Factorization of Polynomials Given by Straight-Line Programs*

Erich Kaltofen

Rensselaer Polytechnic Institute
Department of Computer Science
Troy, New York 12181

and

Mathematical Sciences Research Institute
1000 Centennial Drive
Berkeley, California 94720

Abstract

An algorithm is developed for the factorization of a multivariate polynomial represented by a straight-line program into its irreducible factors. The algorithm is in random polynomial-time as a function in the input size, total degree, and binary coefficient length for the usual coefficient fields and outputs a straight-line program, which with controllably high probability correctly determines the irreducible factors. It also returns the probably correct multiplicities of each distinct factor. If the coefficient field has finite characteristic p and p divides the multiplicities of some irreducible factors our algorithm constructs straight-line programs for the appropriate p-th powers of such factors.

Also a probabilistic algorithm is presented that allows to convert a polynomial given by a straight-line program into its sparse representation. This conversion algorithm is in random-polynomial time in the previously cited parameters and in an upper bound for the number of non-zero monomials permitted in the sparse output. Together with our factorization algorithm we therefore can probabilistically determine all those sparse irreducible factors of a polynomial given by a straight-line program that have less than a given number of monomials. We show that this result is valid without any restriction on the characteristic of the coefficient field.

The first section of this paper also summarizes the history of the polynomial factorization problem, and the last section discusses what questions for this problem remain to be resolved. We have also attempted to provide an extensive list of references on the subject, so that this paper can serve as a starting point for someone without previous knowledge in polynomial factorization.

1. The Problem of Factoring Polynomials

* This material is based upon work supported by the National Science Foundation under Grant No. DCR-85-04391 and by an IBM Faculty Development Award. The results of §6 were originally announced in the paper "Uniform closure properties of p-computable functions" [32]. This paper appears in *Randomness and Computation*, edited by S. Micali, vol. 5 of the *Advances in Computing Research* series, JAI Press Inc., Greenwich, Connecticut, pp. 375-412 (1989).

November 17, 1987

"The invention of divisors of universal quantities," what we refer today as the computation of factors of polynomials, was already taught by Newton in 1673 and the method subsequently published in his Arithmetica Universalis [46]. In 1882 Kronecker [36], pp. 10-13, reduced the problem of factoring multivariate polynomials over algebraic number fields to factoring univariate polynomials over the integers, for which he applied Newton's algorithm. Van der Waerden's influential text [63] discusses those algorithms and suggests that for larger problems they are not very practical. Nonetheless, early computer programs realized this classical approach [23] and verified that it is quite inefficient. The ensuing search for efficient algorithms to factor polynomials is a fine example in the discipline of the design and analysis of algorithms as well as complexity theory and exhibits many of the techniques developed for these subjects.

In 1967 Berlekamp [2] found an algorithm to factor univariate polynomials over moderately sized finite fields in time proportional to the cube of the input degrees. Berlekamp's algorithm is the first evidence that polynomial factorization is not as complex a problem as is integer factoring. However, his algorithm performed badly when applied to large finite fields. Berlekamp's own resolution of this problem in 1970 [3] is remarkable in that by introducing the selection of random field elements the algorithm could be exponentially sped up. Thus the factorization algorithm over large finite fields became one of the forerunners of "randomized" algorithms. We also refer to Rabin's 1976 version of this algorithm [48] for his appealing probability analysis, and to the book [35], §4.6.2, for a discussion of additional work. Recently, the problem of removing the random choices from the algorithm without sacrificing polynomial running-time has been resolved for several special cases by help of interesting new ideas, and we refer to the two exemplary papers [53] and [20]. The performance in practice of the randomized algorithms for univariate polynomial factorization over large finite fields is quite satisfactory and, at the moment, far superior to any known deterministic algorithms.

The advances in factoring polynomials modulo a prime integer suggested to apply these algorithms to factoring polynomials with integer coefficients as well. Zassenhaus in 1969 [67] pointed to the "Hensel Lemma" [18], §4, as a means to reconstruct the integral factors from modular ones. Unlike the factorization algorithm for polynomials over finite fields, however, the reconstruction procedure for the integral polynomial factors from the modular ones can have exponential complexity due to "combinatorical explosion" [3] and [34]. "Probabilistic analysis" [45] and [6] shows, however, that this problem does not arise on "average" inputs and implementations of the Berlekamp-Hensel algorithm for factoring univariate integral polynomials perform quite well, except for very special inputs. However, such inputs can arise and are, in fact, generated by Kronecker's reduction from algebraic number fields, for instance. In 1982 a remarkable diophantine algorithm was found by A. K. Lenstra, H. Lenstra, and Lovász [42] to overcome the combinatorical explosion by a polynomial-time construction. Several more classical problems could then be shown to also belong to the polynomial-time complexity class, for example solvability by radicals [38], factorization of univariate polynomials over algebraic number fields [60] and [37], and the multivariate polynomial factorization problem.

Already in 1971 Musser [44] demonstrated that "Hensel Lifting," as the procedure applying the Hensel lemma is now called, can be also applied to reconstruct multivariate from univariate factors. Combinatorical explosion is still a problem, but not all mappings to the univariate

factorization are categorically bad. This is a consequence of the famous Hilbert Irreducibility Theorem [19], and the first polynomial-time reduction by Kaltofen found in late 1981 is based on an effective deterministic version of that theorem [24], also in [26], §7. A number of different polynomial-time reductions from multivariate to univariate polynomial factorization are known today [4], [13], [26], [31], [40], and [41]. All these algorithms assume that all possible terms count towards the input size, in other words the multivariate polynomials are represented "densely."

If the number of variables in the multivariate factorization problem is allowed to grow with the problem size, then the "sparseness" of the input and output polynomials need to be taken into considerations. Wang upon considering the very sparse examples presented by Claybrook [5] invented several heuristics to cope with the intermediate "expression swell" occurring for sparse inputs and outputs [64]. Zippel in 1979 carried these considerations further by introducing randomization into the Hensel lifting process [69]. In order to make a rigorous analysis of the failure probabilities, an effective probabilistic version of the Hilbert irreducibility theorem was needed. Although already Heintz and Sieveking [17] had provided such a theorem for algebraically closed fields, in 1983 von zur Gathen proved a suitable version for arbitrary coefficient fields [11] and applied it to the sparse factoring problem [14]. In retrospect, Kaltofen's effective Hilbert irreducibility theorem also lent itself to an even simpler probabilistic version [27]. In [14] sparse polynomials are described that possess irreducible factors with super-polynomially more terms. These examples imply that any sparse Hensel Lifting scheme can have more than polynomial running time on certain inputs. It became clear that to to deal with this phenomenon the sparse representation had to be replaced by a more powerful one.

The usage of "straight-line programs" as a means to compute certain polynomials has been developed in the framework of complexity theory in the past decade, refer for example to [57], [58], [47], [59], [50], and [16]. In 1983 von zur Gathen [11] combined his probabilistic Hilbert irreducibility theorem with the probabilistic method of straight-line program evaluation [54] and [21] to find the factor degree pattern of polynomials defined by straight-line computations. A previously known operation on polynomials in straight-line representation is that of taking first order partial derivatives [1]. Although there is evidence that other operations such as higher partial derivatives are inherently complex [62], the greatest common divisor problem of polynomials in straight-line program representation is in probabilistic polynomial-time, as shown by Kaltofen in 1985 [33]. In this paper we show that straight-line programs for the irreducible factors of a polynomial given by a straight-line program can also be found in probabilistic polynomial-time. With Zippel's 1979 sparse polynomial interpolation algorithm [68] our factorization result resolves all problems left open in [69], [14], and [28]. We note that, unlike the randomized solutions for factorization of univariate polynomials over large finite fields, the probabilistic solutions are of the Monte-Carlo kind, "probably correct and always fast." The failure probability can, of course, be made arbitrarily small.

2. Discussion of Results

A straight-line program is a sequence of arithmetic assignments to new variables, the operands of which are either constants, indeterminates, or previously assigned variables. The operators allowed are addition, subtraction, multiplication, and division. Our algorithms treat this straight-line program as a data structure to represent the polynomials computed by them. This representation can define in polynomial-space families of polynomials with exponentially many individual terms, such as determinants by Gaussian elimination sequences. Unlike in the algebraic complexity theory applications, where a straight-line program is a model of computation, our algorithms must not only produce straight-line results of polynomial-length but also perform the transformations efficiently, that is in random polynomial-time.

It appears proper that we explicitly define the model of algebraic computation in which our algorithms can be formulated. Our model is the *sequential probabilistic algebraic random access machine* (RAM), with which we not only manage computations over an abstract algebraic domain but also resolve the question of random element selections from the abstract fields. For concrete domains such as the rational numbers we also establish binary polynomial running time, even if then the algorithms would be formulated on the probabilistic Turing machine model. We think that the algebraic RAM model is in the spirit of new algebraic computing languages such as Scratchpad II [22].

The factorization algorithm presented here takes as input a straight-line program computing a polynomial and outputs a straight-line program and multiplicities for irreducible polynomials that with controllably small error probability determine the irreducible factors of the input polynomial. If the multiplicities are divisible by the characteristic of the coefficient field, our output is slightly different. The factorization algorithm calls a bivariate polynomial factorization procedure and is therefore only effective and of polynomial running time for the usual coefficient fields. We measure the running time as a function in the input size and input polynomial degree. Over the rationals, for instance, we get an algorithm of binary complexity that is a polynomial function in the binary size of the straight-line program determining the input polynomial, in its total degree and the size of the numerators and the common denominator of its rational coefficients, and in the logarithm of the inverse of the probability that the output program incorrectly determines the irreducible factors or their multiplicities.

The key idea of our algorithm, which we will present and analyze in §5, in addition to previously known approaches, is to employ Hensel lifting but to replace the p-adic expansion of the coefficients by the expansions into homogeneous parts of the minor variables. We thus lift all minor variables simultaneously and avoid the variable by variable lifting loop that would compound programs of exponential size. This method can be viewed as a combination of Strassen's trick for eliminating divisions in straight-line computations [57] and Yun's Hensel lifting scheme [66]. If the coefficient field is of positive characteristic p and the multiplicity of an irreducible factor is divisible by p, there arises an additional problem. We can, however, compute a straight-line computation for the appropriate p^k-th power of such a factor.

For completeness we present in §3 our version of Zippel's conversion algorithm [68] from straight-line to sparse polynomial representation. Our algorithm is of polynomial complexity in

the size of the straight-line program defining the input polynomial, in its total degree, and in an upper bound t for the maximal number of monomials permitted in the sparse output. The algorithm produces either a sparsely represented polynomial with no more than t monomials or a message indicating that the input polynomial has more than t terms. The algorithm is Monte-Carlo and can give a wrong answer, that with controllably small probability. Over the rational numbers the algorithm is of binary polynomial running time also in the coefficient size of the input polynomial and the logarithm of the inverse of the failure probability. We believe that our conversion algorithm is a general and useful way in which Zippel's sparse interpolation scheme can be formulated.

Let us now come back to the question of factorizing into sparse polynomials. The examples causing super-polynomial blow-up for the size of the answer have the property that many other factors are very sparse. In general, one may wish to retrieve the sparse factors as such and leave the dense factors in straight-line format. Fortunately, the sparse conversion algorithm discussed before allows to do just that. More precisely, given a bound t we now can probabilistically determine in polynomial-time also in t the sparse format of all irreducible factors with no more than t terms, this without any restriction on characteristic and multiplicities. Moreover, the running time is always polynomial even if we were unlucky in our choice of evaluation points. We think that this finally settles the question of sparse factorization in a very satisfactory manner.

