DOI 10.1007/s10469-020-09565-0
Algebra and Logic, Vol. 58, No. 6, January, 2020
(Russian Original Vol. 58, No. 6, November-December, 2019)

FIELDS OF ALGEBRAIC NUMBERS COMPUTABLE IN POLYNOMIAL TIME. I

P. E. Alaev[1*] **and V. L. Selivanov**[2**]

UDC 510.52+512.62+510.67

Keywords: *field of complex algebraic numbers, ordered field of real algebraic numbers, polynomially computable presentation.*

It is proved that the field of complex algebraic numbers has an isomorphic presentation computable in polynomial time. A similar fact is proved for the ordered field of real algebraic numbers. The constructed polynomially computable presentations are based on a natural presentation of algebraic numbers by rational polynomials. Also new algorithms for computing values of polynomials on algebraic numbers and for solving equations in one variable with algebraic coefficients are presented.

INTRODUCTION

The paper is devoted to investigation of algorithmic properties of the following two fields: the field of complex algebraic numbers $\mathfrak{A}^{\mathrm{C}} = (\mathbb{C}_{\mathrm{alg}}, +, \times)$ and the ordered field of real algebraic numbers $\mathfrak{A}^{\mathrm{R}} = (\mathbb{R}_{\mathrm{alg}}, \leqslant, +, \times)$. A number $\alpha \in \mathbb{C}$ is said to be *algebraic over* $\mathbb{Q}$ if $p(\alpha) = 0$ for some nonzero polynomial $p(x) \in \mathbb{Q}[x]$. If $p(x)$ is a polynomial of least degree with this property, then it is called a *minimal polynomial for* α, and its degree is the *degree of* α. A minimal polynomial is

[*]Supported by RFBR (project No. 17-01-00247) and by the RF Ministry of Science and Higher Education (state assignment to Sobolev Institute of Mathematics, SB RAS, project No. 0314-2019-0002).

[**]The work was carried out at the expense of the subsidy allocated to Kazan (Volga Region) Federal University for the fulfillment of the state assignment in the sphere of scientific activity, project No. 1.13556.2019/13.1.

[1]Sobolev Institute of Mathematics, pr. Akad. Koptyuga 4, Novosibirsk, 630090 Russia. Novosibirsk State University, ul. Pirogova 1, Novosibirsk, 630090 Russia; alaev@math.nsc.ru. [2]Ershov Institute of Informatics Systems, pr. Akad. Lavrent'eva 6, Novosibirsk, 630090 Russia. Kazan (Volga Region) Federal University, ul. Kremlevskaya 18, Kazan, 420008 Russia; vseliv@iis.nsk.su. Translated from *Algebra i Logika*, Vol. 58, No. 6, pp. 673-705, November-December, 2019. Original article submitted July 15, 2018; accepted February 12, 2020.

unique up to multiplication by a constant in $\mathbb{Q}$. Let $\mathbb{C}_{\text{alg}} = \{\alpha \in \mathbb{C} \mid \alpha \text{ is algebraic over } \mathbb{Q}\}$ and $\mathbb{R}_{\text{alg}} = \mathbb{C}_{\text{alg}} \cap \mathbb{R}$. Then $\mathbb{C}_{\text{alg}}$ is a subfield of $(\mathbb{C}, +, \times)$.

If Σ is a finite alphabet then Σ^* denotes the set of all words in the alphabet Σ. A structure $\mathfrak{C}$ of a finite language is *computable in polynomial time* (p-*computable*) if there exists a finite alphabet Σ such that its universe C is a subset of Σ^*, and C and all its operations and relations are computable in polynomial time. Some literature on this subject can be found [1]. We say that a structure $\mathfrak{C}'$ *has a* p-*computable presentation* if there exists a p-computable structure $\mathfrak{C} \cong \mathfrak{C}'$. It is well known that the field of rational numbers $\mathbb{Q}$ has such a presentation. Using known algorithms, it is not hard to show that every finite algebraic extension $\mathbb{Q}(\alpha_1, \dots, \alpha_k)$ also has a p-computable presentation. One of the results of the paper is the following:

THEOREM A. The fields $\mathfrak{A}^{\text{C}}$ and $\mathfrak{A}^{\text{R}}$ have p-computable presentations, where the operations $-x$ and $1/x$ are p-computable.

The theorem cannot be thought of as being truly new. To construct a p-computable structure $\mathfrak{A}_1^{\text{R}} \cong \mathfrak{A}^{\text{R}}$, we code a number $\alpha \in \mathbb{R}_{\text{alg}}$ by a word defining a pair $(p(x), k)$, where $p(x) \in \mathbb{Z}[x]$ is a minimal polynomial such that $p(\alpha) = 0$, and $k \geqslant 1$ is the number of α in the ordered list of all real roots $\alpha_1 < \alpha_2 < \ldots < \alpha_k$ of $p(x)$. This is a quite known presentation which can be found in literature. If α and β are specified by pairs $(p_1(x), k_1)$ and $(p_2(x), k_2)$, then we may define in polynomial time whether $\alpha < \beta$ by applying a standard root isolation algorithm in [2], which is based on the computation of a Sturm sequence and on a known estimation for the minimum distance between real roots of a polynomial $p(x) \in \mathbb{Z}[x]$.

In order to define p-computable operations on pairs, we can use formulas in [3], which are based on the computation of resultants: for example, if $p_1(\alpha) = 0$ and $p_2(\beta) = 0$, then $\alpha + \beta$ is a root of the polynomial $\text{res}_y(p_1(x-y), p_2(y))$. In [3], a pair $(p(x), I)$ is used to code an algebraic number α, where $p(\alpha) = 0$ and $I = (a, b]$ is an interval in $\mathbb{R}$ with rational endpoints such that α is the unique root of $p(x)$ in I. Such an interval is said to be *isolating for* α. Applying a root isolation algorithm, it is not hard to pass from the presentation $(p(x), I)$ to $(p(x), k)$ (or vice versa) in polynomial time, where $k \geqslant 1$ is the number of a root.

If we have an arbitrary polynomial $p(x) \in \mathbb{Z}[x]$ such that $p(\alpha) = 0$, then we can pass from it to a minimal polynomial by applying a polynomial-time algorithm for decomposing a polynomial into irreducible factors [4].

If a p-computable presentation $\mathfrak{A}_1^{\text{R}}$ is constructed, then we may construct a p-computable presentation $\mathfrak{A}_1^{\text{C}} \cong \mathfrak{A}^{\text{C}}$ by coding a complex algebraic number via its real and imaginary parts, which are in $\mathbb{R}_{\text{alg}}$. If necessary, therefore, the proof of Theorem A can be reduced to a combination of a series of known polynomial algorithms and some additional constructions. The proof in the present paper proceeds along essentially different lines.

We recall that a structure $\mathfrak{C}$ of a finite language with universe $C \subseteq \Sigma^*$ is said to be *computable* if C itself and all operations and relations are computable via some algorithms, without any explicit restrictions on the working time. In [5], it was proved that if a field $\mathfrak{F}$ is computable then its algebraic closure also has a computable presentation. Using a computable presentation for $\mathbb{Q}$, we obtain a computable presentation for $\mathfrak{A}^{\mathrm{C}}$. In [6], a similar fact was proved: if an ordered field is computable then its real closure also has a computable presentation. From this, we obtain a computable presentation for $\mathfrak{A}^{\mathrm{R}}$. Theorem A is a natural strengthened version of the facts mentioned. Their general proof can be found in [7].

Let $\mathfrak{A}_1^{\mathrm{R}}$ and $\mathfrak{A}_1^{\mathrm{C}}$ be the above-specified p-computable presentations over an alphabet Σ. A more complicated part of the paper is related to some important algorithms in these structures. For example, we can ask how quickly we can compute the function of iterated addition, i.e., the function taking a word $\alpha_1 * \alpha_2 * \ldots * \alpha_k$ to a word $\alpha_1 + \ldots + \alpha_k$, where $k \geqslant 1$, $\alpha_i \in \mathfrak{A}_1^{\mathrm{R}}$, and $* \notin \Sigma$. Generally, if a field $\mathfrak{F}$ is p-computable then the function has exponential complexity. A more general question asks the following: What is the complexity of the function taking $t(x_1, \ldots, x_k)$ and $\alpha_1 * \alpha_2 * \ldots * \alpha_k$ to $t(\alpha_1, \ldots, \alpha_k)$, where $k \geqslant 1$ and $t(\bar{x}) \in \mathbb{Q}[x_1, \ldots, x_k]$?

We note that Theorem A asserts more than is stated: namely, $\mathbb{R}_{\mathrm{alg}}$ and $\mathbb{C}_{\mathrm{alg}}$ have p-computable presentations based on a quite standard and natural presentation of algebraic numbers, which is closely related to computer algebra. Every question about the complexity of algorithms in $\mathfrak{A}_1^{\mathrm{R}}$ and $\mathfrak{A}_1^{\mathrm{C}}$ is connected with corresponding questions for the algebra of polynomials over $\mathbb{Q}$ and for the computer arithmetic of algebraic numbers, which is of considerable practical importance.

Some algorithms for computing the function $t(x_1, \ldots, x_k), \alpha_1 * \alpha_2 * \ldots * \alpha_k \mapsto t(\alpha_1, \ldots, \alpha_k)$ are well known from literature. If $k = 2$ then we can apply formulas that use resultants similar to those above, and in the general case, obtain a formula with $k-1$ successively applied resultants. To compute the resultants, we have sufficiently good algorithms [8]. Nevertheless, direct applications of such formulas produce algorithms with very bad estimations.

At the same time, the problem can be easily solved for the case $k = 1$, since the arithmetic in a simple algebraic extension $\mathbb{Q}(\alpha_1)$ works very simply. In the general case, the standard algorithm is to reduce the problem to the case $k = 1$. If $\alpha_1, \ldots, \alpha_k \in \mathbb{C}_{\mathrm{alg}}$ then there exists $\theta \in \mathbb{C}_{\mathrm{alg}}$ for which $\mathbb{Q}(\alpha_1, \ldots, \alpha_k) = \mathbb{Q}(\theta)$. Such θ is called a *primitive element*. Moreover, there is a polynomial algorithm that, given two numbers $\alpha_1, \alpha_2 \in \mathbb{C}_{\mathrm{alg}}$, finds $\theta \in \mathbb{C}_{\mathrm{alg}}$ and two polynomials $c_1(x), c_2(x) \in \mathbb{Q}[x]$ such that $\mathbb{Q}(\alpha_1, \alpha_2) = \mathbb{Q}(\theta)$ and $c_i(\theta) = \alpha_i$ for $i = 1, 2$. The algorithm was constructed in [3] and can be found, for example, in [9, Sec. 5.4]. For many specialists in this area, the existence of such an algorithm seems to be a sufficient reason for reducing the whole arithmetic of algebraic numbers to the case of $\mathbb{Q}(\theta)$ (see [10, Sec. 4.2; 11]). The method of searching a primitive element was used in [12] for constructing an algorithm of quantifier elimination in $\mathrm{Th}(\mathfrak{A}^{\mathrm{R}})$. Iterating the algorithm, we can, given a tuple $\alpha_1, \ldots, \alpha_k$, find a primitive element θ and polynomials $c_i(x)$, $i \leqslant k$, such that $\alpha_i = c_i(\theta)$. For fixed k, the method provides a polynomial algorithm for computing $t(\alpha_1, \ldots, \alpha_k)$. However, how quickly its complexity grows with growing k seems to be a nontrivial question.

In [3], some estimations were given for computing θ and $c_i(x)$, $i = 1, 2$, for $k = 2$. However, it has no estimations for the general case, and finding them seems to be an unwieldy computational problem. Apparently, one of the stages of a general algorithm—computing $c_i(x)$ for $i \leqslant k$—requires at least $n^{ck\log(k)}L^d$ steps, where n is the maximal degree of α_i, $i \leqslant k$, L is the total input length of the algorithm, and c and d are fixed constants. Possibly, final estimation would have a still worse form.

In the present paper, we propose another algorithm (Thm. 4) for computing $t(\alpha_1, \dots, \alpha_k)$. Our algorithm does not use resultants and primitive elements and has an estimation for the number of steps of the form $n^{ck}L^d$. Moreover, the estimation can be written in the form $(n_1 n_2 \dots n_k)^c L^d$, where n_i is the degree of α_i for $i \leqslant k$. It looks much better if n_i are essentially different. Also there are examples showing that this estimation cannot be improved, up to polynomial equivalence.

The second problem considered in the paper is finding roots of an equation with algebraic coefficients, i.e., equations of the form $\alpha_e x^e + \dots + \alpha_1 x + \alpha_0 = 0$, where $\alpha_i \in \mathbb{C}_{\text{alg}}$ for $i \leqslant e$. To improve estimations, it is convenient to rewrite an equation in the form

$$t_e(\alpha_1, \dots, \alpha_k)x^e + \dots + t_1(\alpha_1, \dots, \alpha_k)x + t_0(\alpha_1, \dots, \alpha_k) = 0,$$

where $\alpha_1, \dots, \alpha_k \in \mathbb{C}_{\text{alg}}$ and $t_j(\bar{x}) \in \mathbb{Q}[x_1, \dots, x_k]$. The problem is as follows. Given $\alpha_1 * \dots * \alpha_k$ in the constructed presentation $\mathfrak{A}_1^{\mathrm{C}}$ and a list of polynomials $\{t_j(\bar{x})\}_{j \leqslant e}$, we need to find a list of all roots of the equation in $\mathfrak{A}_1^{\mathrm{C}}$. Again we have well-known polynomial algorithms for solving the problem for $k = 1$, and in the general case, we can pass from $\alpha_1, \dots, \alpha_k$ to a primitive element (see [3]). In searching for such as element, here we meet the same difficulties as those described above.

