P-NP problem and complexity in computer algebra

Dima Grigoriev (Lille)

CNRS

07/05/2013, St.Petersbourg

Complexity imposes the restrictions on feasibility of computations.

The main issue of the talk will be the choice of the language of the data in computations. I'll illustrate the importance of the language for efficiency by several results in complexity.

Symbolic computations

First we consider the language of symbolic computations. The input data are symbols, the computations manipulate with intermediate results treated as symbols, the output result is an expression in the input symbols.

An advantage of the symbolic approach is that its result describes the general behaviour in terms of the input. After that if necessary one can substitute in the result the numerical data from the input and obtain a numerical output.

Complexity imposes the restrictions on feasibility of computations.

The main issue of the talk will be the choice of the language of the data in computations. I'll illustrate the importance of the language for efficiency by several results in complexity.

Symbolic computations

First we consider the language of symbolic computations. The input data are symbols, the computations manipulate with intermediate results treated as symbols, the output result is an expression in the input symbols.

An advantage of the symbolic approach is that its result describes the general behaviour in terms of the input. After that if necessary one can substitute in the result the numerical data from the input and obtain a numerical output.

Complexity imposes the restrictions on feasibility of computations.

The main issue of the talk will be the choice of the language of the data in computations. I'll illustrate the importance of the language for efficiency by several results in complexity.

Symbolic computations

First we consider the language of symbolic computations. The input data are symbols, the computations manipulate with intermediate results treated as symbols, the output result is an expression in the input symbols.

An advantage of the symbolic approach is that its result describes the general behaviour in terms of the input. After that if necessary one can substitute in the result the numerical data from the input and obtain a numerical output.

Complexity imposes the restrictions on feasibility of computations.

The main issue of the talk will be the choice of the language of the data in computations. I'll illustrate the importance of the language for efficiency by several results in complexity.

Symbolic computations

First we consider the language of symbolic computations. The input data are symbols, the computations manipulate with intermediate results treated as symbols, the output result is an expression in the input symbols.

An advantage of the symbolic approach is that its result describes the general behaviour in terms of the input. After that if necessary one can substitute in the result the numerical data from the input and obtain a numerical output.

Complexity imposes the restrictions on feasibility of computations.

The main issue of the talk will be the choice of the language of the data in computations. I'll illustrate the importance of the language for efficiency by several results in complexity.

Symbolic computations

First we consider the language of symbolic computations. The input data are symbols, the computations manipulate with intermediate results treated as symbols, the output result is an expression in the input symbols.

An advantage of the symbolic approach is that its result describes the general behaviour in terms of the input. After that if necessary one can substitute in the result the numerical data from the input and obtain a numerical output.

Complexity imposes the restrictions on feasibility of computations.

The main issue of the talk will be the choice of the language of the data in computations. I'll illustrate the importance of the language for efficiency by several results in complexity.

Symbolic computations

First we consider the language of symbolic computations. The input data are symbols, the computations manipulate with intermediate results treated as symbols, the output result is an expression in the input symbols.

An advantage of the symbolic approach is that its result describes the general behaviour in terms of the input. After that if necessary one can substitute in the result the numerical data from the input and obtain a numerical output.

Polynomial complexity of symbolic linear algebra

Especially successful the symbolic approach has shown to be in algebraic computations. Algebra fits well for algorithms, it is not by chance that both words algebra and algorithm stem from the same name of Al-Horesmi (X-th century).

The basic area for algebraic (as well as differential) computations is linear algebra. It deals with matrices $A = (a_{ij})$, its entries a_{ij}, $1 \leq i \leq m$, $1 \leq j \leq n$ are treated as symbols.

The classical Gaussian elimination allows one to bring A to a canonical (trapezium) form from which one can easily yield a basis of the space of solutions of a linear system. The Gaussian elimination can be viewed as a tree-like symbolic algorithm with branchings according to vanishing certain intermediate algebraic expressions in a_{ij}.

The number of algebraic operations in Gaussian elimination is polynomial in m, n, and this number is called the *algebraic complexity*.

Polynomial complexity of symbolic linear algebra

Especially successful the symbolic approach has shown to be in algebraic computations. Algebra fits well for algorithms, it is not by chance that both words algebra and algorithm stem from the same name of Al-Horesmi (X-th century).

The basic area for algebraic (as well as differential) computations is linear algebra. It deals with matrices $A = (a_{ij})$, its entries a_{ij}, $1 \leq i \leq m$, $1 \leq j \leq n$ are treated as symbols.

The classical Gaussian elimination allows one to bring A to a canonical (trapezium) form from which one can easily yield a basis of the space of solutions of a linear system. The Gaussian elimination can be viewed as a tree-like symbolic algorithm with branchings according to vanishing certain intermediate algebraic expressions in a_{ij}.

The number of algebraic operations in Gaussian elimination is polynomial in m, n, and this number is called the *algebraic complexity*.

Polynomial complexity of symbolic linear algebra

Especially successful the symbolic approach has shown to be in algebraic computations. Algebra fits well for algorithms, it is not by chance that both words algebra and algorithm stem from the same name of Al-Horesmi (X-th century).

The basic area for algebraic (as well as differential) computations is linear algebra. It deals with matrices $A = (a_{ij})$, its entries a_{ij}, $1 \le i \le m$, $1 \le j \le n$ are treated as symbols.

The classical Gaussian elimination allows one to bring A to a canonical (trapezium) form from which one can easily yield a basis of the space of solutions of a linear system. The Gaussian elimination can be viewed as a tree-like symbolic algorithm with branchings according to vanishing certain intermediate algebraic expressions in a_{ij}.

The number of algebraic operations in Gaussian elimination is polynomial in m, n, and this number is called the *algebraic complexity*.

Polynomial complexity of symbolic linear algebra

Especially successful the symbolic approach has shown to be in algebraic computations. Algebra fits well for algorithms, it is not by chance that both words algebra and algorithm stem from the same name of Al-Horesmi (X-th century).

The basic area for algebraic (as well as differential) computations is linear algebra. It deals with matrices $A = (a_{ij})$, its entries a_{ij}, $1 \leq i \leq m$, $1 \leq j \leq n$ are treated as symbols.

The classical Gaussian elimination allows one to bring A to a canonical (trapezium) form from which one can easily yield a basis of the space of solutions of a linear system. The Gaussian elimination can be viewed as a tree-like symbolic algorithm with branchings according to vanishing certain intermediate algebraic expressions in a_{ij}.

The number of algebraic operations in Gaussian elimination is polynomial in m, n, and this number is called the *algebraic complexity*.

Polynomial complexity of symbolic linear algebra

Especially successful the symbolic approach has shown to be in algebraic computations. Algebra fits well for algorithms, it is not by chance that both words algebra and algorithm stem from the same name of Al-Horesmi (X-th century).

The basic area for algebraic (as well as differential) computations is linear algebra. It deals with matrices $A = (a_{ij})$, its entries a_{ij}, $1 \leq i \leq m$, $1 \leq j \leq n$ are treated as symbols.

The classical Gaussian elimination allows one to bring A to a canonical (trapezium) form from which one can easily yield a basis of the space of solutions of a linear system. The Gaussian elimination can be viewed as a tree-like symbolic algorithm with branchings according to vanishing certain intermediate algebraic expressions in a_{ij}.

The number of algebraic operations in Gaussian elimination is polynomial in m, n, and this number is called the *algebraic complexity*.

Polynomial complexity of symbolic linear algebra

Especially successful the symbolic approach has shown to be in algebraic computations. Algebra fits well for algorithms, it is not by chance that both words algebra and algorithm stem from the same name of Al-Horesmi (X-th century).

The basic area for algebraic (as well as differential) computations is linear algebra. It deals with matrices $A = (a_{ij})$, its entries a_{ij}, $1 \leq i \leq m$, $1 \leq j \leq n$ are treated as symbols.

The classical Gaussian elimination allows one to bring A to a canonical (trapezium) form from which one can easily yield a basis of the space of solutions of a linear system. The Gaussian elimination can be viewed as a tree-like symbolic algorithm with branchings according to vanishing certain intermediate algebraic expressions in a_{ij}.

The number of algebraic operations in Gaussian elimination is polynomial in m, n, and this number is called the *algebraic complexity*.

Polynomial complexity of symbolic linear algebra

Especially successful the symbolic approach has shown to be in algebraic computations. Algebra fits well for algorithms, it is not by chance that both words algebra and algorithm stem from the same name of Al-Horesmi (X-th century).

The basic area for algebraic (as well as differential) computations is linear algebra. It deals with matrices $A = (a_{ij})$, its entries a_{ij}, $1 \leq i \leq m$, $1 \leq j \leq n$ are treated as symbols.

The classical Gaussian elimination allows one to bring A to a canonical (trapezium) form from which one can easily yield a basis of the space of solutions of a linear system. The Gaussian elimination can be viewed as a tree-like symbolic algorithm with branchings according to vanishing certain intermediate algebraic expressions in a_{ij}.

The number of algebraic operations in Gaussian elimination is polynomial in m, n, and this number is called the *algebraic complexity*.

Polynomial complexity of symbolic linear algebra (continued)

The running time of the algorithm has to take into account the *bit complexity*, so the number of operations with bits. Therefore, it is necessary to bound the bit size of the entries of the intermediate matrices in the course of carrying out the Gaussian elimination. To this end it appears that each entry is the quotient of two minors of the input matrix. Whence we conclude that the bit complexity of the Gaussian elimination is polynomial.

Polynomial complexity is viewed as the first approximation of an algorithm to be efficient. This view is compatible with the P-NP problem.

In the period of 1969-1986 the degree of the polynomial complexity bound was improved from 1.5 in the Gaussian elimination to 1.19 due to the efforts of Strassen, Pan, Schönhage, Bini-Capovani-Lotti-Romani, Coppersmith-Winograd, and this improved algorithm is still in the paradigm of symbolic computations.

Polynomial complexity of symbolic linear algebra (continued)

The running time of the algorithm has to take into account the *bit complexity*, so the number of operations with bits. Therefore, it is necessary to bound the bit size of the entries of the intermediate matrices in the course of carrying out the Gaussian elimination. To this end it appears that each entry is the quotient of two minors of the input matrix. Whence we conclude that the bit complexity of the Gaussian elimination is polynomial.

Polynomial complexity is viewed as the first approximation of an algorithm to be efficient. This view is compatible with the P-NP problem.

In the period of 1969-1986 the degree of the polynomial complexity bound was improved from 1.5 in the Gaussian elimination to 1.19 due to the efforts of Strassen, Pan, Schönhage, Bini-Capovani-Lotti-Romani, Coppersmith-Winograd, and this improved algorithm is still in the paradigm of symbolic computations.

Polynomial complexity of symbolic linear algebra (continued)

The running time of the algorithm has to take into account the *bit complexity*, so the number of operations with bits. Therefore, it is necessary to bound the bit size of the entries of the intermediate matrices in the course of carrying out the Gaussian elimination. To this end it appears that each entry is the quotient of two minors of the input matrix. Whence we conclude that the bit complexity of the Gaussian elimination is polynomial.

Polynomial complexity is viewed as the first approximation of an algorithm to be efficient. This view is compatible with the P-NP problem.

In the period of 1969-1986 the degree of the polynomial complexity bound was improved from 1.5 in the Gaussian elimination to 1.19 due to the efforts of Strassen, Pan, Schönhage, Bini-Capovani-Lotti-Romani, Coppersmith-Winograd, and this improved algorithm is still in the paradigm of symbolic computations.

Polynomial complexity of symbolic linear algebra (continued)

The running time of the algorithm has to take into account the *bit complexity*, so the number of operations with bits. Therefore, it is necessary to bound the bit size of the entries of the intermediate matrices in the course of carrying out the Gaussian elimination. To this end it appears that each entry is the quotient of two minors of the input matrix. Whence we conclude that the bit complexity of the Gaussian elimination is polynomial.

Polynomial complexity is viewed as the first approximation of an algorithm to be efficient. This view is compatible with the P-NP problem.

In the period of 1969-1986 the degree of the polynomial complexity bound was improved from 1.5 in the Gaussian elimination to 1.19 due to the efforts of Strassen, Pan, Schönhage, Bini-Capovani-Lotti-Romani, Coppersmith-Winograd, and this improved algorithm is still in the paradigm of symbolic computations.

Polynomial complexity of symbolic linear algebra (continued)

The running time of the algorithm has to take into account the *bit complexity*, so the number of operations with bits. Therefore, it is necessary to bound the bit size of the entries of the intermediate matrices in the course of carrying out the Gaussian elimination. To this end it appears that each entry is the quotient of two minors of the input matrix. Whence we conclude that the bit complexity of the Gaussian elimination is polynomial.

Polynomial complexity is viewed as the first approximation of an algorithm to be efficient. This view is compatible with the P-NP problem.

In the period of 1969-1986 the degree of the polynomial complexity bound was improved from 1.5 in the Gaussian elimination to 1.19 due to the efforts of Strassen, Pan, Schönhage, Bini-Capovani-Lotti-Romani, Coppersmith-Winograd, and this improved algorithm is still in the paradigm of symbolic computations.

Complexity of polynomial factoring

The next challenging computational problem in algebra is polynomial factoring, so for a polynomial $f \in F[X_1, \dots, X_n]$ to find its irreducible factors $f = f_1 \cdots f_s$. It was studied by Newton, Bézout, Gauss, Kronecker. In the textbooks one can find the Kronecker's procedure for the field $F = \mathbb{Q}$ of rational numbers whose complexity is exponential. After the beginning of the development of the complexity theory a question was posed, whether one can factor polynomials within polynomial complexity?

The history of attempts to answer this question was rather long. The first step was made by D.K.Faddeev-A.I.Skopin (1959) who have designed a polynomial complexity algorithm to test whether a univariate polynomial $f \in GF(p^m)[X]$ over a finite field $GF(p^m)$ is irreducible. The algorithm was never published, and later it was rediscovered by Berlekamp (1968). Then the algorithm was modified by Rabin (1979) to factor polynomials within *probabilistic* polynomial complexity.

Complexity of polynomial factoring

The next challenging computational problem in algebra is polynomial factoring, so for a polynomial $f \in F[X_1, \dots, X_n]$ to find its irreducible factors $f = f_1 \cdots f_s$. It was studied by Newton, Bézout, Gauss, Kronecker. In the textbooks one can find the Kronecker's procedure for the field $F = \mathbb{Q}$ of rational numbers whose complexity is exponential. After the beginning of the development of the complexity theory a question was posed, whether one can factor polynomials within polynomial complexity?

The history of attempts to answer this question was rather long. The first step was made by D.K.Faddeev-A.I.Skopin (1959) who have designed a polynomial complexity algorithm to test whether a univariate polynomial $f \in GF(p^m)[X]$ over a finite field $GF(p^m)$ is irreducible. The algorithm was never published, and later it was rediscovered by Berlekamp (1968). Then the algorithm was modified by Rabin (1979) to factor polynomials within *probabilistic* polynomial complexity.

Complexity of polynomial factoring

The next challenging computational problem in algebra is polynomial factoring, so for a polynomial $f \in F[X_1, \dots, X_n]$ to find its irreducible factors $f = f_1 \cdots f_s$. It was studied by Newton, Bézout, Gauss, Kronecker. In the textbooks one can find the Kronecker's procedure for the field $F = \mathbb{Q}$ of rational numbers whose complexity is exponential. After the beginning of the development of the complexity theory a question was posed, whether one can factor polynomials within polynomial complexity?

The history of attempts to answer this question was rather long. The first step was made by D.K.Faddeev-A.I.Skopin (1959) who have designed a polynomial complexity algorithm to test whether a univariate polynomial $f \in GF(p^m)[X]$ over a finite field $GF(p^m)$ is irreducible. The algorithm was never published, and later it was rediscovered by Berlekamp (1968). Then the algorithm was modified by Rabin (1979) to factor polynomials within *probabilistic* polynomial complexity.

Complexity of polynomial factoring

The next challenging computational problem in algebra is polynomial factoring, so for a polynomial $f \in F[X_1, \dots, X_n]$ to find its irreducible factors $f = f_1 \cdots f_s$. It was studied by Newton, Bézout, Gauss, Kronecker. In the textbooks one can find the Kronecker's procedure for the field $F = \mathbb{Q}$ of rational numbers whose complexity is exponential. After the beginning of the development of the complexity theory a question was posed, whether one can factor polynomials within polynomial complexity?

The history of attempts to answer this question was rather long. The first step was made by D.K.Faddeev-A.I.Skopin (1959) who have designed a polynomial complexity algorithm to test whether a univariate polynomial $f \in GF(p^m)[X]$ over a finite field $GF(p^m)$ is irreducible. The algorithm was never published, and later it was rediscovered by Berlekamp (1968). Then the algorithm was modified by Rabin (1979) to factor polynomials within *probabilistic* polynomial complexity.

Complexity of polynomial factoring

The next challenging computational problem in algebra is polynomial factoring, so for a polynomial $f \in F[X_1, \ldots, X_n]$ to find its irreducible factors $f = f_1 \cdots f_s$. It was studied by Newton, Bézout, Gauss, Kronecker. In the textbooks one can find the Kronecker's procedure for the field $F = \mathbb{Q}$ of rational numbers whose complexity is exponential. After the beginning of the development of the complexity theory a question was posed, whether one can factor polynomials within polynomial complexity?

The history of attempts to answer this question was rather long. The first step was made by D.K.Faddeev-A.I.Skopin (1959) who have designed a polynomial complexity algorithm to test whether a univariate polynomial $f \in GF(p^m)[X]$ over a finite field $GF(p^m)$ is irreducible. The algorithm was never published, and later it was rediscovered by Berlekamp (1968). Then the algorithm was modified by Rabin (1979) to factor polynomials within *probabilistic* polynomial complexity.

Complexity of polynomial factoring

The next challenging computational problem in algebra is polynomial factoring, so for a polynomial $f \in F[X_1, \dots, X_n]$ to find its irreducible factors $f = f_1 \cdots f_s$. It was studied by Newton, Bézout, Gauss, Kronecker. In the textbooks one can find the Kronecker's procedure for the field $F = \mathbb{Q}$ of rational numbers whose complexity is exponential. After the beginning of the development of the complexity theory a question was posed, whether one can factor polynomials within polynomial complexity?

The history of attempts to answer this question was rather long. The first step was made by D.K.Faddeev-A.I.Skopin (1959) who have designed a polynomial complexity algorithm to test whether a univariate polynomial $f \in GF(p^m)[X]$ over a finite field $GF(p^m)$ is irreducible. The algorithm was never published, and later it was rediscovered by Berlekamp (1968). Then the algorithm was modified by Rabin (1979) to factor polynomials within *probabilistic* polynomial complexity.

Complexity of polynomial factoring (continued)

The question whether it is possible to factor polynomials in $GF(p^m)[X]$ within the *deterministic* polynomial complexity, remains open. Now a common conjecture is that one can do it under the assumption of the extended Riemann hypothesis.

When the field $F = \mathbb{Q}$ the situation is better. For univariate polynomials Lenstra-Lenstra-Lovasz (1982) have invented an algorithm which factors polynomials from $\mathbb{Q}[X]$ within polynomial complexity. After that Chistov-G. (1982) have designed an algorithm which factors multivariable polynomials within polynomial complexity, in particular from $\mathbb{Q}[X_1, \ldots, X_n]$ or with algebraic number coefficients $\overline{\mathbb{Q}}[X_1, \ldots, X_n]$. For finite fields $GF(p^m)[X_1, \ldots, X_n]$ our algorithm reduces (within polynomial complexity) factoring to univariate polynomials from $GF(p^m)[X]$ (discussed earlier).

These polynomial complexity algorithms involve quite sophisticated mathematics, it is also the feature of other advanced algorithm in the complexity theory.

Complexity of polynomial factoring (continued)

The question whether it is possible to factor polynomials in $GF(p^m)[X]$ within the *deterministic* polynomial complexity, remains open. Now a common conjecture is that one can do it under the assumption of the extended Riemann hypothesis.

When the field $F = \mathbb{Q}$ the situation is better. For univariate polynomials Lenstra-Lenstra-Lovasz (1982) have invented an algorithm which factors polynomials from $\mathbb{Q}[X]$ within polynomial complexity. After that Chistov-G. (1982) have designed an algorithm which factors multivariable polynomials within polynomial complexity, in particular from $\mathbb{Q}[X_1, \ldots, X_n]$ or with algebraic number coefficients $\overline{\mathbb{Q}}[X_1, \ldots, X_n]$. For finite fields $GF(p^m)[X_1, \ldots, X_n]$ our algorithm reduces (within polynomial complexity) factoring to univariate polynomials from $GF(p^m)[X]$ (discussed earlier).

These polynomial complexity algorithms involve quite sophisticated mathematics, it is also the feature of other advanced algorithm in the complexity theory.

Complexity of polynomial factoring (continued)

The question whether it is possible to factor polynomials in $GF(p^m)[X]$ within the *deterministic* polynomial complexity, remains open. Now a common conjecture is that one can do it under the assumption of the extended Riemann hypothesis.

When the field $F = \mathbb{Q}$ the situation is better. For univariate polynomials Lenstra-Lenstra-Lovasz (1982) have invented an algorithm which factors polynomials from $\mathbb{Q}[X]$ within polynomial complexity. After that Chistov-G. (1982) have designed an algorithm which factors multivariable polynomials within polynomial complexity, in particular from $\mathbb{Q}[X_1, \dots, X_n]$ or with algebraic number coefficients $\overline{\mathbb{Q}}[X_1, \dots, X_n]$. For finite fields $GF(p^m)[X_1, \dots, X_n]$ our algorithm reduces (within polynomial complexity) factoring to univariate polynomials from $GF(p^m)[X]$ (discussed earlier).

These polynomial complexity algorithms involve quite sophisticated mathematics, it is also the feature of other advanced algorithm in the complexity theory.

Complexity of polynomial factoring (continued)

The question whether it is possible to factor polynomials in $GF(p^m)[X]$ within the *deterministic* polynomial complexity, remains open. Now a common conjecture is that one can do it under the assumption of the extended Riemann hypothesis.

When the field $F = \mathbb{Q}$ the situation is better. For univariate polynomials Lenstra-Lenstra-Lovasz (1982) have invented an algorithm which factors polynomials from $\mathbb{Q}[X]$ within polynomial complexity. After that Chistov-G. (1982) have designed an algorithm which factors multivariable polynomials within polynomial complexity, in particular from $\mathbb{Q}[X_1, \ldots, X_n]$ or with algebraic number coefficients $\overline{\mathbb{Q}}[X_1, \ldots, X_n]$.

For finite fields $GF(p^m)[X_1, \ldots, X_n]$ our algorithm reduces (within polynomial complexity) factoring to univariate polynomials from $GF(p^m)[X]$ (discussed earlier).

These polynomial complexity algorithms involve quite sophisticated mathematics, it is also the feature of other advanced algorithm in the complexity theory.

Complexity of polynomial factoring (continued)

The question whether it is possible to factor polynomials in $GF(p^m)[X]$ within the *deterministic* polynomial complexity, remains open. Now a common conjecture is that one can do it under the assumption of the extended Riemann hypothesis.

When the field $F = \mathbb{Q}$ the situation is better. For univariate polynomials Lenstra-Lenstra-Lovasz (1982) have invented an algorithm which factors polynomials from $\mathbb{Q}[X]$ within polynomial complexity. After that Chistov-G. (1982) have designed an algorithm which factors multivariable polynomials within polynomial complexity, in particular from $\mathbb{Q}[X_1, \dots, X_n]$ or with algebraic number coefficients $\overline{\mathbb{Q}}[X_1, \dots, X_n]$. For finite fields $GF(p^m)[X_1, \dots, X_n]$ our algorithm reduces (within polynomial complexity) factoring to univariate polynomials from $GF(p^m)[X]$ (discussed earlier).

These polynomial complexity algorithms involve quite sophisticated mathematics, it is also the feature of other advanced algorithm in the complexity theory.

Complexity of polynomial factoring (continued)

The question whether it is possible to factor polynomials in $GF(p^m)[X]$ within the *deterministic* polynomial complexity, remains open. Now a common conjecture is that one can do it under the assumption of the extended Riemann hypothesis.

When the field $F = \mathbb{Q}$ the situation is better. For univariate polynomials Lenstra-Lenstra-Lovasz (1982) have invented an algorithm which factors polynomials from $\mathbb{Q}[X]$ within polynomial complexity. After that Chistov-G. (1982) have designed an algorithm which factors multivariable polynomials within polynomial complexity, in particular from $\mathbb{Q}[X_1, \ldots, X_n]$ or with algebraic number coefficients $\overline{\mathbb{Q}}[X_1, \ldots, X_n]$. For finite fields $GF(p^m)[X_1, \ldots, X_n]$ our algorithm reduces (within polynomial complexity) factoring to univariate polynomials from $GF(p^m)[X]$ (discussed earlier).

These polynomial complexity algorithms involve quite sophisticated mathematics, it is also the feature of other advanced algorithm in the complexity theory.

Symbolic solving systems of polynomial equations

Now we proceed to the problem of solving systems of polynomial equations
$f_i = 0,\ 1 \leq i \leq k,\ f_i \in F[X_1, \dots, X_n]$
with solutions $x = (x_1, \dots, x_n) \in F^n$.
Actually, the origin itself of algebra is due to this problem, while the historical development of algebra has left its origin quite away. With the appearance of the complexity theory this original goal is revisited.

We'll suppose that the field F is algebraically closed, for example $F = \mathbb{C}, \overline{\mathbb{Q}}$ is the field of complex or algebraic numbers. It is known that for some classes of fields the problem of solvability of a system of equations over this field is algorithmically undecidable.

Now arises a conceptual question, what does it mean to solve a system of polynomial equations?

Symbolic solving systems of polynomial equations

Now we proceed to the problem of solving systems of polynomial equations
$f_i = 0,\ 1 \leq i \leq k,\ f_i \in F[X_1, \ldots, X_n]$
with solutions $x = (x_1, \ldots, x_n) \in F^n$.
Actually, the origin itself of algebra is due to this problem, while the historical development of algebra has left its origin quite away. With the appearance of the complexity theory this original goal is revisited.

We'll suppose that the field F is algebraically closed, for example $F = \mathbb{C}, \overline{\mathbb{Q}}$ is the field of complex or algebraic numbers. It is known that for some classes of fields the problem of solvability of a system of equations over this field is algorithmically undecidable.

Now arises a conceptual question, what does it mean to solve a system of polynomial equations?

Symbolic solving systems of polynomial equations

Now we proceed to the problem of solving systems of polynomial equations
$f_i = 0,\ 1 \leq i \leq k,\ f_i \in F[X_1, \dots, X_n]$
with solutions $x = (x_1, \dots, x_n) \in F^n$.
Actually, the origin itself of algebra is due to this problem, while the historical development of algebra has left its origin quite away. With the appearance of the complexity theory this original goal is revisited.

We'll suppose that the field F is algebraically closed, for example $F = \mathbb{C}, \overline{\mathbb{Q}}$ is the field of complex or algebraic numbers. It is known that for some classes of fields the problem of solvability of a system of equations over this field is algorithmically undecidable.

Now arises a conceptual question, what does it mean to solve a system of polynomial equations?

Symbolic solving systems of polynomial equations

Now we proceed to the problem of solving systems of polynomial equations
$f_i = 0,\ 1 \le i \le k,\ f_i \in F[X_1, \dots, X_n]$
with solutions $x = (x_1, \dots, x_n) \in F^n$.
Actually, the origin itself of algebra is due to this problem, while the historical development of algebra has left its origin quite away. With the appearance of the complexity theory this original goal is revisited.

We'll suppose that the field F is algebraically closed, for example $F = \mathbb{C}, \overline{\mathbb{Q}}$ is the field of complex or algebraic numbers. It is known that for some classes of fields the problem of solvability of a system of equations over this field is algorithmically undecidable.

Now arises a conceptual question, what does it mean to solve a system of polynomial equations?

Symbolic solving systems of polynomial equations

Now we proceed to the problem of solving systems of polynomial equations
$f_i = 0,\ 1 \leq i \leq k,\ f_i \in F[X_1, \dots, X_n]$
with solutions $x = (x_1, \dots, x_n) \in F^n$.
Actually, the origin itself of algebra is due to this problem, while the historical development of algebra has left its origin quite away. With the appearance of the complexity theory this original goal is revisited.

We'll suppose that the field F is algebraically closed, for example $F = \mathbb{C}, \overline{\mathbb{Q}}$ is the field of complex or algebraic numbers. It is known that for some classes of fields the problem of solvability of a system of equations over this field is algorithmically undecidable.

Now arises a conceptual question, what does it mean to solve a system of polynomial equations?