The next section introduces the model of probabilistic algebraic RAMs, defines straight-line programs, and summarizes results needed from other sources. In §4 we present the conversion algorithm to sparse format and in §5 the theorems on probabilistically preserving the factor degree pattern. §6 contains the straight-line polynomial factorization algorithm. We conclude in §7 with a discussion of open problems in connection with the polynomial factorization problem.

3. Definitions and Previous Results

We now repeat the main notions and results presented in [33]. We denote the field of rational numbers by **Q** and the finite field with q elements by $\mathbf{F}_q$. An *algebraic RAM over F*, F a field, has a CPU which is controlled by a finite sequence of labelled instructions and which has access to an infinite address and data memory (see fig. 1).

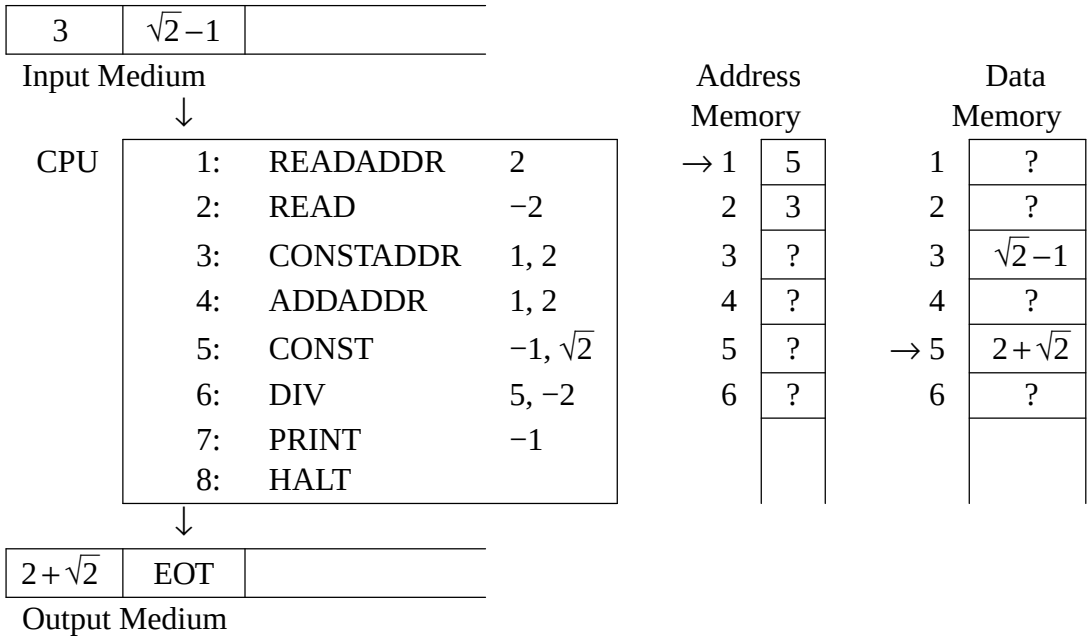

Fig. 1: Algebraic RAM over $\mathbf{Q}(\sqrt{2})$.

The split into two memories, one that facilitates pointer manipulation for array processing as well as maintaining a stack for recursive procedures, and another memory in which the algebraic arithmetic is carried out, is also reflected in other models for algebraic computations such as the parallel arithmetic networks in [12] or by the omnipresence of the built-in type Integer in the Scratchpad II language [22]. Each word in address memory can hold an integral address and each word in data memory can store an element in F. The CPU also has access to an input and an output medium. The instructions in the CPU may have one or two operands which typically are integers. The operands refer to words in address or data memory depending whether the instruction is an address or a data instruction. Indirect addressing is indicated by a negative operand. For completeness the micro-code for a full instruction set is given in fig. 2.

The *arithmetic time* and *space complexity* of an algebraic RAM for a given input are defined as the number of instructions executed and the highest memory address referenced, respectively. It is not always realistic to charge for each arithmetic operation in F one time unit. We will consider encoding data in binary and define as size(a), $a \in F$, where F is a concrete field such as **Q** or $\mathbf{F}_q$, the number of bits needed to represent a. Then the cost and space of an arithmetic instruction depends on the size of its operands. The *binary* time and space complexity of an algebraic RAM over F is derived by charging for each arithmetic step in F as many units as are needed to carry out the computation on a multitape Turing machine. Notice that we

Instruction		*Description*
ADD{ADDR}	i, j	$Op_i \leftarrow Op_i + Op_j$ (see below).
SUB{ADDR}	i, j	$Op_i \leftarrow Op_i - Op_j$.
MULT{ADDR}	i, j	$Op_i \leftarrow Op_i \times Op_j$.
DIVADDR	i, j	$Op_i \leftarrow \lfloor Op_i / Op_j \rfloor$.
DIV	i, j	$Op_i \leftarrow Op_i / Op_j$.
CONST{ADDR}	i, c	$Op_i \leftarrow c$.
MOVE{ADDR}	i, j	$Op_i \leftarrow Op_j$.
JMP{ADDR}	l	Execution continues at program label l.
JMPZ{ADDR}	i, l	If $Op_i = 0$ then execution continues at program label l.
JMPGZADDR	i, l	If $Op_i > 0$ then execution continues at program label l.
READ{ADDR}	i	The input medium is advanced and the next item is read into Op_i.
PRINT{ADDR}	i	The output medium is advanced and Op_i is written onto the medium.
HALT		An EOT marker is written onto the output tape and execution terminates.

$$Op_i = \begin{cases} \text{AM}[i] & \text{if } i > 0 \text{ and address} \\ \text{DM}[i] & \text{if } i > 0 \text{ and data} \\ \text{AM}[\text{AM}[-i]] & \text{if } i < 0 \text{ and address} \\ \text{DM}[\text{AM}[-i]] & \text{if } i < 0 \text{ and data} \end{cases} \text{ instruction}$$

AM = address memory, DM = data memory

AM[$-i$] must be positive, otherwise an interrupt occurs.

Fig. 2: Summary of algebraic RAM instructions

generally assume that the field arithmetic can be carried out in polynomial binary complexity with respect to the size of the operands. What that implies in particular is that elements in $\mathbf{F}_q$, say, always require $O(\log(q))$ representation size indendent whether they are residues of small integral value or not. For READ, PRINT, CONST, MOVE, or JMPZ instructions we charge as many units as is the size of the transferred or tested element.

We also apply this "logarithmic cost criterion" to the address computations and assume that every address is represented as a binary integer. The binary cost for performing address arithmetic is again the Turing machine cost. For indirect addressing we add the size of the final address to the binary time and space cost of the corresponding instruction. We note that in most circumstances the binary cost for performing address arithmetic is by far dominated by the binary cost of the algebraic operations and that for all practical purposes the largest storage location is of constant size. But our more precise measure has its advantages. First, all binary polynomial-time algorithms on algebraic RAMs are also polynomial-time in the Turing machine model. Second, the true binary complexity is measured if we can use the address memory for more than address computations, e.g. for hashing with sophisticated signatures. Another such example is that of selecting random field elements.

A *probabilistic* algebraic RAM is endowed with the additional instruction

RANDOM{ADDR} i, j

with the following meaning. Into Op_i an element $\in F$ (or address) is stored that was uniformly and randomly polled from a set R of elements (or integers) with card(R) equal to the address operand Op_j (see fig. 2 for the definition of Op). The selection of R is unknown except all its elements $a \in R$ have size(a) = $O(\log Op_j)$. This model of randomized algebraic computation overcomes the problem of how to actually generate a "random" rational number, say, and, as we will show later, the failure probabilities can in our circumstances be fully analyzed. Now we only note that for a non-zero polynomial f the probability

$$\text{Prob}(f(a_1,\ldots,a_n) = 0 \mid a_i \in R) \leq \frac{\deg(f)}{\text{card}(R)}, \qquad (\dagger)$$

see [54].

Our algorithms will read as input, produce as intermediate results, and print as output straight-line programs. Let us first precisely define what we mean, see also [56].

Definition: Let F be a field, $X = \{x_1, \ldots, x_n\}$ a set of indeterminates. Then $P = (X, V, C, S)$ is an *algebraic straight-line program* over $K = F(x_1, \ldots, x_n)$ if

(SLP1) $S = \{s_1, \ldots, s_k\} \subset F$, $V = \{v_1, \ldots, v_l\}$, $V \cap K = \emptyset$. X is called the set of *inputs*, V the set of (program) *variables*, S the set of *scalars*.

(SLP2) $C = (v_\lambda \leftarrow v'_\lambda \circ_\lambda v''_\lambda)_{\lambda=1,\ldots,l}$ with $\circ_\lambda \in \{+, -, \times, \div\}$, $v'_\lambda, v''_\lambda \in S \cup X \cup \{v_1, \ldots, v_{\lambda-1}\}$ for all $\lambda = 1, \ldots, l$. C is called the *computation sequence* and l the *length* of P, l = len(P).

(SLP3) For all $\lambda = 1, \ldots, l$ there exists sem(v_λ) $\in K$, the *semantics* of v_λ, such that

$$\begin{aligned}
&\text{sem}(a) = a \text{ if } a \in S \cup X,\\
&\text{sem}(v_\lambda) = \text{sem}(v'_\lambda) \pm \text{sem}(v''_\lambda) \text{ if } \circ_\lambda = \pm,\\
&\text{sem}(v_\lambda) = \text{sem}(v'_\lambda)\,\text{sem}(v''_\lambda) \text{ if } \circ_\lambda = \times,\\
&\text{sem}(v''_\lambda) \neq 0 \text{ and } \text{sem}(v_\lambda) = \text{sem}(v'_\lambda)/\text{sem}(v''_\lambda) \text{ if } \circ_\lambda = \div.
\end{aligned}$$

The *set of elements* computed by P is sem(P) = $\bigcup_{\lambda=1}^{l} \{\text{sem}(v_\lambda)\}$. □

We say $f \in F[x_1, \ldots, x_n]$ is *given by* the straight-line program $P = (X, V, C, S)$ if $f \in$ sem(P). Notice that we use the notation $f \in$ sem(P) with the implied understanding that we also know the $v_\lambda \in V$ with f = sem(v_λ). Straight-line programs are originally meant to be evaluated at points $\phi(x_i) \in F$. It can happen that such an evaluation is impossible due to a division by zero. We say that P is *defined at* ϕ: $\{x_1, \ldots, x_n\} \to F$ if a division by zero does not occur during evaluation of P at $\phi(x_i)$ in place of x_i, $1 \leq i \leq n$.

Here we will not describe a concrete data structure that can be used to represent straight-line programs on an algebraic RAM. It is fairly easy to conceive of suitable ones, e.g. labeled directed acyclic multigraphs could be used. A more intricate data structure was used for the first implementation of our algorithms and is described in [8]. At this point it is convenient to define

the *element size* of a straight-line program as

$$\text{el-size}(P) = \sum_{v_\lambda^* \in X \cup S,\, * \in \{',''\}} \text{size}(v_\lambda^*).$$

Notice that the actual size of P is in bits

$$O(\text{len}(P) \log \text{len}(P) + \text{el-size}(P)),$$

since it takes $\text{size}(v_\lambda) = O(\log(\lambda))$ bits to represent v_λ in address memory.

We now reproduce the input and output specifications of those algorithms presented in [33], which we will need for the algorithms discussed in this paper.

Algorithm *Zero-Division Test*

Input: A straight-line program $P = (\{x_1,\ldots, x_n\}, V, C, \{s_1,\ldots, s_m\})$ of length l over $\mathbf{Q}(x_1,\ldots, x_n)$, $a_\nu \in \mathbf{Q}$, $1 \le \nu \le n$, and a failure probability $\varepsilon \ll 1$.

