(fmatrixk)$SM(k,FFX))::SM(k,FFX)
  lambda  := (fmatrixk *$SM(k,FFX) b)::V(FFX)

  -- construct the auxiliary univariate polynomial and its zeros
  helppoly : UP(X,FFX)
  el : FFX
  bchpoly := (monom$UP(X,FFX))(k,1$FFX)
  for i in 0..k-1 repeat
    el := lambda(i::NNI)
    helppoly := (monom$UP(X,FFX))(i::NNI,el)
    bchpoly :=bchpoly + helppoly
  roots := zeros(bchpoly)

  -- get the discrete logarithms of the roots
  alphas := disclogarithms(omega,roots)

  -- add the p-adic expansions of the alphas to the result
  monomial : MREP
  for j in 0..k-1 repeat
    monomial := [0$FF,(pAdic(alphas(j::NNI),q))]
    sparserep := append(sparserep,list(monomial)$SREP)
```

```
    -- compute the coefficients of the polynomial
    for i in 0..k-1 repeat
      for j in 0..k-1 repeat
        fmatrixk(i,j) := roots(j)**i
    for i in 0..k-1 repeat
      b(i):=fvector(i)
    fmatrixk := (inverse(fmatrixk)$SM(k,FFX))::SM(k,FFX)
    coefficients := (fmatrixk *$SM(k,FFX) b)::V(FFX)
    for i in 0..k-1 repeat
      (sparserep(i)).coeff := retract(coefficients(i))$FFX
    if constflag then  -- add the constant term to the result
      monomial  := [retract(fvector(2*t+2))$FFX,(pAdic(0,q))]
      sparserep := append(sparserep,list(monomial)$SREP)
    sparserep


  -- definitions of local functions

  getEvaluations(blackbox,w,t) ==
    evalvector  : V(FFX)      := new(2*t+1,0$FFX)
    argument    : L(FFX)      := nil$L(FFX)
    computed    : V(Boolean)  := new(2*t,false)
    helparg     : V(FFX)      := new(n,0$FFX)
    helphelp    : NNI         := 1
    valueatzero : FFX
    q           : NNI      := size()$FF
    sold        : NNI
    snew        : NNI
    -- generate the null argument
    for j in 0..n-1 repeat
      argument       := cons(0$FFX,argument)
    valueatzero      := blackbox(argument)
    evalvector(2*t)  := valueatzero
    if valueatzero ^= 0 then t := (t - 1)::NNI
    -- generate the first argument
    for j in 0..n-1 repeat
      argument    := cons(1$FFX,argument)
      helparg(j)  := w ** helphelp
      helphelp := helphelp * q
    evalvector(0) := blackbox(argument) - valueatzero
    for i in 1..(2*t-1) repeat
      -- generate the next argument
      for j in 0..n-1 repeat
        argument(j::NNI) := argument(j)*helparg(j)
      if not computed(i) then
        evalvector(i) := blackbox(argument) - valueatzero
        sold := i
        snew := q * i
        while (snew < 2*t) repeat
          evalvector(snew) := evalvector(sold) ** q
          computed(snew) := true
          sold := snew
          snew := snew * q
    evalvector

  zeros(unipoly) ==
    rootlist : L(FFX)     := nil$L(FFX)
    val,mval : FFX
    monval   : UP(X,FFX)
    mon      : UP(X,FFX) := monom(1,1)$UP(X,FFX)
    -- generate the list of all roots of the polynomial unipoly
```

```
    for i in 1..size()$FFX while degree(unipoly) >0  repeat
      val := index(i :: PositiveInteger)$FFX
      mval := HornerEval(unipoly,val)$UP(X,FFX)
      mval ^= 0$FFX => "next value"
      monval := mon-(val :: UP(X,FFX))
      rootlist := cons(val,rootlist)
      unipoly := (unipoly exquo monval) :: UP(X,FFX)
    rootlist

  disclogarithms(w,lffx) ==
    v      : FFX         := w
    log    : NNI         := 1
    k      : NNI         := #lffx
    number : NNI         := k
    vlog   : V(NNI)      := new(k,0)
    help   : V(Boolean)  := new(k,false)
    while number > 0 repeat
      for i in 0..k-1 repeat
        if not help(i) then
          if lffx(i) = v then
            vlog(i) := log
            help(i) := true
            number  := (number - 1)::NNI
      v := v * w
      log := log + 1
    vlog

  pAdic(a:NNI,p:NNI) ==
    padic : L(NNI) := nil$L(NNI)
    for i in 0..n-1 repeat
      digit := (a div p).remainder
      padic := append(padic, list(digit::NNI))
      a := (a div p).quotient::NNI
    padic
```

The package CPFF provides the functions `createpoly` to generate a random polynomial over finite fields and `convertrat` to generate a mapping (black box) which gives the evaluations in the corresponding field extension:

```
)abbreviation package CPFF CreatePolynomialsOverFiniteFields

++ CreatePolynomialsoverFiniteFields contains functions for
++ generating polynomials over finite fields in order to test the
++ polynomial interpolation algorithms of Clausen, Grabmeier, Karpinski
++ and of Grigoriev, Karpinski, Singer

CreatePolynomialsOverFiniteFields
   (VarList:List(Expression),FF:FiniteField,s:PositiveInteger) : with

    createpoly : NNI -> MP
      ++ createpoly(t) creates a random t-sparse polynomial
      ++ over the ring R in the variables given by VarList

    convertpol : MP -> Mapping(FFX,List(FFX))
      ++ convertpol(poly) generates a black box representing
      ++ the polynomial poly evaluated in the finite field extension
      ++ FFX of FF of degree s
```

Examples:

1. Let $n = 2, t = 3, ff = \mathrm{GF}(3)$