Symbolic solving systems of polynomial equations

Now we proceed to the problem of solving systems of polynomial equations
$f_i = 0,\ 1 \leq i \leq k,\ f_i \in F[X_1, \dots, X_n]$
with solutions $x = (x_1, \dots, x_n) \in F^n$.
Actually, the origin itself of algebra is due to this problem, while the historical development of algebra has left its origin quite away. With the appearance of the complexity theory this original goal is revisited.

We'll suppose that the field F is algebraically closed, for example $F = \mathbb{C}, \overline{\mathbb{Q}}$ is the field of complex or algebraic numbers. It is known that for some classes of fields the problem of solvability of a system of equations over this field is algorithmically undecidable.

Now arises a conceptual question, what does it mean to solve a system of polynomial equations?

Gröbner bases: monomial orderings

A widely spread approach to the problem of solving systems of polynomial equations involves Gröbner bases. The latter is a fundamental notion, first perhaps, introduced by Janet (1924) in a more general setting of differential operators rather than polynomials. Afterwards, this notion was rediscovered by Ritt (1930), Hironaka (1964) and finally by Gröbner (1965) and nowdays is called after the name of the latter.

Let me briefly remind the idea of Gröbner bases. Fix a linear well ordering $\prec$ on the (integer) vectors of exponents $i_1, \dots, i_n \geq 0$ being compatible with the addition: if $a \prec b$ then $a + c \prec b + c$. Linear ordering means that any two vectors are comparable, and well ordering means that any set of vectors contains the minimal one.

For any polynomial $f \in F[X_1, \dots, X_n]$ denote by $\mathrm{lm}(f)$ its leading (with respect to the fixed ordering) monomial. Denote by $\langle f_1, \dots, f_k \rangle \subset F[X_1, \dots, X_n]$ the ideal generated by $f_1, \dots, f_k$.

Gröbner bases: monomial orderings

A widely spread approach to the problem of solving systems of polynomial equations involves Gröbner bases. The latter is a fundamental notion, first perhaps, introduced by Janet (1924) in a more general setting of differential operators rather than polynomials. Afterwards, this notion was rediscovered by Ritt (1930), Hironaka (1964) and finally by Gröbner (1965) and nowdays is called after the name of the latter.

Let me briefly remind the idea of Gröbner bases. Fix a linear well ordering $\prec$ on the (integer) vectors of exponents $i_1, \dots, i_n \geq 0$ being compatible with the addition: if $a \prec b$ then $a + c \prec b + c$. Linear ordering means that any two vectors are comparable, and well ordering means that any set of vectors contains the minimal one.

For any polynomial $f \in F[X_1, \dots, X_n]$ denote by $\mathrm{lm}(f)$ its leading (with respect to the fixed ordering) monomial. Denote by $\langle f_1, \dots, f_k \rangle \subset F[X_1, \dots, X_n]$ the ideal generated by $f_1, \dots, f_k$.

Gröbner bases: monomial orderings

A widely spread approach to the problem of solving systems of polynomial equations involves Gröbner bases. The latter is a fundamental notion, first perhaps, introduced by Janet (1924) in a more general setting of differential operators rather than polynomials. Afterwards, this notion was rediscovered by Ritt (1930), Hironaka (1964) and finally by Gröbner (1965) and nowdays is called after the name of the latter.

Let me briefly remind the idea of Gröbner bases. Fix a linear well ordering $\prec$ on the (integer) vectors of exponents $i_1, \dots, i_n \geq 0$ being compatible with the addition: if $a \prec b$ then $a + c \prec b + c$. Linear ordering means that any two vectors are comparable, and well ordering means that any set of vectors contains the minimal one.

For any polynomial $f \in F[X_1, \dots, X_n]$ denote by $\mathrm{lm}(f)$ its leading (with respect to the fixed ordering) monomial. Denote by $\langle f_1, \dots, f_k \rangle \subset F[X_1, \dots, X_n]$ the ideal generated by $f_1, \dots, f_k$.

Gröbner bases: monomial orderings

A widely spread approach to the problem of solving systems of polynomial equations involves Gröbner bases. The latter is a fundamental notion, first perhaps, introduced by Janet (1924) in a more general setting of differential operators rather than polynomials. Afterwards, this notion was rediscovered by Ritt (1930), Hironaka (1964) and finally by Gröbner (1965) and nowdays is called after the name of the latter.

Let me briefly remind the idea of Gröbner bases. Fix a linear well ordering $\prec$ on the (integer) vectors of exponents $i_1, \dots, i_n \geq 0$ being compatible with the addition: if $a \prec b$ then $a + c \prec b + c$. Linear ordering means that any two vectors are comparable, and well ordering means that any set of vectors contains the minimal one.

For any polynomial $f \in F[X_1, \dots, X_n]$ denote by $\mathrm{lm}(f)$ its leading (with respect to the fixed ordering) monomial. Denote by $\langle f_1, \dots, f_k \rangle \subset F[X_1, \dots, X_n]$ the ideal generated by $f_1, \dots, f_k$.

Gröbner bases: monomial orderings

A widely spread approach to the problem of solving systems of polynomial equations involves Gröbner bases. The latter is a fundamental notion, first perhaps, introduced by Janet (1924) in a more general setting of differential operators rather than polynomials. Afterwards, this notion was rediscovered by Ritt (1930), Hironaka (1964) and finally by Gröbner (1965) and nowdays is called after the name of the latter.

Let me briefly remind the idea of Gröbner bases. Fix a linear well ordering $\prec$ on the (integer) vectors of exponents $i_1, \dots, i_n \geq 0$ being compatible with the addition: if $a \prec b$ then $a + c \prec b + c$. Linear ordering means that any two vectors are comparable, and well ordering means that any set of vectors contains the minimal one.

For any polynomial $f \in F[X_1, \dots, X_n]$ denote by $\mathrm{lm}(f)$ its leading (with respect to the fixed ordering) monomial. Denote by $\langle f_1, \dots, f_k \rangle \subset F[X_1, \dots, X_n]$ the ideal generated by $f_1, \dots, f_k$.

Gröbner bases: monomial orderings

A widely spread approach to the problem of solving systems of polynomial equations involves Gröbner bases. The latter is a fundamental notion, first perhaps, introduced by Janet (1924) in a more general setting of differential operators rather than polynomials. Afterwards, this notion was rediscovered by Ritt (1930), Hironaka (1964) and finally by Gröbner (1965) and nowdays is called after the name of the latter.

Let me briefly remind the idea of Gröbner bases. Fix a linear well ordering $\prec$ on the (integer) vectors of exponents $i_1, \dots, i_n \geq 0$ being compatible with the addition: if $a \prec b$ then $a + c \prec b + c$. Linear ordering means that any two vectors are comparable, and well ordering means that any set of vectors contains the minimal one.

For any polynomial $f \in F[X_1, \dots, X_n]$ denote by $\mathrm{lm}(f)$ its leading (with respect to the fixed ordering) monomial. Denote by $\langle f_1, \dots, f_k \rangle \subset F[X_1, \dots, X_n]$ the ideal generated by $f_1, \dots, f_k$.

Gröbner bases: monomial orderings

A widely spread approach to the problem of solving systems of polynomial equations involves Gröbner bases. The latter is a fundamental notion, first perhaps, introduced by Janet (1924) in a more general setting of differential operators rather than polynomials. Afterwards, this notion was rediscovered by Ritt (1930), Hironaka (1964) and finally by Gröbner (1965) and nowdays is called after the name of the latter.

Let me briefly remind the idea of Gröbner bases. Fix a linear well ordering $\prec$ on the (integer) vectors of exponents $i_1, \dots, i_n \geq 0$ being compatible with the addition: if $a \prec b$ then $a + c \prec b + c$. Linear ordering means that any two vectors are comparable, and well ordering means that any set of vectors contains the minimal one.

For any polynomial $f \in F[X_1, \dots, X_n]$ denote by $\mathrm{lm}(f)$ its leading (with respect to the fixed ordering) monomial. Denote by $\langle f_1, \dots, f_k \rangle \subset F[X_1, \dots, X_n]$ the ideal generated by $f_1, \dots, f_k$.

Gröbner bases: definition and division with remainder

$g_1, \dots, g_s \in F[X_1, \dots, X_n]$ form a **Gröbner basis** if $\mathrm{lm}\langle g_1, \dots, g_s\rangle = \langle \mathrm{lm} g_1, \dots, \mathrm{lm} g_s\rangle$.

The meaning of a Gröbner basis is that it allows one to generalize the division with remainder with respect to $g_1, \dots, g_s$. Namely, if for a polynomial $f \in F[X_1, \dots, X_n]$ its leading monomial $\mathrm{lm} f \in \langle \mathrm{lm} g_1, \dots, \mathrm{lm} g_s\rangle$, i. e. $\mathrm{lm} f \in \langle \mathrm{lm} g_i\rangle$ for some $1 \le i \le s$ then one can divide with remainder $f = Q \cdot g_i + R$ where $\mathrm{lm} R \prec \mathrm{lm} f$. Otherwise, if $\mathrm{lm} f \notin \langle \mathrm{lm} g_1, \dots, \mathrm{lm} g_s\rangle$ then $f \notin \langle g_1, \dots, g_s\rangle$.

Gröbner bases approach consists in constructing a Gröbner basis of a given ideal and allows one to answer the question on solvability of a system of polynomial equations $f_i = 0$, $1 \le i \le k$. The latter is equivalent to that $1 \notin \langle f_1, \dots, f_k\rangle$ due to the Hilbert's Nullstellensatz which is equivalent in its turn to that 1 is not among the elements of the Gröbner basis of the ideal $\langle f_1, \dots, f_k\rangle$.

Gröbner bases: definition and division with remainder

$g_1, \ldots, g_s \in F[X_1, \ldots, X_n]$ form a **Gröbner basis** if $\mathrm{lm}\langle g_1, \ldots, g_s\rangle = \langle \mathrm{lm} g_1, \ldots, \mathrm{lm} g_s\rangle$.

The meaning of a Gröbner basis is that it allows one to generalize the division with remainder with respect to $g_1, \ldots, g_s$. Namely, if for a polynomial $f \in F[X_1, \ldots, X_n]$ its leading monomial $\mathrm{lm} f \in \langle \mathrm{lm} g_1, \ldots, \mathrm{lm} g_s\rangle$, i. e. $\mathrm{lm} f \in \langle \mathrm{lm} g_i\rangle$ for some $1 \le i \le s$ then one can divide with remainder $f = Q \cdot g_i + R$ where $\mathrm{lm} R \prec \mathrm{lm} f$. Otherwise, if $\mathrm{lm} f \notin \langle \mathrm{lm} g_1, \ldots, \mathrm{lm} g_s\rangle$ then $f \notin \langle g_1, \ldots, g_s\rangle$.

Gröbner bases approach consists in constructing a Gröbner basis of a given ideal and allows one to answer the question on solvability of a system of polynomial equations $f_i = 0$, $1 \le i \le k$. The latter is equivalent to that $1 \notin \langle f_1, \ldots, f_k\rangle$ due to the Hilbert's Nullstellensatz which is equivalent in its turn to that 1 is not among the elements of the Gröbner basis of the ideal $\langle f_1, \ldots, f_k\rangle$.

Gröbner bases: definition and division with remainder

$g_1, \ldots, g_s \in F[X_1, \ldots, X_n]$ form a **Gröbner basis** if $\mathrm{lm}\langle g_1, \ldots, g_s\rangle = \langle \mathrm{lm} g_1, \ldots, \mathrm{lm} g_s\rangle$.

The meaning of a Gröbner basis is that it allows one to generalize the division with remainder with respect to $g_1, \ldots, g_s$. Namely, if for a polynomial $f \in F[X_1, \ldots, X_n]$ its leading monomial $\mathrm{lm} f \in \langle \mathrm{lm} g_1, \ldots, \mathrm{lm} g_s\rangle$, i. e. $\mathrm{lm} f \in \langle \mathrm{lm} g_i\rangle$ for some $1 \le i \le s$ then one can divide with remainder $f = Q \cdot g_i + R$ where $\mathrm{lm} R \prec \mathrm{lm} f$. Otherwise, if $\mathrm{lm} f \notin \langle \mathrm{lm} g_1, \ldots, \mathrm{lm} g_s\rangle$ then $f \notin \langle g_1, \ldots, g_s\rangle$.

Gröbner bases approach consists in constructing a Gröbner basis of a given ideal and allows one to answer the question on solvability of a system of polynomial equations $f_i = 0$, $1 \le i \le k$. The latter is equivalent to that $1 \notin \langle f_1, \ldots, f_k\rangle$ due to the Hilbert's Nullstellensatz which is equivalent in its turn to that 1 is not among the elements of the Gröbner basis of the ideal $\langle f_1, \ldots, f_k\rangle$.

Gröbner bases: definition and division with remainder

$g_1, \ldots, g_s \in F[X_1, \ldots, X_n]$ form a **Gröbner basis** if $\mathrm{lm}\langle g_1, \ldots, g_s\rangle = \langle \mathrm{lm} g_1, \ldots, \mathrm{lm} g_s\rangle$.

The meaning of a Gröbner basis is that it allows one to generalize the division with remainder with respect to $g_1, \ldots, g_s$. Namely, if for a polynomial $f \in F[X_1, \ldots, X_n]$ its leading monomial $\mathrm{lm} f \in \langle \mathrm{lm} g_1, \ldots, \mathrm{lm} g_s\rangle$, i. e. $\mathrm{lm} f \in \langle \mathrm{lm} g_i\rangle$ for some $1 \leq i \leq s$ then one can divide with remainder $f = Q \cdot g_i + R$ where $\mathrm{lm} R \prec \mathrm{lm} f$. Otherwise, if $\mathrm{lm} f \notin \langle \mathrm{lm} g_1, \ldots, \mathrm{lm} g_s\rangle$ then $f \notin \langle g_1, \ldots, g_s\rangle$.

Gröbner bases approach consists in constructing a Gröbner basis of a given ideal and allows one to answer the question on solvability of a system of polynomial equations $f_i = 0$, $1 \leq i \leq k$. The latter is equivalent to that $1 \notin \langle f_1, \ldots, f_k\rangle$ due to the Hilbert's Nullstellensatz which is equivalent in its turn to that 1 is not among the elements of the Gröbner basis of the ideal $\langle f_1, \ldots, f_k\rangle$.

Gröbner bases: definition and division with remainder

$g_1, \ldots, g_s \in F[X_1, \ldots, X_n]$ form a **Gröbner basis** if $\mathrm{lm}\langle g_1, \ldots, g_s\rangle = \langle \mathrm{lm} g_1, \ldots, \mathrm{lm} g_s\rangle$.

The meaning of a Gröbner basis is that it allows one to generalize the division with remainder with respect to $g_1, \ldots, g_s$. Namely, if for a polynomial $f \in F[X_1, \ldots, X_n]$ its leading monomial $\mathrm{lm} f \in \langle \mathrm{lm} g_1, \ldots, \mathrm{lm} g_s\rangle$, i. e. $\mathrm{lm} f \in \langle \mathrm{lm} g_i\rangle$ for some $1 \leq i \leq s$ then one can divide with remainder $f = Q \cdot g_i + R$ where $\mathrm{lm} R \prec \mathrm{lm} f$. Otherwise, if $\mathrm{lm} f \notin \langle \mathrm{lm} g_1, \ldots, \mathrm{lm} g_s\rangle$ then $f \notin \langle g_1, \ldots, g_s\rangle$.

Gröbner bases approach consists in constructing a Gröbner basis of a given ideal and allows one to answer the question on solvability of a system of polynomial equations $f_i = 0$, $1 \leq i \leq k$. The latter is equivalent to that $1 \notin \langle f_1, \ldots, f_k\rangle$ due to the Hilbert's Nullstellensatz which is equivalent in its turn to that 1 is not among the elements of the Gröbner basis of the ideal $\langle f_1, \ldots, f_k\rangle$.

Gröbner bases: definition and division with remainder

$g_1, \dots, g_s \in F[X_1, \dots, X_n]$ form a **Gröbner basis** if $\mathrm{lm}\langle g_1, \dots, g_s \rangle = \langle \mathrm{lm} g_1, \dots, \mathrm{lm} g_s \rangle$.

The meaning of a Gröbner basis is that it allows one to generalize the division with remainder with respect to $g_1, \dots, g_s$. Namely, if for a polynomial $f \in F[X_1, \dots, X_n]$ its leading monomial $\mathrm{lm} f \in \langle \mathrm{lm} g_1, \dots, \mathrm{lm} g_s \rangle$, i. e. $\mathrm{lm} f \in \langle \mathrm{lm} g_i \rangle$ for some $1 \le i \le s$ then one can divide with remainder $f = Q \cdot g_i + R$ where $\mathrm{lm} R \prec \mathrm{lm} f$. Otherwise, if $\mathrm{lm} f \notin \langle \mathrm{lm} g_1, \dots, \mathrm{lm} g_s \rangle$ then $f \notin \langle g_1, \dots, g_s \rangle$.

Gröbner bases approach consists in constructing a Gröbner basis of a given ideal and allows one to answer the question on solvability of a system of polynomial equations $f_i = 0$, $1 \le i \le k$. The latter is equivalent to that $1 \notin \langle f_1, \dots, f_k \rangle$ due to the Hilbert's Nullstellensatz which is equivalent in its turn to that 1 is not among the elements of the Gröbner basis of the ideal $\langle f_1, \dots, f_k \rangle$.

Gröbner bases: definition and division with remainder

$g_1, \ldots, g_s \in F[X_1, \ldots, X_n]$ form a **Gröbner basis** if $\mathrm{lm}\langle g_1, \ldots, g_s\rangle = \langle \mathrm{lm} g_1, \ldots, \mathrm{lm} g_s\rangle$.

The meaning of a Gröbner basis is that it allows one to generalize the division with remainder with respect to $g_1, \ldots, g_s$. Namely, if for a polynomial $f \in F[X_1, \ldots, X_n]$ its leading monomial $\mathrm{lm} f \in \langle \mathrm{lm} g_1, \ldots, \mathrm{lm} g_s\rangle$, i. e. $\mathrm{lm} f \in \langle \mathrm{lm} g_i\rangle$ for some $1 \leq i \leq s$ then one can divide with remainder $f = Q \cdot g_i + R$ where $\mathrm{lm} R \prec \mathrm{lm} f$. Otherwise, if $\mathrm{lm} f \notin \langle \mathrm{lm} g_1, \ldots, \mathrm{lm} g_s\rangle$ then $f \notin \langle g_1, \ldots, g_s\rangle$.

Gröbner bases approach consists in constructing a Gröbner basis of a given ideal and allows one to answer the question on solvability of a system of polynomial equations $f_i = 0$, $1 \leq i \leq k$. The latter is equivalent to that $1 \notin \langle f_1, \ldots, f_k\rangle$ due to the Hilbert's Nullstellensatz which is equivalent in its turn to that 1 is not among the elements of the Gröbner basis of the ideal $\langle f_1, \ldots, f_k\rangle$.

Gröbner bases: definition and division with remainder

$g_1, \ldots, g_s \in F[X_1, \ldots, X_n]$ form a **Gröbner basis** if $\mathrm{lm}\langle g_1, \ldots, g_s\rangle = \langle \mathrm{lm} g_1, \ldots, \mathrm{lm} g_s\rangle$.

The meaning of a Gröbner basis is that it allows one to generalize the division with remainder with respect to $g_1, \ldots, g_s$. Namely, if for a polynomial $f \in F[X_1, \ldots, X_n]$ its leading monomial $\mathrm{lm} f \in \langle \mathrm{lm} g_1, \ldots, \mathrm{lm} g_s\rangle$, i. e. $\mathrm{lm} f \in \langle \mathrm{lm} g_i\rangle$ for some $1 \leq i \leq s$ then one can divide with remainder $f = Q \cdot g_i + R$ where $\mathrm{lm} R \prec \mathrm{lm} f$. Otherwise, if $\mathrm{lm} f \notin \langle \mathrm{lm} g_1, \ldots, \mathrm{lm} g_s\rangle$ then $f \notin \langle g_1, \ldots, g_s\rangle$.

Gröbner bases approach consists in constructing a Gröbner basis of a given ideal and allows one to answer the question on solvability of a system of polynomial equations $f_i = 0$, $1 \leq i \leq k$. The latter is equivalent to that $1 \notin \langle f_1, \ldots, f_k\rangle$ due to the Hilbert's Nullstellensatz which is equivalent in its turn to that 1 is not among the elements of the Gröbner basis of the ideal $\langle f_1, \ldots, f_k\rangle$.

Gröbner bases: applications and complexity

If a system $f_i = 0$, $1 \leq i \leq k$ has a finite number of solutions, one can read them from the Gröbner basis. Otherwise, if a system has an infinite number of solutions, it is difficult to extract one of them from the Gröbner basis. However, the Gröbner basis allows one to provide some partial information about the solutions, in particular the dimension of the variety of solutions.

Buchberger (1973) has suggested an algorithm for computing a Gröbner basis of an arbitrary polynomial ideal. There is an example due to Mair-Meyer (1982) of an ideal whose Gröbner basis has necessary double-exponential size. Thus, from the complexity point of view the Gröbner bases are not satisfiable, although in computer experiments the Buchberger's algorithm runs quite fast. This means that the worst-case examples like the one due to Mair-Meyer are not typical. On the other hand, the double-exponential complexity upper bound on Gröbner bases was established by Bayer, Giusti, Mora-Möller (1983).

Gröbner bases: applications and complexity

If a system $f_i = 0,\ 1 \leq i \leq k$ has a finite number of solutions, one can read them from the Gröbner basis. Otherwise, if a system has an infinite number of solutions, it is difficult to extract one of them from the Gröbner basis. However, the Gröbner basis allows one to provide some partial information about the solutions, in particular the dimension of the variety of solutions.

Buchberger (1973) has suggested an algorithm for computing a Gröbner basis of an arbitrary polynomial ideal. There is an example due to Mair-Meyer (1982) of an ideal whose Gröbner basis has necessary double-exponential size. Thus, from the complexity point of view the Gröbner bases are not satisfiable, although in computer experiments the Buchberger's algorithm runs quite fast. This means that the worst-case examples like the one due to Mair-Meyer are not typical. On the other hand, the double-exponential complexity upper bound on Gröbner bases was established by Bayer, Giusti, Mora-Möller (1983).

Gröbner bases: applications and complexity

If a system $f_i = 0,\ 1 \leq i \leq k$ has a finite number of solutions, one can read them from the Gröbner basis. Otherwise, if a system has an infinite number of solutions, it is difficult to extract one of them from the Gröbner basis. However, the Gröbner basis allows one to provide some partial information about the solutions, in particular the dimension of the variety of solutions.

Buchberger (1973) has suggested an algorithm for computing a Gröbner basis of an arbitrary polynomial ideal. There is an example due to Mair-Meyer (1982) of an ideal whose Gröbner basis has necessary double-exponential size. Thus, from the complexity point of view the Gröbner bases are not satisfiable, although in computer experiments the Buchberger's algorithm runs quite fast. This means that the worst-case examples like the one due to Mair-Meyer are not typical. On the other hand, the double-exponential complexity upper bound on Gröbner bases was established by Bayer, Giusti, Mora-Möller (1983).

Gröbner bases: applications and complexity

If a system $f_i = 0,\ 1 \leq i \leq k$ has a finite number of solutions, one can read them from the Gröbner basis. Otherwise, if a system has an infinite number of solutions, it is difficult to extract one of them from the Gröbner basis. However, the Gröbner basis allows one to provide some partial information about the solutions, in particular the dimension of the variety of solutions.

Buchberger (1973) has suggested an algorithm for computing a Gröbner basis of an arbitrary polynomial ideal. There is an example due to Mair-Meyer (1982) of an ideal whose Gröbner basis has necessary double-exponential size. Thus, from the complexity point of view the Gröbner bases are not satisfiable, although in computer experiments the Buchberger's algorithm runs quite fast. This means that the worst-case examples like the one due to Mair-Meyer are not typical. On the other hand, the double-exponential complexity upper bound on Gröbner bases was established by Bayer, Giusti, Mora-Möller (1983).

Gröbner bases: applications and complexity

If a system $f_i = 0,\ 1 \leq i \leq k$ has a finite number of solutions, one can read them from the Gröbner basis. Otherwise, if a system has an infinite number of solutions, it is difficult to extract one of them from the Gröbner basis. However, the Gröbner basis allows one to provide some partial information about the solutions, in particular the dimension of the variety of solutions.

Buchberger (1973) has suggested an algorithm for computing a Gröbner basis of an arbitrary polynomial ideal. There is an example due to Mair-Meyer (1982) of an ideal whose Gröbner basis has necessary double-exponential size. Thus, from the complexity point of view the Gröbner bases are not satisfiable, although in computer experiments the Buchberger's algorithm runs quite fast. This means that the worst-case examples like the one due to Mair-Meyer are not typical. On the other hand, the double-exponential complexity upper bound on Gröbner bases was established by Bayer, Giusti, Mora-Möller (1983).

Gröbner bases: applications and complexity

If a system $f_i = 0,\ 1 \le i \le k$ has a finite number of solutions, one can read them from the Gröbner basis. Otherwise, if a system has an infinite number of solutions, it is difficult to extract one of them from the Gröbner basis. However, the Gröbner basis allows one to provide some partial information about the solutions, in particular the dimension of the variety of solutions.

Buchberger (1973) has suggested an algorithm for computing a Gröbner basis of an arbitrary polynomial ideal. There is an example due to Mair-Meyer (1982) of an ideal whose Gröbner basis has necessary double-exponential size. Thus, from the complexity point of view the Gröbner bases are not satisfiable, although in computer experiments the Buchberger's algorithm runs quite fast. This means that the worst-case examples like the one due to Mair-Meyer are not typical. On the other hand, the double-exponential complexity upper bound on Gröbner bases was established by Bayer, Giusti, Mora-Möller (1983).

Gröbner bases: applications and complexity

If a system $f_i = 0,\ 1 \leq i \leq k$ has a finite number of solutions, one can read them from the Gröbner basis. Otherwise, if a system has an infinite number of solutions, it is difficult to extract one of them from the Gröbner basis. However, the Gröbner basis allows one to provide some partial information about the solutions, in particular the dimension of the variety of solutions.

Buchberger (1973) has suggested an algorithm for computing a Gröbner basis of an arbitrary polynomial ideal. There is an example due to Mair-Meyer (1982) of an ideal whose Gröbner basis has necessary double-exponential size. Thus, from the complexity point of view the Gröbner bases are not satisfiable, although in computer experiments the Buchberger's algorithm runs quite fast. This means that the worst-case examples like the one due to Mair-Meyer are not typical. On the other hand, the double-exponential complexity upper bound on Gröbner bases was established by Bayer, Giusti, Mora-Möller (1983).

Choice of algorithmic language: algebra or geometry?

Hilbert's Nullstellensatz provides a duality between the variety of solutions of a system of polynomial equations (so to say, geometry), and on the other hand, the radical of the ideal generated by the system (so to say, algebra). Gröbner bases fit well for manipulations with ideals (in particular, a Gröbner basis allows one to test membership to the ideal), but Gröbner bases do not help much to answer geometric questions on the variety of solutions.

That is why Chistov-G.(1983) have introduced a different (geometric) language to solve systems of polynomial equations.

Choice of algorithmic language: algebra or geometry?

Hilbert's Nullstellensatz provides a duality between the variety of solutions of a system of polynomial equations (so to say, geometry), and on the other hand, the radical of the ideal generated by the system (so to say, algebra). Gröbner bases fit well for manipulations with ideals (in particular, a Gröbner basis allows one to test membership to the ideal), but Gröbner bases do not help much to answer geometric questions on the variety of solutions.

That is why Chistov-G.(1983) have introduced a different (geometric) language to solve systems of polynomial equations.

Choice of algorithmic language: algebra or geometry?

Hilbert's Nullstellensatz provides a duality between the variety of solutions of a system of polynomial equations (so to say, geometry), and on the other hand, the radical of the ideal generated by the system (so to say, algebra). Gröbner bases fit well for manipulations with ideals (in particular, a Gröbner basis allows one to test membership to the ideal), but Gröbner bases do not help much to answer geometric questions on the variety of solutions.

That is why Chistov-G.(1983) have introduced a different (geometric) language to solve systems of polynomial equations.

Geometric language for solving systems of polynomial equations

Denote by
$V := V(f_1, \dots, f_k) = \{x := (x_1, \dots, x_n) \in F^n : f_i(x) = 0, 1 \le i \le k\}$
the variety of solutions of a system. There is a unique decomposition $V = \cup_j V_j$ of V into its *irreducible components*. Our algorithm finds all V_j.

If $k = 1$ then the variety $V(f_1)$ is a hypersurface (so, has the codimension 1) in F^n, its irreducible components $V(f_1) = \cup_{1 \le j \le s} V_j$ are also hypersurfaces being in a bijective correspondence with the irreducible factors of the polynomial $f_1 = \prod_{1 \le j \le s} g_j$, i. e. $V_j = V(g_j)$. Thus, the polynomial factoring problem is a particular case of the one of solving systems of polynomial equations in our setting.