Output: An integer p such that P is defined at ψ with $\psi(x_\nu) = a_\nu \bmod p$, $\psi(s_\mu) = s_\mu \bmod p$, or "failure". The latter occurs with probability $< \varepsilon$ in case P is defined at ϕ given by $\phi(x_\nu) = a_\nu$. □

Algorithm *Evaluation*

Input: As in algorithm Zero-Division Test. Furthermore an index λ, $1 \le \lambda \le l$, and a bound B_λ.

Output: Either "failure" (that with probability $< \varepsilon$ in case P is defined at ϕ) or $e_\lambda = \text{sem}(\phi(v_\lambda))$ provided that

$$|\text{num}(e_\lambda)|, |\text{den}(e_\lambda)| \le B_\lambda. \quad \square$$

Both algorithms have a binary complexity of order $(l \log(B) \log(1/\varepsilon)^{O(1)}$ on a probabilistic algebraic RAM over $\mathbf{Q}$, where $B = \max(\text{size}(a_\nu), \text{size}(s_\mu), B_\lambda)$ [33].

Algorithm *Polynomial Coefficients*

Input: $f \in F[x_1,\ldots, x_n]$ given by a straight-line program $P = (\{x_1,\ldots, x_n\}, V, C, S)$ over $F(x_1,\ldots, x_n)$ of length l, a failure probability $\varepsilon \ll 1$, and a bound $d \ge \deg_{x_1}(f)$.

Output: Either "failure", this with probability $< \varepsilon$, or a straight-line program $Q = (\{x_2,\ldots, x_n\}, V_Q, C_Q, S_Q)$ over $F(x_2,\ldots, x_n)$ such that

$$\{c_0,\ldots, c_d\} \subset \text{sem}(Q) \quad \text{and} \quad \text{len}(Q) = O(l\, d + M(d) \log d),$$

where $c_\delta \in F[x_2,\ldots, x_n]$ satisfies

$$f = \sum_{\delta=0}^{d} c_\delta(x_2,\ldots, x_n) x_1^\delta.$$

Here and later $M(d)$ denotes a function dominating the time for multiplying polynomials in $F[x]$ of maximum degree d. Notice that for arbitrary fields the best known upper bound for $M(d)$ is $O(d \log(d) \log\log(d))$ [51]. □

The running-time of this algorithm is summarized by the following theorem, which is typical for our theory.

Theorem 3.1: Algorithm Polynomial Coefficients does not fail with probability $> 1 - \varepsilon$. It requires polynomially many arithmetic steps in d and l on a probabilistic algebraic RAM over F. For $F = \mathbf{Q}$ and $F = \mathbf{F}_q$ its binary complexity is also polynomial in el-size(P) and $\log(1/\varepsilon)$ [33], Theorem 5.1. □

The Polynomial Coefficients algorithm requires the knowledge of a bound $d \geq \deg_{x_1}(f)$. If no such bound is given, we can probabilistically guess the degree by running our algorithm for

$$d = 1, 2, 4, \ldots, 2^k, \cdot\ \cdot\ \cdot.$$

Let $f_k(x_1, \ldots, x_n)$ be the interpolation polynomial that is produced for the k-th run. We then choose $a_1, \ldots, a_n \in R$ randomly and probabilistically test whether

$$f(a_1, \ldots, a_n) - f_k(a_1, \ldots, a_n) = 0.$$

This test can be performed by a simple modification of the Zero-Division Test algorithm, and the chance that the difference is falsely determined as 0 can be made smaller than ε. The probability that the randomly selected a_i certify the inequality of f and f_k can by (†) be made exponentially close to 1. Of course, by further testing $c_\delta(x_2, \ldots, x_n)$ for zero, $\delta = 2^k, 2^k - 1, \cdot\ \cdot\ \cdot$ we can get a probabilistic estimate for the actual degree $\deg_{x_1}(f)$. This procedure has expected polynomial running time in $\deg_{x_1}(f)$, and can be made quite efficient by computing the f_k incrementally [8]. The total degree of f can be similarly estimated by testing $\deg_{x_1}(\tilde{f})$, where

$$\tilde{f}(x_1, \ldots, x_n) = f(x_1, x_2 + b_2 x_1, \ldots, x_n + b_n x_1)$$

with b_i randomly selected [33], Lemma 5.1. A similar algorithm is also described in [11], Remark 5.4. More general, one can even probabilistically determine the degrees of the numerator and denominator of a rational function computed by the input program, and therefore one can probabilistically test whether it computes a polynomial to start with, cf [30],. Corollary 4.1.

4. Conversion into Sparse Polynomials

We now discuss our version of Zippel's [68] sparse interpolation algorithm for converting a polynomial from its straight-line to its sparse representation. The *sparse representation* for

$$f(x_1,\ldots,x_n) = \sum_{(e_1,\ldots,e_n)\in J} c_{e_1,\ldots,e_n} x_1^{e_1}\cdot\cdot\cdot x_n^{e_n}, \quad 0 \neq c_{e_1,\ldots,e_n} \in F, \quad J \subset \mathbf{N}^n$$

is the vector

$$((e_1,\ldots,e_n,\ c_{e_1,\ldots,e_n}))_{(e_1,\ldots,e_n)\in J}.$$

Here **N** denotes the set of non-negative integers. We write $\mathrm{mon}(f) = \mathrm{card}(J)$, the number of monomials in f, and $\mathrm{skel}(f) = J$, the *skeleton* or set of non-zero monomial exponents of f. Zippel's algorithm is based on the idea that during the variable by variable interpolation process any zero coefficient is, with high probability, the image of a zero polynomial. We first present the algorithm for general fields. Extra difficulties arising from coefficient size growth are dealt with afterwards.

Algorithm *Sparse Conversion*

Input: $f \in F[x_1,\ldots,x_n]$ given by a straight-line program P of length l. Furthermore, a bound $d_0 \geq \max_{1\leq i\leq n}\{\deg_{x_i}(f)\}$, the allowed failure probability $\varepsilon \ll 1$, and an upper bound $t \leq (d_0+1)^n$ for the number of monomials permitted in the answer.

Output: Either "failure" (that with probability $< \varepsilon$), or the representation of a sparse polynomial with no more than t monomials, or the message "f has (probably) more than t monomials." The latter two outputs are correct with probability $> 1-\varepsilon$.

Step R (Select Initial Evaluation Points): From a set $R \subset F$ with

$$\mathrm{card}(R) > \frac{1}{\varepsilon}\max\left(n\deg(f)(d_0+1)^n,\ (n(d_0+1)t+1)\,2^{l+1} + n\deg(f)(d_0+1)t\right)$$

select random elements $a_2,\ldots,a_n \in R$. Notice that if $\deg(f)$ is not known one can use the crude upper bound $\deg(f) \leq nd_0$.

Step L (Interpolation Loop): **For** $i \leftarrow 1,\ldots,n$ **Do** Step I. Then return $\sum c_{e_1,\ldots,e_n} x_1^{e_1}\cdot\cdot\cdot x_n^{e_n}$, $c_{e_1,\ldots,e_n} \neq 0$.

Step I (Interpolate One More Variable): At this point we have with high probability correctly computed the sparse representation of

$$f(x_1,\ldots,x_{i-1},a_i,\ldots,a_n) = \sum_{(e_1,\ldots,e_{i-1})\in J_i} c_{e_1,\ldots,e_{i-1}} x_1^{e_1}\cdot\cdot\cdot x_{i-1}^{e_{i-1}},\ 0 \neq c_{e_1,\ldots,e_{i-1}} \in F,\ J_i \subset \mathbf{N}^{i-1}.$$

For $i = 1$ we have $J_1 = \{\varnothing\}$. We need not know $c_\varnothing = f(a_1,\ldots,a_n)$. Set

$$j_i \leftarrow \mathrm{card}(\{(e_1,\ldots,e_{i-1},\delta) \mid (e_1,\ldots,e_{i-1}) \in J_i,\ 0 \leq \delta \leq d_{e_1,\ldots,e_{i-1}}\}),$$

where $d_{e_1,\ldots,e_{i-1}} = \min(d_0, \deg(f) - e_1 - \cdots - e_{i-1})$.

For $k \leftarrow 1, \ldots, j_i$ **Do**

From the subset R select random points $b_{k,1}, b_{k,2}, \ldots, b_{k,i} \in R$. Compute

$$g_{k,i} = f(b_{k,1}, \ldots, b_{k,i}, a_{i+1}, \ldots, a_n)$$

by evaluating P at $\phi_{k,i}(x_\mu) = b_{k,\mu}$, $1 \le \mu \le i$, $\phi_{k,i}(x_\nu) = a_\nu$, $i+1 \le \nu \le n$. If P is not defined at $\phi_{k,i}$ return "failure".

Solve the j_i by j_i linear system

$$\sum_{(e_1, \ldots, e_{i-1}) \in J_i} \sum_{\delta=0}^{d_{e_1,\ldots,e_{i-1}}} \gamma_{e_1,\ldots,e_{i-1},\delta}\, b_{k,1}^{e_1} \cdot \cdots \cdot b_{k,i-1}^{e_{i-1}}\, b_{k,i}^{\delta} = g_{k,i}, \quad 1 \le k \le j_i, \tag{$\dagger$}$$

in the indeterminates $\gamma_{e_1,\ldots,e_{i-1},\delta}$. If the system is singular, report "failure".

Set $c_{e_1,\ldots,e_i} = \gamma_{e_1,\ldots,e_i}$, where the RHS ranges over all non-zero components of the solution of the above system. Notice that the subscripts of these components define the set J_{i+1}. If the number of those non-zero coefficients becomes more than t, return "input polynomial has (probably) more than t monomials." □

The challenging part is the verification of the failure and incorrectness probabilities. For this, it is helpful to prove the following lemma.

Lemma 4.1: Let $J_i \subset \mathbf{N}^i$, $\mathrm{card}(J_i) = j_i < \infty$. Then

$$\Delta_i = \det([\beta_{k,1}^{e_1} \cdot \cdots \cdot \beta_{k,i}^{e_i}]_{(e_1,\ldots,e_i) \in J_i;\ k=1,\ldots,j_i})$$

is a non-zero polynomial in $F[\beta_{1,1}, \ldots, \beta_{j_i,i}]$.

Proof: Simply observe that the monomial contributed by the main diagonal of the determinant is unique. □

We now have the following theorem:

Theorem 4.1: Algorithm Sparse Conversion does not fail and outputs the correct result with probability $1 - 2\varepsilon$. In that case it requires

$$O(n(l\, d_0\, t + d_0^3 t^3))$$

arithmetic steps on a probabilistic algebraic RAM over a (sufficiently large) field F.

Proof: Each of the $j_i \le (d_0+1)t$ evaluation in step I requires $O(l)$ arithmetic steps. Solving the j_i by j_i system takes $O(j_i^3)$ steps. Notice that this bound also includes setting up the linear system from J_i and $g_{k,i}$. Step I is executed n times, which shows the stated complexity.