```
poly := createpoly(3)$CPFF(vl,ff,2)
```

$x_1x_2 + 2$ Type: P[x[1],x[2]]GF 3

```
black:= convertpol(poly)$CPFF(vl,ff,2)
```

```
theMap(%G15989,111)     Type: (L FFX(GF 3,2) -> FFX(GF 3,2))
```

```
interpol(black,3)$CGK(ff,2)
```

```
[[coeff= 1,alpha= [1,1]],[coeff= 2,alpha= [0,0]]]

Type: L Record(coeff: GF 3,alpha: L NNI)    .338 sec (EV)
```

2. Let $n = 2, t = 5, ff = \mathrm{GF}(3)$

```
poly := createpoly(5)$CPFF(vl,ff,2)
```

$(2x_2^2 + 1)x_1^2 + (x_2^2 + x_2)x_1 + 2x_2^2$ Type: P[x[1],x[2]]GF 3

```
black:= convertpol(poly)$CPFF(vl,ff,2)
```

```
theMap(%G15989,396)      Type: (L FFX(GF 3,2) -> FFX(GF 3,2))
```

```
interpol(black,5)$CGK(ff,2)
```

```
[[coeff= 1,alpha= [2,0]], [coeff= 1,alpha= [1,2]],
 [coeff= 2,alpha= [0,2]], [coeff= 1,alpha= [1,1]],
 [coeff= 2,alpha= [2,2]]]

Type: L Record(coeff: GF 3,alpha: L NNI)   .802 sec (EV)
```

3. Let $n = 4, t = 7, ff = \mathrm{GF}(7)$

```
poly := createpoly(7)$CPFF(vl,ff,4)
```

$3x_4x_2^6x_1^6 + 5x_4^6x_3^3x_2^2x_1^4 + 5x_4^3x_3^5x_2^6x_1^3 + (5x_4^3x_3^3x_2^6 + 3x_4x_2^5 + 2x_4^4x_3^4x_2)x_1^2$
Type: P[x[1],x[2],x[3],x[4]]GF 7

```
black:= convertpol(poly)$CPFF(vl,ff,4)
```

```
theMap(%G15989,450)          Type: (L FFX(GF 7,4) -> FFX(GF 7,4))
```

```
interpol(black,7)$CGK(ff,4)
```

```
[[coeff= 3,alpha= [6,6,0,1]], [coeff= 2,alpha= [2,1,4,4]],
 [coeff= 5,alpha= [4,2,3,6]], [coeff= 5,alpha= [2,6,3,3]],
 [coeff= 3,alpha= [2,5,0,1]], [coeff= 5,alpha= [3,6,5,3]]]

Type: L Record(coeff: GF 7,alpha: L NNI)    28.822 sec (EV)
```

A.3 The NC-Interpolation Algorithm [GKS 88]

In this section we give the implementation of the NC-interpolation algorithm of Grigoriev, Karpinski and Singer for t-sparse polynomials over finite fields GF(q).

The polynomial which is to be reconstructed is given by a black box. It enables us to evaluate the polynomial over an arbitrary field extension of GF(q).
We implement the black box as a mapping (cf. A.2) parametrized with the degree of the field extension.
Since it is not (yet) possible to specify mappings with parametrized domain and codomain, we assume the black box to evaluate the polynomial in the field extension GF(q^s) for some fixed degree s.
The given implementation works in the logarithmic field extension GF(q^s) with

$$s = \lceil \log_q(4qn(n-1)\binom{t^2}{2} + 2) \rceil .$$

We abstain from the possibility to carry out the zero test for coefficient polynomials in the slight extension GF($q^{\bar{s}}$) with $\bar{s} = \lceil \log_q(4qn(n-1)\binom{t}{2} + 2) \rceil$; we also use the extension GF(q^s).

The package `GKS(FF,s)` provides the function `interpol`.
`interpol(black,t,n)` computes the sparse representation of the t-sparse polynomial in n variables over the finite field FF given by the mapping `black`.

The set of partial solution in each recursion step is implemented by a list of lists of the occurring exponents. This list is initialized in the basic step by computing the occurring exponents for $x_1, \ldots, x_n$. By padding this list by *nil* we accomplish that the algorithm is also usable for values of n not representing some power of two.
While doing the recursion step, we place the $(2 \cdot \beta)$-th component in to the (β)-th component as the new partial solution, if the $(2 \cdot \beta + 1)$-th component of the list is *nil*.

Furthermore, we make use of the fact that in the last recursion step the coefficient polynomials corresponding to the partial solutions are constants. So we can omit the loops for the zero test of the coefficient polynomials (cf. step 13 in 4.3.5); only the vector $P_{0,t}$ has to be calculated. The non-zero components of this vector give us the coefficients of the polynomial.

```
)abbreviation package GKS InterpolationGrigorievKarpinskiSinger88

InterpolationGrigorievKarpinskiSinger88
 (FF : FiniteField , s : PositiveInteger) :
  public == private where

   FFX  ==> FiniteFieldExtension(FF,s)
   I    ==> Integer
   BO   ==> Boolean
   SI   ==> SmallInteger
   NNI  ==> NonNegativeInteger
   V    ==> Vector
   L    ==> List
   MREP ==> Record(coeff : FF, expon : V(NNI))
   SREP ==> L(MREP)
   UP   ==> UnivariatePoly
   M    ==> Matrix
   SM   ==> SquareMatrix
```