How to give algorithmically an irreducible component V_j of V?

Geometric language for solving systems of polynomial equations

Denote by
$V := V(f_1, \dots, f_k) = \{x := (x_1, \dots, x_n) \in F^n \,:\, f_i(x) = 0,\, 1 \le i \le k\}$
the variety of solutions of a system. There is a unique decomposition $V = \cup_j V_j$ of V into its *irreducible components*. Our algorithm finds all V_j.

If $k = 1$ then the variety $V(f_1)$ is a hypersurface (so, has the codimension 1) in F^n, its irreducible components $V(f_1) = \cup_{1 \le j \le s} V_j$ are also hypersurfaces being in a bijective correspondence with the irreducible factors of the polynomial $f_1 = \prod_{1 \le j \le s} g_j$, i. e. $V_j = V(g_j)$. Thus, the polynomial factoring problem is a particular case of the one of solving systems of polynomial equations in our setting.

How to give algorithmically an irreducible component V_j of V?

Geometric language for solving systems of polynomial equations

Denote by
$V := V(f_1, \dots, f_k) = \{x := (x_1, \dots, x_n) \in F^n \,:\, f_i(x) = 0,\ 1 \le i \le k\}$
the variety of solutions of a system. There is a unique decomposition $V = \cup_j V_j$ of V into its *irreducible components*. Our algorithm finds all V_j.

If $k = 1$ then the variety $V(f_1)$ is a hypersurface (so, has the codimension 1) in F^n, its irreducible components $V(f_1) = \cup_{1 \le j \le s} V_j$ are also hypersurfaces being in a bijective correspondence with the irreducible factors of the polynomial $f_1 = \prod_{1 \le j \le s} g_j$, i. e. $V_j = V(g_j)$.
Thus, the polynomial factoring problem is a particular case of the one of solving systems of polynomial equations in our setting.

How to give algorithmically an irreducible component V_j of V?

Geometric language for solving systems of polynomial equations

Denote by
$V := V(f_1, \dots, f_k) = \{x := (x_1, \dots, x_n) \in F^n \ : \ f_i(x) = 0, 1 \leq i \leq k\}$
the variety of solutions of a system. There is a unique decomposition $V = \cup_j V_j$ of V into its *irreducible components*. Our algorithm finds all V_j.

If $k = 1$ then the variety $V(f_1)$ is a hypersurface (so, has the codimension 1) in F^n, its irreducible components $V(f_1) = \cup_{1 \leq j \leq s} V_j$ are also hypersurfaces being in a bijective correspondence with the irreducible factors of the polynomial $f_1 = \prod_{1 \leq j \leq s} g_j$, i. e. $V_j = V(g_j)$. Thus, the polynomial factoring problem is a particular case of the one of solving systems of polynomial equations in our setting.

How to give algorithmically an irreducible component V_j of V?

Geometric language for solving systems of polynomial equations

Denote by
$V := V(f_1, \dots, f_k) = \{x := (x_1, \dots, x_n) \in F^n \, : \, f_i(x) = 0, \, 1 \leq i \leq k\}$
the variety of solutions of a system. There is a unique decomposition $V = \cup_j V_j$ of V into its *irreducible components*. Our algorithm finds all V_j.

If $k = 1$ then the variety $V(f_1)$ is a hypersurface (so, has the codimension 1) in F^n, its irreducible components $V(f_1) = \cup_{1 \leq j \leq s} V_j$ are also hypersurfaces being in a bijective correspondence with the irreducible factors of the polynomial $f_1 = \prod_{1 \leq j \leq s} g_j$, i. e. $V_j = V(g_j)$. Thus, the polynomial factoring problem is a particular case of the one of solving systems of polynomial equations in our setting.

How to give algorithmically an irreducible component V_j of V?

Algorithmic representation of an irreducible variety

We represent an irreducible component V_j in two dual ways. The first one is by a system of polynomials $h_1, \dots, h_l \in F[X_1, \dots, X_n]$ whose variety of zeroes $V_j = V(h_1, \dots, h_l)$ coincides with V_j.

The second way is by means of a *generic point* of V_j. Let the dimension $\dim(V_j) = m$. Our algorithm yields a transcendental basis $X_{i_1}, \dots, X_{i_m}$ of V_j among $X_1, \dots, X_n$ and constructs explicitly an isomorphism

$$F(V_j) \sim F(X_{i_1}, \dots, X_{i_m})[\theta]$$

of the *field* $F(V_j)$ *of rational functions on* V_j. Herein $\theta = \alpha_1 \cdot X_1 + \cdots + \alpha_n \cdot X_n$ is a linear combination of $X_1, \dots, X_n$. For the *primitive element* θ a minimal polynomial $\phi \in F(X_{i_1}, \dots, X_{i_m})[Z]$ is produced where $\phi(\theta) = 0$. The algorithm gives the isomorphism with the help of rational functions

$$X_t = p_t(X_{i_1}, \dots, X_{i_m}, \theta)/q(X_{i_1}, \dots, X_{i_m}, \theta),\ 1 \le t \le n.$$

Algorithmic representation of an irreducible variety

We represent an irreducible component V_j in two dual ways. The first one is by a system of polynomials $h_1, \dots, h_l \in F[X_1, \dots, X_n]$ whose variety of zeroes $V_j = V(h_1, \dots, h_l)$ coincides with V_j.

The second way is by means of a *generic point* of V_j. Let the dimension $\dim(V_j) = m$. Our algorithm yields a transcendental basis $X_{i_1}, \dots, X_{i_m}$ of V_j among $X_1, \dots, X_n$ and constructs explicitly an isomorphism

$$F(V_j) \sim F(X_{i_1}, \dots, X_{i_m})[\theta]$$

of the *field $F(V_j)$ of rational functions on V_j*. Herein $\theta = \alpha_1 \cdot X_1 + \cdots + \alpha_n \cdot X_n$ is a linear combination of $X_1, \dots, X_n$. For the *primitive element* θ a minimal polynomial $\phi \in F(X_{i_1}, \dots, X_{i_m})[Z]$ is produced where $\phi(\theta) = 0$. The algorithm gives the isomorphism with the help of rational functions

$$X_t = p_t(X_{i_1}, \dots, X_{i_m}, \theta)/q(X_{i_1}, \dots, X_{i_m}, \theta),\ 1 \le t \le n.$$

Algorithmic representation of an irreducible variety

We represent an irreducible component V_j in two dual ways. The first one is by a system of polynomials $h_1, \dots, h_l \in F[X_1, \dots, X_n]$ whose variety of zeroes $V_j = V(h_1, \dots, h_l)$ coincides with V_j.

The second way is by means of a *generic point* of V_j. Let the dimension $\dim(V_j) = m$. Our algorithm yields a transcendental basis $X_{i_1}, \dots, X_{i_m}$ of V_j among $X_1, \dots, X_n$ and constructs explicitly an isomorphism

$$F(V_j) \sim F(X_{i_1}, \dots, X_{i_m})[\theta]$$

of the *field $F(V_j)$ of rational functions on V_j*. Herein $\theta = \alpha_1 \cdot X_1 + \cdots + \alpha_n \cdot X_n$ is a linear combination of $X_1, \dots, X_n$. For the *primitive element* θ a minimal polynomial $\phi \in F(X_{i_1}, \dots, X_{i_m})[Z]$ is produced where $\phi(\theta) = 0$. The algorithm gives the isomorphism with the help of rational functions

$$X_t = p_t(X_{i_1}, \dots, X_{i_m}, \theta)/q(X_{i_1}, \dots, X_{i_m}, \theta),\ 1 \le t \le n.$$

Algorithmic representation of an irreducible variety

We represent an irreducible component V_j in two dual ways. The first one is by a system of polynomials $h_1, \dots, h_l \in F[X_1, \dots, X_n]$ whose variety of zeroes $V_j = V(h_1, \dots, h_l)$ coincides with V_j.

The second way is by means of a *generic point* of V_j. Let the dimension $\dim(V_j) = m$. Our algorithm yields a transcendental basis $X_{i_1}, \dots, X_{i_m}$ of V_j among $X_1, \dots, X_n$ and constructs explicitly an isomorphism

$$F(V_j) \sim F(X_{i_1}, \dots, X_{i_m})[\theta]$$

of the *field* $F(V_j)$ *of rational functions on* V_j. Herein $\theta = \alpha_1 \cdot X_1 + \cdots + \alpha_n \cdot X_n$ is a linear combination of $X_1, \dots, X_n$. For the *primitive element* θ a minimal polynomial $\phi \in F(X_{i_1}, \dots, X_{i_m})[Z]$ is produced where $\phi(\theta) = 0$. The algorithm gives the isomorphism with the help of rational functions

$$X_t = p_t(X_{i_1}, \dots, X_{i_m}, \theta)/q(X_{i_1}, \dots, X_{i_m}, \theta),\ 1 \le t \le n.$$

Algorithmic representation of an irreducible variety

We represent an irreducible component V_j in two dual ways. The first one is by a system of polynomials $h_1, \dots, h_l \in F[X_1, \dots, X_n]$ whose variety of zeroes $V_j = V(h_1, \dots, h_l)$ coincides with V_j.

The second way is by means of a *generic point* of V_j. Let the dimension $\dim(V_j) = m$. Our algorithm yields a transcendental basis $X_{i_1}, \dots, X_{i_m}$ of V_j among $X_1, \dots, X_n$ and constructs explicitly an isomorphism

$$F(V_j) \sim F(X_{i_1}, \dots, X_{i_m})[\theta]$$

of the *field $F(V_j)$ of rational functions on V_j*. Herein $\theta = \alpha_1 \cdot X_1 + \cdots + \alpha_n \cdot X_n$ is a linear combination of $X_1, \dots, X_n$. For the *primitive element* θ a minimal polynomial $\phi \in F(X_{i_1}, \dots, X_{i_m})[Z]$ is produced where $\phi(\theta) = 0$. The algorithm gives the isomorphism with the help of rational functions

$$X_t = p_t(X_{i_1}, \dots, X_{i_m}, \theta)/q(X_{i_1}, \dots, X_{i_m}, \theta),\ 1 \le t \le n.$$

Algorithmic representation of an irreducible variety

We represent an irreducible component V_j in two dual ways. The first one is by a system of polynomials $h_1, \ldots, h_l \in F[X_1, \ldots, X_n]$ whose variety of zeroes $V_j = V(h_1, \ldots, h_l)$ coincides with V_j.

The second way is by means of a *generic point* of V_j. Let the dimension $\dim(V_j) = m$. Our algorithm yields a transcendental basis $X_{i_1}, \ldots, X_{i_m}$ of V_j among $X_1, \ldots, X_n$ and constructs explicitly an isomorphism

$$F(V_j) \sim F(X_{i_1}, \ldots, X_{i_m})[\theta]$$

of the *field $F(V_j)$ of rational functions on V_j*. Herein $\theta = \alpha_1 \cdot X_1 + \cdots + \alpha_n \cdot X_n$ is a linear combination of $X_1, \ldots, X_n$. For the *primitive element* θ a minimal polynomial $\phi \in F(X_{i_1}, \ldots, X_{i_m})[Z]$ is produced where $\phi(\theta) = 0$. The algorithm gives the isomorphism with the help of rational functions

$$X_t = p_t(X_{i_1}, \ldots, X_{i_m}, \theta)/q(X_{i_1}, \ldots, X_{i_m}, \theta),\ 1 \leq t \leq n.$$

Algorithmic representation of an irreducible variety

We represent an irreducible component V_j in two dual ways. The first one is by a system of polynomials $h_1, \dots, h_l \in F[X_1, \dots, X_n]$ whose variety of zeroes $V_j = V(h_1, \dots, h_l)$ coincides with V_j.

The second way is by means of a *generic point* of V_j. Let the dimension $\dim(V_j) = m$. Our algorithm yields a transcendental basis $X_{i_1}, \dots, X_{i_m}$ of V_j among $X_1, \dots, X_n$ and constructs explicitly an isomorphism

$$F(V_j) \sim F(X_{i_1}, \dots, X_{i_m})[\theta]$$

of the *field $F(V_j)$ of rational functions on V_j*. Herein $\theta = \alpha_1 \cdot X_1 + \cdots + \alpha_n \cdot X_n$ is a linear combination of $X_1, \dots, X_n$. For the *primitive element* θ a minimal polynomial $\phi \in F(X_{i_1}, \dots, X_{i_m})[Z]$ is produced where $\phi(\theta) = 0$. The algorithm gives the isomorphism with the help of rational functions

$$X_t = p_t(X_{i_1}, \dots, X_{i_m}, \theta)/q(X_{i_1}, \dots, X_{i_m}, \theta),\ 1 \le t \le n.$$

Algorithmic representation of an irreducible variety

We represent an irreducible component V_j in two dual ways. The first one is by a system of polynomials $h_1, \dots, h_l \in F[X_1, \dots, X_n]$ whose variety of zeroes $V_j = V(h_1, \dots, h_l)$ coincides with V_j.

The second way is by means of a *generic point* of V_j. Let the dimension $\dim(V_j) = m$. Our algorithm yields a transcendental basis $X_{i_1}, \dots, X_{i_m}$ of V_j among $X_1, \dots, X_n$ and constructs explicitly an isomorphism

$$F(V_j) \sim F(X_{i_1}, \dots, X_{i_m})[\theta]$$

of the *field $F(V_j)$ of rational functions on V_j*. Herein $\theta = \alpha_1 \cdot X_1 + \cdots + \alpha_n \cdot X_n$ is a linear combination of $X_1, \dots, X_n$. For the *primitive element* θ a minimal polynomial $\phi \in F(X_{i_1}, \dots, X_{i_m})[Z]$ is produced where $\phi(\theta) = 0$. The algorithm gives the isomorphism with the help of rational functions

$$X_t = p_t(X_{i_1}, \dots, X_{i_m}, \theta)/q(X_{i_1}, \dots, X_{i_m}, \theta),\ 1 \le t \le n.$$

Algorithmic representation of an irreducible variety

We represent an irreducible component V_j in two dual ways. The first one is by a system of polynomials $h_1, \ldots, h_l \in F[X_1, \ldots, X_n]$ whose variety of zeroes $V_j = V(h_1, \ldots, h_l)$ coincides with V_j.

The second way is by means of a *generic point* of V_j. Let the dimension $\dim(V_j) = m$. Our algorithm yields a transcendental basis $X_{i_1}, \ldots, X_{i_m}$ of V_j among $X_1, \ldots, X_n$ and constructs explicitly an isomorphism

$$F(V_j) \sim F(X_{i_1}, \ldots, X_{i_m})[\theta]$$

of the *field $F(V_j)$ of rational functions on V_j*. Herein $\theta = \alpha_1 \cdot X_1 + \cdots + \alpha_n \cdot X_n$ is a linear combination of $X_1, \ldots, X_n$. For the *primitive element* θ a minimal polynomial $\phi \in F(X_{i_1}, \ldots, X_{i_m})[Z]$ is produced where $\phi(\theta) = 0$. The algorithm gives the isomorphism with the help of rational functions

$$X_t = p_t(X_{i_1}, \ldots, X_{i_m}, \theta)/q(X_{i_1}, \ldots, X_{i_m}, \theta),\ 1 \le t \le n.$$

Algorithmic representation of an irreducible variety

We represent an irreducible component V_j in two dual ways. The first one is by a system of polynomials $h_1, \dots, h_l \in F[X_1, \dots, X_n]$ whose variety of zeroes $V_j = V(h_1, \dots, h_l)$ coincides with V_j.

The second way is by means of a *generic point* of V_j. Let the dimension $\dim(V_j) = m$. Our algorithm yields a transcendental basis $X_{i_1}, \dots, X_{i_m}$ of V_j among $X_1, \dots, X_n$ and constructs explicitly an isomorphism

$$F(V_j) \sim F(X_{i_1}, \dots, X_{i_m})[\theta]$$

of the *field $F(V_j)$ of rational functions on V_j*. Herein $\theta = \alpha_1 \cdot X_1 + \cdots + \alpha_n \cdot X_n$ is a linear combination of $X_1, \dots, X_n$. For the *primitive element* θ a minimal polynomial $\phi \in F(X_{i_1}, \dots, X_{i_m})[Z]$ is produced where $\phi(\theta) = 0$. The algorithm gives the isomorphism with the help of rational functions

$$X_t = p_t(X_{i_1}, \dots, X_{i_m}, \theta)/q(X_{i_1}, \dots, X_{i_m}, \theta),\ 1 \leq t \leq n.$$

Complexity of solving systems of polynomial equations

The complexity of the described generic point is exponential which is much less than the mentioned double-exponential complexity bound on Gröbner bases. One cannot expect an essentially better bound since the problem of solvability of polynomial equations is NP-hard. Moreover, the exponential bound is close to sharp if we want to find irreducible components even in case of a finite number of solutions (rather than just to answer the question on solvability of a system).

The algorithm constructs the irreducible components V_j recursively, and in the course of recursion both representations: by a system of equations for V_j and by its generic point are crucial and their duality is exploited. In fact, the achieved improvement of the complexity bound is mainly due to the right choice of the language of representation of an irreducible variety in two dual ways. The duality means that the generic point allows one to produce points of the variety (informally speaking, builds the variety from inside), while the equations provide the restrictions on the variety (informally speaking, from outside).

Complexity of solving systems of polynomial equations

The complexity of the described generic point is exponential which is much less than the mentioned double-exponential complexity bound on Gröbner bases. One cannot expect an essentially better bound since the problem of solvability of polynomial equations is NP-hard. Moreover, the exponential bound is close to sharp if we want to find irreducible components even in case of a finite number of solutions (rather than just to answer the question on solvability of a system).

The algorithm constructs the irreducible components V_j recursively, and in the course of recursion both representations: by a system of equations for V_j and by its generic point are crucial and their duality is exploited. In fact, the achieved improvement of the complexity bound is mainly due to the right choice of the language of representation of an irreducible variety in two dual ways. The duality means that the generic point allows one to produce points of the variety (informally speaking, builds the variety from inside), while the equations provide the restrictions on the variety (informally speaking, from outside).

Complexity of solving systems of polynomial equations

The complexity of the described generic point is exponential which is much less than the mentioned double-exponential complexity bound on Gröbner bases. One cannot expect an essentially better bound since the problem of solvability of polynomial equations is NP-hard. Moreover, the exponential bound is close to sharp if we want to find irreducible components even in case of a finite number of solutions (rather than just to answer the question on solvability of a system).

The algorithm constructs the irreducible components V_j recursively, and in the course of recursion both representations: by a system of equations for V_j and by its generic point are crucial and their duality is exploited. In fact, the achieved improvement of the complexity bound is mainly due to the right choice of the language of representation of an irreducible variety in two dual ways. The duality means that the generic point allows one to produce points of the variety (informally speaking, builds the variety from inside), while the equations provide the restrictions on the variety (informally speaking, from outside).

Complexity of solving systems of polynomial equations

The complexity of the described generic point is exponential which is much less than the mentioned double-exponential complexity bound on Gröbner bases. One cannot expect an essentially better bound since the problem of solvability of polynomial equations is NP-hard. Moreover, the exponential bound is close to sharp if we want to find irreducible components even in case of a finite number of solutions (rather than just to answer the question on solvability of a system).

The algorithm constructs the irreducible components V_j recursively, and in the course of recursion both representations: by a system of equations for V_j and by its generic point are crucial and their duality is exploited. In fact, the achieved improvement of the complexity bound is mainly due to the right choice of the language of representation of an irreducible variety in two dual ways. The duality means that the generic point allows one to produce points of the variety (informally speaking, builds the variety from inside), while the equations provide the restrictions on the variety (informally speaking, from outside).

Complexity of solving systems of polynomial equations

The complexity of the described generic point is exponential which is much less than the mentioned double-exponential complexity bound on Gröbner bases. One cannot expect an essentially better bound since the problem of solvability of polynomial equations is NP-hard. Moreover, the exponential bound is close to sharp if we want to find irreducible components even in case of a finite number of solutions (rather than just to answer the question on solvability of a system).

The algorithm constructs the irreducible components V_j recursively, and in the course of recursion both representations: by a system of equations for V_j and by its generic point are crucial and their duality is exploited. In fact, the achieved improvement of the complexity bound is mainly due to the right choice of the language of representation of an irreducible variety in two dual ways. The duality means that the generic point allows one to produce points of the variety (informally speaking, builds the variety from inside), while the equations provide the restrictions on the variety (informally speaking, from outside).

Complexity of solving systems of polynomial equations

The complexity of the described generic point is exponential which is much less than the mentioned double-exponential complexity bound on Gröbner bases. One cannot expect an essentially better bound since the problem of solvability of polynomial equations is NP-hard. Moreover, the exponential bound is close to sharp if we want to find irreducible components even in case of a finite number of solutions (rather than just to answer the question on solvability of a system).

The algorithm constructs the irreducible components V_j recursively, and in the course of recursion both representations: by a system of equations for V_j and by its generic point are crucial and their duality is exploited. In fact, the achieved improvement of the complexity bound is mainly due to the right choice of the language of representation of an irreducible variety in two dual ways. The duality means that the generic point allows one to produce points of the variety (informally speaking, builds the variety from inside), while the equations provide the restrictions on the variety (informally speaking, from outside).

Complexity of solving systems of polynomial equations

The complexity of the described generic point is exponential which is much less than the mentioned double-exponential complexity bound on Gröbner bases. One cannot expect an essentially better bound since the problem of solvability of polynomial equations is NP-hard. Moreover, the exponential bound is close to sharp if we want to find irreducible components even in case of a finite number of solutions (rather than just to answer the question on solvability of a system).

The algorithm constructs the irreducible components V_j recursively, and in the course of recursion both representations: by a system of equations for V_j and by its generic point are crucial and their duality is exploited. In fact, the achieved improvement of the complexity bound is mainly due to the right choice of the language of representation of an irreducible variety in two dual ways. The duality means that the generic point allows one to produce points of the variety (informally speaking, builds the variety from inside), while the equations provide the restrictions on the variety (informally speaking, from outside).

Complexity of solving systems of polynomial equations

The complexity of the described generic point is exponential which is much less than the mentioned double-exponential complexity bound on Gröbner bases. One cannot expect an essentially better bound since the problem of solvability of polynomial equations is NP-hard. Moreover, the exponential bound is close to sharp if we want to find irreducible components even in case of a finite number of solutions (rather than just to answer the question on solvability of a system).

The algorithm constructs the irreducible components V_j recursively, and in the course of recursion both representations: by a system of equations for V_j and by its generic point are crucial and their duality is exploited. In fact, the achieved improvement of the complexity bound is mainly due to the right choice of the language of representation of an irreducible variety in two dual ways. The duality means that the generic point allows one to produce points of the variety (informally speaking, builds the variety from inside), while the equations provide the restrictions on the variety (informally speaking, from outside).

Complexity of solving systems of polynomial equations

The complexity of the described generic point is exponential which is much less than the mentioned double-exponential complexity bound on Gröbner bases. One cannot expect an essentially better bound since the problem of solvability of polynomial equations is NP-hard. Moreover, the exponential bound is close to sharp if we want to find irreducible components even in case of a finite number of solutions (rather than just to answer the question on solvability of a system).

The algorithm constructs the irreducible components V_j recursively, and in the course of recursion both representations: by a system of equations for V_j and by its generic point are crucial and their duality is exploited. In fact, the achieved improvement of the complexity bound is mainly due to the right choice of the language of representation of an irreducible variety in two dual ways. The duality means that the generic point allows one to produce points of the variety (informally speaking, builds the variety from inside), while the equations provide the restrictions on the variety (informally speaking, from outside).

Producing solutions of a system of equation

When $\dim V_j = 0$, so V_j is just a point, the generic point outputs V_j explicitly. When $\dim V_j > 0$, so V_j is infinite, the generic point allows one to produce as many points as one wishes. Also the generic point exhibits the dimension of V_j.

Involving generic points one can test whether a variety is a subvariety of another one.

Producing solutions of a system of equation

When $\dim V_j = 0$, so V_j is just a point, the generic point outputs V_j explicitly. When $\dim V_j > 0$, so V_j is infinite, the generic point allows one to produce as many points as one wishes. Also the generic point exhibits the dimension of V_j.

Involving generic points one can test whether a variety is a subvariety of another one.

Producing solutions of a system of equation

When $\dim V_j = 0$, so V_j is just a point, the generic point outputs V_j explicitly. When $\dim V_j > 0$, so V_j is infinite, the generic point allows one to produce as many points as one wishes. Also the generic point exhibits the dimension of V_j.

Involving generic points one can test whether a variety is a subvariety of another one.

Producing solutions of a system of equation

When $\dim V_j = 0$, so V_j is just a point, the generic point outputs V_j explicitly. When $\dim V_j > 0$, so V_j is infinite, the generic point allows one to produce as many points as one wishes. Also the generic point exhibits the dimension of V_j.

Involving generic points one can test whether a variety is a subvariety of another one.

P-NP problem and solving systems of polynomial equations

Consider the following system of $n+1$ quadratic equations in n variables

$$X_i^2 = X_i,\ 1 \le i \le n, \quad c_1 \cdot X_1 + \cdots + c_n \cdot X_n = c$$

called the **KNAPSACK Problem**

P=NP is equivalent to that there is an algorithm with polynomial complexity to test whether the KNAPSACK Problem has a solution.

P-NP problem and solving systems of polynomial equations

Consider the following system of $n+1$ quadratic equations in n variables

$$X_i^2 = X_i,\ 1 \le i \le n, \quad c_1 \cdot X_1 + \cdots + c_n \cdot X_n = c$$

called the **KNAPSACK Problem**

P=NP is equivalent to that there is an algorithm with polynomial complexity to test whether the KNAPSACK Problem has a solution.

P-NP problem and solving systems of polynomial equations

Consider the following system of $n + 1$ quadratic equations in n variables

$$X_i^2 = X_i,\ 1 \leq i \leq n, \quad c_1 \cdot X_1 + \cdots + c_n \cdot X_n = c$$

called the **KNAPSACK Problem**

P=NP is equivalent to that there is an algorithm with polynomial complexity to test whether the KNAPSACK Problem has a solution.

Complexity of quantifier elimination in the first-order theory of algebraically closed fields

The problem of solving systems of polynomial equations is a particular case of the one of quantifier elimination. Namely, let a formula

$$\exists X_{11} \cdots \exists X_{1n_1} \forall X_{21} \cdots \forall X_{2n_2} \cdots \exists X_{a1} \cdots \exists X_{an_a} Q$$

be given where Q is a quantifier-free formula with atomic subformulas of the type $f = 0$ for polynomials $f \in F[X_{11} \ldots, X_{1n_1}, X_{21}, \ldots, X_{2n_2}, \ldots, X_{a1}, \ldots, X_{an_a}, X_1, \ldots, X_n]$. The problem is to find an equivalent quantifier-free formula with atomic subformulas of the type $g = 0$ for polynomials $g \in F[X_1, \ldots, X_n]$. Such a quantifier-free formula exists due to Tarski-Seidenberg theorem (1930). The complexity of the latter theorem is enormous. Heintz (1982) has designed a better algorithm for quantifier elimination with the double-exponential complexity. Chistov-G.(1984) have suggested a quantifier elimination method with a further improvement of the complexity.

Complexity of quantifier elimination in the first-order theory of algebraically closed fields

The problem of solving systems of polynomial equations is a particular case of the one of quantifier elimination. Namely, let a formula

$$\exists X_{11} \cdots \exists X_{1n_1} \forall X_{21} \cdots \forall X_{2n_2} \cdots \exists X_{a1} \cdots \exists X_{an_a} Q$$

be given where Q is a quantifier-free formula with atomic subformulas of the type $f = 0$ for polynomials $f \in F[X_{11} \ldots, X_{1n_1}, X_{21}, \ldots, X_{2n_2}, \ldots, X_{a1}, \ldots, X_{an_a}, X_1, \ldots, X_n]$.

The problem is to find an equivalent quantifier-free formula with atomic subformulas of the type $g = 0$ for polynomials $g \in F[X_1, \ldots, X_n]$. Such a quantifier-free formula exists due to Tarski-Seidenberg theorem (1930). The complexity of the latter theorem is enormous. Heintz (1982) has designed a better algorithm for quantifier elimination with the double-exponential complexity. Chistov-G.(1984) have suggested a quantifier elimination method with a further improvement of the complexity.