In the paper, we come up with a different algorithm for solving the problem mentioned. Again our algorithm does not use resultants or primitive elements and has an estimation for working time of the form $(n_1 \dots n_k)^c L^d$ (Thm. 8).

In [13, 14], it was proved that for every infinite p-computable structure $\mathfrak{A}$, there exists a p-computable structure $\mathfrak{A}' \cong \mathfrak{A}$ such that $\mathfrak{A}$ and $\mathfrak{A}'$ are not p-computably isomorphic. Therefore, the fields $\mathfrak{A}^{\mathrm{R}}$ and $\mathfrak{A}^{\mathrm{C}}$ have other, essentially different p-computable presentations. In a subsequent paper, we intend to discuss other presentations of $\mathfrak{A}^{\mathrm{R}}$ and $\mathfrak{A}^{\mathrm{C}}$ known in literature, and general questions concerning the uniqueness of such presentations.

1. BASIC ARITHMETIC ALGORITHMS

As our basic model of computation we use multi-tape Turing machines (see, e.g., [15, Sec. 1.6] for additional details). Let Σ be a finite alphabet and $f : A \to \Sigma^*$, where $A \subseteq (\Sigma^*)^n$. We say that f is *computable on a k-tape Turing machine T in $t(\bar{x})$ steps*, where $t : A \to \mathbb{N}$, if $k \geqslant n + 1$ and we can write words $\bar{x} = x_1, \dots, x_n$ in A on the first n tapes and run T, which stops in at most $t(\bar{x})$ steps with $f(\bar{x})$ written on the $(n+1)$th tape. (A more detailed definition can be found in [14]).

If $t(\bar{x}) = c|\bar{x}|^m$ for $|\bar{x}| \neq 0$, where $c, m \in \mathbb{N}$ and $|\bar{x}| = \max\limits_{i \leqslant n}\{|x_i|\}$, then $f(\bar{x})$ is said to be computable on T *in polynomial time* (p-*computable*) and so on.

For a natural number n, the word $\mathrm{b}(n) = a_k a_{k-1} \dots a_1 a_0$ in the alphabet $\{0, 1\}$ is the *binary representation* of n, where $n = a_k 2^k + a_{k-1} 2^{k-1} + \dots + a_1 2 + a_0$, $a_i \in \{0, 1\}$, and $a_k \neq 0$ for $k \neq 0$.

In this case $\mathrm{L}(n) = |\mathrm{b}(n)|$ is called the *length of* n. If $n \neq 0$ then $\mathrm{L}(n) = [\log_2(n)] + 1$, where $[\beta]$ is the integer part of $\beta \in \mathbb{R}$.

If $a = -n$, where $n \geqslant 1$, then we define $\mathrm{b}(a) = 0\mathrm{b}(n)$, and again $\mathrm{L}(a) = |\mathrm{b}(a)|$. If α is a rational number of the form $\frac{a}{b}$, where $a, b \in \mathbb{Z}$ are relatively prime and $b > 0$, then we put $\mathrm{b}(\alpha) = \mathrm{b}(a) * \mathrm{b}(b)$, which is a word in the alphabet $\{0, 1, *\}$, and $\mathrm{L}(\alpha) = |\mathrm{b}(\alpha)|$. This definition somewhat conflicts with the previous one because an integer a can be written both in the form $\mathrm{b}(a)$ and in the form $\mathrm{b}(a) * 1$. Whether we deal with integers or rational numbers will usually be clear from the context. We point out some well-known estimates for the arithmetic in $\mathbb{Z}$.

LEMMA 1. Let $x, x_1, \ldots, x_k \in \mathbb{Z}$ and $k \geqslant 1$. Then:

(a) $\mathrm{L}(-x) \leqslant \mathrm{L}(x) + 1$;

(b) $\mathrm{L}(x_1 + x_2) \leqslant \max\{\mathrm{L}(x_1), \mathrm{L}(x_2)\} + 1$;

(c) $\mathrm{L}(x_1 + x_2 + \ldots + x_k) \leqslant \max\limits_{i \leqslant k}\{\mathrm{L}(x_i)\} + \mathrm{L}(k)$;

(d) $\mathrm{L}(x_1 \cdot x_2 \cdot \ldots \cdot x_k) \leqslant \mathrm{L}(x_1) + \mathrm{L}(x_2) + \ldots + \mathrm{L}(x_k)$.

In the formulation of the next lemma, we use the symbol c which will also appear in what follows. Often the working time of an algorithm is of the form $cf(L)$, where L is the length of input data and c is a fixed constant. To simplify formulations, we use the same symbol c (even if its value is different in different situations) unless this leads to confusion. Usually, our constants are not big, not exceeding 100. Whenever we deal with an algorithm on integers, we tacitly mean that the integers are specified by their binary representations. For instance, in Lemma 2(a) below we speak about an algorithm that takes the word $\mathrm{b}(x)$ as input and returns the word $\mathrm{b}(-x)$. A number $x \geqslant 0$ can sometimes be given in the *unary form* 1^x, where a^x stands for a word of length x consisting only of symbols a. By $\gcd[x, y]$ we denote the greatest common divisor of x and y.

LEMMA 2. Let $x, x_1, \ldots, x_k \in \mathbb{Z}$, $k \geqslant 1$, and $M = \max\limits_{i \leqslant k}\{\mathrm{L}(x_i)\}$. Then, for some constant $c \geqslant 1$, the following hold:

(a) $-x$ is computed from x in time $c\mathrm{L}(x)$;

(b) $x_1 + x_2$ is computed from x_1, x_2 in time $c \max\{\mathrm{L}(x_1), \mathrm{L}(x_2)\}$;

(c) $x_1 + x_2 + \ldots + x_k$ is computed from the word $\mathrm{b}(x_1) * \ldots * \mathrm{b}(x_k)$ in time $ck(M + \mathrm{L}(k))$;

(d) $x_1 \cdot x_2$ is computed from x_1, x_2 in time $c\mathrm{L}(x_1) \cdot \mathrm{L}(x_2)$;

(e) $x_1 \cdot x_2 \cdot \ldots \cdot x_k$ is computed from the word $\mathrm{b}(x_1) * \ldots * \mathrm{b}(x_k)$ in time ck^2M^2.

LEMMA 3. Let $x \in \mathbb{N}$ and $x_1, x_2 \in \mathbb{Z}$. Then, for some constant $c \geqslant 1$, the following hold:

(a) the transfer from $\mathrm{b}(x)$ to 1^x and back can be done in time $c(x + 1)$;

(b) given x_1, x_2, where $x_2 \neq 0$, we can compute numbers $q, r \in \mathbb{Z}$ such that $x_1 = qx_2 + r$, where $0 \leqslant r < |x_2|$, in time $c\mathrm{L}(x_1) \cdot \mathrm{L}(x_2)$;

(c) given x_1, x_2, where $x_1 \neq 0$ or $x_2 \neq 0$, we can compute $\gcd[x_1, x_2]$ in time $c \max\{\mathrm{L}(x_1), \mathrm{L}(x_2)\}^3$.

Similar estimates can also be given for arithmetical operations in $\mathbb{Q}$. Since these yield an essentially worse result, all computations are, as a rule, done only with integers. We only observe that the functions sending the word $\mathrm{b}(a_1) * \mathrm{b}(a_2) * \ldots * \mathrm{b}(a_k)$ to $\mathrm{b}(a_1 + \ldots + a_k)$ and $\mathrm{b}(a_1 \cdot \ldots \cdot a_k)$, where $a_i \in \mathbb{Q}$ for $i \leqslant k$, are p-computable.

If $p(x) \in \mathbb{Q}[x]$ and the variable x is fixed, then $\mathrm{b}(p(x)) = \mathrm{b}(a_n) * \ldots * \mathrm{b}(a_1) * \mathrm{b}(a_0)$, where $p(x) = a_n x^n + \ldots + a_1 x + a_0$ and $a_n \neq 0$ for $n \neq 0$. As usual, $\mathrm{L}(p(x)) = |\mathrm{b}(p(x))|$. When speaking about algorithms on $\mathbb{Q}[x]$, we tacitly identify $p(x)$ with $\mathrm{b}(p(x))$. Note that the degree of a polynomial satisfies the relation $\deg[p(x)] \leqslant \mathrm{L}(p(x))$. In this encoding, the function sending $p(x) \in \mathbb{Q}[x]$ and $a \in \mathbb{Q}$ to $p(a) \in \mathbb{Q}$ is p-computable.

Let $\mathbb{Z}[i] = \{a + ib \mid a, b \in \mathbb{Z}\}$ and $\mathbb{Q}[i] = \{a + ib \mid a, b \in \mathbb{Q}\}$. Defining for these sets natural binary encodings as $\mathrm{b}(a + ib) = \mathrm{b}(a) * \mathrm{b}(b)$ and their length as $\mathrm{L}(a + ib) = |\mathrm{b}(a + ib)|$, we may speak about algorithms on $\mathbb{Z}[i]$ and $\mathbb{Q}[i]$.

LEMMA 4. Let $z, z_1, \ldots, z_k \in \mathbb{Z}[i]$, $k \geqslant 1$, and $M = \max\limits_{i \leqslant k}\{\mathrm{L}(z_i)\}$. Then the following hold:

(a) $\mathrm{L}(-z) \leqslant \mathrm{L}(z) + 2$;

(b) $\mathrm{L}(z_1 + \ldots + z_k) \leqslant 2(M + \mathrm{L}(k))$;

(c) $\mathrm{L}(z_1 \cdot \ldots \cdot z_k) \leqslant 2kM$.

Proof. (c) Let $z_j = x_j + iy_j$, where $x_j, y_j \in \mathbb{Z}$. Then $\mathrm{L}(x_j), \mathrm{L}(y_j) \leqslant M - 2$ for $j \leqslant k$. Removing the parenthesis in $(x_1 + iy_1)(x_2 + iy_2) \ldots (x_k + iy_k)$, we see that $z = z_1 \ldots z_k = x + iy$, where x and y are sums of 2^{k-1} monomials, and each monomial a has the form $\pm x_{j_1} \ldots x_{j_t} y_{e_1} \ldots y_{e_s}$, where $t + s = k$. Then $\mathrm{L}(a) \leqslant k(M - 2) + 1$ and $\mathrm{L}(x), \mathrm{L}(y) \leqslant k(M - 2) + 1 + \mathrm{L}(2^{k-1})$, where $\mathrm{L}(2^{k-1}) = k$. Therefore, $\mathrm{L}(z) \leqslant \mathrm{L}(x) + \mathrm{L}(y) + 1 \leqslant 2kM$. □

Since the length of the product $z_1 \cdot \ldots \cdot z_k$ grows linearly in k, it is easy to see that the functions sending the word $\mathrm{b}(z_1) * \ldots * \mathrm{b}(z_k)$ to $\mathrm{b}(z_1 + \ldots + z_k)$ and $\mathrm{b}(z_1 \cdot \ldots \cdot z_k)$, where $z_i \in \mathbb{Z}[i]$ for $i \leqslant k$, are p-computable. The same is true for the case $z_i \in \mathbb{Q}[i]$.

2. ENCODING ALGEBRAIC NUMBERS

For a commutative ring R with 1 and without divisors of 0, by $R[x]$ we denote the set of all polynomials in a variable x with coefficients in R. We will mainly work with polynomials in $\mathbb{Z}[x]$ and $\mathbb{Q}[x]$. Now we briefly recall some properties of polynomials. If $p(x) = a_n x^n + \ldots + a_1 x + a_0$, where $a_i \in \mathbb{Q}$ and $a_n \neq 0$ for $n \neq 0$, then n is called the *degree of* $p(x)$ and is denoted by $\deg[p(x)]$. The number a_n is the *leading coefficient* of $p(x)$. If $a_n = 1$ or ($n = 0$ and $a_n = 0$), then the polynomial is said to be *normalized.* If $p_1(x), p_2(x) \in \mathbb{Q}[x]$ and $p_2(x) \neq 0$ then there exist unique polynomials $q(x), r(x) \in \mathbb{Q}[x]$ such that $p_1(x) = q(x)p_2(x) + r(x)$, where $\deg[r(x)] < \deg[p_2(x)]$ or $r(x) = 0$.

We say that $p_2(x)$ *divides* $p_1(x)$ in $\mathbb{Q}[x]$, $p_2(x) \,|\, p_1(x)$, if there is $q(x) \in \mathbb{Q}[x]$ with $q(x)p_2(x) = p_1(x)$. A polynomial $p_h(x)$ is a *greatest common divisor* of $p_1(x)$ and $p_2(x)$ if $p_h(x) \,|\, p_i(x)$ for $i = 1, 2$ and the fact that $q(x) \,|\, p_i(x)$ for $i = 1, 2$ implies $q(x) \,|\, p_h(x)$. A greatest common divisor always exists and is unique up to multiplication by a nonzero constant in $\mathbb{Q}$. The normalized greatest common divisor is denoted by $\gcd[p_1(x), p_2(x)]$.