```
SUP  ==> SparseUnivariatePolynomial(FF)
E    ==> Expression
DS   ==> DisplayPackage
OUT  ==> OutputPackage

public == with

 interpol : (Mapping(FFX,L(FFX)),NNI,NNI) -> SREP
   ++ interpol(blackbox,t,n) generates the sparse representation
   ++ of the t-sparse polynomial in n variables over the
   ++ Galois-Field with q elements using the NC-Algorithm
   ++ of Grigoriev, Karpinski and Singer. The polynomial is
   ++ given by the function blackbox evaluating the
   ++ polynomial in the field extension of degree s

private == add

 -- local functions

 cauchy : (NNI,NNI) -> V(I)
   -- cauchy(N,n) constructs the (N*n) Cauchy matrix
   -- the relevant elements are stored in a vector

 test set : (NNI,NNI,NNI,FFX,V(I)) -> M(FFX)
   -- test set(N,n,t,omega,C) generates set of evaluation points
   -- stored like the Cauchy matrix C in a space efficient way

 kartprod : (L(V(NNI)), L(V(NNI))) -> L(V(NNI))
   -- build up S(alpha,2*beta) # S(alpha,2*beta+1)


 interpol(blackbox,t,n) ==

   sparserep    : SREP          := nil$SREP
   coefficients : L(FFX)        := nil$L(FFX)
   SetofExp     : L(L(V(NNI))) := nil$L(L(V(NNI)))
   candidates   : L(V(NNI))     := nil$L(V(NNI))
   omega        : FFX           := primitiveElt()$FFX
   q            : NNI           := size()$FF
   qs           : NNI           := size()$FFX
   fvector      : V(FFX)        := new(q,0$FFX)
   localExp     : L(V(NNI))     := nil$L(V(NNI))

   -- calculate the dimension N
   N := (ceil((qs-1)/(4*n*q))$QuotientField(I))::NNI

   -- get the cauchy matrix
   C := cauchy(N,n)

   -- get the set of test points
   TestSet := test set(N,n,t,omega,C)

   -- list the elements of FF
   gfq : V(FFX) := new(q,1$FFX)
   for k in 0..q-1 repeat
     gfq(k) := (index((k+1)::PositiveInteger)$FF)::FFX

   -- generate the transformation matrix of the basic step
   A : SM(q,FFX) := new(q,q,1$FFX)
   for j in 1..q-1 repeat
     for k in 0..q-1 repeat
       A(k,j) := gfq(k) * A(k,j-1)
```

```
A := (inverse(A)$SM(q,FFX))::SM(q,FFX)

for beta in 0..n-1 repeat

  -- basic step

  localExp := nil$L(V(NNI))        -- S(1,beta)
  added : V(B0) := new(q,false)
  temp : FFX
  for l in 0..t repeat
    arg := nil$L(FFX)
    if l = 0 then
      for j in 0..n-1 repeat
        arg := cons(1$FFX,arg)
    if ((l > 0) and (l < t)) then
      for j in n-2..0 by -1 repeat
        arg := cons(TestSet(l-1,j),arg)
    if l = t then
      for j in 0..n-1 repeat
        arg := cons(0$FFX,arg)
    for i in 0..N-1 repeat
      if ((l > 0) and (l < t)) then
        temp := TestSet(l-1,n+i-1)
        arg := append(arg,list(temp)$L(FFX))
      old := arg(beta)
      for k in 0..q-1 repeat
        arg(beta) := gfq(k)
        fvector(k) := blackbox(arg)
      fvector := (A *$SM(q,FFX) fvector)::V(FFX)
      for k in 0..q-1 repeat
        if fvector(k) ~= 0 then  -- member of S(1,beta)
          if not added(k) then
            localExp := cons([k]::V(NNI),localExp)
            added(k) := true
      arg(beta) := old
      arg := drop(arg,1)
      if (l=0) or (l=t) then i := N
    -- end of i-loop
  -- end of l-loop
  SetofExp := append(SetofExp,list(localExp)$L(L(V(NNI))))
-- end of beta-loop and basic step

-- calculate number of recursion steps
m : NNI := 0
pow := 1
while n > pow repeat
  m := m + 1
  pow := pow * 2
for i in n+1..2**m repeat      -- padding with nil
  SetofExp := append(SetofExp,list(nil$L(V(NNI)))$L(L(V(NNI))))

for alpha in 1..m repeat

  for beta in 0..(2**(m-alpha)::NNI-1)::NNI repeat

    if SetofExp(2*beta+1) = nil$L(V(NNI)) then
      SetofExp(beta) := SetofExp(2*beta)
    else

      -- recursion step

      -- construct the set of candidates, i.e the set S
```

```
candidates := kartprod(SetofExp(2*beta),SetofExp(2*beta+1))
lenofcan := #candidates(0)  -- length of elements in S
numofcan := #candidates     -- number of elements in S
vhelp    := new(lenofcan,1$NNI)
SetofExp(2*beta) := nil$List(V(NNI))
SetofExp(2*beta+1) := nil$List(V(NNI))

-- find a row of the cauchy matrix which seperates the
-- elements of candidates
sepind : NNI := 0       -- index of seperator row
found : B0 := false
while not found repeat
  sum : V(NNI) := new(numofcan,0)
  found := true
  for j in 0..numofcan-1 repeat
    if found then
      for i in 0..lenofcan-1 repeat
        ad := (C(sepind+i) * (candidates(j)(i)))::NNI
        sum(j) := sum(j) + ad
      for i in 0.. j-1 repeat
        if sum(i) = sum(j) then
          found := false
          sepind := sepind + 1
        -- end of if
      -- end of for
    -- edn of if
  -- end of for
-- end of while

-- construct the transformation matrix
Omega : V(FFX) := new(numofcan)
for k in 0..numofcan-1 repeat
  Omega(k) := omega ** sum(k)
B : SM(numofcan,FFX) := new(numofcan,numofcan,1$FFX)
for i in 1..numofcan-1 repeat
  for j in 0..numofcan-1 repeat
    B(i,j) := Omega(j) * B(i-1,j)
B := (inverse(B)$SM(numofcan,FFX))::SM(numofcan,FFX)