Complexity of quantifier elimination in the first-order theory of algebraically closed fields

The problem of solving systems of polynomial equations is a particular case of the one of quantifier elimination. Namely, let a formula

$$\exists X_{11} \cdots \exists X_{1n_1} \forall X_{21} \cdots \forall X_{2n_2} \cdots \exists X_{a1} \cdots \exists X_{an_a} Q$$

be given where Q is a quantifier-free formula with atomic subformulas of the type $f = 0$ for polynomials
$f \in F[X_{11} \dots, X_{1n_1}, X_{21}, \dots, X_{2n_2}, \dots, X_{a1}, \dots, X_{an_a}, X_1, \dots, X_n]$.

The problem is to find an equivalent quantifier-free formula with atomic subformulas of the type $g = 0$ for polynomials $g \in F[X_1, \dots, X_n]$. Such a quantifier-free formula exists due to Tarski-Seidenberg theorem (1930). The complexity of the latter theorem is enormous. Heintz (1982) has designed a better algorithm for quantifier elimination with the double-exponential complexity. Chistov-G.(1984) have suggested a quantifier elimination method with a further improvement of the complexity.

Complexity of quantifier elimination in the first-order theory of algebraically closed fields

The problem of solving systems of polynomial equations is a particular case of the one of quantifier elimination. Namely, let a formula

$$\exists X_{11} \cdots \exists X_{1n_1} \forall X_{21} \cdots \forall X_{2n_2} \cdots \exists X_{a1} \cdots \exists X_{an_a} Q$$

be given where Q is a quantifier-free formula with atomic subformulas of the type $f = 0$ for polynomials
$f \in F[X_{11} \dots, X_{1n_1}, X_{21}, \dots, X_{2n_2}, \dots, X_{a1}, \dots, X_{an_a}, X_1, \dots, X_n]$.
The problem is to find an equivalent quantifier-free formula with atomic subformulas of the type $g = 0$ for polynomials $g \in F[X_1, \dots, X_n]$. Such a quantifier-free formula exists due to Tarski-Seidenberg theorem (1930). The complexity of the latter theorem is enormous. Heintz (1982) has designed a better algorithm for quantifier elimination with the double-exponential complexity. Chistov-G.(1984) have suggested a quantifier elimination method with a further improvement of the complexity.

Complexity of quantifier elimination in the first-order theory of algebraically closed fields

The problem of solving systems of polynomial equations is a particular case of the one of quantifier elimination. Namely, let a formula

$$\exists X_{11} \cdots \exists X_{1n_1} \forall X_{21} \cdots \forall X_{2n_2} \cdots \exists X_{a1} \cdots \exists X_{an_a} Q$$

be given where Q is a quantifier-free formula with atomic subformulas of the type $f = 0$ for polynomials
$f \in F[X_{11} \ldots, X_{1n_1}, X_{21}, \ldots, X_{2n_2}, \ldots, X_{a1}, \ldots, X_{an_a}, X_1, \ldots, X_n]$.

The problem is to find an equivalent quantifier-free formula with atomic subformulas of the type $g = 0$ for polynomials $g \in F[X_1, \ldots, X_n]$. Such a quantifier-free formula exists due to Tarski-Seidenberg theorem (1930). The complexity of the latter theorem is enormous. Heintz (1982) has designed a better algorithm for quantifier elimination with the double-exponential complexity. Chistov-G.(1984) have suggested a quantifier elimination method with a further improvement of the complexity.

Complexity of quantifier elimination in the first-order theory of algebraically closed fields

The problem of solving systems of polynomial equations is a particular case of the one of quantifier elimination. Namely, let a formula

$$\exists X_{11} \cdots \exists X_{1n_1} \forall X_{21} \cdots \forall X_{2n_2} \cdots \exists X_{a1} \cdots \exists X_{an_a} Q$$

be given where Q is a quantifier-free formula with atomic subformulas of the type $f = 0$ for polynomials
$f \in F[X_{11} \ldots, X_{1n_1}, X_{21}, \ldots, X_{2n_2}, \ldots, X_{a1}, \ldots, X_{an_a}, X_1, \ldots, X_n]$.

The problem is to find an equivalent quantifier-free formula with atomic subformulas of the type $g = 0$ for polynomials $g \in F[X_1, \ldots, X_n]$. Such a quantifier-free formula exists due to Tarski-Seidenberg theorem (1930). The complexity of the latter theorem is enormous. Heintz (1982) has designed a better algorithm for quantifier elimination with the double-exponential complexity. Chistov-G.(1984) have suggested a quantifier elimination method with a further improvement of the complexity.

Complexity of quantifier elimination in the first-order theory of algebraically closed fields

The problem of solving systems of polynomial equations is a particular case of the one of quantifier elimination. Namely, let a formula

$$\exists X_{11} \cdots \exists X_{1n_1} \forall X_{21} \cdots \forall X_{2n_2} \cdots \exists X_{a1} \cdots \exists X_{an_a} Q$$

be given where Q is a quantifier-free formula with atomic subformulas of the type $f = 0$ for polynomials
$f \in F[X_{11} \dots, X_{1n_1}, X_{21}, \dots, X_{2n_2}, \dots, X_{a1}, \dots, X_{an_a}, X_1, \dots, X_n]$.
The problem is to find an equivalent quantifier-free formula with atomic subformulas of the type $g = 0$ for polynomials $g \in F[X_1, \dots, X_n]$. Such a quantifier-free formula exists due to Tarski-Seidenberg theorem (1930). The complexity of the latter theorem is enormous. Heintz (1982) has designed a better algorithm for quantifier elimination with the double-exponential complexity. Chistov-G.(1984) have suggested a quantifier elimination method with a further improvement of the complexity.

Eliminating a single quantifier block: projecting a variety

It suffices to eliminate a single existential quantifier block in a formula $\exists Y_1 \cdots \exists Y_{n_1} Q$ where a quantifier-free formula Q contains atomic subformulas of the type $f = 0$ for polynomials $f \in F[Y_1, \dots, Y_{n_1}, X_1, \dots, X_n]$. In a different language the formula determines a projection W of the set $V(Q) \subset F^{n_1+n}$ of the points satisfying Q, in the space F^n with the coordinates $X_1, \dots, X_n$.

The general idea of the elimination is a "parametrizing" of the algorithm solving systems of polynomial equations. So, the algorithm treats Q as a system of polynomial equations and inequalities in the variables $Y_1, \dots, Y_{n_1}$ with parameters $X_1, \dots, X_n$. Applying to Q the algorithm solving systems of equations leads to several branchings according to whether certain polynomials in $X_1, \dots, X_n$ vanish. Thus, we get a tree-like algebraic algorithm in the variables $X_1, \dots, X_n$. Each leaf L of this tree provides some algebraic conditions in $X_1, \dots, X_n$ which determine a set $U_L \subset F^n$ being pairwise disjoint for different leaves. The required projection W is the union of appropriate sets U_L.

Eliminating a single quantifier block: projecting a variety

It suffices to eliminate a single existential quantifier block in a formula $\exists Y_1 \cdots \exists Y_{n_1} Q$ where a quantifier-free formula Q contains atomic subformulas of the type $f = 0$ for polynomials $f \in F[Y_1, \dots, Y_{n_1}, X_1, \dots, X_n]$. In a different language the formula determines a projection W of the set $V(Q) \subset F^{n_1+n}$ of the points satisfying Q, in the space F^n with the coordinates $X_1, \dots, X_n$.

The general idea of the elimination is a "parametrizing" of the algorithm solving systems of polynomial equations. So, the algorithm treats Q as a system of polynomial equations and inequalities in the variables $Y_1, \dots, Y_{n_1}$ with parameters $X_1, \dots, X_n$. Applying to Q the algorithm solving systems of equations leads to several branchings according to whether certain polynomials in $X_1, \dots, X_n$ vanish. Thus, we get a tree-like algebraic algorithm in the variables $X_1, \dots, X_n$. Each leaf L of this tree provides some algebraic conditions in $X_1, \dots, X_n$ which determine a set $U_L \subset F^n$ being pairwise disjoint for different leaves. The required projection W is the union of appropriate sets U_L.

Eliminating a single quantifier block: projecting a variety

It suffices to eliminate a single existential quantifier block in a formula $\exists Y_1 \cdots \exists Y_{n_1} Q$ where a quantifier-free formula Q contains atomic subformulas of the type $f = 0$ for polynomials $f \in F[Y_1, \ldots, Y_{n_1}, X_1, \ldots, X_n]$. In a different language the formula determines a projection W of the set $V(Q) \subset F^{n_1+n}$ of the points satisfying Q, in the space F^n with the coordinates $X_1, \ldots, X_n$.

The general idea of the elimination is a "parametrizing" of the algorithm solving systems of polynomial equations. So, the algorithm treats Q as a system of polynomial equations and inequalities in the variables $Y_1, \ldots, Y_{n_1}$ with parameters $X_1, \ldots, X_n$. Applying to Q the algorithm solving systems of equations leads to several branchings according to whether certain polynomials in $X_1, \ldots, X_n$ vanish. Thus, we get a tree-like algebraic algorithm in the variables $X_1, \ldots, X_n$. Each leaf L of this tree provides some algebraic conditions in $X_1, \ldots, X_n$ which determine a set $U_L \subset F^n$ being pairwise disjoint for different leaves. The required projection W is the union of appropriate sets U_L.

Eliminating a single quantifier block: projecting a variety

It suffices to eliminate a single existential quantifier block in a formula $\exists Y_1 \cdots \exists Y_{n_1} Q$ where a quantifier-free formula Q contains atomic subformulas of the type $f = 0$ for polynomials $f \in F[Y_1, \ldots, Y_{n_1}, X_1, \ldots, X_n]$. In a different language the formula determines a projection W of the set $V(Q) \subset F^{n_1+n}$ of the points satisfying Q, in the space F^n with the coordinates $X_1, \ldots, X_n$.

The general idea of the elimination is a "parametrizing" of the algorithm solving systems of polynomial equations. So, the algorithm treats Q as a system of polynomial equations and inequalities in the variables $Y_1, \ldots, Y_{n_1}$ with parameters $X_1, \ldots, X_n$. Applying to Q the algorithm solving systems of equations leads to several branchings according to whether certain polynomials in $X_1, \ldots, X_n$ vanish. Thus, we get a tree-like algebraic algorithm in the variables $X_1, \ldots, X_n$. Each leaf L of this tree provides some algebraic conditions in $X_1, \ldots, X_n$ which determine a set $U_L \subset F^n$ being pairwise disjoint for different leaves. The required projection W is the union of appropriate sets U_L.

Eliminating a single quantifier block: projecting a variety

It suffices to eliminate a single existential quantifier block in a formula $\exists Y_1 \cdots \exists Y_{n_1} Q$ where a quantifier-free formula Q contains atomic subformulas of the type $f = 0$ for polynomials $f \in F[Y_1, \dots, Y_{n_1}, X_1, \dots, X_n]$. In a different language the formula determines a projection W of the set $V(Q) \subset F^{n_1+n}$ of the points satisfying Q, in the space F^n with the coordinates $X_1, \dots, X_n$.

The general idea of the elimination is a "parametrizing" of the algorithm solving systems of polynomial equations. So, the algorithm treats Q as a system of polynomial equations and inequalities in the variables $Y_1, \dots, Y_{n_1}$ with parameters $X_1, \dots, X_n$. Applying to Q the algorithm solving systems of equations leads to several branchings according to whether certain polynomials in $X_1, \dots, X_n$ vanish. Thus, we get a tree-like algebraic algorithm in the variables $X_1, \dots, X_n$. Each leaf L of this tree provides some algebraic conditions in $X_1, \dots, X_n$ which determine a set $U_L \subset F^n$ being pairwise disjoint for different leaves. The required projection W is the union of appropriate sets U_L.

Eliminating a single quantifier block: projecting a variety

It suffices to eliminate a single existential quantifier block in a formula $\exists Y_1 \cdots \exists Y_{n_1} Q$ where a quantifier-free formula Q contains atomic subformulas of the type $f = 0$ for polynomials $f \in F[Y_1, \dots, Y_{n_1}, X_1, \dots, X_n]$. In a different language the formula determines a projection W of the set $V(Q) \subset F^{n_1+n}$ of the points satisfying Q, in the space F^n with the coordinates $X_1, \dots, X_n$.

The general idea of the elimination is a "parametrizing" of the algorithm solving systems of polynomial equations. So, the algorithm treats Q as a system of polynomial equations and inequalities in the variables $Y_1, \dots, Y_{n_1}$ with parameters $X_1, \dots, X_n$. Applying to Q the algorithm solving systems of equations leads to several branchings according to whether certain polynomials in $X_1, \dots, X_n$ vanish. Thus, we get a tree-like algebraic algorithm in the variables $X_1, \dots, X_n$. Each leaf L of this tree provides some algebraic conditions in $X_1, \dots, X_n$ which determine a set $U_L \subset F^n$ being pairwise disjoint for different leaves.

The required projection W is the union of appropriate sets U_L

Eliminating a single quantifier block: projecting a variety

It suffices to eliminate a single existential quantifier block in a formula $\exists Y_1 \cdots \exists Y_{n_1} Q$ where a quantifier-free formula Q contains atomic subformulas of the type $f = 0$ for polynomials $f \in F[Y_1, \dots, Y_{n_1}, X_1, \dots, X_n]$. In a different language the formula determines a projection W of the set $V(Q) \subset F^{n_1+n}$ of the points satisfying Q, in the space F^n with the coordinates $X_1, \dots, X_n$.

The general idea of the elimination is a "parametrizing" of the algorithm solving systems of polynomial equations. So, the algorithm treats Q as a system of polynomial equations and inequalities in the variables $Y_1, \dots, Y_{n_1}$ with parameters $X_1, \dots, X_n$. Applying to Q the algorithm solving systems of equations leads to several branchings according to whether certain polynomials in $X_1, \dots, X_n$ vanish. Thus, we get a tree-like algebraic algorithm in the variables $X_1, \dots, X_n$. Each leaf L of this tree provides some algebraic conditions in $X_1, \dots, X_n$ which determine a set $U_L \subset F^n$ being pairwise disjoint for different leaves. The required projection W is the union of appropriate sets U_L.

Complexity of quantifier elimination

The complexity of this quantifier elimination algorithm is exponential for a fixed number a of quantifier alternations and depends double-exponentially on a, so being sharp in accordance with the lower bound due to Davenport-Heintz (1986).

Complexity of quantifier elimination

The complexity of this quantifier elimination algorithm is exponential for a fixed number a of quantifier alternations and depends double-exponentially on a, so being sharp in accordance with the lower bound due to Davenport-Heintz (1986).

Complexity of quantifier elimination

The complexity of this quantifier elimination algorithm is exponential for a fixed number a of quantifier alternations and depends double-exponentially on a, so being sharp in accordance with the lower bound due to Davenport-Heintz (1986).

Solving systems of polynomial inequalities over the reals

Now consider polynomials $f_1, \ldots, f_k \in \mathbb{R}[X_1, \ldots, X_n]$ with real coefficients, and we are looking for solutions of a system of polynomial inequalities $f_i \geq 0$, $1 \leq i \leq k$. Solutions are real algebraic vectors, so we need to specify how the algorithm describes a real algebraic number. While studying (complex) algebraic numbers $b \in \overline{\mathbb{Q}}$ it suffices to indicate a minimal polynomial $h \in \mathbb{Q}[Z]$ such that $h(b) = 0$: all the roots of h form a conjugacy class and all the conjugate roots are equivalent.

When in addition, $b \in \mathbb{R} \cap \overline{\mathbb{Q}}$ is a real algebraic number, the algorithm specifies this particular root of h by means of indicating a real interval which contains this unique root b of h.

The algorithm tests whether a system $f_i \geq 0$, $1 \leq i \leq k$ has a real solution, and if yes then outputs one such solution. The coordinates of this solution are real algebraic numbers given by the algorithm with the help of intervals as described.

Solving systems of polynomial inequalities over the reals

Now consider polynomials $f_1, \dots, f_k \in \mathbb{R}[X_1, \dots, X_n]$ with real coefficients, and we are looking for solutions of a system of polynomial inequalities $f_i \geq 0$, $1 \leq i \leq k$. Solutions are real algebraic vectors, so we need to specify how the algorithm describes a real algebraic number. While studying (complex) algebraic numbers $b \in \overline{\mathbb{Q}}$ it suffices to indicate a minimal polynomial $h \in \mathbb{Q}[Z]$ such that $h(b) = 0$: all the roots of h form a conjugacy class and all the conjugate roots are equivalent.

When in addition, $b \in \mathbb{R} \cap \overline{\mathbb{Q}}$ is a real algebraic number, the algorithm specifies this particular root of h by means of indicating a real interval which contains this unique root b of h.

The algorithm tests whether a system $f_i \geq 0$, $1 \leq i \leq k$ has a real solution, and if yes then outputs one such solution. The coordinates of this solution are real algebraic numbers given by the algorithm with the help of intervals as described.

Solving systems of polynomial inequalities over the reals

Now consider polynomials $f_1, \ldots, f_k \in \mathbb{R}[X_1, \ldots, X_n]$ with real coefficients, and we are looking for solutions of a system of polynomial inequalities $f_i \geq 0$, $1 \leq i \leq k$. Solutions are real algebraic vectors, so we need to specify how the algorithm describes a real algebraic number. While studying (complex) algebraic numbers $b \in \overline{\mathbb{Q}}$ it suffices to indicate a minimal polynomial $h \in \mathbb{Q}[Z]$ such that $h(b) = 0$: all the roots of h form a conjugacy class and all the conjugate roots are equivalent.

When in addition, $b \in \mathbb{R} \cap \overline{\mathbb{Q}}$ is a real algebraic number, the algorithm specifies this particular root of h by means of indicating a real interval which contains this unique root b of h.

The algorithm tests whether a system $f_i \geq 0$, $1 \leq i \leq k$ has a real solution, and if yes then outputs one such solution. The coordinates of this solution are real algebraic numbers given by the algorithm with the help of intervals as described.

Solving systems of polynomial inequalities over the reals

Now consider polynomials $f_1, \ldots, f_k \in \mathbb{R}[X_1, \ldots, X_n]$ with real coefficients, and we are looking for solutions of a system of polynomial inequalities $f_i \geq 0$, $1 \leq i \leq k$. Solutions are real algebraic vectors, so we need to specify how the algorithm describes a real algebraic number. While studying (complex) algebraic numbers $b \in \overline{\mathbb{Q}}$ it suffices to indicate a minimal polynomial $h \in \mathbb{Q}[Z]$ such that $h(b) = 0$: all the roots of h form a conjugacy class and all the conjugate roots are equivalent.

When in addition, $b \in \mathbb{R} \cap \overline{\mathbb{Q}}$ is a real algebraic number, the algorithm specifies this particular root of h by means of indicating a real interval which contains this unique root b of h.

The algorithm tests whether a system $f_i \geq 0$, $1 \leq i \leq k$ has a real solution, and if yes then outputs one such solution. The coordinates of this solution are real algebraic numbers given by the algorithm with the help of intervals as described.

Solving systems of polynomial inequalities over the reals

Now consider polynomials $f_1, \dots, f_k \in \mathbb{R}[X_1, \dots, X_n]$ with real coefficients, and we are looking for solutions of a system of polynomial inequalities $f_i \geq 0$, $1 \leq i \leq k$. Solutions are real algebraic vectors, so we need to specify how the algorithm describes a real algebraic number. While studying (complex) algebraic numbers $b \in \overline{\mathbb{Q}}$ it suffices to indicate a minimal polynomial $h \in \mathbb{Q}[Z]$ such that $h(b) = 0$: all the roots of h form a conjugacy class and all the conjugate roots are equivalent.

When in addition, $b \in \mathbb{R} \cap \overline{\mathbb{Q}}$ is a real algebraic number, the algorithm specifies this particular root of h by means of indicating a real interval which contains this unique root b of h.

The algorithm tests whether a system $f_i \geq 0$, $1 \leq i \leq k$ has a real solution, and if yes then outputs one such solution. The coordinates of this solution are real algebraic numbers given by the algorithm with the help of intervals as described.

Solving systems of polynomial inequalities over the reals

Now consider polynomials $f_1, \dots, f_k \in \mathbb{R}[X_1, \dots, X_n]$ with real coefficients, and we are looking for solutions of a system of polynomial inequalities $f_i \geq 0$, $1 \leq i \leq k$. Solutions are real algebraic vectors, so we need to specify how the algorithm describes a real algebraic number. While studying (complex) algebraic numbers $b \in \overline{\mathbb{Q}}$ it suffices to indicate a minimal polynomial $h \in \mathbb{Q}[Z]$ such that $h(b) = 0$: all the roots of h form a conjugacy class and all the conjugate roots are equivalent.

When in addition, $b \in \mathbb{R} \cap \overline{\mathbb{Q}}$ is a real algebraic number, the algorithm specifies this particular root of h by means of indicating a real interval which contains this unique root b of h.

The algorithm tests whether a system $f_i \geq 0$, $1 \leq i \leq k$ has a real solution, and if yes then outputs one such solution. The coordinates of this solution are real algebraic numbers given by the algorithm with the help of intervals as described.

Solving systems of polynomial inequalities over the reals

Now consider polynomials $f_1, \dots, f_k \in \mathbb{R}[X_1, \dots, X_n]$ with real coefficients, and we are looking for solutions of a system of polynomial inequalities $f_i \geq 0$, $1 \leq i \leq k$. Solutions are real algebraic vectors, so we need to specify how the algorithm describes a real algebraic number. While studying (complex) algebraic numbers $b \in \overline{\mathbb{Q}}$ it suffices to indicate a minimal polynomial $h \in \mathbb{Q}[Z]$ such that $h(b) = 0$: all the roots of h form a conjugacy class and all the conjugate roots are equivalent.

When in addition, $b \in \mathbb{R} \cap \overline{\mathbb{Q}}$ is a real algebraic number, the algorithm specifies this particular root of h by means of indicating a real interval which contains this unique root b of h.

The algorithm tests whether a system $f_i \geq 0$, $1 \leq i \leq k$ has a real solution, and if yes then outputs one such solution. The coordinates of this solution are real algebraic numbers given by the algorithm with the help of intervals as described.

Complexity of solving systems of polynomial inequalities

The complexity of this algorithm is exponential (G.-Vorobjov (1984)), and again one cannot expect much better bound because the problem of solvability of systems of polynomial inequalities is NP-hard.

Moreover, solvability of a system of just two inequalities, one being a cubic and another linear, is NP-hard. On the contrary, G.-Pasechnik (2004) have designed an algorithm which solves a system of *quadratic* inequalities $f_i \geq 0, 1 \leq i \leq k$ within the complexity polynomial for any fixed k.

Complexity of solving systems of polynomial inequalities

The complexity of this algorithm is exponential (G.-Vorobjov (1984)), and again one cannot expect much better bound because the problem of solvability of systems of polynomial inequalities is NP-hard.

Moreover, solvability of a system of just two inequalities, one being a cubic and another linear, is NP-hard. On the contrary, G.-Pasechnik (2004) have designed an algorithm which solves a system of *quadratic* inequalities $f_i \geq 0, 1 \leq i \leq k$ within the complexity polynomial for any fixed k.

Complexity of solving systems of polynomial inequalities

The complexity of this algorithm is exponential (G.-Vorobjov (1984)), and again one cannot expect much better bound because the problem of solvability of systems of polynomial inequalities is NP-hard.

Moreover, solvability of a system of just two inequalities, one being a cubic and another linear, is NP-hard. On the contrary, G.-Pasechnik (2004) have designed an algorithm which solves a system of *quadratic* inequalities $f_i \geq 0$, $1 \leq i \leq k$ within the complexity polynomial for any fixed k.

Complexity of solving systems of polynomial inequalities

The complexity of this algorithm is exponential (G.-Vorobjov (1984)), and again one cannot expect much better bound because the problem of solvability of systems of polynomial inequalities is NP-hard.

Moreover, solvability of a system of just two inequalities, one being a cubic and another linear, is NP-hard. On the contrary, G.-Pasechnik (2004) have designed an algorithm which solves a system of *quadratic* inequalities $f_i \geq 0,\ 1 \leq i \leq k$ within the complexity polynomial for any fixed k.

Infinitesimals in symbolic computations

One of the important tools in the algorithm solving systems of polynomial inequalities is explicit using infinitesimals. To illustrate, consider a particular problem of verifying existence and finding (provided it does exist) a real zero $x \in \mathbb{R}^n$ of a polynomial $f \in \mathbb{R}[X_1, \dots, X_n]$. When f is *non-singular*, i. e. the system $f = \frac{\partial f}{\partial X_1} = \cdots = \frac{\partial f}{\partial X_n} = 0$ has no *complex* zeroes one can verify existence and find a real zero of f by means of reduction to the complex case (so-called, the *critical points method*). Now let f be singular.

Introduce an **infinitesimal** ϵ. Formally, consider an ordered field $\mathbb{R}(\epsilon)$ with the ordering $0 < \epsilon < a$ for any $0 < a \in \mathbb{R}$ and its *real closure* $\widetilde{\mathbb{R}(\epsilon)}$. Then the polynomial $f^2 - \epsilon \in \widetilde{\mathbb{R}(\epsilon)}[X_1, \dots, X_n]$ is non-singular, and one can verify existence and find a zero $y \in (\widetilde{\mathbb{R}(\epsilon)})^n$ of $f^2 - \epsilon$. Here we exploit the Tarski transfer principle for real closed fields. Then, informally, the algorithm substitutes 0 instead of ϵ in y, the resulting $y(0) \in \mathbb{R}^n$ is a real zero of f.

Infinitesimals in symbolic computations

One of the important tools in the algorithm solving systems of polynomial inequalities is explicit using infinitesimals. To illustrate, consider a particular problem of verifying existence and finding (provided it does exist) a real zero $x \in \mathbb{R}^n$ of a polynomial $f \in \mathbb{R}[X_1, \dots, X_n]$. When f is *non-singular*, i. e. the system $f = \frac{\partial f}{\partial X_1} = \cdots = \frac{\partial f}{\partial X_n} = 0$ has no *complex* zeroes one can verify existence and find a real zero of f by means of reduction to the complex case (so-called, the *critical points method*). Now let f be singular.

Introduce an **infinitesimal** ϵ. Formally, consider an ordered field $\mathbb{R}(\epsilon)$ with the ordering $0 < \epsilon < a$ for any $0 < a \in \mathbb{R}$ and its *real closure* $\widetilde{\mathbb{R}(\epsilon)}$. Then the polynomial $f^2 - \epsilon \in \widetilde{\mathbb{R}(\epsilon)}[X_1, \dots, X_n]$ is non-singular, and one can verify existence and find a zero $y \in (\widetilde{\mathbb{R}(\epsilon)})^n$ of $f^2 - \epsilon$. Here we exploit the Tarski transfer principle for real closed fields. Then, informally, the algorithm substitutes 0 instead of ϵ in y, the resulting $y(0) \in \mathbb{R}^n$ is a real zero of f.

Infinitesimals in symbolic computations

One of the important tools in the algorithm solving systems of polynomial inequalities is explicit using infinitesimals. To illustrate, consider a particular problem of verifying existence and finding (provided it does exist) a real zero $x \in \mathbb{R}^n$ of a polynomial $f \in \mathbb{R}[X_1, \dots, X_n]$. When f is *non-singular*, i. e. the system $f = \frac{\partial f}{\partial X_1} = \cdots = \frac{\partial f}{\partial X_n} = 0$ has no *complex* zeroes one can verify existence and find a real zero of f by means of reduction to the complex case (so-called, the *critical points method*). Now let f be singular.

Introduce an **infinitesimal** ϵ. Formally, consider an ordered field $\mathbb{R}(\epsilon)$ with the ordering $0 < \epsilon < a$ for any $0 < a \in \mathbb{R}$ and its *real closure* $\widetilde{\mathbb{R}(\epsilon)}$. Then the polynomial $f^2 - \epsilon \in \widetilde{\mathbb{R}(\epsilon)}[X_1, \dots, X_n]$ is non-singular, and one can verify existence and find a zero $y \in (\widetilde{\mathbb{R}(\epsilon)})^n$ of $f^2 - \epsilon$. Here we exploit the Tarski transfer principle for real closed fields. Then, informally, the algorithm substitutes 0 instead of ϵ in y, the resulting $y(0) \in \mathbb{R}^n$ is a real zero of f.

Infinitesimals in symbolic computations

One of the important tools in the algorithm solving systems of polynomial inequalities is explicit using infinitesimals. To illustrate, consider a particular problem of verifying existence and finding (provided it does exist) a real zero $x \in \mathbb{R}^n$ of a polynomial $f \in \mathbb{R}[X_1, \dots, X_n]$. When f is *non-singular*, i. e. the system $f = \frac{\partial f}{\partial X_1} = \cdots = \frac{\partial f}{\partial X_n} = 0$ has no *complex* zeroes one can verify existence and find a real zero of f by means of reduction to the complex case (so-called, the *critical points method*). Now let f be singular.