A polynomial $p(x) \in \mathbb{Q}[x]$ is *irreducible* in $\mathbb{Q}[x]$ if $\deg[p(x)] \geqslant 1$ and the equality $p(x) = p_1(x)p_2(x)$ implies that $\deg[p_1(x)] = 0$ or $\deg[p_2(x)] = 0$. If $p(x)$ is an irreducible polynomial and $p(x) \,|\, p_1(x)p_2(x)$, then $p(x) \,|\, p_1(x)$ or $p(x) \,|\, p_2(x)$. Every nonzero polynomial $p(x) \in \mathbb{Q}[x]$ is representable as

$$p(x) = ap_1(x)p_2(x)\dots p_k(x),$$

where $a \in \mathbb{Q}$ and $p_i(x)$ are normalized irreducible polynomials, and such a representation is unique up to factor permutation.

The *derivative* $p'(x)$ of a polynomial $p(x) = a_nx^n + a_{n-1}x^{n-1} + \ldots + a_2x^2 + a_1x + a_0$ is defined as $na_nx^{n-1} + (n-1)a_{n-1}x^{n-2} + \ldots + 2a_2x + a_1$. If

$$p(x) = a[p_1(x)]^{e_1}[p_2(x)]^{e_2}\dots[p_k(x)]^{e_k},$$

where $a \in \mathbb{Q}$, $p_i(x)$ are normalized irreducible polynomials, $e_i \geqslant 1$, and $p_i(x) \neq p_j(x)$ for $i \neq j$, then

$$\gcd[p(x), p'(x)] = [p_1(x)]^{e_1-1}[p_2(x)]^{e_2-1}\dots[p_k(x)]^{e_k-1}.$$

In particular, $p(x)/\gcd[p(x), p'(x)] = ap_1(x)\dots p_k(x)$.

A polynomial $p(x)$ is *square free* if, in the factorization above, $e_i = 1$ for $i \leqslant k$. This is equivalent to $\gcd[p(x), p'(x)] = 1$. In every field that extends $\mathbb{Q}$, the roots of $p(x)$ and $p(x)/\gcd[p(x), p'(x)]$ coincide.

If $p(x) \in \mathbb{Z}[x]$ and $\alpha_1 < \alpha_2 < \ldots < \alpha_n$ are all its distinct real roots then the *separator* of $p(x)$ is defined as $\Delta_p = \min\limits_{1 \leqslant i < j \leqslant n} |\alpha_i - \alpha_j|$. For $n \leqslant 1$ or $p(x) = 0$, we put $\Delta_p = \infty$. The next result gives a lower bound for the separator.

THEOREM 1 (Mahler, see [16, Sec. 7.2.12]). Let $p(x) \in \mathbb{Z}[x]$ be a polynomial of degree $n \geqslant 2$ and $\gcd[p(x), p'(x)] = 1$. Then

$$\Delta_p \geqslant \frac{\sqrt{3}}{n^{\frac{n+2}{2}}|p(x)|_1^{n-1}}.$$

Here $|p(x)|_1 = |a_n| + |a_{n-1}| + \ldots + |a_0|$ if $p(x) = a_nx^n + a_{n-1}x^{n-1} + \ldots + a_0$. The theorem implies

PROPOSITION 1. There is a p-computable function that, given a polynomial $p(x) \in \mathbb{Q}[x]$, finds a number $\delta_p \in \mathbb{Q}$ such that $0 < \delta_p \leqslant \Delta_p$.

Proof. If $\deg[p(x)] \leqslant 1$ then $\Delta_p = \infty$, and we can set $\delta_p = 1$. Let $\deg[p(x)] = n \geqslant 2$. A brief proof sketch is as follows. First we compute $p'(x)$, find $\gcd[p(x), p'(x)]$, and then replace $p(x)$ by the square-free polynomial $p(x)/\gcd[p(x), p'(x)]$. A complication is associated with the fact that the standard Euclidean algorithm of finding $\gcd[p(x), p'(x)]$ is not polynomial.

A modified version of the algorithm mentioned is the method of subresultant residues based on works of Silvester (1853) and Gabicht (1948) (see [16, Sec. 5.2.3]). It constructs in polynomial time a sequence of polynomials which differs from the usual sequence in Euclid's algorithm by some additional factors, and thus finds $\gcd[p(x), p'(x)]$.

Let $p(x)$ be square free. If we multiply it by the product of denominators of all coefficients we obtain $p(x) \in \mathbb{Z}[x]$. Find then $k = |p(x)|_1$ and use the formula given in Theorem 1, replacing $\sqrt{3}$ by 1 and $n^{\frac{n+2}{2}}$ by n^{n+2}. The computation of k^{n-1} requires $c(n-1)^2\mathrm{L}(k)^2$ steps, and since $\mathrm{L}(p(x)) \geqslant n$, this time is polynomial; the same holds for n^{n+2}. $\square$

The next theorem is based on an algorithm for approximating real roots of a polynomial, which uses Sturm's method. Being a minor modification of a corresponding algorithm in [2], it can be thought of as being well known.

THEOREM 2. There exists a p-computable function that, given a nonzero polynomial $p(x) \in \mathbb{Q}[x]$, $k \in \omega$, and $\varepsilon \in \mathbb{Q}$, $\varepsilon > 0$, solves the following two problems:

(a) determines whether $p(x)$ has at least k distinct real roots;

(b) if it has and $\alpha_1 < \alpha_2 < \ldots < \alpha_m$ are all real roots of $p(x)$ then it finds $a, b \in \mathbb{Q}$ such that $\alpha_k \in (a,b)$ and $|a-b| \leqslant \varepsilon$.

Proof. We give a brief proof sketch. Arguing as in Proposition 1, we may assume that $p(x)$ is square free. Find a Sturm sequence $\operatorname{sseq}(x) = (p(x), p'(x), p_3(x), \ldots, p_h(x))$. It is constructed in essentially the same way as the sequence in Euclid's algorithm and can be computed in polynomial time.

If $a_1, a_2, \ldots, a_n$ is a sequence in $\mathbb{R}$, then the *sign alternation number* in this sequence is the number of pairs (i,j) such that $1 \leqslant i < j \leqslant n$, $a_i \cdot a_j < 0$, and $a_t = 0$ for $i < t < j$. Sturm's theorem says that the number of roots of $p(x)$ in the interval $(a,b]$ equals $v(a) - v(b)$ if $v(a)$ is the sign alternation number in the sequence $\operatorname{sseq}(a)$ for $a \in \mathbb{R}$. Since the sequence of polynomials is found in polynomial time, its components have bounded length, and the function that computes $v(a)$ from $a \in \mathbb{Q}$ and $p(x)$ is p-computable. Because $v(a) \leqslant h \leqslant \deg[p(x)]$, it does not matter whether we speak about $\mathrm{b}(v(a))$ or about $1^{v(a)}$.

From the coefficients of $p(x)$, we can find a number $M \in \mathbb{Q}$ such that all roots of $p(x)$ are in the interval $(-M, M)$. If $v(-M) - v(M) < k$ then α_k does not exist. Otherwise, we subsequently divide the interval $(-M, M)$ in half searching for α_k: namely, we define a sequence of intervals (a_t, b_t) such that $|a_t - b_t| = \frac{M}{2^{t-1}}$ and $\alpha_k \in (a_t, b_t)$ for all t. If (a_t, b_t) is found then we compute $c_t = \frac{a_t + b_t}{2}$, check whether c_t is a root of $p(x)$, and find $v(c_t)$. With $v(a_t)$ and $v(b_t)$ computed, it is not hard to determine which of the intervals (a_t, c_t) and (c_t, b_t) contains α_k.

If in the search process we find a precise value $\alpha_k \in \mathbb{Q}$, then $(\alpha_k - \varepsilon/2, \alpha_k + \varepsilon/2)$ is the desired interval. Otherwise, the process continues until $|a_t - b_t| \leqslant \varepsilon$. Therefore, the last step t_0 is determined by the formula $\frac{M}{2^{t_0-1}} \leqslant \varepsilon$, which is equivalent to $\log_2(\frac{M}{\varepsilon}) + 1 \leqslant t_0$. Put $t_0 = [\log_2(\frac{M}{\varepsilon})] + 2$.

By construction, $a_t = \frac{Ma'_t}{2^t}$ and $b_t = \frac{Mb'_t}{2^t}$, where $-2^t \leqslant a'_t < b'_t \leqslant 2^t$. In particular, $a'_0 = -1$, $b'_0 = 1$, and $a'_{t+1}, b'_{t+1} \in \{2a'_t, 2b'_t, a'_t + b'_t\}$. We obtain $\mathrm{L}(a'_t), \mathrm{L}(b'_t) \leqslant c(\mathrm{L}(M) + t_0)$ and $t_0 \leqslant c(\mathrm{L}(M) + \mathrm{L}(\varepsilon))$, where $c \in \omega$. The total estimate for the number of steps is polynomial. $\square$

Let $p(x) \in \mathbb{Q}[x]$ and $k \geqslant 1$. We say that a pair $(p(x), k)$ *encodes a number* $\alpha \in \mathbb{R}$ if $p(x) \neq 0$, $p(x)$ has at least k real roots, and $\alpha = \alpha_k$, where $\alpha_1 < \alpha_2 < \ldots < \alpha_n$ are all real roots of $p(x)$. We say that pairs $(p(x), k)$ and $(q(x), m)$ are *equivalent* ($(p(x), k) \sim (q(x), m)$) if they encode the same number. By the standard binary representation of a pair $(p(x), k)$ we mean the word $\mathrm{b}(p(x)) * \mathrm{b}(k)$.

COROLLARY 1. A set of pairs $(p(x), k)$ which encode some number and an equivalence relation on such pairs are p-computable.

Proof. Verification that $(p(x), k)$ encodes some number is straightforward from Theorem 2. Let $(p(x), k)$ encode a number $\alpha \in \mathbb{R}$ and $(q(x), m)$ encode a number β. Then α and β are roots of the polynomial $r(x) = p(x)q(x)$. Applying Proposition 1, we find $\delta_r \in \mathbb{Q}$ such that the distance between distinct roots of $r(x)$ is at least δ_r. We set $\varepsilon = \delta_r/3$, and using Theorem 2, find intervals $I_\alpha = (a_1, b_1)$ and $I_\beta = (a_2, b_2)$ such that $\alpha \in I_\alpha$, $\beta \in I_\beta$, and $|a_i - b_i| \leqslant \varepsilon$ for $i = 1, 2$. If $\alpha = \beta$ then $I_\alpha \cap I_\beta \neq \varnothing$. If $\alpha \neq \beta$ and $I_\alpha \cap I_\beta \neq \varnothing$, then the distance between two distinct roots of $r(x)$ is at most 2ε, which is impossible. Therefore, $\alpha = \beta \Leftrightarrow I_\alpha \cap I_\beta \neq \varnothing$, which is a p-computable condition. $\square$

COROLLARY 2. Let a pair $(p(x), k)$ encode $\alpha \in \mathbb{R}$ and $q(x) \in \mathbb{Q}[x]$. Then there is a p-computable function that, given $p(x)$, k, and $q(x)$, determines whether α is a root of $q(x)$, and if yes, then it finds $m \leqslant \deg[q(x)]$ such that $(p(x), k) \sim (q(x), m)$.

Proof. We simply go through all $m \leqslant \deg[q(x)]$, and for each such m, verify the condition $(p(x), k) \sim (q(x), m)$ by using Corollary 1. $\square$

COROLLARY 3. Let pairs $(p(x), k)$ and $(q(x), m)$ encode numbers α and β, respectively. Given these pairs, we can determine in polynomial time whether $\alpha < \beta$, and if yes, find $\varepsilon \in \mathbb{Q}$ such that $|\beta - \alpha| \geqslant \varepsilon > 0$.

Proof. We construct $r(x) = p(x)q(x)$, and using Corollary 2, find $k_1, m_1 \in \omega$ for which $(p(x), k) \sim (r(x), k_1)$ and $(q(x), m) \sim (r(x), m_1)$. Then $\alpha < \beta \Leftrightarrow k_1 < m_1$ and $|\alpha - \beta| \geqslant \delta_r$, where δ_r is the number found by applying Prop. 1. $\square$

Whenever we specify an algebraic number by a minimal polynomial, the number is determined up to a constant. To provide uniqueness, we should normalize it in some way. In order to stay within $\mathbb{Z}[x]$, this can be done by using primitive polynomials. A polynomial $p(x) \in \mathbb{Z}[x]$ is *primitive* if $p(x) \neq 0$ and the greatest common divisor of its coefficients is 1. Every $p(x) \in \mathbb{Q}[x] \setminus \{0\}$ may be represented as $p(x) = ap^*(x)$, where $a \in \mathbb{Q}$ and $p^*(x)$ is a primitive polynomial. Such a representation is unique up to a sign: if $p^*(x) = bp_1^*(x)$, where $b \in \mathbb{Q}$ and $p^*(x), p_1^*(x)$ are primitive, then $b \in \{1, -1\}$.