-- construct the seperator set
SepSet : M(FFX) := new(numofcan,lenofcan,1$FFX)
for i in 1..numofcan-1 repeat
  for j in 0..lenofcan-1 repeat
    SepSet(i,j) := ( omega ** C(sepind+j) ) * SepSet(i-1,j)

fvector := new(numofcan,0$FFX)
for l in 0..t repeat
  arg := nil$L(FFX)
  if l = 0 then
    for j in 0..n-1 repeat
      arg := cons(1$FFX,arg)
  if ((l > 0) and (l < t)) then
    for j in n-2..0 by -1 repeat
      arg := cons(TestSet(l-1,j),arg)
  if l = t then
    for j in 0..n-1 repeat
      arg := cons(0$FFX,arg)
  for i in 0..N-1 repeat
    if ((l>0) and (l < t)) then
      arg := append(arg,list(TestSet(l-1,n+i-1))$L(FFX))
    old := arg(beta)
    for k in 0..numofcan-1 repeat
```

```
            for r in 0..lenofcan-1 repeat
              arg(beta*lenofcan+r) := SepSet(k,r)
            fvector(k) := blackbox(arg)
          fvector := (B *$SM(numofcan,FFX) fvector)::V(FFX)
          for indexu in 0..numofcan-1 repeat
            if fvector(indexu) ~= 0 then      -- member of S(alpha+1,beta)
              vhelp := candidates(indexu)
              if not member(vhelp,SetofExp(beta)) then
                SetofExp(beta) := cons(vhelp,SetofExp(beta))
                if alpha = m then        -- solutions are the coefficients
                  coefficients := cons(fvector(indexu),coefficients)
          arg(beta) := old
          arg := drop(arg,1)
          if (l=0) or (l=t) then i := N
        -- end of i-loop
        if alpha = m then l := t     -- partial polynomials are constants
      -- end of l-loop
    -- end of else clause
  -- end of beta-loop
-- end of alpha-loop

-- convert into elements of FF and build up the sparserep
listofExp : L(V(NNI)) := SetofExp(0)
k : NNI := #listofExp
monomial : MREP
for j in 0..k-1 repeat
  monomial := [retract(coefficients(j))$FFX,listofExp(j)]
  sparserep := append(sparserep,list(monomial)$SREP)

sparserep


cauchy(N,n) ==
  p := (nextPrime(2*N)$IntegerPrimesPackage)::SI
  Cauchy : V(I) := new((N+n-1)::NNI,0$I)
  for i in 0..n+N-2 repeat
    temp   := ((i+2)::I)::SI
    temp   := invmod(temp,p)$SI
    Cauchy(i) := convert(temp)$SI
  Cauchy

test set(N,n,t,omega,C) ==
  tset : M(FFX) := new(t-1,(n+N-1)::NNI,1$FFX)
  if t>1 then
    for i in 0..N+n-2 repeat
      tset(0,i) := omega ** C(i)
    for l in 1..t-2 repeat
      for i in 0..N+n-2 repeat
        tset(l,i) := tset(l-1,i) * tset(1,i)
  tset

kartprod(l1 : L(V(NNI)), l2 : L(V(NNI))) ==
  l : L(V(NNI)) := nil$L(V(NNI))
  v : V(NNI)
  v1: V(NNI)
  v2: V(NNI)
  for i in 0..(#l1-1) repeat
    for j in 0..(#l2-1) repeat
      v1 := l1.i
      v2 := l2.j
      v := new(#v1+#v2)
      for k in 0..(#v1-1) repeat
```

```
      v(k) := v1(k)
    for k in #v1..(#v1+#v2-1) repeat
      v(k) := v2(k-#v1)
    l := cons(v,l)
l
```

Examples:

1. Let $n = 2, q = 3, t = 3$ $s = 7$

```
poly:=createpoly(t)$CPFF(vl,ff,s)
```

$(2x_2^2 + 2x_2)x_1^2 + 2x_2^2$ Type: P[x[1],x[2]]GF 3

```
black:=convertpol(poly)$CPFF(vl,ff,s)
```

```
   theMap(%G15851,922)     Type: (L FFX(GF 3,7) -> FFX(GF 3,7))
```

```
interpol(black,t,n)$GKS(ff,s)
```

```
   [[coeff= 2,expon= [2,2]],[coeff= 2,expon= [2,1]],
    [coeff= 2,expon= [0,2]]]

   Type: L Record(coeff: GF 3,expon: V NNI)      19.893 (EV)
```

2. Let $n = 4, q = 3, t = 4$ $s = 9$

```
poly:=createpoly(t)$CPFF(vl,ff,s)
```

$x_3x_2^2x_1^2 + (x_4^2x_2^2 + 2)x_1 + 2x_4x_3^2x_2^2$ Type: P[x[1],x[2],x[3],x[4]]GF 3

```
black:=convertpol(poly)$CPFF(vl,ff,s)
```

```
   theMap(%G15851,196)     Type: (L FFX(GF 3,9) -> FFX(GF 3,9))
```

```
interpol(black,t,n)$GKS(ff,s)
```

```
   [[coeff= 1,expon= [1,2,0,2]], [coeff= 2,expon= [1,0,0,0]],
    [coeff= 1,expon= [2,2,1,0]], [coeff= 2,expon= [0,2,2,1]]]

   Type: L Record(coeff: GF 3,expon: V NNI)      3846.092 (EV)
```

3. Let $n = 3, q = 7, t = 3$ $s = 5$

```
poly:=createpoly(t)$CPFF(vl,ff,s)
```

$x_3^2x_2^5x_1^4 + 3x_3^5x_2^6x_1^2$ Type: P[x[1],x[2],x[3]]GF 7

```
black:=convertpol(poly)$CPFF(vl,ff,s)

   theMap(%G15851,181)      Type: (L FFX(GF 7,5) -> FFX(GF 7,5))

interpol(black,t,n)$GKS(ff,s)

   [[coeff= 1,expon= [4,5,2]],[coeff= 3,expon= [2,6,5]]]