Introduce an **infinitesimal** ϵ. Formally, consider an ordered field $\mathbb{R}(\epsilon)$ with the ordering $0 < \epsilon < a$ for any $0 < a \in \mathbb{R}$ and its *real closure* $\widetilde{\mathbb{R}(\epsilon)}$. Then the polynomial $f^2 - \epsilon \in \widetilde{\mathbb{R}(\epsilon)}[X_1, \dots, X_n]$ is non-singular, and one can verify existence and find a zero $y \in (\widetilde{\mathbb{R}(\epsilon)})^n$ of $f^2 - \epsilon$. Here we exploit the Tarski transfer principle for real closed fields. Then, informally, the algorithm substitutes 0 instead of ϵ in y, the resulting $y(0) \in \mathbb{R}^n$ is a real zero of f.

Infinitesimals in symbolic computations

One of the important tools in the algorithm solving systems of polynomial inequalities is explicit using infinitesimals. To illustrate, consider a particular problem of verifying existence and finding (provided it does exist) a real zero $x \in \mathbb{R}^n$ of a polynomial $f \in \mathbb{R}[X_1, \dots, X_n]$. When f is *non-singular*, i. e. the system $f = \frac{\partial f}{\partial X_1} = \cdots = \frac{\partial f}{\partial X_n} = 0$ has no *complex* zeroes one can verify existence and find a real zero of f by means of reduction to the complex case (so-called, the *critical points method*). Now let f be singular.

Introduce an **infinitesimal** ϵ. Formally, consider an ordered field $\mathbb{R}(\epsilon)$ with the ordering $0 < \epsilon < a$ for any $0 < a \in \mathbb{R}$ and its *real closure* $\widetilde{\mathbb{R}(\epsilon)}$. Then the polynomial $f^2 - \epsilon \in \widetilde{\mathbb{R}(\epsilon)}[X_1, \dots, X_n]$ is non-singular, and one can verify existence and find a zero $y \in (\widetilde{\mathbb{R}(\epsilon)})^n$ of $f^2 - \epsilon$. Here we exploit the Tarski transfer principle for real closed fields. Then, informally, the algorithm substitutes 0 instead of ϵ in y, the resulting $y(0) \in \mathbb{R}^n$ is a real zero of f.

Infinitesimals in symbolic computations

One of the important tools in the algorithm solving systems of polynomial inequalities is explicit using infinitesimals. To illustrate, consider a particular problem of verifying existence and finding (provided it does exist) a real zero $x \in \mathbb{R}^n$ of a polynomial $f \in \mathbb{R}[X_1, \dots, X_n]$. When f is *non-singular*, i. e. the system $f = \frac{\partial f}{\partial X_1} = \cdots = \frac{\partial f}{\partial X_n} = 0$ has no *complex* zeroes one can verify existence and find a real zero of f by means of reduction to the complex case (so-called, the *critical points method*). Now let f be singular.

Introduce an **infinitesimal** ϵ. Formally, consider an ordered field $\mathbb{R}(\epsilon)$ with the ordering $0 < \epsilon < a$ for any $0 < a \in \mathbb{R}$ and its *real closure* $\widetilde{\mathbb{R}(\epsilon)}$.

Then the polynomial $f^2 - \epsilon \in \widetilde{\mathbb{R}(\epsilon)}[X_1, \dots, X_n]$ is non-singular, and one can verify existence and find a zero $y \in (\widetilde{\mathbb{R}(\epsilon)})^n$ of $f^2 - \epsilon$. Here we exploit the Tarski transfer principle for real closed fields. Then, informally, the algorithm substitutes 0 instead of ϵ in y, the resulting $y(0) \in \mathbb{R}^n$ is a real zero of f.

Infinitesimals in symbolic computations

One of the important tools in the algorithm solving systems of polynomial inequalities is explicit using infinitesimals. To illustrate, consider a particular problem of verifying existence and finding (provided it does exist) a real zero $x \in \mathbb{R}^n$ of a polynomial $f \in \mathbb{R}[X_1, \dots, X_n]$. When f is *non-singular*, i. e. the system $f = \frac{\partial f}{\partial X_1} = \cdots = \frac{\partial f}{\partial X_n} = 0$ has no *complex* zeroes one can verify existence and find a real zero of f by means of reduction to the complex case (so-called, the *critical points method*). Now let f be singular.

Introduce an **infinitesimal** ϵ. Formally, consider an ordered field $\mathbb{R}(\epsilon)$ with the ordering $0 < \epsilon < a$ for any $0 < a \in \mathbb{R}$ and its *real closure* $\widetilde{\mathbb{R}(\epsilon)}$. Then the polynomial $f^2 - \epsilon \in \widetilde{\mathbb{R}(\epsilon)}[X_1, \dots, X_n]$ is non-singular, and one can verify existence and find a zero $y \in (\widetilde{\mathbb{R}(\epsilon)})^n$ of $f^2 - \epsilon$. Here we exploit the Tarski transfer principle for real closed fields. Then, informally, the algorithm substitutes 0 instead of ϵ in y, the resulting $y(0) \in \mathbb{R}^n$ is a real zero of f.

Infinitesimals in symbolic computations

One of the important tools in the algorithm solving systems of polynomial inequalities is explicit using infinitesimals. To illustrate, consider a particular problem of verifying existence and finding (provided it does exist) a real zero $x \in \mathbb{R}^n$ of a polynomial $f \in \mathbb{R}[X_1, \dots, X_n]$. When f is *non-singular*, i. e. the system $f = \frac{\partial f}{\partial X_1} = \cdots = \frac{\partial f}{\partial X_n} = 0$ has no *complex* zeroes one can verify existence and find a real zero of f by means of reduction to the complex case (so-called, the *critical points method*). Now let f be singular.

Introduce an **infinitesimal** ϵ. Formally, consider an ordered field $\mathbb{R}(\epsilon)$ with the ordering $0 < \epsilon < a$ for any $0 < a \in \mathbb{R}$ and its *real closure* $\widetilde{\mathbb{R}(\epsilon)}$. Then the polynomial $f^2 - \epsilon \in \widetilde{\mathbb{R}(\epsilon)}[X_1, \dots, X_n]$ is non-singular, and one can verify existence and find a zero $y \in (\widetilde{\mathbb{R}(\epsilon)})^n$ of $f^2 - \epsilon$. Here we exploit the Tarski transfer principle for real closed fields. Then, informally, the algorithm substitutes 0 instead of ϵ in y, the resulting $y(0) \in \mathbb{R}^n$ is a real zero of f.

Infinitesimals in symbolic computations

One of the important tools in the algorithm solving systems of polynomial inequalities is explicit using infinitesimals. To illustrate, consider a particular problem of verifying existence and finding (provided it does exist) a real zero $x \in \mathbb{R}^n$ of a polynomial $f \in \mathbb{R}[X_1, \dots, X_n]$. When f is *non-singular*, i. e. the system $f = \frac{\partial f}{\partial X_1} = \cdots = \frac{\partial f}{\partial X_n} = 0$ has no *complex* zeroes one can verify existence and find a real zero of f by means of reduction to the complex case (so-called, the *critical points method*). Now let f be singular.

Introduce an **infinitesimal** ϵ. Formally, consider an ordered field $\mathbb{R}(\epsilon)$ with the ordering $0 < \epsilon < a$ for any $0 < a \in \mathbb{R}$ and its *real closure* $\widetilde{\mathbb{R}(\epsilon)}$. Then the polynomial $f^2 - \epsilon \in \widetilde{\mathbb{R}(\epsilon)}[X_1, \dots, X_n]$ is non-singular, and one can verify existence and find a zero $y \in (\widetilde{\mathbb{R}(\epsilon)})^n$ of $f^2 - \epsilon$. Here we exploit the Tarski transfer principle for real closed fields. Then, informally, the algorithm substitutes 0 instead of ϵ in y, the resulting $y(0) \in \mathbb{R}^n$ is a real zero of f.

Infinitesimals in symbolic computations

One of the important tools in the algorithm solving systems of polynomial inequalities is explicit using infinitesimals. To illustrate, consider a particular problem of verifying existence and finding (provided it does exist) a real zero $x \in \mathbb{R}^n$ of a polynomial $f \in \mathbb{R}[X_1, \dots, X_n]$. When f is *non-singular*, i. e. the system $f = \frac{\partial f}{\partial X_1} = \cdots = \frac{\partial f}{\partial X_n} = 0$ has no *complex* zeroes one can verify existence and find a real zero of f by means of reduction to the complex case (so-called, the *critical points method*). Now let f be singular.

Introduce an **infinitesimal** ϵ. Formally, consider an ordered field $\mathbb{R}(\epsilon)$ with the ordering $0 < \epsilon < a$ for any $0 < a \in \mathbb{R}$ and its *real closure* $\widetilde{\mathbb{R}(\epsilon)}$. Then the polynomial $f^2 - \epsilon \in \widetilde{\mathbb{R}(\epsilon)}[X_1, \dots, X_n]$ is non-singular, and one can verify existence and find a zero $y \in (\widetilde{\mathbb{R}(\epsilon)})^n$ of $f^2 - \epsilon$. Here we exploit the Tarski transfer principle for real closed fields. Then, informally, the algorithm substitutes 0 instead of ϵ in y, the resulting $y(0) \in \mathbb{R}^n$ is a real zero of f.

Leibniz’ vs. Newton’s approaches in symbolic computations

This idea of explicit involving infinitesimals in the symbolic algorithms has appeared to be fruitful for improving complexity. It is in a spirit of the language of Leibniz in analysis vs. the language of Newton based on the concept of the limit.

Leibniz' vs. Newton's approaches in symbolic computations

This idea of explicit involving infinitesimals in the symbolic algorithms has appeared to be fruitful for improving complexity. It is in a spirit of the language of Leibniz in analysis vs. the language of Newton based on the concept of the limit.

Quantifier elimination in the first-order theory of real closed fields

Similar to the case of the complex field one considers formulas of the type

$$\exists X_{11} \cdots \exists X_{1n_1} \forall X_{21} \cdots \forall X_{2n_2} \cdots \exists X_{a1} \cdots \exists X_{an_a} Q$$

where the quantifier-free formula Q as its atomic subformulas contains inequalities of the form $f \geq 0$ for polynomials $f \in \mathbb{R}[x_{11}, \dots, X_{as_a}, X_1, \dots, X_n]$.

Tarski (1930): a quantifier elimination method for these formulas, its complexity is enormous. Collins (1973): a quantifier elimination procedure with the double-exponential complexity.

Similar to the complex field case a quantifier elimination algorithm was designed by G. (1984), Heintz-Roy (1986) by means of parametrizing the algorithm solving systems of polynomial inequalities. The complexity of this algorithm is exponential for a fixed number a of quantifier alternations. The latter bound is sharp due to the example of Davenport-Heintz (1986).

Quantifier elimination in the first-order theory of real closed fields

Similar to the case of the complex field one considers formulas of the type

$$\exists X_{11} \cdots \exists X_{1n_1} \forall X_{21} \cdots \forall X_{2n_2} \cdots \exists X_{a1} \cdots \exists X_{an_a} Q$$

where the quantifier-free formula Q as its atomic subformulas contains inequalities of the form $f \geq 0$ for polynomials $f \in \mathbb{R}[x_{11}, \dots, X_{as_a}, X_1, \dots, X_n]$.

Tarski (1930): a quantifier elimination method for these formulas, its complexity is enormous. Collins (1973): a quantifier elimination procedure with the double-exponential complexity.

Similar to the complex field case a quantifier elimination algorithm was designed by G. (1984), Heintz-Roy (1986) by means of parametrizing the algorithm solving systems of polynomial inequalities. The complexity of this algorithm is exponential for a fixed number a of quantifier alternations. The latter bound is sharp due to the example of Davenport-Heintz (1986).

Quantifier elimination in the first-order theory of real closed fields

Similar to the case of the complex field one considers formulas of the type

$$\exists X_{11} \cdots \exists X_{1n_1} \forall X_{21} \cdots \forall X_{2n_2} \cdots \exists X_{a1} \cdots \exists X_{an_a} Q$$

where the quantifier-free formula Q as its atomic subformulas contains inequalities of the form $f \geq 0$ for polynomials $f \in \mathbb{R}[x_{11}, \ldots, X_{as_a}, X_1, \ldots, X_n]$.

Tarski (1930): a quantifier elimination method for these formulas, its complexity is enormous. Collins (1973): a quantifier elimination procedure with the double-exponential complexity.

Similar to the complex field case a quantifier elimination algorithm was designed by G. (1984), Heintz-Roy (1986) by means of parametrizing the algorithm solving systems of polynomial inequalities. The complexity of this algorithm is exponential for a fixed number a of quantifier alternations. The latter bound is sharp due to the example of Davenport-Heintz (1986).

Quantifier elimination in the first-order theory of real closed fields

Similar to the case of the complex field one considers formulas of the type

$$\exists X_{11} \cdots \exists X_{1n_1} \forall X_{21} \cdots \forall X_{2n_2} \cdots \exists X_{a1} \cdots \exists X_{an_a} Q$$

where the quantifier-free formula Q as its atomic subformulas contains inequalities of the form $f \geq 0$ for polynomials $f \in \mathbb{R}[x_{11}, \dots, X_{as_a}, X_1, \dots, X_n]$.

Tarski (1930): a quantifier elimination method for these formulas, its complexity is enormous. Collins (1973): a quantifier elimination procedure with the double-exponential complexity.

Similar to the complex field case a quantifier elimination algorithm was designed by G. (1984), Heintz-Roy (1986) by means of parametrizing the algorithm solving systems of polynomial inequalities. The complexity of this algorithm is exponential for a fixed number a of quantifier alternations. The latter bound is sharp due to the example of Davenport-Heintz (1986).

Quantifier elimination in the first-order theory of real closed fields

Similar to the case of the complex field one considers formulas of the type

$$\exists X_{11} \cdots \exists X_{1n_1} \forall X_{21} \cdots \forall X_{2n_2} \cdots \exists X_{a1} \cdots \exists X_{an_a} Q$$

where the quantifier-free formula Q as its atomic subformulas contains inequalities of the form $f \geq 0$ for polynomials $f \in \mathbb{R}[x_{11}, \dots, X_{as_a}, X_1, \dots, X_n]$.

Tarski (1930): a quantifier elimination method for these formulas, its complexity is enormous. Collins (1973): a quantifier elimination procedure with the double-exponential complexity.

Similar to the complex field case a quantifier elimination algorithm was designed by G. (1984), Heintz-Roy (1986) by means of parametrizing the algorithm solving systems of polynomial inequalities. The complexity of this algorithm is exponential for a fixed number a of quantifier alternations. The latter bound is sharp due to the example of Davenport-Heintz (1986).

Quantifier elimination in the first-order theory of real closed fields

Similar to the case of the complex field one considers formulas of the type

$$\exists X_{11} \cdots \exists X_{1n_1} \forall X_{21} \cdots \forall X_{2n_2} \cdots \exists X_{a1} \cdots \exists X_{an_a} Q$$

where the quantifier-free formula Q as its atomic subformulas contains inequalities of the form $f \geq 0$ for polynomials $f \in \mathbb{R}[x_{11}, \ldots, X_{as_a}, X_1, \ldots, X_n]$.

Tarski (1930): a quantifier elimination method for these formulas, its complexity is enormous. Collins (1973): a quantifier elimination procedure with the double-exponential complexity.

Similar to the complex field case a quantifier elimination algorithm was designed by G. (1984), Heintz-Roy (1986) by means of parametrizing the algorithm solving systems of polynomial inequalities. The complexity of this algorithm is exponential for a fixed number a of quantifier alternations. The latter bound is sharp due to the example of Davenport-Heintz (1986).

Quantifier elimination in the first-order theory of real closed fields

Similar to the case of the complex field one considers formulas of the type

$$\exists X_{11} \cdots \exists X_{1n_1} \forall X_{21} \cdots \forall X_{2n_2} \cdots \exists X_{a1} \cdots \exists X_{an_a} Q$$

where the quantifier-free formula Q as its atomic subformulas contains inequalities of the form $f \geq 0$ for polynomials $f \in \mathbb{R}[x_{11}, \dots, X_{as_a}, X_1, \dots, X_n]$.

Tarski (1930): a quantifier elimination method for these formulas, its complexity is enormous. Collins (1973): a quantifier elimination procedure with the double-exponential complexity.

Similar to the complex field case a quantifier elimination algorithm was designed by G. (1984), Heintz-Roy (1986) by means of parametrizing the algorithm solving systems of polynomial inequalities. The complexity of this algorithm is exponential for a fixed number a of quantifier alternations. The latter bound is sharp due to the example of Davenport-Heintz (1986).

Complexity of constructing connected components of a semialgebraic set

A semialgebraic set
$S := S(f_1 \geq 0, \dots, f_k \geq 0) = \{x \in \mathbb{R}^n \,:\, f_1(x) \geq 0, \dots, f_k(x) \geq 0\}$
is the set of points satisfying a system of inequalities.

Similar to the complex field case a question arises how to output the set S algorithmically? A relevant language appears to be the connected components $S = \sqcup_j S_j$. How to find the connected components S_j?

One can get S_j applying the algorithm by Collins (1973) which provides the *cylindrical algebraic decomposition* of a semialgebraic set. Moreover, the cylindrical algebraic decomposition allows one to obtain the topological structure of a semialgebraic set, in particular, the homology groups. But the complexity of Collins' method is double-exponential. G.-Vorobjov (1988): an algorithm for finding connected components within exponential complexity. It is an open problem whether one can find the topological structure within exponential complexity?

Complexity of constructing connected components of a semialgebraic set

A semialgebraic set
$S := S(f_1 \geq 0, \ldots, f_k \geq 0) = \{x \in \mathbb{R}^n \,:\, f_1(x) \geq 0, \ldots, f_k(x) \geq 0\}$
is the set of points satisfying a system of inequalities.

Similar to the complex field case a question arises how to output the set S algorithmically? A relevant language appears to be the connected components $S = \sqcup_j S_j$. How to find the connected components S_j?

One can get S_j applying the algorithm by Collins (1973) which provides the *cylindrical algebraic decomposition* of a semialgebraic set. Moreover, the cylindrical algebraic decomposition allows one to obtain the topological structure of a semialgebraic set, in particular, the homology groups. But the complexity of Collins' method is double-exponential. G.-Vorobjov (1988): an algorithm for finding connected components within exponential complexity. It is an open problem whether one can find the topological structure within exponential complexity?

Complexity of constructing connected components of a semialgebraic set

A semialgebraic set
$S := S(f_1 \geq 0, \dots, f_k \geq 0) = \{x \in \mathbb{R}^n \,:\, f_1(x) \geq 0, \dots, f_k(x) \geq 0\}$
is the set of points satisfying a system of inequalities.

Similar to the complex field case a question arises how to output the set S algorithmically? A relevant language appears to be the connected components $S = \sqcup_j S_j$. How to find the connected components S_j?

One can get S_j applying the algorithm by Collins (1973) which provides the *cylindrical algebraic decomposition* of a semialgebraic set. Moreover, the cylindrical algebraic decomposition allows one to obtain the topological structure of a semialgebraic set, in particular, the homology groups. But the complexity of Collins' method is double-exponential. G.-Vorobjov (1988): an algorithm for finding connected components within exponential complexity. It is an open problem whether one can find the topological structure within exponential complexity?

Complexity of constructing connected components of a semialgebraic set

A semialgebraic set
$S := S(f_1 \geq 0, \ldots, f_k \geq 0) = \{x \in \mathbb{R}^n : f_1(x) \geq 0, \ldots, f_k(x) \geq 0\}$
is the set of points satisfying a system of inequalities.

Similar to the complex field case a question arises how to output the set S algorithmically? A relevant language appears to be the connected components $S = \sqcup_j S_j$. How to find the connected components S_j?

One can get S_j applying the algorithm by Collins (1973) which provides the *cylindrical algebraic decomposition* of a semialgebraic set. Moreover, the cylindrical algebraic decomposition allows one to obtain the topological structure of a semialgebraic set, in particular, the homology groups. But the complexity of Collins' method is double-exponential. G.-Vorobjov (1988): an algorithm for finding connected components within exponential complexity. It is an open problem whether one can find the topological structure within exponential complexity?

Complexity of constructing connected components of a semialgebraic set

A semialgebraic set
$S := S(f_1 \geq 0, \ldots, f_k \geq 0) = \{x \in \mathbb{R}^n \,:\, f_1(x) \geq 0, \ldots, f_k(x) \geq 0\}$
is the set of points satisfying a system of inequalities.

Similar to the complex field case a question arises how to output the set S algorithmically? A relevant language appears to be the connected components $S = \sqcup_j S_j$. How to find the connected components S_j?

One can get S_j applying the algorithm by Collins (1973) which provides the *cylindrical algebraic decomposition* of a semialgebraic set.

Moreover, the cylindrical algebraic decomposition allows one to obtain the topological structure of a semialgebraic set, in particular, the homology groups. But the complexity of Collins' method is double-exponential. G.-Vorobjov (1988): an algorithm for finding connected components within exponential complexity. It is an open problem whether one can find the topological structure within exponential complexity?

Complexity of constructing connected components of a semialgebraic set

A semialgebraic set
$S := S(f_1 \geq 0, \ldots, f_k \geq 0) = \{x \in \mathbb{R}^n : f_1(x) \geq 0, \ldots, f_k(x) \geq 0\}$
is the set of points satisfying a system of inequalities.

Similar to the complex field case a question arises how to output the set S algorithmically? A relevant language appears to be the connected components $S = \sqcup_j S_j$. How to find the connected components S_j?

One can get S_j applying the algorithm by Collins (1973) which provides the *cylindrical algebraic decomposition* of a semialgebraic set. Moreover, the cylindrical algebraic decomposition allows one to obtain the topological structure of a semialgebraic set, in particular, the homology groups. But the complexity of Collins' method is double-exponential. G.-Vorobjov (1988): an algorithm for finding connected components within exponential complexity. It is an open problem whether one can find the topological structure within exponential complexity?

Complexity of constructing connected components of a semialgebraic set

A semialgebraic set
$S := S(f_1 \geq 0, \dots, f_k \geq 0) = \{x \in \mathbb{R}^n : f_1(x) \geq 0, \dots, f_k(x) \geq 0\}$
is the set of points satisfying a system of inequalities.

Similar to the complex field case a question arises how to output the set S algorithmically? A relevant language appears to be the connected components $S = \sqcup_j S_j$. How to find the connected components S_j?

One can get S_j applying the algorithm by Collins (1973) which provides the *cylindrical algebraic decomposition* of a semialgebraic set. Moreover, the cylindrical algebraic decomposition allows one to obtain the topological structure of a semialgebraic set, in particular, the homology groups. But the complexity of Collins' method is double-exponential. G.-Vorobjov (1988): an algorithm for finding connected components within exponential complexity. It is an open problem whether one can find the topological structure within exponential complexity?

Complexity of constructing connected components of a semialgebraic set

A semialgebraic set
$S := S(f_1 \geq 0, \ldots, f_k \geq 0) = \{x \in \mathbb{R}^n : f_1(x) \geq 0, \ldots, f_k(x) \geq 0\}$
is the set of points satisfying a system of inequalities.

Similar to the complex field case a question arises how to output the set S algorithmically? A relevant language appears to be the connected components $S = \sqcup_j S_j$. How to find the connected components S_j?

One can get S_j applying the algorithm by Collins (1973) which provides the *cylindrical algebraic decomposition* of a semialgebraic set. Moreover, the cylindrical algebraic decomposition allows one to obtain the topological structure of a semialgebraic set, in particular, the homology groups. But the complexity of Collins' method is double-exponential. G.-Vorobjov (1988): an algorithm for finding connected components within exponential complexity. It is an open problem whether one can find the topological structure within exponential complexity?

Complexity of constructing connected components of a semialgebraic set

A semialgebraic set
$S := S(f_1 \geq 0, \dots, f_k \geq 0) = \{x \in \mathbb{R}^n : f_1(x) \geq 0, \dots, f_k(x) \geq 0\}$
is the set of points satisfying a system of inequalities.

Similar to the complex field case a question arises how to output the set S algorithmically? A relevant language appears to be the connected components $S = \sqcup_j S_j$. How to find the connected components S_j?

One can get S_j applying the algorithm by Collins (1973) which provides the *cylindrical algebraic decomposition* of a semialgebraic set. Moreover, the cylindrical algebraic decomposition allows one to obtain the topological structure of a semialgebraic set, in particular, the homology groups. But the complexity of Collins' method is double-exponential. G.-Vorobjov (1988): an algorithm for finding connected components within exponential complexity. It is an open problem whether one can find the topological structure within exponential complexity?

Complexity of Nullstellensatz

Remind that Hilbert's Nullstellensatz states that a system of polynomials $f_1, \dots, f_k \in \mathbb{C}[X_1, \dots, X_n]$ has no complex zeroes iff the ideal $\langle f_1, \dots, f_k \rangle \ni 1$ contains 1. Equivalently, $1 = \sum_{1 \le i \le k} h_i \cdot f_i$ for suitable polynomials $h_1, \dots, h_k \in \mathbb{C}[X_1, \dots, X_n]$.

Let $\deg(f_i) < d$, $1 \le i \le k$. First the double-exponential bound on $\deg(h_i) < d^{2^{O(n)}}$ was established by Hermann (1926). The original Hilbert's proof (1890) was non-constructive. Brownawell (1986), Giusti-Heintz (1988): $\deg(h_i) < d^{O(n)}$ being sharp (Sometimes, the latter bound is called the *Effective Nullstellensatz*).

For the problem of *membership to an ideal*, so whether $g = \sum_{1 \le i \le k} h_i \cdot f_i$ for given $g, f_1, \dots, f_k$ (rather than of membership of 1 to an ideal as in the Nullstellensatz), much worse double-exponential bound $\deg(h_i) < d^{2^{O(n)}}$ is sharp (due to the example of Mair-Meyer).

Complexity of Nullstellensatz

Remind that Hilbert's Nullstellensatz states that a system of polynomials $f_1, \dots, f_k \in \mathbb{C}[X_1, \dots, X_n]$ has no complex zeroes iff the ideal $\langle f_1, \dots, f_k \rangle \ni 1$ contains 1. Equivalently, $1 = \sum_{1 \le i \le k} h_i \cdot f_i$ for suitable polynomials $h_1, \dots, h_k \in \mathbb{C}[X_1, \dots, X_n]$.

Let $\deg(f_i) < d$, $1 \le i \le k$. First the double-exponential bound on $\deg(h_i) < d^{2^{O(n)}}$ was established by Hermann (1926). The original Hilbert's proof (1890) was non-constructive. Brownawell (1986), Giusti-Heintz (1988): $\deg(h_i) < d^{O(n)}$ being sharp (Sometimes, the latter bound is called the *Effective Nullstellensatz*).

For the problem of *membership to an ideal*, so whether $g = \sum_{1 \le i \le k} h_i \cdot f_i$ for given $g, f_1, \dots, f_k$ (rather than of membership of 1 to an ideal as in the Nullstellensatz), much worse double-exponential bound $\deg(h_i) < d^{2^{O(n)}}$ is sharp (due to the example of Mair-Meyer).

Complexity of Nullstellensatz

Remind that Hilbert's Nullstellensatz states that a system of polynomials $f_1, \dots, f_k \in \mathbb{C}[X_1, \dots, X_n]$ has no complex zeroes iff the ideal $\langle f_1, \dots, f_k \rangle \ni 1$ contains 1. Equivalently, $1 = \sum_{1 \le i \le k} h_i \cdot f_i$ for suitable polynomials $h_1, \dots, h_k \in \mathbb{C}[X_1, \dots, X_n]$.

Let $\deg(f_i) < d$, $1 \le i \le k$. First the double-exponential bound on $\deg(h_i) < d^{2^{O(n)}}$ was established by Hermann (1926). The original Hilbert's proof (1890) was non-constructive. Brownawell (1986), Giusti-Heintz (1988): $\deg(h_i) < d^{O(n)}$ being sharp (Sometimes, the latter bound is called the *Effective Nullstellensatz*).

For the problem of *membership to an ideal*, so whether $g = \sum_{1 \le i \le k} h_i \cdot f_i$ for given $g, f_1, \dots, f_k$ (rather than of membership of 1 to an ideal as in the Nullstellensatz), much worse double-exponential bound $\deg(h_i) < d^{2^{O(n)}}$ is sharp (due to the example of Mair-Meyer).

Complexity of Nullstellensatz

Remind that Hilbert's Nullstellensatz states that a system of polynomials $f_1, \dots, f_k \in \mathbb{C}[X_1, \dots, X_n]$ has no complex zeroes iff the ideal $\langle f_1, \dots, f_k \rangle \ni 1$ contains 1. Equivalently, $1 = \sum_{1 \le i \le k} h_i \cdot f_i$ for suitable polynomials $h_1, \dots, h_k \in \mathbb{C}[X_1, \dots, X_n]$.