We now analyze the probabilistic behavior of the algorithm. Let us first assume that the algorithm does not fail. A correct answer is returned provided the system (†) captures for all i every non-zero monomial coefficient of $f(x_1, \ldots, x_i, a_{i+1}, \ldots, a_n)$. Let

$$f(x_1, \ldots, x_i, \alpha_{i+1}, \ldots, \alpha_n) = \sum_{(e_1,\ldots,e_i) \in \hat{J}_{i+1}} \hat{c}_{e_1,\ldots,e_i}\, x_1^{e_1} \cdot \cdots \cdot x_i^{e_i},$$

$0 \neq \hat{c}_{e_1,\ldots,e_i} \in F[\alpha_{i+1},\ldots,\alpha_n]$, and let

$$\sigma_i = \prod_{(e_1,\ldots,e_i)\in \hat{J}_{i+1}} \hat{c}_{e_1,\ldots,e_i}, \quad \deg(\sigma_i) \leq \mathrm{mon}(f)\deg(f).$$

Notice that in general $\hat{J}_{i+1} \supseteq J_{i+1}$. But $\sigma_i(a_{i+1},\ldots,a_n) \neq 0$ implies that $\hat{J}_{i+1} = J_{i+1}$, which in turn means that the unique solution to (†) must determine $f(x_1,\ldots,x_i,a_{i+1},\ldots,a_n)$. Since $\mathrm{mon}(f) \leq (d_0+1)^n$ the probability

$$\mathrm{Prob}(\sigma_i(a_{i+1},\ldots,a_n) \neq 0 \text{ for all } 1 \leq i \leq n)$$

is not less than

$$1 - \sum_{i=1}^{n} \frac{\deg(\sigma_i)}{\mathrm{card}(R)} \geq 1 - \frac{n\,\mathrm{mon}(f)\deg(f)}{\mathrm{card}(R)} \geq 1 - \frac{n\deg(f)(d_0+1)^n}{\mathrm{card}(R)} > 1 - \varepsilon.$$

We now estimate the failure probability. We define the events

$$E_0 = \{(a_2,\ldots,a_n) \mid P \text{ is defined at } \phi_0(x_1) = x_1,\ \phi_0(x_i) = a_i,\ 2 \leq i \leq n\}$$

and

$$E_{k,i} = \{(b_{k,1},\ldots,b_{k,i}) \mid P \text{ is defined at } \phi_{k,i}\}.$$

As in [33], Lemma 4.2, we have

$$\mathrm{Prob}(E_0), \quad \mathrm{Prob}(E_{k,i}|E_0) \geq 1 - \frac{2^{l+1}}{\mathrm{card}(R)}.$$

Since $j_i \leq (d_0+1)t$ we get

$$\mathrm{Prob}(E_0 \cap \bigcap_{\substack{i=1,\ldots,n \\ k=1,\ldots,j_i}} E_{k,i}) \geq (1 - \frac{2^{l+1}}{\mathrm{card}(R)}) \prod_{\substack{i=1,\ldots,n \\ k=1,\ldots,j_i}} (1 - \frac{2^{l+1}}{\mathrm{card}(R)})$$

$$\geq 1 - \frac{(n(d_0+1)t+1)2^{l+1}}{\mathrm{card}(R)}.$$

Now by lemma 4.1 for a given i the coefficient matrix for (†) is non-singular with probability $\geq$

$$1 - \frac{\deg(\Delta_i)}{\mathrm{card}(R)} \geq 1 - \frac{\deg(f)j_i}{\mathrm{card}(R)} \geq 1 - \frac{\deg(f)(d_0+1)t}{\mathrm{card}(R)}.$$

Thus all n arising systems are non-singular with probability $\geq 1 - n\deg(f)(d_0+1)t/\mathrm{card}(R)$. Therefore, the algorithm fails with probability $\leq$

$$\frac{1}{\mathrm{card}(R)}\left((n(d_0+1)t+1)\,2^{l+1} + n\deg(f)(d_0+1)t\right) \quad \square$$

We wish to remark that the input parameters d_0 and t need not be specified beforehand. In §3 we have discussed how to probabilistically determine $d_i = \deg_{x_i}(f)$. In fact, the Sparse Conversion algorithm runs more efficiently if we use d_i in place of d_0 for the i-th iteration of step I.

The parameter t is used only to abort execution in case f has too many monomials or that we are interpolating with unlucky evaluation points. By adjusting card(R) appropriately we can achieve expected polynomial running time also in mon(f) without the input parameter t. In the context of an actual computer algebra system we prefer our formulation of the algorithm, whose running time is independent of bad random choices.

We now discuss the complications arising for $F = \mathbf{Q}$. Our requirement is to accomplish binary polynomial-time complexity. It is clear that we must include the *coefficient size* of f,

$$\text{c-size}(f) = \max_{(e_1,\ldots,e_n)\in \text{skel}(f)} \{\text{size}(c_{e_1,\ldots,e_n})\},$$

into our input parameters. One might think that all we have to do is use the Evaluation algorithm of §3 inside the FOR loop of step I and adjust the failure probability accordingly. Unfortunately, there exists a theoretical possibility that size($g_{k,i}$) is exponential in n. This would happen, for instance, if all denominators of $c_{e_1,\ldots,e_n}$ were distinct primes and mon(f) were exponential in n. A way to overcome this problem is to perform the entire conversion modulo p, p an integer that has been tested to be a prime with probability $\geq 1 - \varepsilon$ [55], [49], and retrieve the rational coefficients of f by a continued fraction approximation from the coefficients of f mod p as in step C of the cited Evaluation algorithm. The pseudo-prime p must be selected such that also with probability $\geq 1 - \varepsilon$, P is defined at $\phi(x_v) = x_v$, $\phi(s) = s \bmod p$ for all $s \in S$ (cf. the Zero-Division Test algorithm cited in §3), and such that

$$p \geq \text{card}(R), \quad 2^{2\text{c-size}(f)+1}.$$

In practice, it is better to work modulo $\bar{p}^k$ at the danger of increasing the failure probability. Then one avoids the generation of the rather large pseudo-prime p, and one can also solve the linear system (†) $\bar{p}$-adically [8]. For the record, let us state the following theorem.

Theorem 4.2: For $F = \mathbf{Q}$, algorithm Sparse Conversion, if used in conjunction with a probabilistic primality test, the Zero-Division Test algorithm, and a recovery procedure for rational numbers from their modular images, can complete and determine a correct answer with probability $\geq 1 - 3\varepsilon$. Its binary running time is polynomial in l, d_0, log(1/ε), t, el-size(P) and the additional input parameter that is a bound for c-size(f). □

An interesting result concerning the conversion of a straight-line program to a sparse rational function is a direct consequence of this theorem and the Numerator and Denominator algorithm in [30].

Corollary 4.1: Let f/g be given by a straight-line program P, $f, g \in F[x_1, \ldots, x_n]$, GCD($f$, g) = 1, $d \geq \deg(f), \deg(g)$, $0 < \varepsilon \ll 1$. In order to avoid ambiguity assume that the coefficient of the lexicographically first monomial in g is 1. Provided the sparse representation of f, respectively g, has less than t monomials, it can be computed correctly with probability $> 1 - \varepsilon$ on a probabilistic algebraic RAM over F in polynomially many arithmetic steps in len(P), d, and t. In case $F = \mathbf{Q}$ the binary running time is also polynomial in el-size(P), log(1/ε), and c-size(f), respectively c-size(g). □

Before we can apply theorem 4.2 to the Polynomial GCD algorithm in [33] we must introduce a slightly more restricted notion of coefficient size of f, that where the coefficients are already brought to a common denominator. Assume that

$$c_{e_1,\ldots,e_n} = \frac{u_{e_1,\ldots,e_n}}{u_*}, \quad u_{e_1,\ldots,e_n}, u_* \in \mathbf{Z} \text{ for all } (e_1,\ldots,e_n) \in \text{skel}(f).$$

Then the *combined coefficient size* of f is defined as

$$\text{cc-size}(f) = \text{size}(u_*) + \max_{(e_1,\ldots,e_n) \in \text{skel}(f)} \{\text{size}(u_{e_1,\ldots,e_n})\}.$$

Now since the size of the coefficients of integral multivariate polynomial factors can be polynomially bounded [15], Chapter III, §4, Lemma II, we obtain from the straight-line GCD algorithm in [33] the following typical corollary.

Corollary 4.2: Let $f_\rho \in F[x_1, \ldots, x_n]$ be given by a straight-line program P, $d \geq \deg(f_\rho)$, $1 \leq \rho \leq r$, $g = \text{GCD}_{1\leq\rho\leq r}(f_\rho)$, $0 < \varepsilon \ll 1$. Provided the sparse representation of g has less than t monomials, it can be computed correctly with probability $> 1-\varepsilon$ on a probabilistic algebraic RAM over F in polynomially many arithmetic steps in len(P), d, and t. In case $F = \mathbf{Q}$ the binary running time is also polynomial in el-size(P), log(1/ε), and $\min_{1\leq\rho\leq r}$ {cc-size(f_ρ)}. □

Notice that we cannot yet prove the above corollary for c-size(f_ρ) replacing cc-size(f_ρ). Therefore, one might question whether our restriction is reasonable. The answer is that for three large subclasses of polynomial representations, namely

Sparse polynomials, Formulas, and Determinants,

the combined coefficient size as well as the degrees are polynomially related to the input size. In fact, we know of no example for a straight-line program representing a polynomial of polynomially bounded degree and coefficient size, but where the combined coefficient size becomes exponential.

We shall conclude this section with a remark on counting the number of monomials. Clearly, the Sparse Conversion algorithm can probabilistically produce the number of monomials in a polynomial given by a straight-line program in time polynomial in the unary representation of that count. One may question whether it is possible to find the number of monomials in binary in random polynomial-time. This is most likely not the case due to the fact that the evaluation of 0-1 permanents is #P-hard [61]. For if we replace all 1 entries in a 0-1 matrix by indeterminates $x_{i,j}$, i the corresponding row and j the corresponding column, then the number of monomials in the determinant of the new matrix is equal to the permanent of the original 0-1 matrix. Therefore the problem of counting the number of monomials in families of polynomials with polynomially bounded degree and straight-line computation length, which Valiant calls p-computable [62], is #P-hard.

5. Evaluation and Factor Degree Pattern

It is crucial for our Factoring algorithm that the Hensel lifting is started with true factor images. Fortunately, the effective versions of the Hilbert irreducibility theorem [11] and [27] make it possible to probabilistically enforce this assumption. In this section we present a theorem (Theorem 5.2) on the probabilities that certain evaluations preserve the *factor degree pattern* that determines the number of irreducible factors, their multiplicities, and their total degrees. The argument is essentially the same as that in [11], Theorem 3.6, but with our effective version of the Hilbert irreducibility theorem (Theorem 5.1). The main advantage of this change is that the evaluations are simpler and the probability of success is higher.

Theorem 5.1 (*Effective Hilbert Irreducibility Theorem*): Let $f(x_1, \ldots, x_n) \in F[x_1, \ldots, x_n]$, F a field, have total degree d and be irreducible. Furthermore, assume that x_2 occurs in f, that is $\deg_{x_2}(f) > 0$. If $\mathrm{char}(F) = p > 0$ we require that each coefficient of f in F possesses a p-th root in F. A sufficient condition for this to be true is that F be perfect. Let $R \subset F$ and let $a_1, a_3, \ldots, a_n, b_3, \ldots, b_n$, be random elements in R. Then the probability

$$\mathrm{Prob}(f(x_1 + a_1, x_2, b_3 x_1 + a_3, \ldots, b_n x_1 + a_n)$$

$$\text{becomes reducible in } F[x_1, x_2]) \le \frac{4d\ 2^d}{\mathrm{card}(R)}.$$

For a proof see [27], Theorem 3. □

In the following association between two polynomials f and g is denoted by $f \sim g$ and means that $f = cg$ with $0 \neq c \in F$. The factor degree pattern of $f \in F[x_1, \ldots, x_n]$ is defined as a lexicographically ordered vector $((d_i, e_i))_{i=1,\ldots,r}$ such that for $f = \prod_{i=1}^{r} h_i^{e_i}$, $h_i \in F[x_1, \ldots, x_n]$,

$$h_i \text{ irreducible}, \quad d_i = \deg(h_i) \ge 1, \quad 1 \le i \le r, \quad h_i \nmid h_j, \quad 1 \le i \neq j \le r.$$

We want to apply theorem 5.1 to the irreducible factors of a multivariate polynomial. However, theorem 5.1 will only apply to those factors that depend on x_2. Therefore we need the following notion. The *primitive part* of a polynomial with respect to a variable is the polynomial divided by the GCD of all (polynomial) coefficients of that variable. We denote it by $\mathrm{pp}_x(.)$, where x is the corresponding variable. In particular, if no factors are independent of x we call the polynomial primitive in x.