   Type: L Record(coeff: GF 7,expon: V NNI)      507.061 (EV)
```

A.4 The rational NC-Interpolation Algorithm [GK 88]

The enumeration technique used in the interpolation algorithm of Grigoriev and Karpinski for sparse rational functions corresponds to enumeration technique of the NC-algorithm for sparse polynomials implemented in the previous section. The used data structures for partial solutions can be directly transferred.

Our task is to generate a nonreducible representation of the rational function. This is accomplished by selecting the minimal pair (k_1, k_2) (cf. 5.3) corresponding to a valid representation. In the presented implementation we make use of the fact that only pairs (k_1, k_2) have to be considered which corresponds exactly to existing candidates of partial solutions. For this purpose we sort the candidates according to the sum of their exponents. The occurring values correspond to the pairs (k_1, k_2) which must be tested.

The package `GKRAT()` provides the function `interpol`. `interpol(black,n,t,d)` computes the sparse representation with respect to the algorithm [GK 88]:

```
)abbreviation package GKRAT RationalInterpolationGrigorievKarpinski88

RationalInterpolationGrigorievKarpinski88() : public == private where

  NNI    ==> NonNegativeInteger
  RN     ==> RationalNumber
  I      ==> Integer
  V      ==> Vector
  L      ==> List
  MREP   ==> Record(coeff : I, expon : V(NNI))
  SREP   ==> L(MREP)
  RATREP ==> Record(numerator : SREP, denominator : SREP)
  VLI    ==> Record(num : V(L(V(NNI))), den : V(L(V(NNI))))

  public == with

    interpol : (Mapping(RN,V(I)),NNI,NNI,NNI) -> RATREP
      ++ interpol(blackbox,n,t,d) reconstructs the rational function
      ++ over the integers given by the blackbox using the NC-algorithm
      ++ Grigoriev and Karpinski. n is the number of variables,
      ++ the degree in each variable is bounded by d and t is a bound
      ++ of the number of non-zero terms in the denominator and numerator.

  private == add

    -- local functions
```

```
firstprimes   : I -> V(I)
  -- firstprimes(n) calculates the first n prime numbers

norm : V(NNI) -> NNI
  -- norm(vector) is the sum of components of vector

less? : (V(NNI),V(NNI)) -> Boolean
  -- less?(v1,v1) <=> ( norm(v1) < norm(v2) )

kartprod : (L(V(NNI)), L(V(NNI))) -> L(V(NNI))
  -- constructs S(alpha,2*beta) * S(alpha,2*beta+1)
  -- and sorts the resulting list according to less?


interpol(blackbox, n, t, d) ==

  -- declarations
  ratfuncrep : RATREP
  SetofExp   : VLI
  S1         : L(V(NNI))    := nil$L(V(NNI))
  S2         : L(V(NNI))    := nil$L(V(NNI))
  primes     : V(I)         := firstprimes(n)
  coeffind   : L(NNI)
  numnum     : NNI
  numden     : NNI

  -- calculate the number of recursion steps
  m : NNI := 0
  pow : NNI := 1
  while n > pow repeat
    m := m + 1
    pow := 2 * pow
  SetofExp := [ new(pow,nil$L(V(NNI))), new(pow,nil$L(V(NNI))) ]

  for j in 0..(n-1) repeat

    -- basic step

    SetofExp(num)(j) := nil$List(V(NNI))
    SetofExp(den)(j) := nil$List(V(NNI))

    -- determine function values
    EvalRes : Matrix(RN) := new(t,2*d+1,1)  -- results of the evaluations
    EvalArg : V(I) := new(n,1)  -- argument for the blackbox
    for i in 0..(t-1) repeat
      for l in 0..(2*d) repeat
        EvalArg(j) := l+1
        EvalRes(i,l) := blackbox(EvalArg)
      for r in 0..(n-1) repeat  -- next prime power
        EvalArg(r) := EvalArg(r) * primes(r)

    -- test all pairs k1,k2

    minflag := false  -- a pair corresp. to the nonreducible repres. has been found
    for k1 in 0..(d-1) repeat
      for k2 in 0..(d-1) repeat
        if not minflag then
          TestSet1 : L(V(NNI)) := nil$L(V(NNI))
          TestSet2 : L(V(NNI)) := nil$L(V(NNI))
          for i in 0..(t-1) repeat
```

```
            -- construct the matrix A(k1,k2)
            A : Matrix(RN) := new(2*d+1,k1+k2+2,0)
            for l1 in 0..2*d repeat
              A(l1,0) := 1
            for l1 in 0..2*d repeat
              for l2 in 1..k1 repeat
                A(l1,l2) := (l1+1)*A(l1,l2-1)
            for l1 in 0..2*d repeat
              A(l1,k1+1) := -EvalRes(i,l1)
            for l1 in 0..2*d repeat
              for l2 in k1+2..k1+1+k2 repeat
                A(l1,l2) := (l1+1)*A(l1,l2-1)

            -- determine the rank of A(k1,k2)
            mainrank : NNI := rank(A)
            -- test if the s-th row is linear independent
            vh1 : V(NNI)
            vh2 : V(NNI)
            tempvec : V(RN) := new(2*d+1,1)
            for s in k1+1+k2..0 by -1 repeat
              for l1 in 0..2*d repeat
                tempvec(l1) := A(l1,s)
                A(l1,s) := 0
              testrank : NNI := rank(A)
              for l1 in 0..2*d repeat
                A(l1,s) := tempvec.l1
              if mainrank = testrank then
                if s < k1+1 then
                  vh1 := [s]::V(NNI)
                  if not (vh1 in TestSet1) then
                    TestSet1 := cons(vh1,TestSet1)
                if s > k1 then
                  vh2 := [(s-k1-1)::NNI]::V(NNI)
                  if not (vh2 in TestSet2) then
                    TestSet2 := cons(vh2,TestSet2)
          -- end of i-loop
          -- does k1,k2 correspond to the nonreducible representation ?
          vh1 := [k1]::V(NNI)
          vh2 := [k2]::V(NNI)
          if member(vh1,TestSet1) and member(vh2,TestSet2) then
            SetofExp(num)(j) := TestSet1
            SetofExp(den)(j) := TestSet2
            minflag := true
        -- end of if not minflag
      -- end of k2-loop
    -- end of k1-loop
    -- end of basicstep
  -- end of j-loop


  for alpha in 1..m repeat

    for beta in 0..(2**(m-alpha)::NNI-1)::NNI repeat

      if SetofExp(num)(2*beta+1) = nil$L(V(NNI)) then
        SetofExp(num)(beta) := SetofExp(num)(2*beta)
        SetofExp(den)(beta) := SetofExp(den)(2*beta)
      else