THEOREM 3 [4, Thm. 3.6]. There exists a p-computable function that, given a polynomial $p(x) \in \mathbb{Q}[x]$, finds a tuple of polynomials $p_1(x), \ldots, p_n(x) \in \mathbb{Q}[x]$ irreducible in $\mathbb{Q}[x]$ such that $p(x) = p_1(x) \ldots p_n(x)$. In particular, we can determine in polynomial time whether $p(x)$ is irreducible in $\mathbb{Q}[x]$.

Proof. The algorithm in [4, Thm. 3.6] is described only for primitive polynomials $p(x) \in \mathbb{Z}[x]$. In the general case we should represent $p(x)$ as $ap^*(x)$, where $a \in \mathbb{Q}$ and $p^*(x)$ is primitive. $\square$

We say that $(q(x), m)$ is a *natural code* for $\alpha \in \mathbb{R}_{\mathrm{alg}}$ if $q(x)$ is a primitive irreducible polynomial in $\mathbb{Z}[x]$ with positive leading coefficient, and the pair $(q(x), m)$ encodes α.

COROLLARY 4. Given a pair $(p(x), k)$, which encodes $\alpha \in \mathbb{R}_{\mathrm{alg}}$, we can find in polynomial time a unique natural code $(q(x), m)$, which encodes the same number.

Proof. We find a decomposition $p(x) = p_1(x) \cdot \ldots \cdot p_n(x)$ into irreducible factors and then, by an exhaustive search, find a pair (i, m) such that $i \leqslant n$, $m \leqslant \deg[p_i(x)]$, and $(p(x), k) \sim (p_i(x), m)$. To pass from $p_i(x) \in \mathbb{Q}[x]$ to a primitive polynomial, we multiply $p_i(x)$ by the product of denominators of its coefficients, and then divide by their greatest common divisor. All these computations are doable in polynomial time.

The uniqueness of the natural code is well known: if $q(\alpha) = 0$ and $q(x)$ is irreducible then it is a minimal polynomial for α which is unique up to a constant in $\mathbb{Q}$. $\square$

Let $(q(x), m)$ be a natural code for α. The degree $\deg[q(x)]$ is called the *degree of a number* α and is denoted $\deg[\alpha]$. Let $\mathrm{b}(\alpha) = \mathrm{b}(q(x)) * \mathrm{b}(m)$ and $R_{\mathrm{alg}} = \{\mathrm{b}(\alpha) \mid \alpha \in \mathbb{R}_{\mathrm{alg}}\}$. We will sometimes identify α with $\mathrm{b}(\alpha)$.

It is known that a number $\gamma = \alpha + i\beta$, where $\alpha, \beta \in \mathbb{R}$, is algebraic iff both α and β are algebraic. A constructive proof of this fact is implicitly contained in Lemma 10. If $\gamma \in \mathbb{C}_{\mathrm{alg}}$ then we define $\mathrm{b}(\gamma) = \mathrm{b}(\alpha) + \mathrm{b}(\beta)$, which is a word in the alphabet $\{0, 1, *, +\}$. Let $C_{\mathrm{alg}} = \{\mathrm{b}(\gamma) \mid \gamma \in \mathbb{C}_{\mathrm{alg}}\}$. Speaking about $\mathbb{C}_{\mathrm{alg}}$, we sometimes identify $\gamma \in \mathbb{C}_{\mathrm{alg}}$ with $\mathrm{b}(\gamma)$. As in the case of reals, $\deg[\gamma]$ is the minimal degree of a polynomial $q(x) \in \mathbb{Z}[x] \setminus \{0\}$ for which $q(\gamma) = 0$. Put $\deg_{\mathrm{R}}[\gamma] = \max\{\deg[\alpha], \deg[\beta]\}$.

3. PSEUDO-DIVISION OF POLYNOMIALS

First, we need exact estimates for a pseudo-division algorithm. Let $p(x), q(x) \in \mathbb{Z}[x]$, $q(x) \neq 0$, $n = \deg[p(x)] \geqslant k = \deg[q(x)]$, and b be the leading coefficient of $q(x)$. *Pseudo-division of* $p(x)$ *by* $q(x)$ is searching for unique polynomials $p_*(x)$ and $r(x) \in \mathbb{Z}[x]$ for which

$$b^{n-k+1} p(x) = p_*(x) q(x) + r(x),$$

where $\deg[r(x)] < \deg[q(x)]$ or $r(x) = 0$. The polynomial $r(x)$ is called the *pseudo-remainder* and $p_*(x)$ is called the *pseudo-quotient*. Pseudo-division differs from the usual division of polynomials in $\mathbb{Q}[x]$ in that it is always possible not going beyond $\mathbb{Z}[x]$. If $n < k$, then $p_*(x) = 0$ and $r(x) = p(x)$.

PROPOSITION 2. The operation of pseudo-division $p(x), q(x) \mapsto p_*(x), r(x)$ in $\mathbb{Z}[x]$ is p-computable.

More precisely, let $p(x) = a_n x^n + \ldots + a_1 x + a_0$ and $q(x) = b_k x^k + \ldots + b_1 x + b_0$, where $a_i, b_j \in \mathbb{Z}$ and $\mathrm{L}(a_i), \mathrm{L}(b_j) < M$ for all i and all j. Then, for $n \neq 0$, the length of coefficients of $p_*(x)$ and $r(x)$ is at most $2n(M + \mathrm{L}(k))$, while the working time of the algorithm with $n, k \neq 0$ for some $c \geqslant 1$ can be estimated as

$$ckn^2(M + L(k))^2.$$

Proof. The division algorithm for polynomials is well known. Let $n \geqslant k \geqslant 1$. Define $b = b_k$ and

$$\begin{cases} p_0(x) = b^{n-k+1}p(x), \\ p_{s+1}(x) = p_s(x) - c_s x^{n-k-s} q(x) \end{cases}$$

for all $s \leqslant n-k$. The numbers $c_s \in \mathbb{Z}$ are chosen so that the coefficients at x^{n-s} in $p_s(x)$ and $c_s x^{n-k-s}q(x)$ will coincide. Then $\deg[p_s(x)] \leqslant n-s$ and

$$p_0(x) = \left(\sum_{s=0}^{n-k} c_s x^{n-k-s}\right) q(x) + p_{(n-k)+1}(x),$$

and we can put $r(x) = p_{(n-k)+1}(x)$ and $p_*(x) = \sum\limits_{s=0}^{n-k} c_s x^{n-k-s}$. Assuming that $c_s = 0$ for $s < 0$, it is easy to deduce an inductive formula

$$c_s = b^{n-k} a_{n-s} - \frac{1}{b} \sum_{j=0}^{k-1} (c_{s-k+j} b_j)$$

for all $s \leqslant n-k$. The formula follows directly from the obvious inductive formula for the coefficients of $p_s(x)$. By induction on $s \leqslant n-k$, it is easy to show that $b^{(n-k)-s} \mid c_s$, and hence all c_s are integers. From the formula above, we derive the inductive estimate

$$L(c_s) \leqslant (n-k+1)M + s(M + \mathrm{L}(k+1)).$$

Therefore, $\mathrm{L}(c_s) \leqslant L_0 = (2n-2k+1)M + (n-k)L(k+1)$ for all $s \leqslant n-k$. An estimate for coefficients of $r(x)$ is $L_0 + M + L(k+1)$; for $k \neq 0$, it does not exceed $2n(M + L(k))$. If $k = 0$ then $p_*(x) = b^{n-k}p(x)$.

Consequently, the same estimate holds for all numbers involved in intermediate computations for obtaining $\{c_s\}_{s \leqslant n-k}$. Clearly, most of the working time is taken by arithmetical operations; the cost of reading and writing intermediate values is obviously smaller. For instance, while inductively computing c_s, we save the sequence $\mathrm{b}(c_0) * \mathrm{b}(c_1) * \ldots * \mathrm{b}(c_s)$ on a separate tape, and every time we read from it the last k elements, which requires at most ckL_0 steps.

During the computation of one c_s, most of the time is consumed by multiplications that require $ckn(M + \mathrm{L}(k))^2$ steps; for all $s \leqslant n-k$, this takes $ckn^2(M + \mathrm{L}(k))^2$ steps. Every coefficient of $r(x)$ requires at most k multiplications of c_s and coefficients of $q(x)$, followed by k additions, which does not exceed the estimate $ckn(M + \mathrm{L}(k))^2$. For all k coefficients, this yields $ck^2n(M + \mathrm{L}(k))^2$ steps. All the above expressions do not exceed the total estimate. □

4. COMPUTING VALUES OF POLYNOMIALS

We fix a natural lexicographic order on a set ω^*. If $I \subseteq \omega^*$, $I = \{i_0 < i_1 < \ldots < i_n\}$, a_i, $i \in I$, are words in some alphabet, and b is a new symbol, then we define

$$\mathrm{C}^b_{i \in I} a_i = a_{i_0} b a_{i_1} b \ldots b a_{i_n}.$$

Fix a sequence of variables $x_1, x_2, \ldots$. Every polynomial $t(\bar{x})$ in $\mathbb{Q}[x_1, \ldots, x_n]$ is representable as

$$t(\bar{x}) = \sum_{(s_1,\ldots,s_k)\in I} q_{s_1,\ldots,s_k} x_1^{s_1} x_2^{s_2} \ldots x_k^{s_k},$$

where $I \subseteq \omega^*$, $I \neq \varnothing$, $s_k \neq 0$ for $k \neq 0$, $q_{s_1,\ldots,s_k} \in \mathbb{Q}$, and $q_{s_1,\ldots,s_k} \neq 0$ for $k \neq 0$. Define

$$\mathrm{b}(t(\bar{x})) = 1^m * \mathrm{C}^+_{(s_1,\ldots,s_k)\in I} \mathrm{b}(q_{s_1,\ldots,s_k}) * \mathrm{b}(s_1) * \ldots * \mathrm{b}(s_k),$$

where $m = \max\limits_{(s_1,\ldots,s_k)\in I}\{s_i\}$. As usual, by $\mathrm{L}(t(\bar{x}))$ we denote $|\mathrm{b}(t(\bar{x}))|$. The added word 1^m makes this encoding polynomially equivalent to the above-mentioned encoding for $\mathbb{Z}[x]$ with $k = 1$.

PROPOSITION 3. There exists an algorithm that, given $k \geqslant 1$ and nonzero polynomials $p_1(x), \ldots, p_k(x) \in \mathbb{Z}[x]$ and $t(x_1, \ldots, x_k) \in \mathbb{Q}[x_1, \ldots, x_k]$, finds a nonzero polynomial $q(x) \in \mathbb{Z}[x]$ such that if $\alpha_1, \ldots, \alpha_k \in \mathbb{C}$ and $p_i(\alpha_i) = 0$ for $i \leqslant k$ then $q(t(\alpha_1, \ldots, \alpha_k)) = 0$.

More precisely, the algorithm sends words $\mathrm{C}^+_{i\leqslant k}\mathrm{b}(p_i(x))$ and $\mathrm{b}(t(\bar{x}))$ to the word $\mathrm{b}(q(x))$. Let $n_i = \deg[p_i(x)]$ for $i \leqslant k$ and let $n = \max\limits_{i\leqslant k}\{n_i\}$. The working time for $n_1 n_2 \ldots n_k \neq 0$ can be estimated as $(n_1 n_2 \ldots n_k)^c L^d$, or $n^{ck} L^d$, where c and d are fixed constants and L is the total input length. For k fixed, the algorithm is polynomial. Furthermore, $\deg[q(x)] \leqslant n_1 n_2 \ldots n_k$.

Proof. If $n_i = 0$ for some $i \leqslant k$, then the number α_i does not exist, and so we can put $q(x) = 1$. Suppose $n_i \geqslant 1$ for $i \leqslant k$. Denote by b_i the leading coefficient of $p_i(x)$. We may assume that $b_i > 0$. Let $\mathrm{L}(a) < M$ for an arbitrary coefficient a of any polynomial $p_i(x)$, $i \leqslant k$. Below we present several algorithms assuming that, along with the objects specified in the formulation above, they take as inputs polynomials $p_i(x)$, $i \leqslant k$. Let α_i be some numbers in $\mathbb{C}$ for which $p_i(\alpha_i) = 0$. The algorithms do not depend on the choice of these numbers. □

LEMMA 5. There exists a p-computable function that, given $p_i(x)$, where $i \leqslant k$, and 1^s, $s \geqslant 0$, finds a tuple $(c_0^{s,i}, \ldots, c_{n_i-1}^{s,i})$ in $\mathbb{Z}$ such that

$$\alpha_i^s = \frac{1}{b_i^s} \sum_{t<n_i} c_t^{s,i} \alpha_i^t. \tag{1}$$

Furthermore, $\mathrm{L}(c_t^{s,i}) \leqslant 3s(M + \mathrm{L}(n_i))$ for $s \neq 0$, $\mathrm{L}(c_t^{s,i}) = 1$ for $s = 0$, and the working time is bounded by $cn_i^2(s+1)^2(M + \mathrm{L}(n_i))^2$ steps for some constant $c \geqslant 1$.