Theorem 5.2: Let $f \in F[x_1, \ldots, x_n]$, F a perfect field, $d = \deg(f)$, $R \subset F$. Let $a_1, a_3, \ldots, a_n$, $b_3, \ldots, b_n \in R$ be randomly selected elements,

$$f_2 = f(x_1 + a_1, x_2, b_3 x_1 + a_3, \ldots, b_n x_1 + a_n).$$

Then

$$\mathrm{Prob}(\mathrm{pp}_{x_2}(f) \text{ and } \mathrm{pp}_{x_2}(f_2) \text{ have the same factor degree pattern}) \ge 1 - \frac{4d\,2^d + d^3}{\mathrm{card}(R)}.$$

Proof: First we consider all the factors h_i with $\deg_{x_2}(h_i) > 0$. By theorem 5.1,

$$h_{i,2} = h_i(x_1 + a_1,\ x_2,\ b_3x_1 + a_3,\ldots,\ b_nx_1 + a_n)$$

remains irreducible in $F[x_1,\ x_2]$ with probability $\geq 1 - 4d_i2^{d_i}/\text{card}(R)$. It remains to estimate the probability that $\deg(h_{i,2}) = d_i$ and that $h_{i,2} \nmid h_{j,2}$ for all $j \neq i$. Let

$$\hat{h}_i(x_1, x_2, \alpha_1, \alpha_3,\ldots, \alpha_n, \beta_3,\ldots, \beta_n) = h_i(x_1 + \alpha_1,\ x_2,\ \beta_3x_1 + \alpha_3,\ldots,\ \beta_nx_1 + \alpha_n),$$

$\hat{h}_i \in F[x_1,\ x_2,\ \alpha_1,\ \alpha_3\ ,\ldots,\ \alpha_n,\ \beta_3\ ,\ldots,\ \beta_n]$. Clearly, $\deg_{x_1,x_2}(\hat{h}_i) = d_i$. Let

$$0 \neq \pi_i(\beta_3,\ldots,\ \beta_n) \in F[\beta_3,\ldots,\ \beta_n]$$

be the coefficient of a monomial $x_1^{j_1}x_2^{j_2}$, $j_1 + j_2 = d_i$, in $\hat{h}_i$. Now $\deg(\pi_i) \leq d_i$ and

$$\pi_i(b_3,\ldots,\ b_n) \neq 0 \ \text{ implies } \ \deg(h_{i,2}) = d_i.$$

By [54], Lemma 1, this happens with probability $\geq$

$$1 - \frac{\deg(\pi_i)}{\text{card}(R)} \geq 1 - \frac{d_i}{\text{card}(R)}.$$

We finally estimate the chance that $h_{i,2} \nmid h_{j,2}$. First we claim that $\hat{h}_i \nmid \hat{h}_j$, $i \neq j$, in $\bar{F}[x_1,\ x_2]$, $\bar{F} = F(\alpha_1,\ \alpha_3\ ,\ldots,\ \alpha_n,\ \beta_3\ ,\ldots,\ \beta_n)$. For if this were not the case, then there would exist non-zero g_i, $g_j \in F[\alpha_1,\ \alpha_3\ ,\ldots,\ \alpha_n,\ \beta_3\ ,\ldots,\ \beta_n]$, GCD$(g_i,\ g_j) = 1$, such that $(g_i/g_j)\,\hat{h}_i = \hat{h}_j$. Hence either one of $\hat{h}_i$ or $\hat{h}_j$ would have to be reducible in $F[x_1,\ x_2,\ \alpha_1,\ \alpha_3\ ,\ldots,\ \alpha_n,\ \beta_3\ ,\ldots,\ \beta_n]$, say $\hat{h}_i = \hat{h}_i^{(1)}\ \hat{h}_i^{(2)}$. However then

$$h_i = (\hat{h}_i^{(1)}\ \hat{h}_i^{(2)})(x_1 - \alpha_1, x_2, \alpha_1, x_3 - \beta_3(x_1 - \alpha_1),\ldots, x_n - \beta_n(x_1 - \alpha_1), \beta_3,\ldots, \beta_n)$$

would be a non-trivial factorization of h_i which would necessarily have to lie in $F[x_1\ ,\ldots,\ x_n]$, in contradiction to the irreducibility of h_i. This shows non-associativity of $\hat{h}_i$ and $\hat{h}_j$ over $\bar{F}$. We now have two coefficients of $\hat{h}_i$ in $\bar{F}$, that is

$$\hat{h}_i = \cdots + \sigma_i^{(\lambda_1,\lambda_2)}x_1^{\lambda_1}x_2^{\lambda_2} + \cdots + \sigma_i^{(\mu_1,\mu_2)}x_1^{\mu_1}x_2^{\mu_2} + \cdots, \quad \sigma_i^{(\lambda_1,\lambda_2)},\ \sigma_i^{(\mu_1,\mu_2)} \in \bar{F},$$

and two corresponding coefficients in $\hat{h}_j$,

$$\hat{h}_j = \cdots + \sigma_j^{(\lambda_1,\lambda_2)}x_1^{\lambda_1}x_2^{\lambda_2} + \cdots + \sigma_j^{(\mu_1,\mu_2)}x_1^{\mu_1}x_2^{\mu_2} + \cdots, \quad \sigma_j^{(\lambda_1,\lambda_2)},\ \sigma_j^{(\mu_1,\mu_2)} \in \bar{F},$$

such that

$$\tau_{i,j} = \sigma_i^{(\lambda_1,\lambda_2)}\sigma_j^{(\mu_1,\mu_2)} - \sigma_i^{(\mu_1,\mu_2)}\sigma_j^{(\lambda_1,\lambda_2)} \neq 0.$$

Now $\tau_{i,j} \in F[\alpha_1,\ \alpha_3\ ,\ldots,\ \alpha_n,\ \beta_3\ ,\ldots,\ \beta_n]$ and it is relatively easy to see that

$$\tau_{i,j}(a_1,\ a_3,\ldots,\ a_n,\ b_3,\ldots,\ b_n) \neq 0 \ \text{ implies } \ h_{i,2} \nmid h_{j,2}.$$

Since $\deg(\tau_{i,j}) \leq d_i + d_j$ the probability of this event is $\geq 1 - (d_i + d_j)/\text{card}(R)$.

Now we consider those h_i with $\deg_{x_2}(h_i) = 0$. All that must be satisfied for the theorem to hold is that $h_{i,2}$ as defined above is not identical zero. Again the total degree of h_i gets preserved with

probability $d_i/\mathrm{card}(R)$, which is a sufficient condition. Overall, the factor degree pattern is preserved with probability not less than

$$1 - \left(\sum_{i=1}^{r} \frac{4 d_i 2^{d_i}}{\mathrm{card}(R)} + \sum_{i=1}^{r} \frac{d_i}{\mathrm{card}(R)} + \sum_{1 \le i < j \le r} \frac{d_i + d_j}{\mathrm{card}(R)} \right)$$

$$\ge 1 - \left(\frac{4d\ 2^d}{\mathrm{card}(R)} + \frac{d}{\mathrm{card}(R)} + \frac{d(d-1)}{2} \frac{d}{\mathrm{card}(R)} \right) \ge 1 - \frac{4d\ 2^d + d^3}{\mathrm{card}(R)}. \quad \square$$

One can probabilistically enforce that the input polynomial is primitive in x_2 by making the linear substitution $x_i + c_i x_2$ for all x_i, $i \neq 2$, with randomly chosen c_i. This substitution does not effect the factor degree pattern. It should be clear from the above theorem that we thus can probabilistically obtain the factor degree pattern of a polynomial given by a straight-line program by evaluation. We formulated theorem 5.2 in its generality because we will make a slightly different substitution in the Factorization algorithm in §6. Moreover, the theorem in its current form can be used to also compute the degrees of individual variables in the factors. One lets each variable take the role of x_2 and identifies the factors in the different bivariate domains by evaluating that variable at a linear form. However, since this result is not needed in the following, we shall skip the details.

The assumption that the field is perfect can be dropped at the cost of increasing the failure probability somewhat (cf [11],. Lemma 4.2), but since the usual coefficient fields are perfect we do not incorporate this generalization.

6. Straight-Line Factorization

We now describe the algorithm for finding the straight-line factors of a straight-line polynomial. The algorithm is derived from the One-Variable Lifting algorithm in [28], with the homogeneous parts of the minor variables replacing the monomials of the single variable with respect to which is lifted. Note that a homogeneous polynomial of degree d has the form

$$\sum_{e_1+\cdots+e_n=d} c_{e_1,\ldots,e_n} x_1^{e_1} \cdot \cdots \cdot x_n^{e_n}, \quad c_{e_1,\ldots,e_n} \in F.$$

We will compute those homogeneous parts by straight-line programs. The main reason why the answer is polynomial in length is that we only need to add on to the intermediate programs. This is because subsequent homogeneous parts can be computed from previous ones and Strassen's technique of obtaining a homogeneous program need not be applied at each iteration.

Algorithm *Factorization*

Input: $f \in F[x_1, \ldots, x_n]$ given by a straight-line program P of length l, a bound $d \geq \deg(f)$, and an allowed failure probability $\varepsilon \ll 1$.

Output: Either "failure", that with probability $< \varepsilon$, or $e_i \geq 1$ and irreducible $h_i \in F[x_1, \ldots, x_n]$, $1 \leq i \leq r$, given by a straight-line program Q of length

$$\text{len}(Q) = O(d^2 l + d\, M(d^2) \log(d))$$

such that with probability $> 1 - \varepsilon$, $f = \prod_{i=1}^{r} h_i^{e_i}$. (Refer to algorithm Polynomial Coefficients in §3 for the definition of $M(.)$.) In case $p = \text{char}(F)$ divides any e_i, that is $e_i = p^{\hat{e}_i}\, \bar{e}_i$ with $\bar{e}_i$ not divisible by p, we return $\bar{e}_i$ in place of e_i and Q will compute $h_i^{p^{\hat{e}_i}}$.

Step R (Random Points Selection): From a set $R \subset F$ with

$$\text{card}(R) > \frac{6}{\varepsilon} \max(2^{l+2},\, d\,2^d + d^3,\, 2(d+1)^4)$$

select random elements $a_1, \ldots, a_n, b_2, \ldots, b_n, c_1, c_3, \ldots, c_n$. If $F = \mathbf{F}_q$ with q small we can instead work over $\mathbf{F}_{q^p}$, p a prime integer $> d$. By Theorem 6.1 in [11] no additional factors occur.