        -- recursion step

        -- construct the sets S1 and S2
```

```
S1:=kartprod(SetofExp(num)(2*beta),SetofExp(num)(2*beta+1))
S2:=kartprod(SetofExp(den)(2*beta),SetofExp(den)(2*beta+1))
lenofcan := #S1(0)
vh1 := new(lenofcan,1)
vh2 := new(lenofcan,1)

SetofExp(num)(2*beta) := nil$List(V(NNI))
SetofExp(den)(2*beta) := nil$List(V(NNI))
SetofExp(num)(2*beta+1) := nil$List(V(NNI))
SetofExp(den)(2*beta+1) := nil$List(V(NNI))

-- determine the occurring norms of the candidates in S1, S2
index1 : L(NNI) := nil$L(NNI)
for r in (#S1-1)..0 by -1 repeat
  help := norm(S1(r))
  if not(help in index1) then index1 := cons(help,index1)
index2 : L(NNI) := nil$L(NNI)
for r in (#S2-1)..0 by -1 repeat
  help := norm(S2(r))
  if not(help in index2) then index2 := cons(help,index2)

-- evaluations of the black box
EvalRes := new(t,2*t**3+1,1)
EvalArg := new(n,1)
for i in 0..(t-1) repeat
  for l in 0..(2*t**3) repeat
    for k in 0..(beta*lenofcan-1) repeat
      EvalArg(k) := primes(k)**i
    for k in (beta*lenofcan)..((beta+1)*lenofcan-1) repeat
      EvalArg(k) := primes(k-beta*lenofcan)**l
    for k in ((beta+1)*lenofcan)..(n-1) repeat
      EvalArg(k) := primes(k)**i
    EvalRes(i,l) := blackbox(EvalArg)
  if alpha = m then i := t   -- last recursion only once

-- construct the vectors omega1 and omega2
omega1 : V(NNI) := new(#S1)
omega2 : V(NNI) := new(#S2)
for k1 in (0..#S1-1) repeat
  omega1(k1) := 1
  for k in 0..lenofcan-1 repeat
    help := S1(k1)(k)
    help := (primes(k)**help)::NNI
    omega1(k1) := omega1(k1)*help
for k2 in (0..#S2-1) repeat
  omega2(k2) := 1
  for k in 0..lenofcan-1 repeat
    help := S2(k2)(k)
    help := (primes(k)**help)::NNI
    omega2(k2) := omega2(k2)*help

-- test all pairs k1,k2 in index1,index2
altnumk1 : NNI := 0
minflag := false
for k1 in index1 repeat
  if not minflag then
    -- calculate number of elements in S1 with norm <= k1
    numk1 : NNI := altnumk1
    flag  : Boolean := true
    while flag and (numk1 < #S1) repeat
      if norm(S1(numk1)) <= k1
        then numk1 := numk1 + 1
```

```
    else flag  := false
altnumk1 := numk1
altnumk2 : NNI := 0
for k2 in index2 repeat
  if not minflag then
    -- calculate number of elements in S2 with norm <= k2
    numk2 : NNI := altnumk2
    flag  := true
    while flag and (numk2 < #S2) repeat
      if norm(S2(numk2)) <= k2
        then numk2 := numk2 + 1
        else flag  := false
    altnumk2 := numk2
    TestSet1 := nil$List(V(NNI))
    TestSet2 := nil$List(V(NNI))
    if alpha = m then
      coeffind : L(NNI) := nil$L(NNI)
      numnum : NNI := 0
      numden : NNI := 0

    for i in 0..(t-1) repeat
      -- generating the matrix B(k1,k2)
      B : Matrix(RN) := new(2*t**3+1,numk1+numk2,0)
      for l2 in 0..numk1-1 repeat
        B(0,l2) := 1$RN
      for l1 in 1..2*t**3 repeat
        for l2 in 0..numk1-1 repeat
          B(l1,l2) := (omega1(l2))::RN*B(l1-1,l2)
      for l2 in 0..numk2-1 repeat
        B(0,l2+numk1) := -1$RN
      for l1 in 1..2*t**3 repeat
        for l2 in 0..numk2-1 repeat
          help := l2+numk1
          B(l1,help):=(omega2(l2))::RN*B(l1-1,help)
      for l1 in 0..2*t**3 repeat
        for l2 in 0..numk2-1 repeat
          help := l2+numk1
          B(l1,help) := EvalRes(i,l1)*B(l1,help)
      -- evaluating the rank of B
      mainrank := rank(B)

      -- Test if the s-th column is linear independent
      tempvec : V(RN) := new(2*t**3+1,1)
      for s in numk1+numk2-1..0 by -1 repeat
        for l1 in 0..2*t**3 repeat
          tempvec(l1) := B(l1,s)
          B(l1,s) := 0
        testrank := rank(B)
        for l1 in 0..2*t**3 repeat
          B(l1,s) := tempvec(l1)
        if mainrank = testrank then
          if alpha = m then
            coeffind := cons(s::NNI,coeffind)
          if s < numk1 then
            if alpha = m then numnum := numnum + 1
            vh1 := S1(s)
            if not (vh1 in TestSet1) then
              TestSet1 := cons(vh1,TestSet1)
          if s > numk1-1 then
            if alpha = m then numden := numden + 1
            vh2 := S2(s-numk1)
            if not (vh2 in TestSet2) then
```