Proof. If $s < n_i$ then $\alpha_i^s = \frac{1}{b_i^s}(b_i^s \alpha_i^s)$. Suppose $s \geqslant n_i$. By Proposition 2, we obtain $b_i^{s-n_i+1} x^s = p_*(x) p_i(x) + r(x)$, where $r(x) \in \mathbb{Z}[x]$, $\deg[r(x)] < n_i$, and $\mathrm{L}(a) \leqslant 2s(M + \mathrm{L}(n_i))$ if a are coefficients of $r(x)$. These coefficients generate the desired tuple since $p_i(\alpha_i) = 0$ and $\alpha_i^s = \frac{1}{b_i^{s-n_i+1}} r(\alpha_i) = \frac{1}{b_i^s}(b_i^{n_i-1} r(\alpha_i))$. Multiplying the coefficients by $b_i^{n_i-1}$ increases the length to $(n_i - 1)M \leqslant s(M + \mathrm{L}(n_i))$, which yields an estimate of the same form.

The time for pseudo-division is $cn_i s^2(M + \mathrm{L}(n_i))^2$, and the degree $b_i^{n_i-1}$ is computed in $cn_i^2 M^2$ steps, which does not exceed the desired estimate. □

LEMMA 6. There exists an algorithm that, given $1^{t_1} * \ldots * 1^{t_k}$ and $1^{s_1} * \ldots * 1^{s_k}$, where $t_i < n_i$ and $s_i \geqslant 0$, finds a number $d^{s_1,\ldots,s_k}_{t_1,\ldots,t_k} \in \mathbb{Z}$ for which

$$\alpha_1^{s_1}\alpha_2^{s_2}\ldots\alpha_k^{s_k} = \frac{1}{b_1^s b_2^s \ldots b_k^s} \sum_{t_i<n_i} d^{s_1,\ldots,s_k}_{t_1,\ldots,t_k} \alpha_1^{t_1}\alpha_2^{t_2}\ldots\alpha_k^{t_k}, \tag{2}$$

where $s = \max\limits_{i \leqslant k}\{s_i\}$. Furthermore, $\mathrm{L}(d^{s_1,\ldots,s_k}_{t_1,\ldots,t_k}) \leqslant 3ks(M + \mathrm{L}(n))$ for $s \neq 0$ and 1 for $s = 0$, and the working time is bounded by $ck^2n^2(s+1)^2(M + \mathrm{L}(n))^2$.

Proof. Let $e = b_1^{s-s_1} \ldots b_k^{s-s_k}$. Using (1) and removing parenthesis, we have $d^{s_1,\ldots,s_k}_{t_1,\ldots,t_k} = ec^{s_1,1}_{t_1} \ldots c^{s_k,k}_{t_k}$. The estimate for $\mathrm{L}(c^{s,i}_t)$ implies that $\mathrm{L}(d^{s_1,\ldots,s_k}_{t_1,\ldots,t_k}) \leqslant \sum\limits_{i=1}^{k}\big[(s - s_i)M + 3s_i(M + \mathrm{L}(n_i))\big] \leqslant 3ks(M + \mathrm{L}(n))$. The working time involves computing all $c^{s,i}_{t_i}$, $i \leqslant k$, which requires $ckn^2(s+1)^2(M+\mathrm{L}(n))^2$ steps, computing their product in $ck^2(s+1)^2(M+\mathrm{L}(n))^2$ steps, computing e in $ck^2(s+1)^2M^2$ steps, and multiplying these. □

Let $u = n_1n_2\ldots n_k$. Then $u \leqslant n^k$.

LEMMA 7. There exists an algorithm that, given 1^s, $s \geqslant 1$, and a tuple $e, \{e_{t_1,\ldots,t_k}\}_{t_i<n_i}$ in $\mathbb{Z}$, finds a tuple $e_s, \{e^s_{t_1,\ldots,t_k}\}_{t_i<n_i}$ in $\mathbb{Z}$ such that

$$\text{if } \beta = \frac{1}{e}\sum_{t_i<n_i} e_{t_1,\ldots,t_k}\alpha_1^{t_1}\ldots\alpha_k^{t_k}, \text{ then } \beta^s = \frac{1}{e_s}\sum_{t_i<n_i} e^s_{t_1,\ldots,t_k}\alpha_1^{t_1}\ldots\alpha_k^{t_k}. \tag{3}$$

Suppose also that $\mathrm{L}(e), \mathrm{L}(e_{t_1,\ldots,t_k}) \leqslant T$ for $t_i < n_i$, $i \leqslant k$, and $R = T + M + \mathrm{L}(n)$. Then $\mathrm{L}(e_s), \mathrm{L}(e^s_{t_1,\ldots,t_k}) \leqslant 10sknR$.

The algorithm works inductively and the transition from a decomposition for β^s to a decomposition for β^{s+1} takes $csk^2n^4u^3R^2$ steps. To obtain the final estimate, we should multiply the last expression by s.

Proof. In this algorithm, (2) will be used only for the case where $s_i \leqslant 2n$ for all $i \leqslant k$. In such a situation, it seems convenient to think that

$$\alpha_1^{s_1}\alpha_2^{s_2}\ldots\alpha_k^{s_k} = \frac{1}{d}\sum_{t_i<n_i} d^{s_1,\ldots,s_k}_{t_1,\ldots,t_k}\alpha_1^{t_1}\alpha_2^{t_2}\ldots\alpha_k^{t_k},$$

where $d = (b_1 \ldots b_k)^{2n}$. For this, the coefficients in (2) should be multiplied by $(b_1 \ldots b_k)^{2n-s}$. Since $\mathrm{L}(d) \leqslant 2knM$, we may now assume that $\mathrm{L}(d^{s_1,\ldots,s_k}_{t_1,\ldots,t_k}) \leqslant 8kn(M + \mathrm{L}(n)) \leqslant 8knR$, and the time for computing a single coefficient is $ck^2n^4R^2$.

Suppose that equality (3) holds for β^s. Simplifying the expression $\beta^{s+1} = \beta^s\beta$, removing parenthesis, and using (2), we see that $e_{s+1} = e_sed$ and

$$e^{s+1}_{t_1,\ldots,t_k} = \sum_{t'_i,t''_i<n_i}\big[d^{t'_1+t''_1,\ldots,t'_k+t''_k}_{t_1,\ldots,t_k} e^s_{t'_1,\ldots,t'_k} e_{t''_1,\ldots,t''_k}\big]. \tag{4}$$

This implies $\mathrm{L}(e_s), \mathrm{L}(e^s_{t_1,\ldots,t_k}) \leqslant 10sknR$ because $\mathrm{L}(n^{2k}) \leqslant 2k\mathrm{L}(n)$. The same estimate holds for all numbers appearing while using (4) for $e^{s+1}_{t_1,\ldots,t_k}$.

Now we estimate the time of transition from β^s to β^{s+1}. Suppose that numbers $d^{s_1,\dots,s_k}_{t_1,\dots,t_k}$ are already found. Each summand in (4) requires two multiplications, i.e., at most $ck^2n^2sR^2$ steps.

Since the numbers $d^{s_1,\dots,s_k}_{t_1,\dots,t_k}$ permanently appear in our computations, it seems natural to compute them once, save, and use when needed. However, in this case their list with about $2u^2$ entries should be kept on a separate tape, and reading from it a single element requires searching through the entire list. Computing them anew takes only $ck^2n^4R^2$ steps. The total time for one summand in (4) equals $csk^2n^4R^2$.

The formula consists of u^2 summands, hence the total time for computing the multiplications equals $csk^2n^4u^2R^2$. Computing the sums requires at most $cu^2(csknR + \mathrm{L}(u^2))$ steps, $\mathrm{L}(u^2) \leqslant 2k\mathrm{L}(n)$, which is less than the time for multiplications. Since formula (4) is used u times, the whole time for transition from β^s to β^{s+1} can be estimated by the expression specified in the lemma.

We briefly describe the entire algorithm for computing (4). Choose the lexicographic ordering on tuples $(t_1,\dots,t_k)$, where $t_i < n_i$, $i \leqslant k$, one tape keeps a list $\{e^s_{t'_1,\dots,t'_k}\}_{t'_i<n_i}$ in this order, and another one keeps a similar list $\{e_{t''_1,\dots,t''_k}\}_{t''_i<n_i}$. To obtain one $e^{s+1}_{t_1,\dots,t_k}$ we proceed as follows: look through elements of the first list, and for each of these, look through the second list; compute the corresponding products and then sum them all up. In this process we organize counters $(t'_1,\dots,t'_k)$ and $(t''_1,\dots,t''_k)$, working with which is not time-consuming. □

LEMMA 8. There exists an algorithm that, given $\mathrm{b}(t(x_1,\dots,x_k))$, finds a tuple of numbers $e, \{e_{t_1,\dots,t_k}\}_{t_i<n_i}$ in $\mathbb{Z}$ such that if $\beta = t(\alpha_1,\dots,\alpha_k)$ then

$$\beta = \frac{1}{e}\sum_{t_i<n_i} e_{t_1,\dots,t_k}\alpha_1^{t_1}\dots\alpha_k^{t_k}. \tag{5}$$

Suppose also that

$$t(x_1,\dots,x_k) = \sum_{(s_1,\dots,s_k)\in I} q_{s_1,\dots,s_k}x_1^{s_1}x_2^{s_2}\dots x_k^{s_k},$$

$m = \max\limits_{(s_1,\dots,s_k)\in I}\{s_i\}$, and $\mathrm{L}(a) < F$, where a is the numerator or denominator of $q_{s_1,\dots,s_k}$. Then $\mathrm{L}(e), \mathrm{L}(e_{t_1,\dots,t_k}) \leqslant |I|F + 4km(M+\mathrm{L}(n))$, and the working time is bounded by $ck^2m^2n^2u|I|^2F^2(M+\mathrm{L}(n))^2$.

Proof. At a first step, we converge $t(\bar{x})$ to the form $\frac{1}{f}\sum\limits_{(s_1,\dots,s_k)\in I} f_{s_1,\dots,s_k}x_1^{s_1}\dots x_k^{s_k}$, where $f, f_{s_1,\dots,s_k} \in \mathbb{Z}$. Let f be the product of all denominators of coefficients in $t(\bar{x})$. Then $\mathrm{L}(f) \leqslant |I|F$ and its computation requires a run through $\mathrm{b}(t(\bar{x}))$ and multiplication of $|I|$ numbers, i.e., $c(m + |I|(k\mathrm{L}(m) + F) + |I|^2F^2)$ steps, which does not exceed the total estimate. Then we again run through $\mathrm{b}(t(\bar{x}))$ and rewrite it, replacing $q_{s_1,\dots,s_k}$ by $f_{s_1,\dots,s_k} = fq_{s_1,\dots,s_k}$ and replacing the monomial encoding $\mathrm{b}(s_1) * \dots * \mathrm{b}(s_k)$ by $1^{s_1} * \dots * 1^{s_k}$. Furthermore, $\mathrm{L}(f_{s_1,\dots,s_k}) \leqslant |I|F$, and the first operation takes $c|I|^2F^2$ steps while the second one is effected in $ckm|I|$ steps.

Let $b = b_1 \dots b_k$. Moving along the list of monomials and using (2) for each, we obtain formula (5) where $e = fb^m$ and

$$e_{t_1,\ldots,t_k} = \sum_{(s_1,\ldots,s_k)\in I} f_{s_1,\ldots,s_k} b^{m-s} d^{s_1,\ldots,s_k}_{t_1,\ldots,t_k}, \tag{6}$$

where s in every summand stands for $\max\{s_1, \ldots, s_k\}$.

Here $\mathrm{L}(d^{s_1,\ldots,s_k}_{t_1,\ldots,t_k}) \leqslant 3km(M + \mathrm{L}(n))$, and the time of its computation is $ck^2m^2n^2(M + \mathrm{L}(n))^2$. In each sum of form (6), the computation should be repeated $|I|$ times, doing so for each $e_{t_1,\ldots,t_k}$, whose number is u. Therefore, the whole time for computing the numbers $d^{s_1,\ldots,s_k}_{t_1,\ldots,t_k}$ is bounded by $ck^2m^2n^2u|I|(M + \mathrm{L}(n))^2$, which does not exceed the total estimate. Since the length of every component is known, it is not hard to estimate the time needed for multiplications and summations in (6), which also fits in the total estimate. □

Let $\beta = t(\alpha_1, \ldots, \alpha_k)$. It remains to find a nonzero polynomial $q(x) \in \mathbb{Z}[x]$ with $q(\beta) = 0$. Using Lemma 8, we can find a tuple $\bar{e}_1 \in \mathbb{Q}^u$ such that $\beta = \bar{e}_1\bar{a}$, where $\bar{a}$ is a column vector of numbers $\{\alpha_1^{t_1}\alpha_2^{t_2}\ldots\alpha_k^{t_k}\}_{t_i<n_i}$ in some fixed order. Applying Lemma 7, we find tuples $\bar{e}_2, \ldots, \bar{e}_u \in \mathbb{Q}^u$ such that $\beta^s = \bar{e}_s\bar{a}$ for $s \leqslant u$. In addition, $\beta^0 = \bar{e}_0\bar{a} = 1 \cdot \alpha_1^0\alpha_2^0\ldots\alpha_k^0$. Since the dimension of the vector space $\mathbb{Q}^u$ equals u, there is a nonzero tuple $\lambda_0, \lambda_1, \ldots, \lambda_u \in \mathbb{Q}$ for which $\lambda_0\bar{e}_0 + \ldots + \lambda_u\bar{e}_u = 0$. In this case $\lambda_0 + \lambda_1\beta + \ldots + \lambda_u\beta^u = 0$, which gives the required polynomial.