Test whether P is defined at $\phi(x_i) = a_i$, $1 \leq i \leq n$. For $F = \mathbf{Q}$ we call algorithm Zero-Division Test in [33] such that the probability of "failure" even if P were defined at ϕ is less than $\varepsilon/6$. If P turns out to be (probably) undefined at ϕ we return "failure". Otherwise, P is definitely defined at ϕ and we compute the dense representation of

$$f_2 = f(x_1 + c_1 x_2 + a_1,\, x_2 + b_2 x_1 + a_2,\, b_3 x_1 + c_3 x_2 + a_3, \ldots, b_n x_1 + c_n x_2 + a_n).$$

This can be done by evaluation and interpolation similarly to the Sparse Conversion algorithm. If $F = \mathbf{Q}$, a bound for the cc-size(f) must be added to the input parameters and we must again make the probability that "failure" occurs due to the use of modular arithmetic during evaluation less than $\varepsilon/6$.

Step F (Factorization): Factor

$$f_2 = \prod_{i=1}^{r} \tilde{g}_{i,2}^{e_i},$$

$\tilde{g}_{i,2} \in F[x_1, x_2]$ irreducible and pairwise not associated. Notice that by theorem 5.1 f and f_2 have with high probability the same factor degree pattern, that is irreducible factors of f map to pairwise non-associated irreducible factors of f_2 of the same total degrees. For the remainder of the algorithm we will assume that this is the case.

If $\mathrm{char}(F) = p > 0$ divides any of the e_i, say $e_i = p^{\hat{e}_i}\,\bar{e}_i$ with $\bar{e}_i$ not divisible by p, we replace e_i by $\bar{e}_i$ and $\tilde{g}_{i,2}$ by $\tilde{g}_{i,2}^{p^{\hat{e}_i}}$. This replacement guarantees that none of the multiplicities are divisible by the characteristic.

Now set

$$g_{i,0}(x_1) \leftarrow \tilde{g}_{i,2}(x_1, 0) \in F[x_1].$$

Check whether $\mathrm{GCD}(g_{i,0}, g_{j,0}) \sim 1$ for $1 \le i < j \le r$ and whether $\deg(\tilde{g}_{i,2}) = \deg(g_{i,0})$ for $1 \le i \le r$. If not return "failure".

Let

$$\bar{f}(x_1,\ldots,x_n) = f(x_1 + a_1, x_2 + b_2 x_1 + a_2,\ldots,x_n + b_n x_1 + a_n) = \prod_{i=1}^{r} h_i(x_1,\ldots,x_n)^{e_i},$$

and assume that h_i are the factors that correspond to $\tilde{g}_{i,2}$. Notice that the assumptions on the preservation of the total degrees of the factors throughout the evaluation process also imply that

$$\mathrm{ldcf}_{x_1}(\bar{f}) \in F. \qquad (*)$$

Here ldcf_{x_1} denotes the coefficient of the highest power of x_1 and is generally a polynomial in $F[x_2, \ldots, x_n]$. Furthermore, let $\bar{P}$ be a straight-line program computing $\bar{f}$. We write

$$\bar{f}(x_1,\ldots, x_n) = \sum_{j=0}^{d} \sum_{m=0}^{d} \bar{f}_{j,m}(x_2,\ldots, x_n)\, x_1^m,$$

where $\bar{f}_{j,m} \in F[x_2, \ldots, x_n]$ is homogeneous of degree j. We remark that d can now be set to $\deg(\bar{f})$ rather than a bound for it. We will need a straight-line program that computes $\bar{f}_{j,m}$. If we replace x_i by $x_i x_1^{d+1}$, $2 \le i \le n$, in $\bar{P}$ then $\bar{f}_{j,m}$ is the coefficient of $x_1^{j(d+1)+m}$. Therefore by evaluating at x_1 and interpolating as in the Polynomial Coefficients algorithm (§3) we can find a straight-line program Q_0 for $\bar{f}_{j,m}$ of length

$$\mathrm{len}(Q_0) = O(d^2 l + M(d^2)\log(d)).$$

Notice that we need to randomly pick $(d+1)^2$ distinct points at which we interpolate and we must make sure that the straight-line program $\bar{P}$ is defined at those points. If that is not the case, or if for $F = \mathbf{Q}$ we cannot confirm by the Zero-Division Test algorithm (§3) that a point is good, that with probability $< \varepsilon/(6(d+1)^2)$, we return "failure". For more details we refer to step P in the cited Polynomial Coefficients algorithm.

Step H (Hensel Lifting Loop): **For** $k \leftarrow 0, \ldots, d-1$ **Do** step L.

Step L (Lift by One Degree): Let

$$h_i(x_1, \ldots, x_n) = \sum_{m=0}^{d_i} \sum_{j=0}^{d_i} c_{i,j,m}(x_2, \ldots, x_n)\, x_1^m, \quad d_i = \deg(h_i),$$

where $c_{i,j,m}(x_2, \ldots, x_n) \in F[x_2, \ldots, x_n]$ is homogeneous of degree j. At this point we have a straight-line program Q_k over $F(x_2, \ldots, x_n)$ that computes $c_{i,j,m}$ for $1 \le i \le r$, $0 \le j \le k$, $0 \le m \le d_i$, and all $\bar{f}_{j,m}$, $0 \le j, m \le d$. Notice that $c_{i,0,m} \in F$ is the coefficient of x_1^m in $g_{i,0}$ found in step F, and therefore Q_0 need not encode them. Whenever reference to these coefficients is made later, we just encode them as scalars. Notice also that by (*) $c_{i,j,d_i} = 0$ for $j > 0$. We will extend Q_k to Q_{k+1} that also computes $c_{i,k+1,m}$. It is useful to introduce the following polynomials

$$g_{i,k} = \sum_{j=0}^{k} \sum_{m=0}^{d_i} c_{i,j,m} x_1^m, \quad \hat{g}_{i,k+1} = \sum_{m=0}^{d_i} c_{i,k+1,m} x_1^m.$$

Now consider the congruence

$$\prod_{i=1}^{r} (g_{i,k} + \hat{g}_{i,k+1})^{e_i} \equiv \bar{f} \bmod (x_2, \ldots, x_n)^{k+2}. \tag{$\dagger$}$$

Expanding the LHS we get

$$g_{1,0}^{e_1-1} \cdot \cdot \cdot g_{r,0}^{e_r-1} \sum_{i=1}^{r} (e_i \hat{g}_{i,k+1} \prod_{\substack{j=1 \\ j \ne i}}^{r} g_{j,0}) \equiv \bar{f} - \prod_{i=1}^{r} g_{i,k}^{e_i} \bmod (x_2, \ldots, x_n)^{k+2}. \tag{$\ddagger$}$$

By our loop invariant for Q_k

$$(\bar{f} - \prod_{i=1}^{r} g_{i,k}^{e_i}) \bmod (x_2, \ldots, x_n)^{k+2} = t_{k+1} = \sum_{m=0}^{d-1} t_{k+1,m}(x_2, \ldots, x_n)\, x_1^m,$$

where $t_{k+1,m} \in F[x_2, \ldots, x_n]$ is homogeneous of degree $k+1$. Notice that the degree of t_{k+1} in x_1 is $\le d-1$ by the assumption (*). We need a program T_{k+1} that computes $t_{k+1,m}$. However, T_{k+1} does not start from scratch, but references the program variables in Q_k that compute $c_{i,j,m}$ and $\bar{f}_{j,m}$. If T_{k+1} encodes a tree-like bivariate multiplication scheme with those program variables as undetermined coefficients, that can be done in

$$\text{len}(T_{k+1}) = O(M(d^2) \log(d)).$$

Now, since t_{k+1} equals the LHS of ($\ddagger$) $g_{1,0}^{e_1-1} \cdot \cdot \cdot g_{r,0}^{e_r-1}$ must divide t_{k+1} in $F[x_1, \ldots, x_n]$. Notice that this claim might not be valid if $g_{i,0}$ is not an image of h_i, since then the existence of the $\hat{g}_{i,k+1}$ cannot be guaranteed. However, in that case our construction still completes, but the resulting straight-line answer is incorrect. Let

$$u_{k+1} = \sum_{m=0}^{d_1+\cdots+d_r-1} u_{k+1,m}\, x_1^m = \frac{t_{k+1}}{g_{1,0}^{e_1-1} \cdot \cdot \cdot g_{r,0}^{e_r-1}}, \quad u_{k+1,m} \in F[x_2, \ldots, x_n].$$

Again, we need a straight-line program U_{k+1} that computes all $u_{k+1,m}$ from the program variables for $t_{k+1,m}$ in T_{k+1} as indeterminates. Since the leading coefficient in x_1 of $g_{1,0}^{e_1-1} \cdot \cdot \cdot g_{r,0}^{e_r-1}$ is an element in F, the $u_{k+1,m}$ can be determined by simply encoding a univariate polynomial division in x_1 over the coefficient field $F(x_2, \ldots, x_n)$. Therefore we can construct U_{k+1} of length $\mathrm{len}(U_{k+1}) = O(M(d))$. (Actually, the entire divisor is in $F[x_1]$ but our argument also applies to a quadratic lifting procedure, see the remark below the proof of theorem 6.1.) Now consider

$$\frac{u_{k+1}}{g_{1,0} \cdot \cdot \cdot g_{r,0}} = \frac{e_1 \hat{g}_{1,k+1}}{g_{1,0}} + \cdot \cdot \cdot + \frac{e_r \hat{g}_{r,k+1}}{g_{r,0}}.$$

It is clear that $e_i c_{i,j,k+1}$ are the coefficients of the univariate partial fraction decomposition of $u_{k+1}/(g_{1,0} \cdot \cdot \cdot g_{r,0})$ carried out over the field $F(x_2, \ldots, x_n)$. One way to compute these coefficients by a straight-line program $\hat{Q}_{k+1}$ with $\mathrm{len}(\hat{Q}_{k+1}) = O(d^2)$ is to once and for all find $\hat{g}_{i,0}^{(m)} \in F[x_1]$, $0 \le m \le d_1 + \cdot \cdot \cdot + d_r - 1$, with

$$\frac{x_1^m}{g_{1,0} \cdot \cdot \cdot g_{r,0}} = \frac{\hat{g}_{1,0}^{(m)}}{g_{1,0}} + \cdot \cdot \cdot + \frac{\hat{g}_{r,0}^{(m)}}{g_{r,0}}, \quad \deg(\hat{g}_{i,0}^{(m)}) < d_i,$$

and encode the summation

$$\hat{g}_{i,k+1} = \frac{1}{e_i} \sum_{m=0}^{d_1+\cdots+d_r-1} u_{k+1,m} \hat{g}_{i,0}^{(m)}, \quad 1 \le i \le r.$$

We must be able to divide by e_i and here we need the fact that the multiplicities must not be divisible by $\mathrm{char}(F)$. We finally link the programs Q_k, T_{k+1}, U_{k+1}, and $\hat{Q}_{k+1}$ properly together to obtain the program Q_{k+1}. Notice that

$$\mathrm{len}(Q_{k+1}) \le \mathrm{len}(Q_k) + C\, M(d^2) \log(d),$$

where C is an absolute constant. Therefore $\mathrm{len}(Q_{k+1}) = \mathrm{len}(Q_0) + O((k+1)\, M(d^2) \log(d))$.

Step T (Final Translation): From Q_d we obtain Q that computes

$$h_i(x_1 - a_1,\ x_2 - b_2(x_1 - a_1) - a_2, \ldots,\ x_n - b_n(x_1 - a_1) - a_n)$$

by adding in front of Q_d instructions for translating the x_i appropriately. □

The following theorem summarizes the complexity of the above algorithm.