```
                    TestSet2 := cons(vh2,TestSet2)
                if alpha = m then i := t
              -- end of i-loop
              -- does the pair (k1,k2) correspond to the nonreducible representation
              if ( #TestSet1 > 0 ) and ( #TestSet2 > 0 ) then
                vh1 := last(TestSet1)
                vh2 := last(TestSet2)
                if (norm(vh1)=k1) and (norm(vh2)=k2) then
                  if alpha = m then
                    coeffmatrix : Matrix(RN)
                    coeffmatrix := new(2*t**3+1,numnum+numden,0)
                    help := 0
                    for s in coeffind repeat
                      for r in 0..2*t**3 repeat
                        coeffmatrix(r,help) := B(r,s)
                      help := help + 1
                    minflag := true
                    SetofExp(num)(beta) := TestSet1
                    SetofExp(den)(beta) := TestSet2

            -- end of if not minflag in k2-loop
          -- end of k2-loop
        -- end of if not minflag in k1-loop
      -- end of k1-loop
      -- end of recursion step
    -- end of else clause
  -- end of beta-loop
-- end of alpha-loop

-- determine the coefficients
coefflist : V(RN) := nullSpace(coeffmatrix).0
numcoeff : V(I) := new(numnum,0)
dencoeff : V(I) := new(numden,0)
-- transform coefficients into integer values
scalar := 1
for s in 0..numnum+numden-1 repeat
  scalar := lcm(scalar,denom(coefflist(s)))
for s in 0..numnum+numden-1 repeat
  coefflist(s) := scalar * coefflist(s)
for s in 0..numnum-1 repeat
  numcoeff(s) := numer(coefflist(s))
for s in numnum..numnum+numden-1 repeat
  dencoeff((s-numnum):NNI) := numer(coefflist(s))

-- transform the results into output format
sparserepnum : SREP := nil$SREP
sparserepden : SREP := nil$SREP
monomial : MREP
listofExp : L(V(NNI)) := SetofExp(num)(0)
for i in 0..numnum-1 repeat
  monomial := [numcoeff(i),listofExp(i)]
  sparserepnum := append(sparserepnum,list(monomial)$SREP)
listofExp : L(V(NNI)) := SetofExp(den)(0)
for i in 0..numden-1 repeat
  monomial := [dencoeff(i),listofExp(i)]
  sparserepden := append(sparserepden,list(monomial)$SREP)
ratfuncrep := [ sparserepnum, sparserepden]

ratfuncrep


-- definitions of the local functions
```

```
norm(v) ==
  sum : NNI := 0
  for i in 0..(#v-1) repeat
    sum := sum + v(i)
  sum


less?(v1 : V(NNI), v2 : V(NNI)) ==
  (norm(v1) < norm(v2))


kartprod(l1 : L(V(NNI)), l2 : L(V(NNI))) ==
  l : L(V(NNI)) := nil$L(V(NNI))
  v : V(NNI)
  v1: V(NNI)
  v2: V(NNI)
  for i in 0..(#l1-1) repeat
    for j in 0..(#l2-1) repeat
      v1 := l1.i
      v2 := l2.j
      v := new(#v1+#v2)
      for k in 0..(#v1-1) repeat
        v(k) := v1(k)
      for k in #v1..(#v1+#v2-1) repeat
        v(k) := v2(k-#v1)
      l := cons(v,l)
  sort(l,less?)


firstprimes(n) ==
  l : Vector(Integer) := new(n::NonNegativeInteger,1)
  flag : Integer := -1
  if n > 0 then l(0) := 2
  if n > 1 then l(1) := 3
  k : Integer := 2
  m : Integer := 5
  while k < n repeat
    if prime?$IntegerPrimesPackage m then
      l(k) := m
      k := k + 1
    m := m + 3 + flag
    flag := -flag
  l
```

In analogy to the previous implementations we use the operation `createrat(t,d)` for generating a random rational functions and the operation `convertrat` for converting a rational function into a mapping (black box). The package `CR(`$x_1, \ldots, x_n$`)` provides these operations.

Examples:

1. Let $n = 2, t = 2\ d = 2$

```
rf:=createratf(t,d)$CR(vl)
```

$\frac{3x_2x_1+9x_2}{4x_1+1}$ Type: QF P[x[1],x[2]]I

```
black:=convertrat(rf)$CR(vl);
```

```
interpol(black,n,t,d)

   [numerator =   [[coeff= 9,expon= [0,1]],[coeff= 3,expon= [1,1]]],
    denominator = [[coeff= 1,expon= [0,0]],[coeff= 4,expon= [1,0]]]]

Type: Record(numerator   : L Record(coeff: I,expon: V NNI),
             denominator: L Record(coeff: I,expon: V NNI))  2.58 (EV)
```

2. Let $n = 3, t = 3\ d = 4$

```
rf:=createratf(t,d)$CR(vl)
```

$\frac{(2x_3x_2+15x_3^2)x_1}{3x_3^2x_2^2+3x_3)x_1+4}$ Type: QF P[x[1],x[2],x[3]]I

```
black:=convertrat(rf)$CR(vl);

interpol(black,n,t,d)

   [numerator   = [[coeff= 2,expon= [1,1,1]],[coeff= 15,expon= [1,0,2]]],
    denominator = [[coeff= 4,expon= [0,0,0]], [coeff= 3,expon= [1,0,1]],
                   [coeff= 3,expon= [1,2,2]]]]

Type: Record(numerator   : L Record(coeff: I,expon: V NNI),
             denominator: L Record(coeff: I,expon: V NNI))  2220.605 (EV)
```

3. Let $n = 2, t = 4\ d = 2$

```
rf:=createratf(t,d)$CR(vl)
```

$\frac{2x_1+14x_2}{(6x_1+3)x_1+3x_2+9}$ Type: QF P[x[1],x[2]]I

```
black:=convertrat(rf)$CR(vl);

interpol(black,n,t,d)