We form a matrix $A = [\bar{e}_0\bar{e}_1\ldots\bar{e}_u]$ of size $u \times (u + 1)$, considering $\bar{e}_s$ as columns. Let $\bar{\lambda} = (\lambda_0, \ldots, \lambda_u)^\top$ be a column vector. The problem reduces to finding a nonzero solution for the system $A\bar{\lambda} = 0$.

Combining the estimates in Lemmas 7 and 8, we see that the length of numerators and denominators of the numbers in A is bounded by $10knu(|I|F + 5km(M + \mathrm{L}(n))$. Let L be the total input length of the algorithm that is currently under construction. Then the last expression can be rewritten as uL^d, where d is a fixed degree because $L \geqslant 2$.

To find a vector $\bar{\lambda}$, we apply the theorem saying that Gauss's method for solving systems of linear equations over $\mathbb{Q}$ works in polynomial time, which requires subtle estimates for the numbers appearing in realizing the method (see [17, Thm. 3.3]). Gauss's method transforms the system into $[B\,C]\bar{\lambda} = 0$, where B is a diagonal matrix of size $u \times u$, if we admit column permutations (and hence also permutations of elements $\bar{\lambda}$). Setting then $\lambda_u = 1$, we easily find the desired $\bar{\lambda}$.

In the notation of [17], the input size for Gauss's algorithm is the sum of all lengths of elements of the matrix A, i.e., a number of the form u^3L^d, and the working time is bounded by $cu^{3r}L^{dr}$ where $r \geqslant 1$ is a fixed number; the same estimate holds for the length of elements of the vector λ. Since $L \geqslant 2$, the constant c may be removed. The time estimates from Lemmas 7 and 8 also do not exceed this formula. Passing from $\mathbb{Q}[x]$ to $\mathbb{Z}[x]$, we see that $q(x)$ is found in time stated in the formulation. Proposition 3 is proved. □

LEMMA 9. Let $n \geqslant 1$ and $a_1, \ldots, a_n, b_1, \ldots, b_n \in \mathbb{C}$. Then

$$|b_1b_2\ldots b_n - a_1a_2\ldots a_n| \leqslant nM^{n-1}\varepsilon,$$

where $\varepsilon = \max\limits_{i\leqslant n}|b_i - a_i|$ and $M = \max\limits_{i\leqslant n}\{|a_i|, |b_i|\}$.

Proof. It suffices to note that

$$b_1 b_2 \dots b_n - a_1 a_2 \dots a_n = \sum_{t=1}^{n} b_1 \dots b_{t-1} a_{t+1} \dots a_n (b_t - a_t).$$

This can be shown by induction, using the equality $b_1 b_2 \dots b_{n+1} - a_1 a_2 \dots a_{n+1} = b_1 \dots b_n (b_{n+1} - a_{n+1}) + a_{n+1}(b_1 \dots b_n - a_1 \dots a_n)$. From which we derive an estimate for $|b_1 b_2 \dots b_n - a_1 a_2 \dots a_n|$, which completes the proof. □

LEMMA 10. (a) There exists a p-computable function that, given a polynomial $q(x) \in \mathbb{Z}[x] \setminus \{0\}$, finds two polynomials $q_1(x), q_2(x) \in \mathbb{Z}[x] \setminus \{0\}$ such that if $\alpha, \beta \in \mathbb{R}$ and $q(\alpha + i\beta) = 0$ then $q_1(\alpha) = q_2(\beta) = 0$. Furthermore, $\deg[q_1(x)], \deg[q_2(x)] \leqslant \deg[q(x)]^2$.

(b) There exists a p-computable function that realizes the inverse operation: given polynomials $q_1(x), q_2(x) \in \mathbb{Z}[x] \setminus \{0\}$, it finds a polynomial $q(x) \in \mathbb{Z}[x] \setminus \{0\}$ such that if $\alpha, \beta \in \mathbb{R}$ and $q_1(\alpha) = q_2(\beta) = 0$ then $q(\alpha + i\beta) = 0$. Furthermore, $\deg[q(x)] \leqslant 2 \deg[q_1(x)] \deg[q_2(x)]$.

Proof. (a) We suppose that $\gamma = \alpha + i\beta$, where $\alpha, \beta \in \mathbb{R}$, and $q(\gamma) = 0$. It is well known that $q(\bar{\gamma}) = 0$, where $\bar{\gamma} = \alpha - i\beta$. Applying Proposition 3 to $\alpha_1 = \gamma$, $\alpha_2 = \bar{\gamma}$, and $t(x_1, x_2) = \frac{1}{2}(x_1 + x_2)$, we obtain a polynomial $q_1(x) \in \mathbb{Z}[x] \setminus \{0\}$ for which $q_1(\alpha) = 0$ and $\deg[q_1(x)] \leqslant \deg[q(x)]^2$. Applying the same proposition to γ, $\bar{\gamma}$, and $\frac{1}{2}(x_1 - x_2)$, we arrive at a polynomial $p(x)$ such that $\deg[p(x)] \leqslant \deg[q(x)]^2$ and $p(i\beta) = 0$. Let $p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$. If we substitute $i\beta$ for x and compute i^t with $t \leqslant n$ we can easily obtain a polynomial $q_2(x) \in \mathbb{Z}[x]$ for which $q_2(\beta) = 0$ and $\deg[q_2(x)] \leqslant \deg[p(x)]$.

(b) Let $q_1(\alpha) = q_2(\beta) = 0$. If we apply Proposition 3 to $\alpha_1 = \alpha$, $\alpha_2 = \beta$, $\alpha_3 = i$ and to a polynomial $x_1 + x_2 x_3$ we arrive at $q(x)$. □

In particular, $\deg[\gamma] \leqslant 2\deg_{\mathrm{R}}[\gamma]^2$ for every algebraic number $\gamma \in \mathbb{C}$.

LEMMA 11. There exists a p-computable function that, given $\gamma \in C_{\text{alg}}$ and $\varepsilon \in \mathbb{Q}$, $\varepsilon > 0$, finds $a \in \mathbb{Q}[i]$ with $|\gamma - a| \leqslant \varepsilon$. If, in addition, γ is real, then $a \in \mathbb{Q}$.

Proof. Let $\gamma = \alpha + i\beta$, where $\alpha, \beta \in \mathbb{R}$. As input of the algorithm, we use pairs $(p_1(x), m_1)$ and $(p_2(x), m_2)$ that encode α and β, respectively. By Theorem 2, we can find numbers $a_1, a_2 \in \mathbb{Q}$ such that $|\alpha - a_1|, |\beta - a_2| \leqslant \varepsilon/2$. If $a = a_1 + ia_2$ then $|\gamma - a| \leqslant \varepsilon$.

If, in addition, $\beta = 0$, we just set $a = a_1$. □

THEOREM 4. There exists an algorithm that, given $k \geqslant 1$, $\alpha_1, \ldots, \alpha_k \in C_{\text{alg}}$, and $t(x_1, \ldots, x_k) \in \mathbb{Q}[x_1, \ldots, x_k]$, finds $\beta = t(\alpha_1, \ldots, \alpha_k) \in C_{\text{alg}}$.

More precisely, given the words $\mathrm{C}^{\&}_{i \leqslant k} \mathrm{b}(\alpha_i)$ and $\mathrm{b}(t(x_1, \ldots, x_k))$, the algorithm finds a word $\mathrm{b}(\beta)$. Let $n_i = \deg[\alpha_i]$, $i \leqslant k$, and $n = \max\limits_{i \leqslant k}\{n_i\}$. The working time of the algorithm is bounded by $(n_1 n_2 \ldots n_k)^c L^d$, or $n^{ck} L^d$, where c and d are fixed constants and L is the total input length. In particular, for every fixed k, we obtain a p-computable function that computes values for polynomials in C_{alg}. Moreover, $\deg[\beta] \leqslant \prod\limits_{i \leqslant k} \deg[\alpha_i]$.

The **proof** is based on the following lemma.

LEMMA 12. There exists a p-computable function that, given $k \geqslant 1$, $\alpha_1, \ldots, \alpha_k \in C_{\text{alg}}$, a polynomial $t(x_1, \ldots, x_k) \in \mathbb{Q}[x_1, \ldots, x_k]$, and $\varepsilon \in \mathbb{Q}$, $\varepsilon > 0$, finds a number $a \in \mathbb{Q}[i]$ such that $|a - t(\alpha_1, \ldots, \alpha_k)| \leqslant \varepsilon$.

Proof. First, we consider the case $t(x_1, \ldots, x_k) = x_1^{s_1} x_2^{s_2} \ldots x_k^{s_k}$, where $s_k \geqslant 1$. Using Lemma 11 k times, we find numbers $a_i^0 \in \mathbb{Q}[i]$ for which $|\alpha_i - a_i^0| \leqslant 1$ with $i \leqslant k$. Then $|\alpha_i| \leqslant |a_i^0| + 1$. Suppose $b = \max\limits_{i \leqslant k}\{|a_i^0|\} + 1$.

Let $n = s_1 + s_2 + \ldots + s_k$. By the definition of a polynomial encoding, $s_i \leqslant \mathrm{L}(t(\bar{x}))$ for $i \leqslant k$, whence $n \leqslant k\mathrm{L}(t(\bar{x})) \leqslant \mathrm{L}(t(\bar{x}))^2$. The number $\varepsilon_1 = \min\left\{\frac{\varepsilon}{n(b+1)^{n-1}}, 1\right\}$ can be found in polynomial time. Using Lemma 11 again, we find $a_i \in \mathbb{Q}[i]$ such that $|\alpha_i - a_i| \leqslant \varepsilon_1$, and then set $a = a_1^{s_1} a_2^{s_2} \ldots a_k^{s_k}$. By Lemma 9, $|\alpha_1^{s_1} \ldots \alpha_k^{s_k} - a| \leqslant n(b+1)^{n-1}\varepsilon_1 \leqslant \varepsilon$ since $|a_i| \leqslant |\alpha_i| + \varepsilon_1 \leqslant b + 1$.

Consider now the case of an arbitrary polynomial

$$t(\bar{x}) = \sum_{(s_1,\ldots,s_k) \in I} q_{s_1,\ldots,s_k} x_1^{s_1} \ldots x_k^{s_k}.$$

Let $\varepsilon_1 = \varepsilon/|I|$. Moving along the list of monomials, for each $(s_1, \ldots, s_k) \in I$, we find a number $a_{s_1,\ldots,s_k} \in \mathbb{Q}[i]$ such that $|q_{s_1,\ldots,s_k}\alpha_1^{s_1} \ldots \alpha_k^{s_k} - a_{s_1,\ldots,s_k}| \leqslant \varepsilon_1$. To do this, we use the previous algorithm to find $a' \in \mathbb{Q}[i]$ satisfying the condition $|\alpha_1^{s_1} \ldots \alpha_k^{s_k} - a'| \leqslant \varepsilon_1/|q_{s_1,\ldots,s_k}|$, and then set $a_{s_1,\ldots,s_k} = q_{s_1,\ldots,s_k} a'$. Whenever all these numbers are found, we put $a = \sum\limits_{(s_1,\ldots,s_k) \in I} a_{s_1,\ldots,s_k}$. $\square$

The algorithm takes as input polynomials in $\mathbb{Z}[x]$, whose roots are the real and imaginary parts of α_i. Using Lemma 10 k times, we find polynomials $p_i(x) \in \mathbb{Z}[x] \setminus \{0\}$ such that $p_i(\alpha_i) = 0$ for $i \leqslant k$. At the moment, we assume that $n_i = \deg[p_i(x)]$. The equality $n_i = \deg[\alpha_i]$ may fail to hold in general. Let $u = n_1 n_2 \ldots n_k$. Applying Proposition 3, we find $q(x) \in \mathbb{Z}[x] \setminus \{0\}$ such that $q(\beta) = 0$. Since $p_i(x)$ were obtained in polynomial time, this takes $u^c L^d$ steps; a similar estimate holds for the length of $q(x)$, and $\deg[q(x)] \leqslant u$.

Let $\beta = \beta' + i\beta''$. Using Lemma 10 again, we find $q_1(x), q_2(x) \in \mathbb{Z}[x] \setminus \{0\}$ such that $q_1(\beta') = q_2(\beta'') = 0$. Moving through all numbers $s \leqslant \deg[q_1(x)]$, we find a list $\beta_1', \ldots, \beta_e'$ of all real roots of $q_1(x)$, which are encoded by pairs $(q_1(x), s)$, and then find a list $\beta_1'', \ldots, \beta_f''$ of all roots of $q_2(x)$. As a result, we obtain a list of numbers $z_1, \ldots, z_m \in C_{\text{alg}}$ of the form $\beta_s' + i\beta_t''$, among which are all complex roots of $q(x)$.

Computing an estimate δ_{q_1} for the minimal distance between real roots of $q_1(x)$ and computing a similar estimate δ_{q_2} for $q_2(x)$, we obtain $\varepsilon = \min\{\delta_{q_1}, \delta_{q_2}\}$ such that $|z_i - z_j| \geqslant \varepsilon$ for $i \neq j$. By Lemma 12, we find $a \in \mathbb{Q}[i]$ with $|a - \beta| \leqslant \varepsilon/5$, and by Lemma 11, for each $j \leqslant m$ we find $a_j \in \mathbb{Q}[i]$ with $|a_j - z_j| \leqslant \varepsilon/5$. In this case $\beta = z_j \Leftrightarrow |a - a_j| \leqslant 2\varepsilon/5$, and such j can be found by an exhaustive search.