Theorem 6.1: Algorithm Factorization does not fail with probability $> 1 - \varepsilon$. In that case it reduces the problem in polynomially many steps on a probabilistic algebraic RAM over F as a function in l and d to factoring bivariate polynomials. Its answer will be correct with probability $> 1 - \varepsilon$. It requires polynomially many randomly selected field elements. For $F = \mathbf{Q}$ or $F = \mathbf{F}_q$ the algorithm has binary polynomial complexity also in $\log(1/\varepsilon)$, el-size(P), and cc-size(f).

Proof: The arithmetic and binary running time is polynomial as a direct consequence of the results in [33], in particular Theorem 3.1, 4.1, and 5.1. It remains to analyze the failure probabilities of the Factorization algorithm. The only way an incorrect program Q can be produced is that the factor degree patterns of f and f_2 disagree. If $\deg_{x_2}(f) = \deg(f)$, which is true with

probability $> 1 - d/\mathrm{card}(R) > 1 - \varepsilon/12$, then by by theorem 5.2 this happens with probability less than

$$\frac{4d\,2^d + d^3}{\mathrm{card}(R)} < \frac{4\,\varepsilon}{6}.$$

Thus the compound probability of getting an incorrect result is $< \varepsilon$.

"Failure" can occur in six separate circumstances. First, P may be undefined at ϕ, that with probability $< 2^{l+1}/\mathrm{card}(R) < \varepsilon/6$ by Lemma 4.2 of [33]. Second, for $F = \mathbf{Q}$ we might fail to recognize that P is defined at ϕ, but we make this possibility happen with probability $< \varepsilon/6$. Third, for $F = \mathbf{Q}$ the computation of f_2 may fail with probability $< \varepsilon/6$.

Fourth, "failure" can occur if for some $i \neq j$, $\mathrm{GCD}(g_{i,0},\, g_{j,0}) \neq 1$, or $\deg(g_{i,0}) < \deg(\tilde{g}_{i,2})$. Let $\pi_i(\beta_2) = \mathrm{ldcf}_{x_1}(\tilde{g}_{i,2}(x_1,\, \beta_2 x_1 + \alpha_2))$ and let

$$\sigma_{i,j}(\alpha_2,\, \beta_2) = \mathrm{resultant}_{x_1}(\tilde{g}_{i,2}(x_1,\, \beta_2 x_1 + \alpha_2),\, \tilde{g}_{j,2}(x_1,\, \beta_2 x_1 + \alpha_2))$$

over $F[\alpha_2,\, \beta_2,\, x_1]$. It is easy to see that $0 \neq \pi_i\, \sigma_{i,j} \in F[\alpha_2,\, \beta_2]$ and $\pi_i(b_2)\, \sigma_{i,j}(a_2,\, b_2) \neq 0$ implies that the above events are impossible. Now, $\deg(\pi_i) \leq d_i$ and $\deg(\sigma_{i,j}) \leq 2d_i d_j$ and therefore the probability that the above events occur for any $i \neq j$ is less than

$$\sum_{i=1}^{r} \frac{d_i}{\mathrm{card}(R)} + \sum_{1 \leq i < j \leq r} \frac{2d_i d_j}{\mathrm{card}(R)} < \frac{(d_1 + \cdots + d_r)^2}{\mathrm{card}(R)} < \frac{d^2}{\mathrm{card}(R)} < \frac{\varepsilon}{6}.$$

Notice that if P were division-free, this event would be the only one where failure could occur.

Fifth, we may not find good interpolation points in order to produce Q_0. If we try at most $(d+1)^4$ points, the probability that at least $(d+1)^2 = d^*$ points are good can be estimated like in the proof of [33], Theorem 5.1. We shall repeat the argument here. An individual point was not picked earlier with probability $\geq 1 - (d+1)^4/\mathrm{card}(R) > 1 - \varepsilon/12$. $\bar{P}$ is not defined at an individual point substituted for x_1 with probability $< 2^{l+1}/\mathrm{card}(R) < \varepsilon/12$. Hence a suitable point can be found in a block of d^* points with probability $>$

$$1 - (\varepsilon^*)^{d^*} > 1 - \frac{\varepsilon^*}{d^*}, \quad \varepsilon^* = \frac{\varepsilon}{6},$$

because $(1/\varepsilon^*)^{d^*-1} > 2^{d^*} - 1 \geq d^*$ for $\varepsilon^* < 1/2$. Now the probability that a good point occurs in all of the d^* blocks of points is $>$

$$(1 - \frac{\varepsilon^*}{d^*})^{d^*} > 1 - \varepsilon^*,$$

and therefore failure happens with probability $< \varepsilon/6$. Sixth and last, for $F = \mathbf{Q}$ we may not recognize that we have good interpolating points, that for all $(d+1)^2$ points together with probability $< \varepsilon/6$. Summing up these failure probabilities completes the proof. □

We remark that our result in [26] would allow to further reduce the problem on an algebraic RAM over F to univariate factorization. We also mention that the input parameter d can be probabilistically estimated in expected polynomial-time in $\deg(f)$ (§3). Furthermore, the

algorithm could be formulated using quadratic lifting [25] in step L. Then the length of Q could be asymptotically reduced to $O(d^2 l + M(d^2)\log(d))$. Finally we mention that the binary polynomial-time upper bound can be easily generalized to F being an algebraic extension of **Q**.

We now formulate two corollaries to theorem 6.1. The first refers to computing the sparse factorization of f and follows from theorem 4.2.

Corollary 6.1: If in addition to the input parameters of the Factorization algorithm we are given $t > 0$, for $F = \mathbf{Q}$ or $F = \mathbf{F}_q$ we can find in polynomially many binary steps and random bit choices in

$$l,\ d,\ \log(\frac{1}{\varepsilon}),\ \text{el-size}(P),\ \text{cc-size}(f),\ \text{and } t$$

sparse polynomials that with probability $> 1 - \varepsilon$ constitute all irreducible factors of f with no more than t monomials. □

Notice that the above running time is always polynomial independently whether the correct sparse factors were produced or whether other factors are dense. This makes this corollary superior to all previous work on sparse factorization. The second corollary deals with possibly non-uniform closure. Again, in a family of p-computable polynomials the degrees computation lengths are polynomially bounded [62].

Corollary 6.2: Let F be a field of characteristic 0. Then any family of factors of a family of p-computable polynomials is p-computable. □

Notice that this corollary applies even to fields in which arithmetic is recursive but over which polynomial factorization is undecidable [9]. It also shows that a polynomial degree bound is necessarily required. We note that $x^{2^d} - 1$ can be computed with $O(d)$ instructions but it is known that over the complex numbers there exist factors that require $\Omega(2^{d/2}/\sqrt{d})$ computation length [43] and [50]. It would be nice to give such an example where the factors are irreducible polynomials over **Q**.

We have implemented the Factorization algorithm [8]. In order that len(Q) does not become too large, two practically important improvements to the Factorization algorithm as it is described above were made. First, the coefficients $\bar{f}_{j,m}$ are not computed a-priori but as they are needed for each k in the lifting loop. This is accomplished by using the Polynomial Coefficients algorithm in the original version of [33], which is based on Taylor series expansion. Second, the product $\prod_{i=1}^{r} g_{i,k}^{e_i}$ is also computed incrementally using the coefficients determined already for $k-1$ of the same product.

7. Conclusion

Aside from the predecessor paper [33] currently two more of our papers deal with the subject of manipulating polynomials in straight-line representation. In the forthcoming paper [32] we show how to replace the input parameter d in the Factorization algorithm by a degree bound for the individual factors. We also have implemented our algorithms in Lisp with an interface to Macsyma. The details of this first implementation together with practical improvements and our experience on test cases are reported in [8].

The question arises what major unresolved problems in the subject of polynomial factorization remain. It is appropriate to distinguish between theoretical and practical issues. One theoretical question is to remove the necessity of random choices from any of the problems known to lie within probabilistic polynomial-time, say factorization of univariate polynomials over large finite fields. Another problem is to investigate the parallel complexity of polynomial factorization, say for the **NC** model [7]. Kronecker's reduction from algebraic number coefficients [36], [60], and [37], Berlekamp's factorization algorithm over small finite fields [10], Kaltofen's deterministic Hilbert irreducibility theorem [26], §7, and Weinberger's irreducibility test for $\mathbf{Q}[x]$ [65] all lead to **NC** solutions by simply applying known **NC** methods for linear algebra problems. It is open whether factoring in $\mathbf{Q}[x]$ or irreducibility testing in $\mathbf{F}_p[x]$, p large, or in $\mathbf{Q}[x, y]$ can be accomplished in **NC**. We remark that testing a rational dense multivariate polynomial for absolute irreducibility can be shown to be in **NC** [29].

In connection with the Factorization algorithm presented here, we also mention an open question. Assume that a straight-line program computes a polynomial whose degree is exponential in the length of the program. Are then at least its factors of polynomially bounded degree p-computable? A positive answer to this question would show that testing a polynomial for zero in a suitable decision-tree model is polynomial-time related to computing that polynomial (cf [32],. §6, Problem 6). In general the theory of straight-line manipulation of polynomials may be extendable in part to unbounded input degrees, but even for the elimination of divisions problem [57] the answer is not known.

From a pragmatic point of view the main unresolved question is what role the polynomial-time polynomial factorization algorithms should play in computer algebra systems, that is in actually used implementations. The "L^3" algorithm [42] has been considered impractical by even one of the inventors, but that was not meant to imply that this algorithm is useless for polynomial factorization. In fact, using L^3 to recover algebraic numbers from their modular images leads to a practically competitive factoring algorithm for polynomials over algebraic number fields [39]. We submit that careful implementations of different lattice reduction schemes together with the complex root approximation method [52] might out-perform the Berlekamp-Hensel algorithm on hard-to-factor polynomials. The first implementation of the straight-line factorization algorithm is reported in [8]. There its practical merits have been demonstrated on very dense inputs such as symbolic determinants.

In summary, in this paper we were able to contribute to Valiant's algebraic counterpart of the theory of **P** vs. **NP** in the positive, that is establish a polynomial upper bound for a major problem in computational algebra. In fact, it comes to us as a small surprise that p-computable

polynomials are closed under factorization. And we have, finally, put to rest the problem of computing the sparse factorization of a multivariate polynomial.

Note added in proof: Since this chapter has been submitted, progress on several problems discussed can be reported. The sparse conversion problem in Section 4 has been solve more efficiently by Ben-Or and Tiwari [*Proc. 20th Annual ACM Symp. Theory Comput.* 301--309 (1988)], Zippel [*J. Symbolic Comput.* to appear (1990)], and by Lakshman Yagati and the author [*Proc. ISSAC 1988, Springer Lec. Notes Comput. Sci.* to appear (1989)]. John Canny and Barry Trager have made the author aware of a more effective version of the Hilbert Irreducibility Theorem 5.1, that essentially reduces the numerator of the probability bound to $d^{O(1)}$. Such a theorem also follows from methods presented in [K6], Section 5. Finally, Barry Trager and the author [*Proc. 29th Annual Symp. Foundations Comput. Sci.* 296--305 (1988)] have shown that another implicit representation for multivariate polynomials, that of black box programs that merely allow to evaluate the polynomials at given input points, can be used as input and output representation for polynomial-time polynomial factorization.

Acknowledgement: A discussion I had with Barry Trager a few years ago has helped me in developing §4. I also thank Joachim von zur Gathen and Gregory Imirzian for their valuable comments.

References

1. Baur, W. and Strassen, V., "The complexity of partial derivatives," *Theoretical Comp. Sci.*, 22, pp. 317-330 (1983).

2. Berlekamp, E. R., "Factoring polynomials over finite fields," *Bell Systems Tech. J.*, 46, pp. 1853-1859 (1967). Republished in revised form in: E. R. Berlekamp, *Algebraic Coding Theory*, Chapter 6, McGraw-Hill Publ., New York 1968.