   [numerator   = [[coeff= 2,expon= [1,0]],[coeff= 14,expon= [0,1]]],
    denominator = [[coeff= 9,expon= [0,0]], [coeff= 3,expon= [1,0]],
                   [coeff= 3,expon= [0,1]], [coeff= 6,expon= [1,1]]]]

Type: Record(numerator   : L Record(coeff: I,expon: V NNI),
             denominator: L Record(coeff: I,expon: V NNI))  164.642 (EV)
```

References

[AL 86] Adleman, L.M., Lenstra, H.K., *Finding Irreducible Polynomials over Finite Fields*, Proc. 18^{th} ACM STOC (1986), pp. 350–355.

[Be 70] Berlekamp, E.R., *Factoring Polynomials over Large Finite Fields*, Math. Comp. 24 (1970), pp. 713–735.

[BGH 82] Borodin, A., von zur Gathen, J., Hopcroft, J., *Fast Parallel Matrix and GCD Computation*, Proc. 23^{th} IEEE FOCS (1982), pp. 65–71.

[Bl 83] Blahut, R.E., *The Theory and Practice of Error Control Codes*, Addison-Wesley Publishing Co., 1983.

[BT 88] Ben-Or, M., Tiwari, P.A., *A Deterministic Algorithm for Sparse Multivariate Polynomial Interpolation*, Proc. 20^{th} ACM STOC (1988), pp. 301–309.

[CDGK 88] Clausen, M., Dress, A., Grabmeier, J., Karpinski, M., *On Zero-Testing and Interpolation of k-sparse Multivariate Polynomials over Finite Fields*, Research Report No. 8522-CS, University of Bonn.

[CGK 87] Clausen, Grabmeier, J., Karpinski, M., *Efficient Deterministic Interpolation of Multivariate Polynomials over Finite Fields*, Research Report No. 8519-CS, University of Bonn.

[Co 85] Cook, S.A., *A Taxomony of Problems with Fast Parallel Algorithms*, Information and Control 64 (1985), pp. 2–22.

[Ga 83] von zur Gathen, J., *Factoring Sparse Multivariate Polynomials*, Proc. 24^{th} IEEE FOCS (1983), pp. 172–179.

[Ga 84] von zur Gathen, J., *Parallel Algorithms for Algebraic Problems*, SIAM J. Comput. 13 (1984), pp. 808–824.

[Ga 86] von zur Gathen, J., *Representations and Parallel Computation for Rational Functions*, SIAM J. Comput. 15 (1986), pp. 432–452.

[GK 87] Grigoriev, D.Y., Karpinski, M., *The Matching Problem for Bipartite Graphs with Polynomially Bounded Permanent is in NC*, Proc. 28^{th} IEEE FOCS (1997), pp.166–172.

[GK 88] Grigoriev, D.Y., Karpinski, M., *A Deterministic Algorithm for Rational Function Interpolation*, Research Report No. 8526-CS, University of Bonn.

[GKS 88] Grigoriev, D.Y., Karpinski, M., Singer, M.F., *Fast Parallel Algorithms for Sparse Multivariate Polynomial Interpolation over Finite Fields*, Research Report No. 8523-CS, University of Bonn.

[JSW 87] Jenks, R.D., Sutor, R.S., Watt S.M., *Scratchpad II: An Abstract Datatype System for Mathematical Computation*, Trends in Computer Algebra, Springer Lecture Notes in Computer Science Vol. 296.

[Ka 85] Kaltofen, E., *Computing with Polynomials Given by Straight-Line Programs I: Greatest Common Divisors*, Proc. 17^{th} ACM STOC (1985), pp. 131–142.

[KT 88] Kaltofen, E., Trager, B., *Computing with Polynomials Given by Black Boxes for their Evaluations: Greatest Common Divisors, Sparse Factorization, Separation of Numerators and Denominators*, Proc. 29^{th} IEEE FOCS (1988), pp. 296–305. (1988).

[KY 88] Kaltofen, E., Yagati, L., *Improved Sparse Polynomial Interpolation Algorithms*, to appear: Proc. ISSAC (1988), Springer Lecture Notes in Computer Science.

[KR 88] Karp, R.M., Remachandran, V., *A Survey of Parallel Algorithms for Shared-Memory Machines*, Research Report No. UCB/CSD 88/407, University of California, Berkeley (1988), to appear in Handbook of Theoretical Computer Science, North Holland (1989).

[Le 83] Lenstra, A.K., *Factoring Multivariate Polynomials over Finite Fields*, Proc. 15^{th} ACM STOC (1983), pp. 189–192.

[Lo 83] Loos, R., *Computing Rational Zeros of Integer Polynomials by p-adic Expansions*, SIAM J. Comput. 12 (1983), pp. 262–293.

[LN 86] Lidl, R., Niederreiter, H., *Introduction to Finite Fields and their Applications*, Cambridge University Press 1986.

[Mu 86] Mulmuley, K., *A Fast Parallel Algorithm to Compute the Rank of a Matrix over an Arbitrary Field*, Proc. 18^{th} STOC ACM (1986), pp. 338–339.

[St 73] Strassen, V., *Vermeidung von Divisionen*, Reine und Angewandte Mathematik 264 (1973), pp. 184–202.

[Sc 80] Schwartz, T.J.., *Fast Probabilistic Algorithms for Verification of Polynomial Identities*, J. ACM 27,4 (1980), pp. 701–717.

[Ti 87] Tiwari, P.A., *Deterministic Algorithms for Multivariate Polynomial Interpolation*, IBM T.J. Watson Research Center: Technical Report.

[WS 80] Werner, H., Schaback, R., *Praktische Mathematik II*, Springer-Verlag 1980.

[Zi 79] Zippel, R.E., *Probabilistic Algorithms for Sparse Polynomials*, Proc. EUROSAM'79 Springer Lecture Notes in Computer Science Vol 72. (1979), pp. 216–226.

[Zi 88] Zippel, R.E., *Interpolating Polynomials from their Values*, Manuscript, Symbolics Inc. (1988).