If we consider $\alpha_1, \ldots, \alpha_k, t(\bar{x})$ and $q(x)$ as input data of the algorithm, then the last part of the proof can be done in polynomial time. Consequently, the whole estimate for working time has the form $u^c L^d$. In particular, the algorithm is polynomial for $k = 1$ since $\deg[p_1(x)] \leqslant 2\deg_{\mathrm{R}}[\alpha_1]^2$.

Now, in order to achieve the equality $n_i = \deg[\alpha_i]$, we return to the beginning of the proof.

Having found $p_i(x)$, we decompose it into irreducible factors, i.e., $p_i(x) = q_1(x) \dots q_r(x)$. Using the described algorithm to compute $q_1(\alpha), \dots, q_r(\alpha) \in C_{\mathrm{alg}}$, we find j with $q_j(\alpha) = 0$, and then set $p_i(x) = q_j(x)$. $\square$

As mentioned in the Introduction, the estimate for working time in Theorem 4 seems asymptotically better than an estimate obtained by first finding a primitive element for $\mathbb{Q}(\alpha_1, \dots, \alpha_k)$. Now we give an example showing that our estimate is in a sense best possible. In [18], the following fact was proved.

THEOREM 5. Let $p_1, \dots, p_k$ be distinct primes and $n_1, \dots, n_k \geqslant 1$. Then the extension degree satisfies the equality $[\mathbb{Q}(p_1^{1/n_1} + \dots + p_k^{1/n_k}) : \mathbb{Q}] = n_1 n_2 \dots n_k$.

Let $p_1, \dots, p_k$ be the first k primes. Then $p_k \leqslant 2^k$. We consider arbitrary numbers $n_1, \dots, n_k \geqslant 1$ and set $\alpha_i = p_i^{1/n_i}$ for $i \leqslant k$. The number α_i is coded by the pair $(x^{n_i} - p_i, m)$, where $m \in \{1, 2\}$, $\mathrm{L}(\alpha_i) \leqslant 2n_i + k + 5$, and $\mathrm{L}(\alpha_1 * \dots * \alpha_n) \leqslant 2(n_1 + \dots + n_k) + k^2 + 6k$. If $t(x_1, \dots, x_k) = x_1 + \dots + x_k$ then $\mathrm{L}(t) \leqslant k^2 + 6k$. In this case $\mathrm{L}(\alpha_1 + \dots + \alpha_k) \geqslant n_1 n_2 \dots n_k$ since the length of a number is not less than the degree of its minimal polynomial.

This implies that the algorithm in Theorem 4 cannot be polynomial even in computing polynomials of the form $x_1 + \dots + x_k$, and the estimate specified in that theorem is bounded by a polynomial in L and by the best possible lower bound $n_1 n_2 \dots n_k$. Moreover, if, in our example, $n_i = n$ for $i \leqslant k$ and $L^* = |\alpha_1 + \dots + \alpha_k|$, then a lower bound for L^*/L^d as $k \to \infty$ is close to n^k, and therefore the estimate in Theorem 4 cannot be radically improved.

5. POLYNOMIAL PRESENTATIONS FOR $\mathbb{R}_{\mathrm{alg}}$ AND $\mathbb{C}_{\mathrm{alg}}$

THEOREM 6. The ordered field of algebraic reals $\mathfrak{A}^{\mathrm{R}}$ has a p-computable presentation $\mathfrak{A}_1^{\mathrm{R}} = (R_{\mathrm{alg}}, \leqslant, +, \times)$, in which the functions $x \mapsto -x$ and $x \mapsto \frac{1}{x}$ for $x \neq 0$ are also p-computable.

Proof. The set R_{alg} has been constructed above. For $\alpha \in \mathbb{R}_{\mathrm{alg}}$, the transfer from $\mathrm{b}(\alpha) \in R_{\mathrm{alg}}$ to a corresponding word in C_{alg} and vice versa is easy, so Theorem 4 implies that the operations $x + y$, $x \times y$, and $-x$ are p-computable, while Corollary 3 gives p-computability of the order relation. It remains to prove this for $\frac{1}{x}$.

Let $(p(x), k)$ be the natural code for $\alpha \in \mathbb{R}_{\mathrm{alg}}$ and $p(x) = a_n x^n + \dots + a_1 x + a_0$. If $\alpha \neq 0$ then $\frac{1}{\alpha}$ is a root of a polynomial $p_1(y) = y^n p(\frac{1}{y}) = a_n + a_{n-1} y + \dots + a_1 y^{n-1} + a_0 y^n$. Suppose $\alpha > 0$. Using Corollary 3, we find $\varepsilon_0 \in \mathbb{Q}$ with $0 < \varepsilon_0 \leqslant \alpha$ and $\delta_{p_1} \in \mathbb{Q}$ with $0 < \delta_{p_1} \leqslant \Delta_{p_1}$. Let $\varepsilon = \min\left\{\frac{\delta_{p_1} \varepsilon_0^2}{4}, \frac{\varepsilon_0}{2}\right\}$. Now we find $a, b \in \mathbb{Q}$ such that $a < \alpha < b$ and $|b - a| \leqslant \varepsilon$. Then $\alpha - a < \frac{\varepsilon_0}{2}$ and $\alpha - \frac{\varepsilon_0}{2} \geqslant \frac{\varepsilon_0}{2}$, whence $\alpha - \frac{\varepsilon_0}{2} > \alpha - a$ and $a > \frac{\varepsilon_0}{2}$. Furthermore, $\frac{1}{b} < \frac{1}{\alpha} < \frac{1}{a}$ and $\frac{1}{a} - \frac{1}{b} = \frac{b-a}{ab} \leqslant \frac{b-a}{a^2} < \varepsilon \frac{4}{\varepsilon_0^2} \leqslant \delta_{p_1}$.

Since the interval $(\frac{1}{a}, \frac{1}{b})$ contains at most one root of $p_1(y)$, it remains to find $m \leqslant n$ such that the pair $(p_1(y), m)$ encodes a number in this interval. $\square$

THEOREM 7. The field $(\mathbb{C}_{\mathrm{alg}}, +, \times)$ of complex algebraic numbers has a p-computable presentation $\mathfrak{A}_1^{\mathrm{C}} = (C_{\mathrm{alg}}, +, \times)$, in which the operations $-x$ and $\frac{1}{x}$ are also p-computable.

Proof. The set C_{alg} has been defined above; the p-computability of functions $x+y$, $x\times y$, and $-x$ follows directly from Theorem 4. The function $\frac{1}{x}$ is expressed via the operations in $\mathfrak{A}_1^{\mathrm{R}}$ using the formula $\frac{1}{y+iz}=\frac{y-iz}{y^2+z^2}$. □

6. SOLVING EQUATIONS WITH ONE VARIABLE

THEOREM 8. There exists an algorithm that, given $k\geqslant 1$, $\alpha_1,\ldots,\alpha_k\in C_{\mathrm{alg}}$, and polynomials $t_0(\bar{x}),\ldots,t_e(\bar{x})\in\mathbb{Q}[x_1,\ldots,x_k]$, finds a list $\beta_1,\ldots,\beta_g\in C_{\mathrm{alg}}$ of all complex roots of the equation

$$t_e(\alpha_1,\ldots,\alpha_k)x^e+\ldots+t_1(\alpha_1,\ldots,\alpha_k)x+t_0(\alpha_1,\ldots,\alpha_k)=0.$$

More exactly, it maps the words $\mathrm{C}^{\&}_{i\leqslant k}\mathrm{b}(\alpha_i)$ and $\mathrm{C}^{\&}_{p\leqslant e}\mathrm{b}(t_p(\bar{x}))$ into a word $\mathrm{C}^{\&}_{j\leqslant g}\mathrm{b}(\beta_j)$. Let $n_i=\deg[\alpha_i]$ for $i\leqslant k$, $n=\max\limits_{i\leqslant k}\{n_i\}$, and L be the total input length of the algorithm. Its working time can be estimated as $(n_1n_2\ldots n_k)^cL^d$, or $n^{ck}L^d$, where c and d are fixed constants. In particular, we obtain a polynomial root-finding algorithm for equations if k is fixed or $n=1$. In addition, $\deg[\beta_j]\leqslant e\prod\limits_{i\leqslant k}\deg[\alpha_i]$ for $j\leqslant g$.

Proof. First, we consider the case where $t_e(\bar{x})=-1$, i.e., an equation has the form $x^e=t_{e-1}(\bar{\alpha})x^{e-1}+\ldots+t_1(\bar{\alpha})x+t_0(\bar{\alpha})$, where $\bar{\alpha}=\alpha_1,\ldots,\alpha_k$. Each polynomial $t_p(\bar{x})$ is representable as

$$\sum_{(s_1,\ldots,s_k)\in I_p} q^p_{s_1,\ldots,s_k}x_1^{s_1}\ldots x_k^{s_k},$$

where $I_p\subseteq\omega^k$. We will assume that $I_p=I$ for all $p<e$. Computing the product f of denominators of all coefficients of polynomials and then multiplying the polynomials by f, we may also assume that each $t_p(\bar{x})$ has the form $\frac{1}{f}\sum\limits_{(s_1,\ldots,s_k)\in I} f^p_{s_1,\ldots,s_k}x_1^{s_1}\ldots x_k^{s_k}$, where $f, f^p_{s_1,\ldots,s_k}\in\mathbb{Z}$. Denote by m the maximum of powers in all polynomials, i.e., $m=\max\limits_{(s_1,\ldots,s_k)\in I}\{s_i\}$. We may suppose that $m\geqslant 1$. If $e=1$ then the only root of the equation is $t_0(\bar{\alpha})$. A value for this polynomial can be computed in a specified number of steps using Theorem 4. Below we assume that $e\geqslant 2$.

Applying Lemma 10, we find $p_i(x)\in\mathbb{Z}[x]\setminus\{0\}$ such that $p_i(\alpha_i)=0$ for $i\leqslant k$. Arguing as in Theorem 4, we may suppose that $p_i(x)$ is irreducible and $\deg[p_i(x)]=n_i$. Let $b_i>0$ be the leading coefficient of $p_i(x)$. Suppose also that $L(a)<M$ if a is a coefficient of $p_i(x)$, $i\leqslant k$. Put $u=n_1n_2\ldots n_k$.

Let β be a root of the initial equation. Then

$$\beta^e=\sum_{p<e}t_p(\alpha_1,\ldots,\alpha_k)\beta^p=\frac{1}{f}\sum_{p<e}\sum_{(s_1,\ldots,s_k)\in I}f^p_{s_1,\ldots,s_k}\alpha_1^{s_1}\ldots\alpha_k^{s_k}\beta^p.$$

We describe an algorithm that, given 1^s, $s\geqslant 1$, finds a set e_s, $\{e^s_{t_1,\ldots,t_k,p}\}^{p<e}_{t_i<n_i}$ of numbers in $\mathbb{Z}$ such that

$$\beta^s=\frac{1}{e_s}\sum_{t_i<n_i}^{p<e}e^s_{t_1,\ldots,t_k,p}\alpha_1^{t_1}\ldots\alpha_k^{t_k}\beta^p. \tag{7}$$

If $s < e$ then $e_s = 1$ and $e^s_{t_1,\dots,t_k,p} \in \{0,1\}$. Here Lemma 6 is used in the following form: we assume that $\alpha_1^{s_1}\dots\alpha_k^{s_k} = \frac{1}{d}\sum\limits_{t_i<n_i} d^{s_1,\dots,s_k}_{t_1,\dots,t_k}\alpha_1^{t_1}\dots\alpha_k^{t_k}$, where $d = (b_1b_2\dots b_k)^{m+n}$, since it will be applied only for cases with $s_i \leqslant m+n$.

Let $s = e$. Then

$$\beta^s = \frac{1}{fd}\sum_{p<e}\sum_{(s_1,\dots,s_k)\in I}\sum_{t_i<n_i} f^p_{s_1,\dots,s_k} d^{s_1,\dots,s_k}_{t_1,\dots,t_k}\alpha_1^{t_1}\dots\alpha_k^{t_k}\beta^p,$$

i.e., $e_s = fd$ and $e^s_{t_1,\dots,t_k,p} = \sum\limits_{(s_1,\dots,s_k)\in I} f^p_{s_1,\dots,s_k} d^{s_1,\dots,s_k}_{t_1,\dots,t_k}$.