Let $\deg(f_i) < d$, $1 \le i \le k$. First the double-exponential bound on $\deg(h_i) < d^{2^{O(n)}}$ was established by Hermann (1926). The original Hilbert's proof (1890) was non-constructive. Brownawell (1986), Giusti-Heintz (1988): $\deg(h_i) < d^{O(n)}$ being sharp (Sometimes, the latter bound is called the *Effective Nullstellensatz*).

For the problem of *membership to an ideal*, so whether $g = \sum_{1 \le i \le k} h_i \cdot f_i$ for given $g, f_1, \dots, f_k$ (rather than of membership of 1 to an ideal as in the Nullstellensatz), much worse double-exponential bound $\deg(h_i) < d^{2^{O(n)}}$ is sharp (due to the example of Mair-Meyer).

Complexity of Nullstellensatz

Remind that Hilbert's Nullstellensatz states that a system of polynomials $f_1, \ldots, f_k \in \mathbb{C}[X_1, \ldots, X_n]$ has no complex zeroes iff the ideal $\langle f_1, \ldots, f_k \rangle \ni 1$ contains 1. Equivalently, $1 = \sum_{1 \leq i \leq k} h_i \cdot f_i$ for suitable polynomials $h_1, \ldots, h_k \in \mathbb{C}[X_1, \ldots, X_n]$.

Let $\deg(f_i) < d$, $1 \leq i \leq k$. First the double-exponential bound on $\deg(h_i) < d^{2^{O(n)}}$ was established by Hermann (1926). The original Hilbert's proof (1890) was non-constructive. Brownawell (1986), Giusti-Heintz (1988): $\deg(h_i) < d^{O(n)}$ being sharp (Sometimes, the latter bound is called the *Effective Nullstellensatz*).

For the problem of *membership to an ideal*, so whether $g = \sum_{1 \leq i \leq k} h_i \cdot f_i$ for given $g, f_1, \ldots, f_k$ (rather than of membership of 1 to an ideal as in the Nullstellensatz), much worse double-exponential bound $\deg(h_i) < d^{2^{O(n)}}$ is sharp (due to the example of Mair-Meyer).

Complexity of Nullstellensatz

Remind that Hilbert's Nullstellensatz states that a system of polynomials $f_1, \dots, f_k \in \mathbb{C}[X_1, \dots, X_n]$ has no complex zeroes iff the ideal $\langle f_1, \dots, f_k \rangle \ni 1$ contains 1. Equivalently, $1 = \sum_{1 \le i \le k} h_i \cdot f_i$ for suitable polynomials $h_1, \dots, h_k \in \mathbb{C}[X_1, \dots, X_n]$.

Let $\deg(f_i) < d$, $1 \le i \le k$. First the double-exponential bound on $\deg(h_i) < d^{2^{O(n)}}$ was established by Hermann (1926). The original Hilbert's proof (1890) was non-constructive. Brownawell (1986), Giusti-Heintz (1988): $\deg(h_i) < d^{O(n)}$ being sharp (Sometimes, the latter bound is called the *Effective Nullstellensatz*).

For the problem of *membership to an ideal*, so whether $g = \sum_{1 \le i \le k} h_i \cdot f_i$ for given $g, f_1, \dots, f_k$ (rather than of membership of 1 to an ideal as in the Nullstellensatz), much worse double-exponential bound $\deg(h_i) < d^{2^{O(n)}}$ is sharp (due to the example of Mair-Meyer).

Complexity of Nullstellensatz

Remind that Hilbert's Nullstellensatz states that a system of polynomials $f_1, \dots, f_k \in \mathbb{C}[X_1, \dots, X_n]$ has no complex zeroes iff the ideal $\langle f_1, \dots, f_k \rangle \ni 1$ contains 1. Equivalently, $1 = \sum_{1 \le i \le k} h_i \cdot f_i$ for suitable polynomials $h_1, \dots, h_k \in \mathbb{C}[X_1, \dots, X_n]$.

Let $\deg(f_i) < d$, $1 \le i \le k$. First the double-exponential bound on $\deg(h_i) < d^{2^{O(n)}}$ was established by Hermann (1926). The original Hilbert's proof (1890) was non-constructive. Brownawell (1986), Giusti-Heintz (1988): $\deg(h_i) < d^{O(n)}$ being sharp (Sometimes, the latter bound is called the *Effective Nullstellensatz*).

For the problem of *membership to an ideal*, so whether $g = \sum_{1 \le i \le k} h_i \cdot f_i$ for given $g, f_1, \dots, f_k$ (rather than of membership of 1 to an ideal as in the Nullstellensatz), much worse double-exponential bound $\deg(h_i) < d^{2^{O(n)}}$ is sharp (due to the example of Mair-Meyer).

Complexity of Nullstellensatz

Remind that Hilbert's Nullstellensatz states that a system of polynomials $f_1, \ldots, f_k \in \mathbb{C}[X_1, \ldots, X_n]$ has no complex zeroes iff the ideal $\langle f_1, \ldots, f_k \rangle \ni 1$ contains 1. Equivalently, $1 = \sum_{1 \le i \le k} h_i \cdot f_i$ for suitable polynomials $h_1, \ldots, h_k \in \mathbb{C}[X_1, \ldots, X_n]$.

Let $\deg(f_i) < d$, $1 \le i \le k$. First the double-exponential bound on $\deg(h_i) < d^{2^{O(n)}}$ was established by Hermann (1926). The original Hilbert's proof (1890) was non-constructive. Brownawell (1986), Giusti-Heintz (1988): $\deg(h_i) < d^{O(n)}$ being sharp (Sometimes, the latter bound is called the *Effective Nullstellensatz*).

For the problem of *membership to an ideal*, so whether $g = \sum_{1 \le i \le k} h_i \cdot f_i$ for given $g, f_1, \ldots, f_k$ (rather than of membership of 1 to an ideal as in the Nullstellensatz), much worse double-exponential bound $\deg(h_i) < d^{2^{O(n)}}$ is sharp (due to the example of Mair-Meyer).

Complexity of Nullstellensatz

Remind that Hilbert's Nullstellensatz states that a system of polynomials $f_1, \dots, f_k \in \mathbb{C}[X_1, \dots, X_n]$ has no complex zeroes iff the ideal $\langle f_1, \dots, f_k \rangle \ni 1$ contains 1. Equivalently, $1 = \sum_{1 \le i \le k} h_i \cdot f_i$ for suitable polynomials $h_1, \dots, h_k \in \mathbb{C}[X_1, \dots, X_n]$.

Let $\deg(f_i) < d$, $1 \le i \le k$. First the double-exponential bound on $\deg(h_i) < d^{2^{O(n)}}$ was established by Hermann (1926). The original Hilbert's proof (1890) was non-constructive. Brownawell (1986), Giusti-Heintz (1988): $\deg(h_i) < d^{O(n)}$ being sharp (Sometimes, the latter bound is called the *Effective Nullstellensatz*).

For the problem of *membership to an ideal*, so whether $g = \sum_{1 \le i \le k} h_i \cdot f_i$ for given $g, f_1, \dots, f_k$ (rather than of membership of 1 to an ideal as in the Nullstellensatz), much worse double-exponential bound $\deg(h_i) < d^{2^{O(n)}}$ is sharp (due to the example of Mair-Meyer).

Positivstellensatz: its complexity?

To formulate a real field analogue of the Nullstellensatz (called *Positivstellensatz*) one needs to replace the concept of the ideal by the one of the cone. The *cone* $C := C(f_1, \dots, f_k) \ni f_1, \dots, f_k$ for real polynomials $f_1, \dots, f_k \in \mathbb{R}[X_1, \dots, X_n]$ is generated recursively by the following operations

- if $g_1, g_2 \in C$ then $g_1 + g_2 \in C$;
- if $g_1, g_2 \in C$ then $g_1 \cdot g_2 \in C$;
- $g^2 \in C$ for any $g \in \mathbb{R}[X_1, \dots, X_n]$.

The Positivstellensatz claims that a system of inequalities $f_1 \geq 0, \dots, f_k \geq 0$ has no real solution iff $-1 \in C$. The Positivstellensatz generalizes the 17-th Hilbert's problem solved by Artin in 1927.

Unlike the Nullstellensatz, the complexity bound on the Positivstellensatz is unknown. The difficulty is that the existing proofs of the Positivstellensatz involve the model theory (the compactness theorem based on the axiom of choice).

Positivstellensatz: its complexity?

To formulate a real field analogue of the Nullstellensatz (called *Positivstellensatz*) one needs to replace the concept of the ideal by the one of the cone. The *cone* $C := C(f_1, \dots, f_k) \ni f_1, \dots, f_k$ for real polynomials $f_1, \dots, f_k \in \mathbb{R}[X_1, \dots, X_n]$ is generated recursively by the following operations

- if $g_1, g_2 \in C$ then $g_1 + g_2 \in C$;
- if $g_1, g_2 \in C$ then $g_1 \cdot g_2 \in C$;
- $g^2 \in C$ for any $g \in \mathbb{R}[X_1, \dots, X_n]$.

The Positivstellensatz claims that a system of inequalities $f_1 \geq 0, \dots, f_k \geq 0$ has no real solution iff $-1 \in C$. The Positivstellensatz generalizes the 17-th Hilbert's problem solved by Artin in 1927.

Unlike the Nullstellensatz, the complexity bound on the Positivstellensatz is unknown. The difficulty is that the existing proofs of the Positivstellensatz involve the model theory (the compactness theorem based on the axiom of choice).

Positivstellensatz: its complexity?

To formulate a real field analogue of the Nullstellensatz (called *Positivstellensatz*) one needs to replace the concept of the ideal by the one of the cone. The *cone* $C := C(f_1, \dots, f_k) \ni f_1, \dots, f_k$ for real polynomials $f_1, \dots, f_k \in \mathbb{R}[X_1, \dots, X_n]$ is generated recursively by the following operations

- if $g_1, g_2 \in C$ then $g_1 + g_2 \in C$;
- if $g_1, g_2 \in C$ then $g_1 \cdot g_2 \in C$;
- $g^2 \in C$ for any $g \in \mathbb{R}[X_1, \dots, X_n]$.

The Positivstellensatz claims that a system of inequalities $f_1 \geq 0, \dots, f_k \geq 0$ has no real solution iff $-1 \in C$. The Positivstellensatz generalizes the 17-th Hilbert's problem solved by Artin in 1927.

Unlike the Nullstellensatz, the complexity bound on the Positivstellensatz is unknown. The difficulty is that the existing proofs of the Positivstellensatz involve the model theory (the compactness theorem based on the axiom of choice).

Positivstellensatz: its complexity?

To formulate a real field analogue of the Nullstellensatz (called *Positivstellensatz*) one needs to replace the concept of the ideal by the one of the cone. The *cone* $C := C(f_1, \dots, f_k) \ni f_1, \dots, f_k$ for real polynomials $f_1, \dots, f_k \in \mathbb{R}[X_1, \dots, X_n]$ is generated recursively by the following operations

- if $g_1, g_2 \in C$ then $g_1 + g_2 \in C$;
- if $g_1, g_2 \in C$ then $g_1 \cdot g_2 \in C$;
- $g^2 \in C$ for any $g \in \mathbb{R}[X_1, \dots, X_n]$.

The Positivstellensatz claims that a system of inequalities $f_1 \geq 0, \dots, f_k \geq 0$ has no real solution iff $-1 \in C$. The Positivstellensatz generalizes the 17-th Hilbert's problem solved by Artin in 1927.

Unlike the Nullstellensatz, the complexity bound on the Positivstellensatz is unknown. The difficulty is that the existing proofs of the Positivstellensatz involve the model theory (the compactness theorem based on the axiom of choice).

Positivstellensatz: its complexity?

To formulate a real field analogue of the Nullstellensatz (called *Positivstellensatz*) one needs to replace the concept of the ideal by the one of the cone. The *cone* $C := C(f_1, \dots, f_k) \ni f_1, \dots, f_k$ for real polynomials $f_1, \dots, f_k \in \mathbb{R}[X_1, \dots, X_n]$ is generated recursively by the following operations

- if $g_1, g_2 \in C$ then $g_1 + g_2 \in C$;
- if $g_1, g_2 \in C$ then $g_1 \cdot g_2 \in C$;
- $g^2 \in C$ for any $g \in \mathbb{R}[X_1, \dots, X_n]$.

The Positivstellensatz claims that a system of inequalities $f_1 \geq 0, \dots, f_k \geq 0$ has no real solution iff $-1 \in C$. The Positivstellensatz generalizes the 17-th Hilbert's problem solved by Artin in 1927.

Unlike the Nullstellensatz, the complexity bound on the Positivstellensatz is unknown. The difficulty is that the existing proofs of the Positivstellensatz involve the model theory (the compactness theorem based on the axiom of choice).

Positivstellensatz: its complexity?

To formulate a real field analogue of the Nullstellensatz (called *Positivstellensatz*) one needs to replace the concept of the ideal by the one of the cone. The *cone* $C := C(f_1, \dots, f_k) \ni f_1, \dots, f_k$ for real polynomials $f_1, \dots, f_k \in \mathbb{R}[X_1, \dots, X_n]$ is generated recursively by the following operations

- if $g_1, g_2 \in C$ then $g_1 + g_2 \in C$;
- if $g_1, g_2 \in C$ then $g_1 \cdot g_2 \in C$;
- $g^2 \in C$ for any $g \in \mathbb{R}[X_1, \dots, X_n]$.

The Positivstellensatz claims that a system of inequalities $f_1 \geq 0, \dots, f_k \geq 0$ has no real solution iff $-1 \in C$. The Positivstellensatz generalizes the 17-th Hilbert's problem solved by Artin in 1927.

Unlike the Nullstellensatz, the complexity bound on the Positivstellensatz is unknown. The difficulty is that the existing proofs of the Positivstellensatz involve the model theory (the compactness theorem based on the axiom of choice).

Positivstellensatz: its complexity?

To formulate a real field analogue of the Nullstellensatz (called *Positivstellensatz*) one needs to replace the concept of the ideal by the one of the cone. The *cone* $C := C(f_1, \dots, f_k) \ni f_1, \dots, f_k$ for real polynomials $f_1, \dots, f_k \in \mathbb{R}[X_1, \dots, X_n]$ is generated recursively by the following operations

- if $g_1, g_2 \in C$ then $g_1 + g_2 \in C$;
- if $g_1, g_2 \in C$ then $g_1 \cdot g_2 \in C$;
- $g^2 \in C$ for any $g \in \mathbb{R}[X_1, \dots, X_n]$.

The Positivstellensatz claims that a system of inequalities $f_1 \geq 0, \dots, f_k \geq 0$ has no real solution iff $-1 \in C$. The Positivstellensatz generalizes the 17-th Hilbert's problem solved by Artin in 1927.

Unlike the Nullstellensatz, the complexity bound on the Positivstellensatz is unknown. The difficulty is that the existing proofs of the Positivstellensatz involve the model theory (the compactness theorem based on the axiom of choice).

Positivstellensatz: its complexity?

To formulate a real field analogue of the Nullstellensatz (called *Positivstellensatz*) one needs to replace the concept of the ideal by the one of the cone. The *cone* $C := C(f_1, \dots, f_k) \ni f_1, \dots, f_k$ for real polynomials $f_1, \dots, f_k \in \mathbb{R}[X_1, \dots, X_n]$ is generated recursively by the following operations

- if $g_1, g_2 \in C$ then $g_1 + g_2 \in C$;
- if $g_1, g_2 \in C$ then $g_1 \cdot g_2 \in C$;
- $g^2 \in C$ for any $g \in \mathbb{R}[X_1, \dots, X_n]$.

The Positivstellensatz claims that a system of inequalities $f_1 \geq 0, \dots, f_k \geq 0$ has no real solution iff $-1 \in C$. The Positivstellensatz generalizes the 17-th Hilbert's problem solved by Artin in 1927.

Unlike the Nullstellensatz, the complexity bound on the Positivstellensatz is unknown. The difficulty is that the existing proofs of the Positivstellensatz involve the model theory (the compactness theorem based on the axiom of choice).

Positivstellensatz: its complexity?

To formulate a real field analogue of the Nullstellensatz (called *Positivstellensatz*) one needs to replace the concept of the ideal by the one of the cone. The *cone* $C := C(f_1, \ldots, f_k) \ni f_1, \ldots, f_k$ for real polynomials $f_1, \ldots, f_k \in \mathbb{R}[X_1, \ldots, X_n]$ is generated recursively by the following operations

- if $g_1, g_2 \in C$ then $g_1 + g_2 \in C$;
- if $g_1, g_2 \in C$ then $g_1 \cdot g_2 \in C$;
- $g^2 \in C$ for any $g \in \mathbb{R}[X_1, \ldots, X_n]$.

The Positivstellensatz claims that a system of inequalities $f_1 \geq 0, \ldots, f_k \geq 0$ has no real solution iff $-1 \in C$. The Positivstellensatz generalizes the 17-th Hilbert's problem solved by Artin in 1927.

Unlike the Nullstellensatz, the complexity bound on the Positivstellensatz is unknown. The difficulty is that the existing proofs of the Positivstellensatz involve the model theory (the compactness theorem based on the axiom of choice).

Resolution of singularities of varieties

Let V be a variety over a field of characteristic 0. A *smooth* variety U is a **resolution of singularities** of V if $\dim U = \dim V$ and there exists an epimorphism $\phi : U \to V$ such that ϕ is a local isomorphism at a neighborhood of any *smooth* point of V.

Hironaka (1964) has designed an algorithm for resolution of singularities of an arbitrary variety V. His algorithm is quite complicated, and enormously complicated is the proof of its correctness.

Resolution of singularities of varieties

Let V be a variety over a field of characteristic 0. A *smooth* variety U is a **resolution of singularities** of V if $\dim U = \dim V$ and there exists an epimorphism $\phi : U \to V$ such that ϕ is a local isomorphism at a neighborhood of any *smooth* point of V.

Hironaka (1964) has designed an algorithm for resolution of singularities of an arbitrary variety V. His algorithm is quite complicated, and enormously complicated is the proof of its correctness.

Resolution of singularities of varieties

Let V be a variety over a field of characteristic 0. A *smooth* variety U is a **resolution of singularities** of V if $\dim U = \dim V$ and there exists an epimorphism $\phi : U \to V$ such that ϕ is a local isomorphism at a neighborhood of any *smooth* point of V.

Hironaka (1964) has designed an algorithm for resolution of singularities of an arbitrary variety V. His algorithm is quite complicated, and enormously complicated is the proof of its correctness.

Resolution of singularities of varieties

Let V be a variety over a field of characteristic 0. A *smooth* variety U is a **resolution of singularities** of V if $\dim U = \dim V$ and there exists an epimorphism $\phi : U \to V$ such that ϕ is a local isomorphism at a neighborhood of any *smooth* point of V.

Hironaka (1964) has designed an algorithm for resolution of singularities of an arbitrary variety V. His algorithm is quite complicated, and enormously complicated is the proof of its correctness.

Resolution of singularities of varieties

Let V be a variety over a field of characteristic 0. A *smooth* variety U is a **resolution of singularities** of V if $\dim U = \dim V$ and there exists an epimorphism $\phi : U \to V$ such that ϕ is a local isomorphism at a neighborhood of any *smooth* point of V.

Hironaka (1964) has designed an algorithm for resolution of singularities of an arbitrary variety V. His algorithm is quite complicated, and enormously complicated is the proof of its correctness.

Resolution of singularities of varieties

Let V be a variety over a field of characteristic 0. A *smooth* variety U is a **resolution of singularities** of V if $\dim U = \dim V$ and there exists an epimorphism $\phi : U \to V$ such that ϕ is a local isomorphism at a neighborhood of any *smooth* point of V.

Hironaka (1964) has designed an algorithm for resolution of singularities of an arbitrary variety V. His algorithm is quite complicated, and enormously complicated is the proof of its correctness.

Complexity of resolution of singularities

To formulate a complexity bound of resolution of singularities briefly remind the Grzegorczyk's hierarchy $\mathcal{E}^0 \subset \mathcal{E}^1 \subset \cdots$ of the primitive-recursive functions. Each class $\mathcal{E}^k$ is closed under the composition. $\mathcal{E}^0$ contains the functions $x \mapsto x + const$. Class $\mathcal{E}^1$ contains the linear functions $x \mapsto const \cdot x$. Class $\mathcal{E}^2$ contains all the polynomials. If functions $h(X_1, \dots, X_n)$, $g(Z, Y, X_1, \dots, X_n) \in \mathcal{E}^k$ then the function $f(Y, X_1, \dots, X_n)$ defined by recursion

- $f(0, X_1, \dots, X_n) = h(X_1, \dots, X_n)$;
- $f(Y + 1, X_1, \dots, X_n) = g(f(Y, X_1, \dots, X_n), Y, X_1, \dots, X_n)$

belongs to $\mathcal{E}^{k+1}$. In particular, $\mathcal{E}^3$ contains all the towers of the exponential function.

The complexity of resolution of singularities of a variety V with $m = \dim V$ can be bounded by a suitable function from the class $\mathcal{E}^{m+3}$ (Bierstone-G.-Milman-Wlodarczyk (2010)).

Complexity of resolution of singularities

To formulate a complexity bound of resolution of singularities briefly remind the Grzegorczyk's hierarchy $\mathcal{E}^0 \subset \mathcal{E}^1 \subset \cdots$ of the primitive-recursive functions. Each class $\mathcal{E}^k$ is closed under the composition. $\mathcal{E}^0$ contains the functions $x \mapsto x + const$. Class $\mathcal{E}^1$ contains the linear functions $x \mapsto const \cdot x$. Class $\mathcal{E}^2$ contains all the polynomials. If functions $h(X_1, \dots, X_n), g(Z, Y, X_1, \dots, X_n) \in \mathcal{E}^k$ then the function $f(Y, X_1, \dots, X_n)$ defined by recursion

- $f(0, X_1, \dots, X_n) = h(X_1, \dots, X_n)$;
- $f(Y + 1, X_1, \dots, X_n) = g(f(Y, X_1, \dots, X_n), Y, X_1, \dots, X_n)$

belongs to $\mathcal{E}^{k+1}$. In particular, $\mathcal{E}^3$ contains all the towers of the exponential function.

The complexity of resolution of singularities of a variety V with $m = \dim V$ can be bounded by a suitable function from the class $\mathcal{E}^{m+3}$ (Bierstone-G.-Milman-Wlodarczyk (2010)).

Complexity of resolution of singularities

To formulate a complexity bound of resolution of singularities briefly remind the Grzegorczyk's hierarchy $\mathcal{E}^0 \subset \mathcal{E}^1 \subset \cdots$ of the primitive-recursive functions. Each class $\mathcal{E}^k$ is closed under the composition. $\mathcal{E}^0$ contains the functions $x \mapsto x + const$. Class $\mathcal{E}^1$ contains the linear functions $x \mapsto const \cdot x$. Class $\mathcal{E}^2$ contains all the polynomials. If functions $h(X_1, \ldots, X_n), g(Z, Y, X_1, \ldots, X_n) \in \mathcal{E}^k$ then the function $f(Y, X_1, \ldots, X_n)$ defined by recursion

- $f(0, X_1, \ldots, X_n) = h(X_1, \ldots, X_n)$;
- $f(Y + 1, X_1, \ldots, X_n) = g(f(Y, X_1, \ldots, X_n), Y, X_1, \ldots, X_n)$

belongs to $\mathcal{E}^{k+1}$. In particular, $\mathcal{E}^3$ contains all the towers of the exponential function.

The complexity of resolution of singularities of a variety V with $m = \dim V$ can be bounded by a suitable function from the class $\mathcal{E}^{m+3}$ (Bierstone-G.-Milman-Wlodarczyk (2010)).

Complexity of resolution of singularities

To formulate a complexity bound of resolution of singularities briefly remind the Grzegorczyk's hierarchy $\mathcal{E}^0 \subset \mathcal{E}^1 \subset \cdots$ of the primitive-recursive functions. Each class $\mathcal{E}^k$ is closed under the composition. $\mathcal{E}^0$ contains the functions $x \mapsto x + const$. Class $\mathcal{E}^1$ contains the linear functions $x \mapsto const \cdot x$. Class $\mathcal{E}^2$ contains all the polynomials. If functions $h(X_1, \dots, X_n)$, $g(Z, Y, X_1, \dots, X_n) \in \mathcal{E}^k$ then the function $f(Y, X_1, \dots, X_n)$ defined by recursion

- $f(0, X_1, \dots, X_n) = h(X_1, \dots, X_n)$;
- $f(Y + 1, X_1, \dots, X_n) = g(f(Y, X_1, \dots, X_n), Y, X_1, \dots, X_n)$

belongs to $\mathcal{E}^{k+1}$. In particular, $\mathcal{E}^3$ contains all the towers of the exponential function.

The complexity of resolution of singularities of a variety V with $m = \dim V$ can be bounded by a suitable function from the class $\mathcal{E}^{m+3}$ (Bierstone-G.-Milman-Wlodarczyk (2010)).

Complexity of resolution of singularities

To formulate a complexity bound of resolution of singularities briefly remind the Grzegorczyk's hierarchy $\mathcal{E}^0 \subset \mathcal{E}^1 \subset \cdots$ of the primitive-recursive functions. Each class $\mathcal{E}^k$ is closed under the composition. $\mathcal{E}^0$ contains the functions $x \mapsto x + const$. Class $\mathcal{E}^1$ contains the linear functions $x \mapsto const \cdot x$. Class $\mathcal{E}^2$ contains all the polynomials. If functions $h(X_1, \dots, X_n)$, $g(Z, Y, X_1, \dots, X_n) \in \mathcal{E}^k$ then the function $f(Y, X_1, \dots, X_n)$ defined by recursion

- $f(0, X_1, \dots, X_n) = h(X_1, \dots, X_n)$;
- $f(Y+1, X_1, \dots, X_n) = g(f(Y, X_1, \dots, X_n), Y, X_1, \dots, X_n)$

belongs to $\mathcal{E}^{k+1}$. In particular, $\mathcal{E}^3$ contains all the towers of the exponential function.

The complexity of resolution of singularities of a variety V with $m = \dim V$ can be bounded by a suitable function from the class $\mathcal{E}^{m+3}$ (Bierstone-G.-Milman-Wlodarczyk (2010)).

Complexity of resolution of singularities

To formulate a complexity bound of resolution of singularities briefly remind the Grzegorczyk's hierarchy $\mathcal{E}^0 \subset \mathcal{E}^1 \subset \cdots$ of the primitive-recursive functions. Each class $\mathcal{E}^k$ is closed under the composition. $\mathcal{E}^0$ contains the functions $x \mapsto x + const$. Class $\mathcal{E}^1$ contains the linear functions $x \mapsto const \cdot x$. Class $\mathcal{E}^2$ contains all the polynomials. If functions $h(X_1, \ldots, X_n)$, $g(Z, Y, X_1, \ldots, X_n) \in \mathcal{E}^k$ then the function $f(Y, X_1, \ldots, X_n)$ defined by recursion

- $f(0, X_1, \ldots, X_n) = h(X_1, \ldots, X_n)$;
- $f(Y + 1, X_1, \ldots, X_n) = g(f(Y, X_1, \ldots, X_n), Y, X_1, \ldots, X_n)$

belongs to $\mathcal{E}^{k+1}$. In particular, $\mathcal{E}^3$ contains all the towers of the exponential function.

The complexity of resolution of singularities of a variety V with $m = \dim V$ can be bounded by a suitable function from the class $\mathcal{E}^{m+3}$ (Bierstone-G.-Milman-Wlodarczyk (2010)).

Complexity of resolution of singularities

To formulate a complexity bound of resolution of singularities briefly remind the Grzegorczyk's hierarchy $\mathcal{E}^0 \subset \mathcal{E}^1 \subset \cdots$ of the primitive-recursive functions. Each class $\mathcal{E}^k$ is closed under the composition. $\mathcal{E}^0$ contains the functions $x \mapsto x + const$. Class $\mathcal{E}^1$ contains the linear functions $x \mapsto const \cdot x$. Class $\mathcal{E}^2$ contains all the polynomials. If functions $h(X_1, \dots, X_n)$, $g(Z, Y, X_1, \dots, X_n) \in \mathcal{E}^k$ then the function $f(Y, X_1, \dots, X_n)$ defined by recursion

- $f(0, X_1, \dots, X_n) = h(X_1, \dots, X_n)$;
- $f(Y + 1, X_1, \dots, X_n) = g(f(Y, X_1, \dots, X_n), Y, X_1, \dots, X_n)$

belongs to $\mathcal{E}^{k+1}$. In particular, $\mathcal{E}^3$ contains all the towers of the exponential function.