3. Berlekamp, E. R., "Factoring polynomials over large finite fields," *Math. Comp.*, 24, pp. 713-735 (1970).

4. Chistov, A. L. and Grigoryev, D. Yu., "Polynomial-time factoring of multivariable polynomials over a global field," *LOMI preprint E-5-82*, Steklov Institute, Leningrad (1982).

5. Claybrook, B. G., "A new approach to the symbolic factorization of multivariate polynomials," *Artificial Intelligence*, 7, pp. 203-241 (1976).

6. Collins, G. E., "Factoring univariate integral polynomials in polynomial average time," *Proc. EUROSAM '79, Springer Lec. Notes Comp. Sci.*, 72, pp. 317-329 (1979).

7. Cook, S. A., "A taxonomy of problems with fast parallel algorithms," *Inf. Control*, 64, pp. 2-22 (1985).

8. Freeman, T. S., Imirzian, G., Kaltofen, E., and Yagati, Lakshman, "DAGWOOD: A system for manipulating polynomials given by straight-line programs," Tech. Report 86-15, Dept. Comput. Sci., RPI. Preliminary version in *Proc. 1986 ACM Symp. Symbolic Algebraic Comp.*, pp. 169-175, 1986.

9. Fröhlich, A. and Shepherdson, J. C., "Effective procedures in field theory," *Phil. Trans. Roy. Soc., Ser. A*, 248, pp. 407-432 (1955/56).

10. Gathen, J. von zur, "Parallel algorithms for algebraic problems," *SIAM J. Comp.*, 13, pp. 802-824 (1984).

11. Gathen, J. von zur, "Irreducibility of multivariate polynomials," *J. Comp. System Sci.*, 31, pp. 225-264 (1985).

12. Gathen, J. von zur, "Parallel arithmetic computation: A survey," *Proc. MFCS '86, Springer Lec. Notes Comp. Sci.*, 233, pp. 93-112 (1986).

13. Gathen, J. von zur and Kaltofen, E., "Factoring multivariate polynomials over finite fields," *Math. Comp.*, 45, pp. 251-261 (1985).

14. Gathen, J. von zur and Kaltofen, E., "Factoring sparse multivariate polynomials," *J. Comp. System Sci.*, 31, pp. 265-287 (1985).

15. Gelfond, A. O., *Transcendental and Algebraic Numbers*, Dover Publ., New York (1960).

16. Heintz, J., "A note on polynomials with symmetric galois group which are easy to compute," *Theor. Comp. Sci.*, 47, pp. 99-105 (1986).

17. Heintz, J. and Sieveking, M., "Absolute primality of polynomials is decidable in random polynomial-time in the number of variables," *Proc. ICALP '81, Springer Lec. Notes Comp. Sci.*, 115, pp. 16-28 (1981).

18. Hensel, K., *Theorie der algebraischen Zahlen*, Teubner, Leipzig (1908).

19. Hilbert, D., "Über die Irreduzibilität ganzer rationaler Funktionen mit ganzzahligen Koeffizienten," *J. reine angew. Math.*, 110, pp. 104-129 (1892).

20. Huang, M.-D. A., "Riemann hypothesis and finding roots over finite fields," *Proc. 17th ACM Symp. Theory Comp.*, pp. 121-130 (1985).

21. Ibarra, O. H. and Moran, S., "Probabilistic algorithms for deciding equivalence of straight-line programs," *J. ACM*, 30, pp. 217-228 (1983).

22. Jenks, R. D., “A primer: 11 keys to new SCRATCHPAD,” *Proc. EUROSAM ’84, Springer Lec. Notes Comp. Sci.,* 174, pp. 123-147 (1984).

23. Jordan, D. E., Kain, R. Y., and Clapp, L. C., “Symbolic factoring of polynomials in several variables,” *Comm. ACM,* 9, pp. 638-643 (1966).

24. Kaltofen, E., “A polynomial reduction from multivariate to bivariate integral polynomial factorization,” *Proc. 14th Annual ACM Symp. Theory Comp.,* pp. 261-266 (1982).

25. Kaltofen, E., “Polynomial factorization” in *Computer Algebra, 2nd ed.,* ed. B. Buchberger et al, pp. 95-113, Springer Verlag, Vienna (1982).

26. Kaltofen, E., “Polynomial-time reductions from multivariate to bi- and univariate integral polynomial factorization,” *SIAM J. Comp.,* 14, pp. 469-489 (1985).

27. Kaltofen, E., “Effective Hilbert irreducibility,” *Information and Control,* 66, pp. 123-137 (1985).

28. Kaltofen, E., “Computing with polynomials given by straight-line programs II; Sparse factorization,” *Proc. 26th IEEE Symp. Foundations Comp. Sci.,* pp. 451-458 (1985).

29. Kaltofen, E., “Fast parallel absolute irreducibility testing,” *J. Symbolic Computation,* 1, pp. 57-67 (1985).

30. Kaltofen, E., “Uniform closure properties of p-computable functions,” *Proc. 18th ACM Symp. Theory Comp.,* pp. 330-337 (1986).

31. Kaltofen, E., “Deterministic irreducibility testing of polynomials over large finite fields,” *J. Symbolic Comp.,* 3, pp. 77-82 (1987).

32. Kaltofen, E., “Single-factor Hensel lifting and its application to the straight-line complexity of certain polynomials,” *Proc. 19th Annual ACM Symp. Theory Comp.,* pp. 443-452 (1987).

33. Kaltofen, E., “Greatest common divisors of polynomials given by straight-line programs,” *J. ACM,* 35, 1, pp. 231-264 (1988).

34. Kaltofen, E., Musser, D. R., and Saunders, B. D., “A generalized class of polynomials that are hard to factor,” *SIAM J. Comp.,* 12, pp. 473-485 (1983).

35. Knuth, D. E., *The Art of Programming, vol. 2, Semi-Numerical Algorithms, ed. 2,* Addison Wesley, Reading, MA (1981).

36. Kronecker, L., “Grundzüge einer arithmetischen Theorie der algebraischen Grössen,” *J. reine angew. Math.,* 92, pp. 1-122 (1882).

37. Landau, S., “Factoring polynomials over algebraic number fields,” *SIAM J. Comp.,* 14, pp. 184-195 (1985).

38. Landau, S. and Miller, G. L., “Sovability by radicals,” *J. Comp. System Sci.,* 30, pp. 179-208 (1985).

39. Lenstra, A. K., “Lattices and factorization of polynomials over algebraic number fields,” *Proc. EUROCAM ’82, Springer Lec. Notes Comp. Sci.,* 144, pp. 32-39 (1982).

40. Lenstra, A. K., “Factoring multivariate integral polynomials,” *Theoretical Comp. Sci.,* 34, pp. 207-213 (1984).

41. Lenstra, A. K., “Factoring multivariate polynomials over finite fields,” *J. Comput. System Sci.,* 30, pp. 235-248 (1985).

42. Lenstra, A. K., Jr., H. W. Lenstra, and Lovász, L., “Factoring polynomials with rational coefficients,” *Math. Ann.,* 261, pp. 515-534 (1982).

43. Lipton, R. and Stockmeyer, L., “Evaluations of polynomials with superpreconditioning,” *Proc. 8th ACM Symp. Theory Comp.,* pp. 174-180 (1976).

44. Musser, D. R., "Multivariate polynomial factorization," *J. ACM*, 22, pp. 291-308 (1975).

45. Musser, D. R., "On the efficiency of a polynomial irreducibility test," *J. ACM*, 25, pp. 271-282 (1978).

46. Newton, I., *Arithmetica Universalis, 2nd ed.*, London (1728). In English. Reprinted in *The Mathematical Works of Isaac Newton*, vol. 2, D. T. Whiteside, ed., Johnson Reprint Corp., New York, 1967.

47. Paterson, M. S. and Stockmeyer, L. J., "On the number of nonscalar multiplications necessary to evaluate polynomials," *SIAM J. Comp.*, 2, pp. 60-66 (1973).

48. Rabin, M. O., "Probabilistic algorithms in finite fields," *SIAM J. Comp.*, 9, pp. 273-280 (1980).

49. Rabin, M. O., "Probabilistic algorithms for testing primality," *J. Number Theory*, 12, pp. 128-138 (1980).

50. Schnorr, C. P., "Improved lower bounds on the number of multiplications/divisions which are necessary to evaluate polynomials," *Theoretical Comp. Sci.*, 7, pp. 251-261 (1978).

51. Schönhage, A., "Schnelle Multiplikation von Polynomen über Körpern der Charakteristik 2," *Acta Inf.*, 7, pp. 395-398 (1977). (In German).

52. Schönhage, A., "Factorization of univariate integer polynomials by diophantine approximation and an improved basis reduction algorithm," *Proc. ICALP '84, Springer Lec. Notes Comp. Sci.*, 172, pp. 436-447 (1984).

53. Schoof, R. J., "Elliptic curves over finite fields and the computation of square roots mod p," *Math. Comp.*, 44, pp. 483-494 (1985).

54. Schwartz, J. T., "Fast probabilistic algorithms for verification of polynomial identities," *J. ACM*, 27, pp. 701-717 (1980).

55. Solovay, R. M. and Strassen, V., "A fast Monte-Carlo test for primality," *SIAM J. Comp.*, 6, pp. 84-85 (1977). Correction: vol. 7, p. 118 (1978).

56. Strassen, V., "Berechnung und Programm I," *Acta Inf.*, 1, pp. 320-335 (1972). (In German).

57. Strassen, V., "Vermeidung von Divisionen," *J. reine u. angew. Math.*, 264, pp. 182-202 (1973). (In German).

58. Strassen, V., "Die Berechnungskomplexität von elementarsymmetrischen Funktionen und von Interpolationskoeffizienten," *Numer. Math.*, 20, pp. 238-251 (1973). (In German).

59. Strassen, V., "Polynomials with rational coefficients which are hard to compute," *SIAM J. Comp.*, 3, pp. 128-149 (1974).

60. Trager, B. M., "Algebraic factoring and rational function integration," *Proc. 1976 ACM Symp. Symbolic Algebraic Comp.*, pp. 219-228 (1976).

61. Valiant, L., "The complexity of computing the permanent," *Theoretical Comp. Sci.*, 8, pp. 189-201 (1979).

62. Valiant, L., "Reducibility by algebraic projections," *L'Enseignement mathématique*, 28, pp. 253-268 (1982).

63. Waerden, B. L. van der, *Modern Algebra*, F. Ungar Publ. Co., New York (1953).

64. Wang, P. S., "An improved multivariate polynomial factorization algorithm," *Math. Comp.*, 32, pp. 1215-1231 (1978).

65. Weinberger, P. J., "Finding the number of factors of a polynomial," *J. Algorithms*, 5, pp. 180-186 (1984).

66. Yun, D. Y. Y., "The Hensel lemma in algebraic manipulation," Ph.D. Thesis, M.I.T. (1974). Reprint: Garland Publ., New York 1980.

67. Zassenhaus, H., "On Hensel factorization I," *J. Number Theory*, 1, pp. 291-311 (1969).

68. Zippel, R. E., "Probabilistic algorithms for sparse polynomials," *Proc. EUROSAM '79, Springer Lec. Notes Comp. Sci.*, 72, pp. 216-226 (1979).

69. Zippel, R. E., "Newton's iteration and the sparse Hensel algorithm," *Proc. '81 ACM Symp. Symbolic Algebraic Comp.*, pp. 68-72 (1981).