Assume that $s \geqslant e$ and formula (7) holds for β^s. Then

$$\beta^{s+1} = \beta^s\beta = \frac{1}{e_s}\sum_{t_i<n_i}^{1\leqslant p<e} e^s_{t_1,\dots,t_k,p-1}\alpha_1^{t_1}\dots\alpha_k^{t_k}\beta^p + \frac{1}{e_s}\sum_{t'_i<n_i} e^s_{t'_1,\dots,t'_k,e-1}\alpha_1^{t'_1}\dots\alpha_k^{t'_k}\beta^e.$$

Writing out the second summand, we obtain

$$\frac{1}{e_s f}\sum_{t'_i<n_i}\sum_{p<e}\sum_{(s_1,\dots,s_k)\in I} e^s_{t'_1,\dots,t'_k,e-1} f^p_{s_1,\dots,s_k}\alpha_1^{t'_1+s_1}\dots\alpha_k^{t'_k+s_k}\beta^p$$
$$= \frac{1}{e_s fd}\sum_{t_i<n_i}^{p<e}\Big[\sum_{t'_i<n_i}\sum_{(s_1,\dots,s_k)\in I} e^s_{t'_1,\dots,t'_k,e-1} f^p_{s_1,\dots,s_k} d^{t'_1+s_1,\dots,t'_k+s_k}_{t_1,\dots,t_k}\Big]\alpha_1^{t_1}\dots\alpha_k^{t_k}\beta^p.$$

Therefore, $e_{s+1} = e_s fd$ and

$$e^{s+1}_{t_1,\dots,t_k,p} = fde^s_{t_1,\dots,t_k,p-1} + \sum_{t'_i<n_i}\Big[e^s_{t'_1,\dots,t'_k,e-1}\cdot\sum_{(s_1,\dots,s_k)\in I} f^p_{s_1,\dots,s_k} d^{t'_1+s_1,\dots,t'_k+s_k}_{t_1,\dots,t_k}\Big], \tag{8}$$

where the first summand vanishes if $p = 0$.

We estimate the working time of the algorithm. Suppose that $\mathrm{L}(a) < F$, where a is the nominator or denominator of $q^p_{s_1,\dots,s_k}$. To compute f, we go along the list $t_0(\bar{x}),\dots,t_{e-1}(\bar{x})$ and multiply denominators, whose number is $|I_0| + \dots + |I_{e-1}| \leqslant e|I|$. Going along the list again, we rewrite it replacing rational coefficients by integer ones. To obtain $I = I_1\cup\dots\cup I_{e-1}$, we may write down all words $b(s_1)*\dots*b(s_k)$, where $(s_1,\dots,s_k)\in I_p$, $p < e$, and sort them in lexicographic order, removing repetitions. Then, for every $(s_1,\dots,s_k)\in I$, we go along the list of polynomials and extract $f^p_{s_1,\dots,s_k}$, replacing missing coefficients by zeros. As the result, we construct a list $\{f^p_{s_1,\dots,s_k}\}^{p<e}_{(s_1,\dots,s_k)\in I}$, ordered in lexicographic order on I, in polynomial time with respect to L. Hence $L(f), L(f^p_{s_1,\dots,s_k}) < e|I|F$ and the length of the list does not exceed $2e^2|I|^2F$.

Clearly, $\mathrm{L}(d) \leqslant k(m+n)M$. The initial estimation given in Lemma 6 is $\mathrm{L}(d^{s_1,\dots,s_k}_{t_1,\dots,t_k}) \leqslant 3k(m+n)(M+\mathrm{L}(n))$ for $s_i \leqslant m+n$, $i \leqslant k$. Here these coefficients should be multiplied by a number not greater than d, and $\mathrm{L}(d^{s_1,\dots,s_k}_{t_1,\dots,t_k}) \leqslant 4k(m+n)(M+\mathrm{L}(n))$. Let $R = 2e|I|F + 5kmn(M+\mathrm{L}(n))$. Then $\mathrm{L}(d), \mathrm{L}(d^{s_1,\dots,s_k}_{t_1,\dots,t_k}) \leqslant R$. Formula (8) easily implies that $\mathrm{L}(e^s_{t_1,\dots,t_k,p}) \leqslant sR$ for $s \geqslant 1$. The estimation $(s+1)R$ also holds for all intermediate numbers appearing in the computation of (8) for $e^{s+1}_{t_1,\dots,t_k,p}$.

We show that one number $e^{s+1}_{t_1,\dots,t_k,p}$ is computed in $csn^2u|I|R^2$ steps. First, we go along the entire list $\{e^s_{t_1,\dots,t_k,p}\}^{p<e}_{t_i<n_i}$, extracting $e^s_{t_1,\dots,t_k,p-1}$ for $p \neq 0$ and the set $\{e^s_{t'_1,\dots,t'_k,e-1}\}_{t'_i<n_i}$ in $cseuR$ steps. Next, we go through all tuples $(t'_1,\dots,t'_k)$, where $t'_i < n_i$, whose number is u, compute the sum $\sum\limits_{(s_1,\dots,s_k)\in I} f^p_{s_1,\dots,s_k} d^{t'_1+s_1,\dots,t'_k+s_k}_{t_1,\dots,t_k}$ for each, and multiply it by $e^s_{t'_1,\dots,t'_k,e-1}$. Since the lengths of all values are known, it is easy to prove that most of the time is required for computing coefficients $d^{t'_1+s_1,\dots,t'_k+s_k}_{t_1,\dots,t_k}$, which should be done $u|I|$ times, and for multiplying the sum by $e^s_{t'_1,\dots,t'_k,e-1}$, which should be done u times. One coefficient can be computed in cn^2R^2 steps, and computation of one product takes csR^2 steps. The result fits in the general estimation.

We see that the transition from a decomposition for β^s to a decomposition for β^{s+1} can be effected in $csen^2u^2|I|R^2$ steps.

Our further reasoning proceeds as in Theorem 4. Let $\bar{a}$ be a column vector of length eu consisting of numbers $\{\alpha_1^{t_1}\dots\alpha_k^{t_k}\beta^p\}^{p<e}_{t_i<n_i}$ in a fixed order. Applying the previous algorithm, we find tuples $\bar{e}_0,\dots,\bar{e}_{eu} \in \mathbb{Q}^{eu}$ such that $\beta^s = \bar{e}_s\bar{a}$ for $s \leqslant eu$. The initial equation can have many roots β, but these tuples do not depend on the choice of β by construction. There exists a nonzero set $\lambda_0,\lambda_1,\dots,\lambda_{eu} \in \mathbb{Q}$ for which $\lambda_0\bar{e}_0+\dots+\lambda_{eu}\bar{e}_{eu} = 0$. Hence $\lambda_0+\lambda_1\beta+\dots+\lambda_{eu}\beta^{eu} = 0$, and we have found a polynomial $q(x) \in \mathbb{Q}[x]$ of degree not greater than eu such that all roots of the initial equation are its roots.

In order to find $\lambda_0,\dots,\lambda_{eu}$, we consider a matrix $A = [\bar{e}_0\bar{e}_1\dots\bar{e}_{eu}]$. The total length of binary representations of all elements in A has the form u^3L^d, where d is a fixed power. Applying Gauss's polynomial method, we find $q(x)$ in time $u^cL^{d_1}$.

Again we proceed as in Theorem 4: given $q(x)$, we construct polynomials $q_1(x), q_2(x) \in \mathbb{Z}[x]\backslash\{0\}$ such that if $\beta = \beta'+i\beta''$, where $\beta',\beta'' \in \mathbb{R}$, then $q_1(\beta') = q_2(\beta'') = 0$. Going through all real roots of $q_1(x)$ and $q_2(x)$, we create a list $z_1,\dots,z_h \in C_{\text{alg}}$ that includes all roots of $q(x)$. This operation can be effected in polynomial time with respect to $q(x)$.

It remains to check which z_j are roots of the initial equation. To do this, for each $j \leqslant h$, we compute the expression

$$z_j^e - t_{e-1}(\alpha_1,\dots,\alpha_k)z_j^{k-1} - \dots - t_0(\alpha_1,\dots,\alpha_k),$$

which can be treated as a polynomial in $z_j,\alpha_1,\dots,\alpha_k$. The coefficients of the polynomial are computed from coefficients of $t_p(\bar{x})$, $p < e$, in polynomial time. The number h does not exceed $\deg[q_1(x)]\cdot\deg[q_2(x)] \leqslant e^4u^4$, $\deg[z_j] \leqslant eu$, $\deg[\alpha_i] = n_i$ for $i \leqslant k$, and the total number of steps for computing roots can be again estimated via an expression of the form $(n_1n_2\dots n_k)^cL^d$.

Now we consider the general case where $t_e(\alpha_1,\dots,\alpha_k)$ is an arbitrary polynomial. First, compute the value $t_e(\alpha_1,\dots,\alpha_k)$ using Theorem 4. If it is equal to 0, then the problem is reduced to an equation of smaller degree, and further we can proceed by induction.

Suppose $t_e(\alpha_1,\dots,\alpha_k) \neq 0$. Let $\alpha_{k+1} = -1/t_e(\alpha_1,\dots,\alpha_k)$. Then the problem is reduced to the equation

$$x^e = \alpha_{k+1}t_{e-1}(\alpha_1,\dots,\alpha_k)x^{e-1}+\dots+\alpha_{k+1}t_0(\alpha_1,\dots,\alpha_k).$$

Here $\deg[\alpha_{k+1}] \leqslant u$, and $\mathrm{L}(\alpha_{k+1}) \leqslant u^c L^d$ for some constants c and d. Considering the coefficients of the equation as polynomials in $\alpha_1, \dots, \alpha_{k+1}$, we can apply the previous algorithm and obtain a list of roots in $u^{c_1} L^{d_1}$ steps, with new constants c_1 and d_1.

The estimation of the degree of a root follows from the observation that the extension degree $[\mathbb{Q}(\alpha_1, \dots, \alpha_k) : \mathbb{Q}]$ does not exceed u, and each root β is a root of an equation of degree e with coefficients in $\mathbb{Q}(\alpha_1, \dots, \alpha_k)$. Hence $[\mathbb{Q}(\beta) : \mathbb{Q}] \leqslant [\mathbb{Q}(\alpha_1, \dots, \alpha_k, \beta) : \mathbb{Q}] \leqslant eu$. □

COROLLARY 5. There exists a polynomial algorithm that, given $\alpha_0, \dots, \alpha_e \in C_{\mathrm{alg}}$ lying in $\mathbb{Q}[i]$, finds a list $\beta_1, \dots, \beta_g \in C_{\mathrm{alg}}$ of all complex roots of the equation

$$\alpha_e x^e + \alpha_{e-1} x^{e-1} + \ldots + \alpha_1 x + \alpha_0 = 0.$$

Proof. If we set $\alpha = i$ we can consider every α_p as a polynomial $t_p(\alpha)$ for $p \leqslant e$. □

Acknowledgments. We are grateful to S. S. Goncharov for discussing the results of the paper, which allowed us to improve their formulations.

REFERENCES

1. D. Cenzer and J. B. Remmel, “Complexity theoretic model theory and algebra,” in *Handbook of Recursive Mathematics*, Vol. 1, *Recursive Model Theory*, Y. L. Ershov, S. S. Goncharov, A. Nerode, and J. B. Remmel (Eds.), *Stud. Log. Found. Math.*, **138**, Elsevier, Amsterdam (1998), pp. 381-513.
2. G. E. Collins and R. Loos, “Real zeroes of polynomials,” in *Computer Algebra. Symbolic and Algebraic Computations, Comput. Suppl.*, **4**, Springer-Verlag, New York (1982), pp. 83-94.
3. R. Loos, “Computing in algebraic extensions,” in *Computer Algebra: Symbolic and Algebraic Computations, Comput. Suppl.*, **4**, Springer-Verlag, New York (1982), 173-187.
4. A. K. Lenstra, H. W. Lenstra, and L. Lovász, “Factoring polynomials with rational coefficients,” *Math. Ann.*, **261**, 515-534 (1982).
5. M. O. Rabin, “Computable algebra, general theory and theory of computable fields,” *Trans. Am. Math. Soc.*, **95**, 341-360 (1960).
6. Yu. L. Eršov, “Theorie der Numerierungen. III,” *Z. Math. Logik Grundl. Math.*, **23**, No. 4, 289-371 (1977).
7. S. S. Goncharov and Yu. L. Ershov, *Constructive Models, Sib. School Alg. Log.* [in Russian], Nauch. Kniga, Novosibirsk (1999).
8. G. E. Collins, “The calculation of multivariate polynomial resultants,” *J. Assoc. Comput. Mach.*, **18**, No. 4, 515-532 (1971).
9. F. Winkler, *Polynomial Algorithms in Computer Algebra, Texts Monogr. Symb. Comput.*, Springer, Wien (1996).

10. H. Cohen, *A Course in Computational Algebraic Number Theory, Grad. Texts Math.*, **138**, Springer-Verlag, Berlin (1993).

11. C. K. Yap, *Fundamental Problems of Algorithmic Algebra*, Oxford Univ. Press, Oxford (2000).

12. G. E. Collins, "Quantifier elimination for real closed fields by cylindrical decomposition," in *Lect. Notes Comput. Sci.*, **33** (1975), pp. 134-183.

13. P. E. Alaev, "Existence and uniqueness of structures computable in polynomial time," *Algebra and Logic*, **55**, No. 1, 72-76 (2016).

14. P. E. Alaev, "Structures computable in polynomial time. I," *Algebra and Logic*, **55**, No. 6, 421-435 (2016).

15. A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA (1974).

16. A. Akritas, *Elements of Computer Algebra with Applications*, Wiley (1989).

17. A. Schrijver, *Theory of Linear and Integer Programming*, Wiley (1998).

18. Jianping Zhou, "On the degree of extensions generated by finitely many algebraic numbers," *J. Number Th.*, **34**, No. 2, 133-141 (1990).