The complexity of resolution of singularities of a variety V with $m = \dim V$ can be bounded by a suitable function from the class $\mathcal{E}^{m+3}$ (Bierstone-G.-Milman-Wlodarczyk (2010)).

Complexity of resolution of singularities

To formulate a complexity bound of resolution of singularities briefly remind the Grzegorczyk's hierarchy $\mathcal{E}^0 \subset \mathcal{E}^1 \subset \cdots$ of the primitive-recursive functions. Each class $\mathcal{E}^k$ is closed under the composition. $\mathcal{E}^0$ contains the functions $x \mapsto x + const$. Class $\mathcal{E}^1$ contains the linear functions $x \mapsto const \cdot x$. Class $\mathcal{E}^2$ contains all the polynomials. If functions $h(X_1, \dots, X_n)$, $g(Z, Y, X_1, \dots, X_n) \in \mathcal{E}^k$ then the function $f(Y, X_1, \dots, X_n)$ defined by recursion

- $f(0, X_1, \dots, X_n) = h(X_1, \dots, X_n)$;
- $f(Y + 1, X_1, \dots, X_n) = g(f(Y, X_1, \dots, X_n), Y, X_1, \dots, X_n)$

belongs to $\mathcal{E}^{k+1}$. In particular, $\mathcal{E}^3$ contains all the towers of the exponential function.

The complexity of resolution of singularities of a variety V with $m = \dim V$ can be bounded by a suitable function from the class $\mathcal{E}^{m+3}$ (Bierstone-G.-Milman-Wlodarczyk (2010)).

Complexity of resolution of singularities

To formulate a complexity bound of resolution of singularities briefly remind the Grzegorczyk's hierarchy $\mathcal{E}^0 \subset \mathcal{E}^1 \subset \cdots$ of the primitive-recursive functions. Each class $\mathcal{E}^k$ is closed under the composition. $\mathcal{E}^0$ contains the functions $x \mapsto x + const$. Class $\mathcal{E}^1$ contains the linear functions $x \mapsto const \cdot x$. Class $\mathcal{E}^2$ contains all the polynomials. If functions $h(X_1, \dots, X_n)$, $g(Z, Y, X_1, \dots, X_n) \in \mathcal{E}^k$ then the function $f(Y, X_1, \dots, X_n)$ defined by recursion

- $f(0, X_1, \dots, X_n) = h(X_1, \dots, X_n)$;
- $f(Y + 1, X_1, \dots, X_n) = g(f(Y, X_1, \dots, X_n), Y, X_1, \dots, X_n)$

belongs to $\mathcal{E}^{k+1}$. In particular, $\mathcal{E}^3$ contains all the towers of the exponential function.

The complexity of resolution of singularities of a variety V with $m = \dim V$ can be bounded by a suitable function from the class $\mathcal{E}^{m+3}$ (Bierstone-G.-Milman-Wlodarczyk (2010)).

Complexity of resolution of singularities

To formulate a complexity bound of resolution of singularities briefly remind the Grzegorczyk's hierarchy $\mathcal{E}^0 \subset \mathcal{E}^1 \subset \cdots$ of the primitive-recursive functions. Each class $\mathcal{E}^k$ is closed under the composition. $\mathcal{E}^0$ contains the functions $x \mapsto x + const$. Class $\mathcal{E}^1$ contains the linear functions $x \mapsto const \cdot x$. Class $\mathcal{E}^2$ contains all the polynomials. If functions $h(X_1, \dots, X_n)$, $g(Z, Y, X_1, \dots, X_n) \in \mathcal{E}^k$ then the function $f(Y, X_1, \dots, X_n)$ defined by recursion

- $f(0, X_1, \dots, X_n) = h(X_1, \dots, X_n)$;
- $f(Y + 1, X_1, \dots, X_n) = g(f(Y, X_1, \dots, X_n), Y, X_1, \dots, X_n)$

belongs to $\mathcal{E}^{k+1}$. In particular, $\mathcal{E}^3$ contains all the towers of the exponential function.

The complexity of resolution of singularities of a variety V with $m = \dim V$ can be bounded by a suitable function from the class $\mathcal{E}^{m+3}$ (Bierstone-G.-Milman-Wlodarczyk (2010)).

Complexity in symbolic differential algebra

Manipulating with differential polynomials or differential operators differs from manipulating with algebraic polynomials. The principal difference is that unlike the computer algebra, there are no universal methods for solving equations in the differential algebra. That is why a single differential equation can be studied for a couple of centuries. Many quite natural problems in differential algebra are algorithmically undecidable.

Therefore, one can rarely produce algorithms in differential algebra. I'll give two examples of such algorithms. The first one concerns the quantifier elimination in differentially closed fields. While any algebraic equation has a solution in an *algebraically* closed field, any non-linear differential equation has a solution in a *differentially* closed field. Thus, the latter is an uncomprehensible object whose existence is justified by the axiom of choice.

Complexity in symbolic differential algebra

Manipulating with differential polynomials or differential operators differs from manipulating with algebraic polynomials. The principal difference is that unlike the computer algebra, there are no universal methods for solving equations in the differential algebra. That is why a single differential equation can be studied for a couple of centuries. Many quite natural problems in differential algebra are algorithmically undecidable.

Therefore, one can rarely produce algorithms in differential algebra. I'll give two examples of such algorithms. The first one concerns the quantifier elimination in differentially closed fields. While any algebraic equation has a solution in an *algebraically* closed field, any non-linear differential equation has a solution in a *differentially* closed field. Thus, the latter is an uncomprehensible object whose existence is justified by the axiom of choice.

Complexity in symbolic differential algebra

Manipulating with differential polynomials or differential operators differs from manipulating with algebraic polynomials. The principal difference is that unlike the computer algebra, there are no universal methods for solving equations in the differential algebra. That is why a single differential equation can be studied for a couple of centuries. Many quite natural problems in differential algebra are algorithmically undecidable.

Therefore, one can rarely produce algorithms in differential algebra. I'll give two examples of such algorithms. The first one concerns the quantifier elimination in differentially closed fields. While any algebraic equation has a solution in an *algebraically* closed field, any non-linear differential equation has a solution in a *differentially* closed field. Thus, the latter is an uncomprehensible object whose existence is justified by the axiom of choice.

Complexity in symbolic differential algebra

Manipulating with differential polynomials or differential operators differs from manipulating with algebraic polynomials. The principal difference is that unlike the computer algebra, there are no universal methods for solving equations in the differential algebra. That is why a single differential equation can be studied for a couple of centuries. Many quite natural problems in differential algebra are algorithmically undecidable.

Therefore, one can rarely produce algorithms in differential algebra. I'll give two examples of such algorithms. The first one concerns the quantifier elimination in differentially closed fields. While any algebraic equation has a solution in an *algebraically* closed field, any non-linear differential equation has a solution in a *differentially* closed field. Thus, the latter is an uncomprehensible object whose existence is justified by the axiom of choice.

Complexity in symbolic differential algebra

Manipulating with differential polynomials or differential operators differs from manipulating with algebraic polynomials. The principal difference is that unlike the computer algebra, there are no universal methods for solving equations in the differential algebra. That is why a single differential equation can be studied for a couple of centuries. Many quite natural problems in differential algebra are algorithmically undecidable.

Therefore, one can rarely produce algorithms in differential algebra. I'll give two examples of such algorithms. The first one concerns the quantifier elimination in differentially closed fields. While any algebraic equation has a solution in an *algebraically* closed field, any non-linear differential equation has a solution in a *differentially* closed field. Thus, the latter is an uncomprehensible object whose existence is justified by the axiom of choice.

Complexity in symbolic differential algebra

Manipulating with differential polynomials or differential operators differs from manipulating with algebraic polynomials. The principal difference is that unlike the computer algebra, there are no universal methods for solving equations in the differential algebra. That is why a single differential equation can be studied for a couple of centuries. Many quite natural problems in differential algebra are algorithmically undecidable.

Therefore, one can rarely produce algorithms in differential algebra. I'll give two examples of such algorithms. The first one concerns the quantifier elimination in differentially closed fields. While any algebraic equation has a solution in an *algebraically* closed field, any non-linear differential equation has a solution in a *differentially* closed field. Thus, the latter is an uncomprehensible object whose existence is justified by the axiom of choice.

Complexity in symbolic differential algebra

Manipulating with differential polynomials or differential operators differs from manipulating with algebraic polynomials. The principal difference is that unlike the computer algebra, there are no universal methods for solving equations in the differential algebra. That is why a single differential equation can be studied for a couple of centuries. Many quite natural problems in differential algebra are algorithmically undecidable.

Therefore, one can rarely produce algorithms in differential algebra. I'll give two examples of such algorithms. The first one concerns the quantifier elimination in differentially closed fields. While any algebraic equation has a solution in an *algebraically* closed field, any non-linear differential equation has a solution in a *differentially* closed field. Thus, the latter is an uncomprehensible object whose existence is justified by the axiom of choice.

Complexity in symbolic differential algebra

Manipulating with differential polynomials or differential operators differs from manipulating with algebraic polynomials. The principal difference is that unlike the computer algebra, there are no universal methods for solving equations in the differential algebra. That is why a single differential equation can be studied for a couple of centuries. Many quite natural problems in differential algebra are algorithmically undecidable.

Therefore, one can rarely produce algorithms in differential algebra. I'll give two examples of such algorithms. The first one concerns the quantifier elimination in differentially closed fields. While any algebraic equation has a solution in an *algebraically* closed field, any non-linear differential equation has a solution in a *differentially* closed field. Thus, the latter is an uncomprehensible object whose existence is justified by the axiom of choice.

Complexity in symbolic differential algebra

Manipulating with differential polynomials or differential operators differs from manipulating with algebraic polynomials. The principal difference is that unlike the computer algebra, there are no universal methods for solving equations in the differential algebra. That is why a single differential equation can be studied for a couple of centuries. Many quite natural problems in differential algebra are algorithmically undecidable.

Therefore, one can rarely produce algorithms in differential algebra. I'll give two examples of such algorithms. The first one concerns the quantifier elimination in differentially closed fields. While any algebraic equation has a solution in an *algebraically* closed field, any non-linear differential equation has a solution in a *differentially* closed field. Thus, the latter is an uncomprehensible object whose existence is justified by the axiom of choice.

Complexity in symbolic differential algebra

Manipulating with differential polynomials or differential operators differs from manipulating with algebraic polynomials. The principal difference is that unlike the computer algebra, there are no universal methods for solving equations in the differential algebra. That is why a single differential equation can be studied for a couple of centuries. Many quite natural problems in differential algebra are algorithmically undecidable.

Therefore, one can rarely produce algorithms in differential algebra. I'll give two examples of such algorithms. The first one concerns the quantifier elimination in differentially closed fields. While any algebraic equation has a solution in an *algebraically* closed field, any non-linear differential equation has a solution in a *differentially* closed field. Thus, the latter is an uncomprehensible object whose existence is justified by the axiom of choice.

Complexity of quantifier elimination in the first-order theory of differentially closed fields

Consider a formula

$$\exists X_{11} \cdots \exists X_{1n_1} \forall X_{21} \cdots \forall X_{2n_2} \cdots \exists X_{a1} \cdots \exists X_{an_a} Q$$

where atomic subformulas of the quantifier-free formula Q are of the type $f = 0$ for differential polynomials f with respect to m derivatives $\partial/\partial t_1, \dots, \partial/\partial t_m$.

Seidenberg (1956): a quantifier elimination algorithm which yields a quantifier-free formula equivalent over a differentially closed field. Its complexity can be estimated by a suitable function from the Grzegorczyk's class $\mathcal{E}^{m+2}$.

The proof relies on a similar bound (also established by Seidenberg) for the Hilbert's Idealbasissatz: any ascending chain of ideals $I_1 \subset I_2 \subset \cdots \subset F[t_1, \dots, t_m]$ eventually stabilizes.

In case of *ordinary* ($m = 1$) differentially closed fields G. (1986): a quantifier elimination with a triple-exponential complexity bound better than a quadruple-exponential bound in the Seidenberg's algorithm.

Complexity of quantifier elimination in the first-order theory of differentially closed fields

Consider a formula

$$\exists X_{11} \cdots \exists X_{1n_1} \forall X_{21} \cdots \forall X_{2n_2} \cdots \exists X_{a1} \cdots \exists X_{an_a} Q$$

where atomic subformulas of the quantifier-free formula Q are of the type $f = 0$ for differential polynomials f with respect to m derivatives $\partial/\partial t_1, \dots, \partial/\partial t_m$.

Seidenberg (1956): a quantifier elimination algorithm which yields a quantifier-free formula equivalent over a differentially closed field. Its complexity can be estimated by a suitable function from the Grzegorczyk's class $\mathcal{E}^{m+2}$.

The proof relies on a similar bound (also established by Seidenberg) for the Hilbert's Idealbasissatz: any ascending chain of ideals $I_1 \subset I_2 \subset \cdots \subset F[t_1, \dots, t_m]$ eventually stabilizes.

In case of *ordinary* ($m = 1$) differentially closed fields G. (1986): a quantifier elimination with a triple-exponential complexity bound better than a quadruple-exponential bound in the Seidenberg's algorithm.

Complexity of quantifier elimination in the first-order theory of differentially closed fields

Consider a formula

$$\exists X_{11} \cdots \exists X_{1n_1} \forall X_{21} \cdots \forall X_{2n_2} \cdots \exists X_{a1} \cdots \exists X_{an_a} Q$$

where atomic subformulas of the quantifier-free formula Q are of the type $f = 0$ for differential polynomials f with respect to m derivatives $\partial/\partial t_1, \dots, \partial/\partial t_m$.

Seidenberg (1956): a quantifier elimination algorithm which yields a quantifier-free formula equivalent over a differentially closed field. Its complexity can be estimated by a suitable function from the Grzegorczyk's class $\mathcal{E}^{m+2}$.

The proof relies on a similar bound (also established by Seidenberg) for the Hilbert's Idealbasissatz: any ascending chain of ideals $I_1 \subset I_2 \subset \cdots \subset F[t_1, \dots, t_m]$ eventually stabilizes.

In case of *ordinary* ($m = 1$) differentially closed fields G. (1986): a quantifier elimination with a triple-exponential complexity bound better than a quadruple-exponential bound in the Seidenberg's algorithm.

Complexity of quantifier elimination in the first-order theory of differentially closed fields

Consider a formula

$$\exists X_{11} \cdots \exists X_{1n_1} \forall X_{21} \cdots \forall X_{2n_2} \cdots \exists X_{a1} \cdots \exists X_{an_a} Q$$

where atomic subformulas of the quantifier-free formula Q are of the type $f = 0$ for differential polynomials f with respect to m derivatives $\partial/\partial t_1, \dots, \partial/\partial t_m$.

Seidenberg (1956): a quantifier elimination algorithm which yields a quantifier-free formula equivalent over a differentially closed field. Its complexity can be estimated by a suitable function from the Grzegorczyk's class $\mathcal{E}^{m+2}$.

The proof relies on a similar bound (also established by Seidenberg) for the Hilbert's Idealbasissatz: any ascending chain of ideals $I_1 \subset I_2 \subset \cdots \subset F[t_1, \dots, t_m]$ eventually stabilizes.

In case of *ordinary* ($m = 1$) differentially closed fields G. (1986): a quantifier elimination with a triple-exponential complexity bound better than a quadruple-exponential bound in the Seidenberg's algorithm.

Complexity of quantifier elimination in the first-order theory of differentially closed fields

Consider a formula

$$\exists X_{11} \cdots \exists X_{1n_1} \forall X_{21} \cdots \forall X_{2n_2} \cdots \exists X_{a1} \cdots \exists X_{an_a} Q$$

where atomic subformulas of the quantifier-free formula Q are of the type $f = 0$ for differential polynomials f with respect to m derivatives $\partial/\partial t_1, \dots, \partial/\partial t_m$.

Seidenberg (1956): a quantifier elimination algorithm which yields a quantifier-free formula equivalent over a differentially closed field. Its complexity can be estimated by a suitable function from the Grzegorczyk's class $\mathcal{E}^{m+2}$.

The proof relies on a similar bound (also established by Seidenberg) for the Hilbert's Idealbasissatz: any ascending chain of ideals $I_1 \subset I_2 \subset \cdots \subset F[t_1, \dots, t_m]$ eventually stabilizes.

In case of *ordinary* ($m = 1$) differentially closed fields G. (1986): a quantifier elimination with a triple-exponential complexity bound better than a quadruple-exponential bound in the Seidenberg's algorithm.

Complexity of quantifier elimination in the first-order theory of differentially closed fields

Consider a formula

$$\exists X_{11} \cdots \exists X_{1n_1} \forall X_{21} \cdots \forall X_{2n_2} \cdots \exists X_{a1} \cdots \exists X_{an_a} Q$$

where atomic subformulas of the quantifier-free formula Q are of the type $f = 0$ for differential polynomials f with respect to m derivatives $\partial/\partial t_1, \ldots, \partial/\partial t_m$.

Seidenberg (1956): a quantifier elimination algorithm which yields a quantifier-free formula equivalent over a differentially closed field. Its complexity can be estimated by a suitable function from the Grzegorczyk's class $\mathcal{E}^{m+2}$.

The proof relies on a similar bound (also established by Seidenberg) for the Hilbert's Idealbasissatz: any ascending chain of ideals $I_1 \subset I_2 \subset \cdots \subset F[t_1, \ldots, t_m]$ eventually stabilizes.

In case of *ordinary* ($m = 1$) differentially closed fields G. (1986): a quantifier elimination with a triple-exponential complexity bound better than a quadruple-exponential bound in the Seidenberg's algorithm

Complexity of quantifier elimination in the first-order theory of differentially closed fields

Consider a formula

$$\exists X_{11} \cdots \exists X_{1n_1} \forall X_{21} \cdots \forall X_{2n_2} \cdots \exists X_{a1} \cdots \exists X_{an_a} Q$$

where atomic subformulas of the quantifier-free formula Q are of the type $f = 0$ for differential polynomials f with respect to m derivatives $\partial/\partial t_1, \ldots, \partial/\partial t_m$.

Seidenberg (1956): a quantifier elimination algorithm which yields a quantifier-free formula equivalent over a differentially closed field. Its complexity can be estimated by a suitable function from the Grzegorczyk's class $\mathcal{E}^{m+2}$.

The proof relies on a similar bound (also established by Seidenberg) for the Hilbert's Idealbasissatz: any ascending chain of ideals $I_1 \subset I_2 \subset \cdots \subset F[t_1, \ldots, t_m]$ eventually stabilizes.

In case of *ordinary* ($m = 1$) differentially closed fields G. (1986): a quantifier elimination with a triple-exponential complexity bound better than a quadruple-exponential bound in the Seidenberg's algorithm.

Complexity of factoring linear ordinary differential operators

Another algorithm in differential algebra is the one for factoring linear ordinary differential operators $L = \sum_j b_j \cdot \frac{d^j}{dt^j} \in \mathbb{C}(t)[\frac{d}{dt}]$ with rational functions coefficients $b_j \in \mathbb{C}(t)$. The factoring problem is to produce irreducible operators $L_1, \dots, L_k \in \mathbb{C}(t)[\frac{d}{dt}]$ such that $L = L_1 \circ \cdots \circ L_k$. The ring $\mathbb{C}(t)[\frac{d}{dt}]$ of differential operators is not commutative, although has some common features with the polynomial ring, in particular admits both left and right divisions with remainder.

Beke-Schlesinger (1895): factoring algorithm with the triple-exponential complexity. G. (1986): a factoring algorithm with the double-exponential complexity. Conjecture: the complexity of factoring is exponential.

Complexity of factoring linear ordinary differential operators

Another algorithm in differential algebra is the one for factoring linear ordinary differential operators $L = \sum_j b_j \cdot \frac{d^j}{dt^j} \in \mathbb{C}(t)[\frac{d}{dt}]$ with rational functions coefficients $b_j \in \mathbb{C}(t)$. The factoring problem is to produce irreducible operators $L_1, \dots, L_k \in \mathbb{C}(t)[\frac{d}{dt}]$ such that $L = L_1 \circ \cdots \circ L_k$.

The ring $\mathbb{C}(t)[\frac{d}{dt}]$ of differential operators is not commutative, although has some common features with the polynomial ring, in particular admits both left and right divisions with remainder.

Beke-Schlesinger (1895): factoring algorithm with the triple-exponential complexity. G. (1986): a factoring algorithm with the double-exponential complexity. Conjecture: the complexity of factoring is exponential.

Complexity of factoring linear ordinary differential operators

Another algorithm in differential algebra is the one for factoring linear ordinary differential operators $L = \sum_j b_j \cdot \frac{d^j}{dt^j} \in \mathbb{C}(t)[\frac{d}{dt}]$ with rational functions coefficients $b_j \in \mathbb{C}(t)$. The factoring problem is to produce irreducible operators $L_1, \dots, L_k \in \mathbb{C}(t)[\frac{d}{dt}]$ such that $L = L_1 \circ \cdots \circ L_k$. The ring $\mathbb{C}(t)[\frac{d}{dt}]$ of differential operators is not commutative, although has some common features with the polynomial ring, in particular admits both left and right divisions with remainder.

Beke-Schlesinger (1895): factoring algorithm with the triple-exponential complexity. G. (1986): a factoring algorithm with the double-exponential complexity. Conjecture: the complexity of factoring is exponential.

Complexity of factoring linear ordinary differential operators

Another algorithm in differential algebra is the one for factoring linear ordinary differential operators $L = \sum_j b_j \cdot \frac{d^j}{dt^j} \in \mathbb{C}(t)[\frac{d}{dt}]$ with rational functions coefficients $b_j \in \mathbb{C}(t)$. The factoring problem is to produce irreducible operators $L_1, \dots, L_k \in \mathbb{C}(t)[\frac{d}{dt}]$ such that $L = L_1 \circ \cdots \circ L_k$. The ring $\mathbb{C}(t)[\frac{d}{dt}]$ of differential operators is not commutative, although has some common features with the polynomial ring, in particular admits both left and right divisions with remainder.

Beke-Schlesinger (1895): factoring algorithm with the triple-exponential complexity. G. (1986): a factoring algorithm with the double-exponential complexity. Conjecture: the complexity of factoring is exponential.

Complexity of factoring linear ordinary differential operators

Another algorithm in differential algebra is the one for factoring linear ordinary differential operators $L = \sum_j b_j \cdot \frac{d^j}{dt^j} \in \mathbb{C}(t)[\frac{d}{dt}]$ with rational functions coefficients $b_j \in \mathbb{C}(t)$. The factoring problem is to produce irreducible operators $L_1, \dots, L_k \in \mathbb{C}(t)[\frac{d}{dt}]$ such that $L = L_1 \circ \cdots \circ L_k$. The ring $\mathbb{C}(t)[\frac{d}{dt}]$ of differential operators is not commutative, although has some common features with the polynomial ring, in particular admits both left and right divisions with remainder.

Beke-Schlesinger (1895): factoring algorithm with the triple-exponential complexity. G. (1986): a factoring algorithm with the double-exponential complexity. Conjecture: the complexity of factoring is exponential.

Complexity of factoring linear ordinary differential operators

Another algorithm in differential algebra is the one for factoring linear ordinary differential operators $L = \sum_j b_j \cdot \frac{d^j}{dt^j} \in \mathbb{C}(t)[\frac{d}{dt}]$ with rational functions coefficients $b_j \in \mathbb{C}(t)$. The factoring problem is to produce irreducible operators $L_1, \dots, L_k \in \mathbb{C}(t)[\frac{d}{dt}]$ such that $L = L_1 \circ \cdots \circ L_k$. The ring $\mathbb{C}(t)[\frac{d}{dt}]$ of differential operators is not commutative, although has some common features with the polynomial ring, in particular admits both left and right divisions with remainder.

Beke-Schlesinger (1895): factoring algorithm with the triple-exponential complexity. G. (1986): a factoring algorithm with the double-exponential complexity. Conjecture: the complexity of factoring is exponential.

Complexity of factoring linear ordinary differential operators

Another algorithm in differential algebra is the one for factoring linear ordinary differential operators $L = \sum_j b_j \cdot \frac{d^j}{dt^j} \in \mathbb{C}(t)[\frac{d}{dt}]$ with rational functions coefficients $b_j \in \mathbb{C}(t)$. The factoring problem is to produce irreducible operators $L_1, \dots, L_k \in \mathbb{C}(t)[\frac{d}{dt}]$ such that $L = L_1 \circ \cdots \circ L_k$. The ring $\mathbb{C}(t)[\frac{d}{dt}]$ of differential operators is not commutative, although has some common features with the polynomial ring, in particular admits both left and right divisions with remainder.

Beke-Schlesinger (1895): factoring algorithm with the triple-exponential complexity. G. (1986): a factoring algorithm with the double-exponential complexity. Conjecture: the complexity of factoring is exponential.

Approximations and complexity

So far, we considered symbolic computations. Another type of data rather than symbolic, are numeric. Since now we study computations with approximations, a question arises about a relation between approximations and complexity. Roughly speaking, the better approximation one desires to achieve the more time one has to spend.

In this direction the Liouville's theorem on approximation of algebraic numbers is known. Namely, if algebraic numbers $a \neq b$ are roots of polynomials $f(a) = g(b) = 0$, $f, g \in \mathbb{Z}[Y]$, where $\deg(f), \deg(g) < n$ and the integer coefficients of f, g have absolute values less than M then $|a - b| > M^{-O(n)}$. Thus, if one wants to approximate a fixed number a with a sufficiently good rate, one needs to increase the complexity of b determined by a polynomial g.

Can one extend this phenomenon of the trade-off between approximations and complexity to solutions of differential, rather than polynomial equations? It appeared that this holds for two classes of differential equations, and off these two classes the phenomenon fails. Denote by $\exp^{(n)} = \exp \circ \cdots \circ \exp$ the n times iteration of exp.

Approximations and complexity

So far, we considered symbolic computations. Another type of data rather than symbolic, are numeric. Since now we study computations with approximations, a question arises about a relation between approximations and complexity. Roughly speaking, the better approximation one desires to achieve the more time one has to spend.

In this direction the Liouville's theorem on approximation of algebraic numbers is known. Namely, if algebraic numbers $a \neq b$ are roots of polynomials $f(a) = g(b) = 0$, $f, g \in \mathbb{Z}[Y]$, where $\deg(f), \deg(g) < n$ and the integer coefficients of f, g have absolute values less than M then $|a - b| > M^{-O(n)}$. Thus, if one wants to approximate a fixed number a with a sufficiently good rate, one needs to increase the complexity of b determined by a polynomial g.

Can one extend this phenomenon of the trade-off between approximations and complexity to solutions of differential, rather than polynomial equations? It appeared that this holds for two classes of differential equations, and off these two classes the phenomenon fails.
Denote by $\exp^{(n)} = \exp \circ \cdots \circ \exp$ the n times iteration of exp

Approximations and complexity

So far, we considered symbolic computations. Another type of data rather than symbolic, are numeric. Since now we study computations with approximations, a question arises about a relation between approximations and complexity. Roughly speaking, the better approximation one desires to achieve the more time one has to spend.

In this direction the Liouville's theorem on approximation of algebraic numbers is known. Namely, if algebraic numbers $a \neq b$ are roots of polynomials $f(a) = g(b) = 0$, $f, g \in \mathbb{Z}[Y]$, where $\deg(f), \deg(g) < n$ and the integer coefficients of f, g have absolute values less than M then $|a - b| > M^{-O(n)}$. Thus, if one wants to approximate a fixed number a with a sufficiently good rate, one needs to increase the complexity of b determined by a polynomial g.

Can one extend this phenomenon of the trade-off between approximations and complexity to solutions of differential, rather than polynomial equations? It appeared that this holds for two classes of differential equations, and off these two classes the phenomenon fails. Denote by $\exp^{(n)} = \exp \circ \cdots \circ \exp$ the n times iteration of exp

Approximations and complexity

So far, we considered symbolic computations. Another type of data rather than symbolic, are numeric. Since now we study computations with approximations, a question arises about a relation between approximations and complexity. Roughly speaking, the better approximation one desires to achieve the more time one has to spend.

In this direction the Liouville's theorem on approximation of algebraic numbers is known. Namely, if algebraic numbers $a \neq b$ are roots of polynomials $f(a) = g(b) = 0$, $f, g \in \mathbb{Z}[Y]$, where $\deg(f), \deg(g) < n$ and the integer coefficients of f, g have absolute values less than M then $|a - b| > M^{-O(n)}$. Thus, if one wants to approximate a fixed number a with a sufficiently good rate, one needs to increase the complexity of b determined by a polynomial g.

Can one extend this phenomenon of the trade-off between approximations and complexity to solutions of differential, rather than polynomial equations? It appeared that this holds for two classes of differential equations, and off these two classes the phenomenon fails. Denote by $\exp^{(n)} = \exp \circ \cdots \circ \exp$ the n times iteration of exp

Approximations and complexity

So far, we considered symbolic computations. Another type of data rather than symbolic, are numeric. Since now we study computations with approximations, a question arises about a relation between approximations and complexity. Roughly speaking, the better approximation one desires to achieve the more time one has to spend.

In this direction the Liouville's theorem on approximation of algebraic numbers is known. Namely, if algebraic numbers $a \neq b$ are roots of polynomials $f(a) = g(b) = 0$, $f, g \in \mathbb{Z}[Y]$, where $\deg(f)$, $\deg(g) < n$ and the integer coefficients of f, g have absolute values less than M then $|a - b| > M^{-O(n)}$. Thus, if one wants to approximate a fixed number a with a sufficiently good rate, one needs to increase the complexity of b determined by a polynomial g.

Can one extend this phenomenon of the trade-off between approximations and complexity to solutions of differential, rather than polynomial equations? It appeared that this holds for two classes of differential equations, and off these two classes the phenomenon fails. Denote by $\exp^{(n)} = \exp \circ \cdots \circ \exp$ the n times iteration of exp

Approximations and complexity

So far, we considered symbolic computations. Another type of data rather than symbolic, are numeric. Since now we study computations with approximations, a question arises about a relation between approximations and complexity. Roughly speaking, the better approximation one desires to achieve the more time one has to spend.

In this direction the Liouville's theorem on approximation of algebraic numbers is known. Namely, if algebraic numbers $a \neq b$ are roots of polynomials $f(a) = g(b) = 0$, $f, g \in \mathbb{Z}[Y]$, where $\deg(f), \deg(g) < n$ and the integer coefficients of f, g have absolute values less than M then $|a - b| > M^{-O(n)}$. Thus, if one wants to approximate a fixed number a with a sufficiently good rate, one needs to increase the complexity of b determined by a polynomial g.

Can one extend this phenomenon of the trade-off between approximations and complexity to solutions of differential, rather than polynomial equations? It appeared that this holds for two classes of differential equations, and off these two classes the phenomenon fails. Denote by $\exp^{(n)} = \exp \circ \cdots \circ \exp$ the n times iteration of exp.

Approximations and complexity

So far, we considered symbolic computations. Another type of data rather than symbolic, are numeric. Since now we study computations with approximations, a question arises about a relation between approximations and complexity. Roughly speaking, the better approximation one desires to achieve the more time one has to spend.

In this direction the Liouville's theorem on approximation of algebraic numbers is known. Namely, if algebraic numbers $a \neq b$ are roots of polynomials $f(a) = g(b) = 0,\ f, g \in \mathbb{Z}[Y]$, where $\deg(f), \deg(g) < n$ and the integer coefficients of $f,\ g$ have absolute values less than M then $|a - b| > M^{-O(n)}$. Thus, if one wants to approximate a fixed number a with a sufficiently good rate, one needs to increase the complexity of b determined by a polynomial g.

Can one extend this phenomenon of the trade-off between approximations and complexity to solutions of differential, rather than polynomial equations? It appeared that this holds for two classes of differential equations, and off these two classes the phenomenon fails. Denote by $\exp^{(n)} = \exp \circ \cdots \circ \exp$ the n times iteration of exp

Approximations and complexity

So far, we considered symbolic computations. Another type of data rather than symbolic, are numeric. Since now we study computations with approximations, a question arises about a relation between approximations and complexity. Roughly speaking, the better approximation one desires to achieve the more time one has to spend.

In this direction the Liouville's theorem on approximation of algebraic numbers is known. Namely, if algebraic numbers $a \neq b$ are roots of polynomials $f(a) = g(b) = 0,\ f, g \in \mathbb{Z}[Y]$, where $\deg(f), \deg(g) < n$ and the integer coefficients of $f,\ g$ have absolute values less than M then $|a - b| > M^{-O(n)}$. Thus, if one wants to approximate a fixed number a with a sufficiently good rate, one needs to increase the complexity of b determined by a polynomial g.

Can one extend this phenomenon of the trade-off between approximations and complexity to solutions of differential, rather than polynomial equations? It appeared that this holds for two classes of differential equations, and off these two classes the phenomenon fails. Denote by $\exp^{(n)} = \exp \circ \cdots \circ \exp$ the n times iteration of exp

Approximations and complexity

So far, we considered symbolic computations. Another type of data rather than symbolic, are numeric. Since now we study computations with approximations, a question arises about a relation between approximations and complexity. Roughly speaking, the better approximation one desires to achieve the more time one has to spend.

In this direction the Liouville's theorem on approximation of algebraic numbers is known. Namely, if algebraic numbers $a \neq b$ are roots of polynomials $f(a) = g(b) = 0,\ f, g \in \mathbb{Z}[Y]$, where $\deg(f), \deg(g) < n$ and the integer coefficients of $f,\ g$ have absolute values less than M then $|a - b| > M^{-O(n)}$. Thus, if one wants to approximate a fixed number a with a sufficiently good rate, one needs to increase the complexity of b determined by a polynomial g.

Can one extend this phenomenon of the trade-off between approximations and complexity to solutions of differential, rather than polynomial equations? It appeared that this holds for two classes of differential equations, and off these two classes the phenomenon fails.

Denote by $\exp^{(n)} = \exp \circ \cdots \circ \exp$ the n times iteration of exp.

Approximations and complexity

So far, we considered symbolic computations. Another type of data rather than symbolic, are numeric. Since now we study computations with approximations, a question arises about a relation between approximations and complexity. Roughly speaking, the better approximation one desires to achieve the more time one has to spend.

In this direction the Liouville's theorem on approximation of algebraic numbers is known. Namely, if algebraic numbers $a \neq b$ are roots of polynomials $f(a) = g(b) = 0,\ f, g \in \mathbb{Z}[Y]$, where $\deg(f), \deg(g) < n$ and the integer coefficients of $f,\ g$ have absolute values less than M then $|a - b| > M^{-O(n)}$. Thus, if one wants to approximate a fixed number a with a sufficiently good rate, one needs to increase the complexity of b determined by a polynomial g.

Can one extend this phenomenon of the trade-off between approximations and complexity to solutions of differential, rather than polynomial equations? It appeared that this holds for two classes of differential equations, and off these two classes the phenomenon fails. Denote by $\exp^{(n)} = \exp \circ \cdots \circ \exp$ the n times iteration of exp.

Approximations and complexity

So far, we considered symbolic computations. Another type of data rather than symbolic, are numeric. Since now we study computations with approximations, a question arises about a relation between approximations and complexity. Roughly speaking, the better approximation one desires to achieve the more time one has to spend.

In this direction the Liouville's theorem on approximation of algebraic numbers is known. Namely, if algebraic numbers $a \neq b$ are roots of polynomials $f(a) = g(b) = 0,\ f, g \in \mathbb{Z}[Y]$, where $\deg(f), \deg(g) < n$ and the integer coefficients of $f,\ g$ have absolute values less than M then $|a - b| > M^{-O(n)}$. Thus, if one wants to approximate a fixed number a with a sufficiently good rate, one needs to increase the complexity of b determined by a polynomial g.

Can one extend this phenomenon of the trade-off between approximations and complexity to solutions of differential, rather than polynomial equations? It appeared that this holds for two classes of differential equations, and off these two classes the phenomenon fails. Denote by $\exp^{(n)} = \exp \circ \cdots \circ \exp$ the n times iteration of exp.

Approximations and complexity

So far, we considered symbolic computations. Another type of data rather than symbolic, are numeric. Since now we study computations with approximations, a question arises about a relation between approximations and complexity. Roughly speaking, the better approximation one desires to achieve the more time one has to spend.

In this direction the Liouville's theorem on approximation of algebraic numbers is known. Namely, if algebraic numbers $a \neq b$ are roots of polynomials $f(a) = g(b) = 0,\ f, g \in \mathbb{Z}[Y]$, where $\deg(f), \deg(g) < n$ and the integer coefficients of $f,\ g$ have absolute values less than M then $|a - b| > M^{-O(n)}$. Thus, if one wants to approximate a fixed number a with a sufficiently good rate, one needs to increase the complexity of b determined by a polynomial g.

Can one extend this phenomenon of the trade-off between approximations and complexity to solutions of differential, rather than polynomial equations? It appeared that this holds for two classes of differential equations, and off these two classes the phenomenon fails.

Denote by $\exp^{(n)} = \exp \circ \cdots \circ \exp$ the n times iteration of exp

Approximations and complexity

So far, we considered symbolic computations. Another type of data rather than symbolic, are numeric. Since now we study computations with approximations, a question arises about a relation between approximations and complexity. Roughly speaking, the better approximation one desires to achieve the more time one has to spend.

In this direction the Liouville's theorem on approximation of algebraic numbers is known. Namely, if algebraic numbers $a \neq b$ are roots of polynomials $f(a) = g(b) = 0,\ f, g \in \mathbb{Z}[Y]$, where $\deg(f), \deg(g) < n$ and the integer coefficients of f, g have absolute values less than M then $|a - b| > M^{-O(n)}$. Thus, if one wants to approximate a fixed number a with a sufficiently good rate, one needs to increase the complexity of b determined by a polynomial g.

Can one extend this phenomenon of the trade-off between approximations and complexity to solutions of differential, rather than polynomial equations? It appeared that this holds for two classes of differential equations, and off these two classes the phenomenon fails. Denote by $\exp^{(n)} = \exp \circ \cdots \circ \exp$ the n times iteration of exp.

Approximations of iterated solutions of linear ordinary differential equations

To describe the first class of differential equations assume that we possess a device which allows one to yield a solution u of a linear ordinary differential equation $(\sum_j h_j \frac{d^j}{dt^j}) \cdot u = 0$. Also arithmetic operations are admitted in computations.

The main result on the trade-off between the approximations and complexity (G. (1992)): if functions $u(t) \neq v(t)$ are obtained each by applications of the device at most of n times then

$$|u(t) - v(t)| \succ (\exp^{(n)}(t^{O(1)}))^{-1}$$

where the latter relation $\succ$ means that the measure of the real points $t \in \mathbb{R}$ at which this inequality fails, is finite.

Approximations of iterated solutions of linear ordinary differential equations

To describe the first class of differential equations assume that we possess a device which allows one to yield a solution u of a linear ordinary differential equation $(\sum_j h_j \frac{d^j}{dt^j}) \cdot u = 0$. Also arithmetic operations are admitted in computations.

The main result on the trade-off between the approximations and complexity (G. (1992)): if functions $u(t) \neq v(t)$ are obtained each by applications of the device at most of n times then

$$|u(t) - v(t)| \succ (\exp^{(n)}(t^{O(1)}))^{-1}$$

where the latter relation $\succ$ means that the measure of the real points $t \in \mathbb{R}$ at which this inequality fails, is finite.

Approximations of iterated solutions of linear ordinary differential equations

To describe the first class of differential equations assume that we possess a device which allows one to yield a solution u of a linear ordinary differential equation $(\sum_j h_j \frac{d^j}{dt^j}) \cdot u = 0$. Also arithmetic operations are admitted in computations.

The main result on the trade-off between the approximations and complexity (G. (1992)): if functions $u(t) \neq v(t)$ are obtained each by applications of the device at most of n times then

$$|u(t) - v(t)| \succ (\exp^{(n)}(t^{O(1)}))^{-1}$$

where the latter relation $\succ$ means that the measure of the real points $t \in \mathbb{R}$ at which this inequality fails, is finite.

Approximations of iterated solutions of linear ordinary differential equations

To describe the first class of differential equations assume that we possess a device which allows one to yield a solution u of a linear ordinary differential equation $(\sum_j h_j \frac{d^j}{dt^j}) \cdot u = 0$. Also arithmetic operations are admitted in computations.

The main result on the trade-off between the approximations and complexity (G. (1992)): if functions $u(t) \neq v(t)$ are obtained each by applications of the device at most of n times then

$$|u(t) - v(t)| \succ (\exp^{(n)}(t^{O(1)}))^{-1}$$

where the latter relation $\succ$ means that the measure of the real points $t \in \mathbb{R}$ at which this inequality fails, is finite.

Approximations of iterated solutions of linear ordinary differential equations

To describe the first class of differential equations assume that we possess a device which allows one to yield a solution u of a linear ordinary differential equation $(\sum_j h_j \frac{d^j}{dt^j}) \cdot u = 0$. Also arithmetic operations are admitted in computations.

The main result on the trade-off between the approximations and complexity (G. (1992)): if functions $u(t) \neq v(t)$ are obtained each by applications of the device at most of n times then

$$|u(t) - v(t)| \succ (\exp^{(n)}(t^{O(1)}))^{-1}$$

where the latter relation $\succ$ means that the measure of the real points $t \in \mathbb{R}$ at which this inequality fails, is finite.

Approximations of Pfaffian functions

The second class of differential equations for which the trade-off between approximations and complexity holds, is the *Pfaffian functions*. Real functions $u_1, \ldots, u_n$ form a *Pfaffian chain* if

$$\frac{du_i}{dt} = p_i(t, u_1, \ldots, u_i),\ 1 \leq i \leq n$$

for suitable polynomials $p_i \in \mathbb{R}[t, Y_1, \ldots, Y_i]$. Each function u_i, $1 \leq i \leq n$ is called Pfaffian and n is called the *length of the Pfaffian chain.*

In other words, we suppose that besides the arithmetic operations, we are in possession of a device which allows one to solve non-linear ordinary first-order equations. The main result on the trade-off for Pfaffian functions (G. (1992)): if Pfaffian functions $u(t) \neq v(t)$ are given each by a Pfaffian chain of the length n then

$$|u(t) - v(t)| > (\exp^{(n)}(t^{O(1)}))^{-1},\ t >> 0.$$

Approximations of Pfaffian functions

The second class of differential equations for which the trade-off between approximations and complexity holds, is the *Pfaffian functions*. Real functions $u_1, \dots, u_n$ form a *Pfaffian chain* if

$$\frac{du_i}{dt} = p_i(t, u_1, \dots, u_i),\ 1 \leq i \leq n$$

for suitable polynomials $p_i \in \mathbb{R}[t, Y_1, \dots, Y_i]$. Each function u_i, $1 \leq i \leq n$ is called Pfaffian and n is called the *length of the Pfaffian chain*.

In other words, we suppose that besides the arithmetic operations, we are in possession of a device which allows one to solve non-linear ordinary first-order equations. The main result on the trade-off for Pfaffian functions (G. (1992)): if Pfaffian functions $u(t) \neq v(t)$ are given each by a Pfaffian chain of the length n then

$$|u(t) - v(t)| > (\exp^{(n)}(t^{O(1)}))^{-1},\ t >> 0.$$

Approximations of Pfaffian functions

The second class of differential equations for which the trade-off between approximations and complexity holds, is the *Pfaffian functions*. Real functions $u_1, \dots, u_n$ form a *Pfaffian chain* if

$$\frac{du_i}{dt} = p_i(t, u_1, \dots, u_i),\ 1 \le i \le n$$

for suitable polynomials $p_i \in \mathbb{R}[t, Y_1, \dots, Y_i]$. Each function u_i, $1 \le i \le n$ is called Pfaffian and n is called the *length of the Pfaffian chain*.

In other words, we suppose that besides the arithmetic operations, we are in possession of a device which allows one to solve non-linear ordinary first-order equations. The main result on the trade-off for Pfaffian functions (G. (1992)): if Pfaffian functions $u(t) \neq v(t)$ are given each by a Pfaffian chain of the length n then

$$|u(t) - v(t)| > (\exp^{(n)}(t^{O(1)}))^{-1},\ t >> 0.$$

Approximations of Pfaffian functions

The second class of differential equations for which the trade-off between approximations and complexity holds, is the *Pfaffian functions*. Real functions $u_1, \dots, u_n$ form a *Pfaffian chain* if

$$\frac{du_i}{dt} = p_i(t, u_1, \dots, u_i),\ 1 \le i \le n$$

for suitable polynomials $p_i \in \mathbb{R}[t, Y_1, \dots, Y_i]$. Each function $u_i,\ 1 \le i \le n$ is called Pfaffian and n is called the *length of the Pfaffian chain*.

In other words, we suppose that besides the arithmetic operations, we are in possession of a device which allows one to solve non-linear ordinary first-order equations. The main result on the trade-off for Pfaffian functions (G. (1992)): if Pfaffian functions $u(t) \neq v(t)$ are given each by a Pfaffian chain of the length n then

$$|u(t) - v(t)| > (\exp^{(n)}(t^{O(1)}))^{-1},\ t >> 0.$$

Approximations of Pfaffian functions

The second class of differential equations for which the trade-off between approximations and complexity holds, is the *Pfaffian functions*. Real functions $u_1, \dots, u_n$ form a *Pfaffian chain* if

$$\frac{du_i}{dt} = p_i(t, u_1, \dots, u_i),\ 1 \le i \le n$$

for suitable polynomials $p_i \in \mathbb{R}[t, Y_1, \dots, Y_i]$. Each function $u_i,\ 1 \le i \le n$ is called Pfaffian and n is called the *length of the Pfaffian chain*.

In other words, we suppose that besides the arithmetic operations, we are in possession of a device which allows one to solve non-linear ordinary first-order equations. The main result on the trade-off for Pfaffian functions (G. (1992)): if Pfaffian functions $u(t) \ne v(t)$ are given each by a Pfaffian chain of the length n then

$$|u(t) - v(t)| > (\exp^{(n)}(t^{O(1)}))^{-1},\ t >> 0.$$

Approximations of Pfaffian functions

The second class of differential equations for which the trade-off between approximations and complexity holds, is the *Pfaffian functions*. Real functions $u_1, \ldots, u_n$ form a *Pfaffian chain* if

$$\frac{du_i}{dt} = p_i(t, u_1, \ldots, u_i),\ 1 \le i \le n$$

for suitable polynomials $p_i \in \mathbb{R}[t, Y_1, \ldots, Y_i]$. Each function $u_i,\ 1 \le i \le n$ is called Pfaffian and n is called the *length of the Pfaffian chain*.

In other words, we suppose that besides the arithmetic operations, we are in possession of a device which allows one to solve non-linear ordinary first-order equations. The main result on the trade-off for Pfaffian functions (G. (1992)): if Pfaffian functions $u(t) \ne v(t)$ are given each by a Pfaffian chain of the length n then

$$|u(t) - v(t)| > (\exp^{(n)}(t^{O(1)}))^{-1},\ t >> 0.$$

Frontiers of the trade-off between approximations and complexity

Thus, informally, if we deal only with (iterations of) either linear or first-order differential equations then the trade-off between approximations and complexity holds. On the other hand, there was exhibited a family of non-linear second-order ordinary algebraic differential equations such that its solutions cannot be asymptotically separated from zero by any function.

Formulated two results concern the asymptotical approximations on the real line. Similar results were established for the trade-off between approximations and complexity on a real interval for two classes of functions being solutions of appropriate non-linear ordinary differential equations (G. (2001)).

Frontiers of the trade-off between approximations and complexity

Thus, informally, if we deal only with (iterations of) either linear or first-order differential equations then the trade-off between approximations and complexity holds. On the other hand, there was exhibited a family of non-linear second-order ordinary algebraic differential equations such that its solutions cannot be asymptotically separated from zero by any function.

Formulated two results concern the asymptotical approximations on the real line. Similar results were established for the trade-off between approximations and complexity on a real interval for two classes of functions being solutions of appropriate non-linear ordinary differential equations (G. (2001)).

Frontiers of the trade-off between approximations and complexity

Thus, informally, if we deal only with (iterations of) either linear or first-order differential equations then the trade-off between approximations and complexity holds. On the other hand, there was exhibited a family of non-linear second-order ordinary algebraic differential equations such that its solutions cannot be asymptotically separated from zero by any function.

Formulated two results concern the asymptotical approximations on the real line. Similar results were established for the trade-off between approximations and complexity on a real interval for two classes of functions being solutions of appropriate non-linear ordinary differential equations (G. (2001)).

Frontiers of the trade-off between approximations and complexity

Thus, informally, if we deal only with (iterations of) either linear or first-order differential equations then the trade-off between approximations and complexity holds. On the other hand, there was exhibited a family of non-linear second-order ordinary algebraic differential equations such that its solutions cannot be asymptotically separated from zero by any function.

Formulated two results concern the asymptotical approximations on the real line. Similar results were established for the trade-off between approximations and complexity on a real interval for two classes of functions being solutions of appropriate non-linear ordinary differential equations (G. (2001)).

Frontiers of the trade-off between approximations and complexity

Thus, informally, if we deal only with (iterations of) either linear or first-order differential equations then the trade-off between approximations and complexity holds. On the other hand, there was exhibited a family of non-linear second-order ordinary algebraic differential equations such that its solutions cannot be asymptotically separated from zero by any function.

Formulated two results concern the asymptotical approximations on the real line. Similar results were established for the trade-off between approximations and complexity on a real interval for two classes of functions being solutions of appropriate non-linear ordinary differential equations (G. (2001)).

Complexity of black-box computations

So far, we considered two types of data: symbolic and numeric. Now we study a type of data intermediate between symbolic and numeric ones, namely, black-box computations. Assume that a computation contains a black-box which for a given input outputs the value of an a priori unknown function f. Then such a computation has the features of both numeric because the output of the black-box is numeric data, and on the other hand, the computation can treat the outputs of the black-box as symbols since the latter are a priori unknown.

The problem of *black-box interpolation* is to retrieve f. Of course, some information on f should be available. First, let f be a polynomial in n variables with s monomials, f is called s-*sparse*. Emphasize that the degree of f is a priori unknown, while s is given. Let f be defined over a field of characteristic zero, then Ben-Or-Tiwari (1987): an algorithm which retrieves f within polynomial complexity, moreover the algorithm makes just $2 \cdot s + 1$ calls to the black-box. More precisely, herein the complexity is measured as a function of the size of the output f (a priori unknown).

Complexity of black-box computations

So far, we considered two types of data: symbolic and numeric. Now we study a type of data intermediate between symbolic and numeric ones, namely, black-box computations. Assume that a computation contains a black-box which for a given input outputs the value of an a priori unknown function f. Then such a computation has the features of both numeric because the output of the black-box is numeric data, and on the other hand, the computation can treat the outputs of the black-box as symbols since the latter are a priori unknown.

The problem of *black-box interpolation* is to retrieve f. Of course, some information on f should be available. First, let f be a polynomial in n variables with s monomials, f is called *s-sparse*. Emphasize that the degree of f is a priori unknown, while s is given. Let f be defined over a field of characteristic zero, then Ben-Or-Tiwari (1987): an algorithm which retrieves f within polynomial complexity, moreover the algorithm makes just $2 \cdot s + 1$ calls to the black-box. More precisely, herein the complexity is measured as a function of the size of the output f (a priori unknown).

Complexity of black-box computations

So far, we considered two types of data: symbolic and numeric. Now we study a type of data intermediate between symbolic and numeric ones, namely, black-box computations. Assume that a computation contains a black-box which for a given input outputs the value of an a priori unknown function f. Then such a computation has the features of both numeric because the output of the black-box is numeric data, and on the other hand, the computation can treat the outputs of the black-box as symbols since the latter are a priori unknown.

The problem of *black-box interpolation* is to retrieve f. Of course, some information on f should be available. First, let f be a polynomial in n variables with s monomials, f is called *s-sparse*. Emphasize that the degree of f is a priori unknown, while s is given. Let f be defined over a field of characteristic zero, then Ben-Or-Tiwari (1987): an algorithm which retrieves f within polynomial complexity, moreover the algorithm makes just $2 \cdot s + 1$ calls to the black-box. More precisely, herein the complexity is measured as a function of the size of the output f (a priori unknown).

Complexity of black-box computations

So far, we considered two types of data: symbolic and numeric. Now we study a type of data intermediate between symbolic and numeric ones, namely, black-box computations. Assume that a computation contains a black-box which for a given input outputs the value of an a priori unknown function f. Then such a computation has the features of both numeric because the output of the black-box is numeric data, and on the other hand, the computation can treat the outputs of the black-box as symbols since the latter are a priori unknown.

The problem of *black-box interpolation* is to retrieve f. Of course, some information on f should be available. First, let f be a polynomial in n variables with s monomials, f is called *s-sparse*. Emphasize that the degree of f is a priori unknown, while s is given. Let f be defined over a field of characteristic zero, then Ben-Or-Tiwari (1987): an algorithm which retrieves f within polynomial complexity, moreover the algorithm makes just $2 \cdot s + 1$ calls to the black-box. More precisely, herein the complexity is measured as a function of the size of the output f (a priori unknown).

Complexity of black-box computations

So far, we considered two types of data: symbolic and numeric. Now we study a type of data intermediate between symbolic and numeric ones, namely, black-box computations. Assume that a computation contains a black-box which for a given input outputs the value of an a priori unknown function f. Then such a computation has the features of both numeric because the output of the black-box is numeric data, and on the other hand, the computation can treat the outputs of the black-box as symbols since the latter are a priori unknown.

The problem of *black-box interpolation* is to retrieve f. Of course, some information on f should be available. First, let f be a polynomial in n variables with s monomials, f is called *s-sparse*. Emphasize that the degree of f is a priori unknown, while s is given. Let f be defined over a field of characteristic zero, then Ben-Or-Tiwari (1987): an algorithm which retrieves f within polynomial complexity, moreover the algorithm makes just $2 \cdot s + 1$ calls to the black-box. More precisely, herein the complexity is measured as a function of the size of the output f (a priori unknown).

Complexity of black-box computations

So far, we considered two types of data: symbolic and numeric. Now we study a type of data intermediate between symbolic and numeric ones, namely, black-box computations. Assume that a computation contains a black-box which for a given input outputs the value of an a priori unknown function f. Then such a computation has the features of both numeric because the output of the black-box is numeric data, and on the other hand, the computation can treat the outputs of the black-box as symbols since the latter are a priori unknown.

The problem of *black-box interpolation* is to retrieve f. Of course, some information on f should be available. First, let f be a polynomial in n variables with s monomials, f is called s-*sparse*. Emphasize that the degree of f is a priori unknown, while s is given. Let f be defined over a field of characteristic zero, then Ben-Or-Tiwari (1987): an algorithm which retrieves f within polynomial complexity, moreover the algorithm makes just $2 \cdot s + 1$ calls to the black-box. More precisely, herein the complexity is measured as a function of the size of the output f (a priori unknown).

Complexity of black-box computations

So far, we considered two types of data: symbolic and numeric. Now we study a type of data intermediate between symbolic and numeric ones, namely, black-box computations. Assume that a computation contains a black-box which for a given input outputs the value of an a priori unknown function f. Then such a computation has the features of both numeric because the output of the black-box is numeric data, and on the other hand, the computation can treat the outputs of the black-box as symbols since the latter are a priori unknown.

The problem of *black-box interpolation* is to retrieve f. Of course, some information on f should be available. First, let f be a polynomial in n variables with s monomials, f is called *s-sparse*. Emphasize that the degree of f is a priori unknown, while s is given. Let f be defined over a field of characteristic zero, then Ben-Or-Tiwari (1987): an algorithm which retrieves f within polynomial complexity, moreover the algorithm makes just $2 \cdot s + 1$ calls to the black-box. More precisely, herein the complexity is measured as a function of the size of the output f (a priori unknown).

Complexity of black-box computations

So far, we considered two types of data: symbolic and numeric. Now we study a type of data intermediate between symbolic and numeric ones, namely, black-box computations. Assume that a computation contains a black-box which for a given input outputs the value of an a priori unknown function f. Then such a computation has the features of both numeric because the output of the black-box is numeric data, and on the other hand, the computation can treat the outputs of the black-box as symbols since the latter are a priori unknown.

The problem of *black-box interpolation* is to retrieve f. Of course, some information on f should be available. First, let f be a polynomial in n variables with s monomials, f is called *s-sparse*. Emphasize that the degree of f is a priori unknown, while s is given. Let f be defined over a field of characteristic zero, then Ben-Or-Tiwari (1987): an algorithm which retrieves f within polynomial complexity, moreover the algorithm makes just $2 \cdot s + 1$ calls to the black-box. More precisely, herein the complexity is measured as a function of the size of the output f (a priori unknown).

Complexity of black-box computations

So far, we considered two types of data: symbolic and numeric. Now we study a type of data intermediate between symbolic and numeric ones, namely, black-box computations. Assume that a computation contains a black-box which for a given input outputs the value of an a priori unknown function f. Then such a computation has the features of both numeric because the output of the black-box is numeric data, and on the other hand, the computation can treat the outputs of the black-box as symbols since the latter are a priori unknown.

The problem of *black-box interpolation* is to retrieve f. Of course, some information on f should be available. First, let f be a polynomial in n variables with s monomials, f is called *s-sparse*. Emphasize that the degree of f is a priori unknown, while s is given. Let f be defined over a field of characteristic zero, then Ben-Or-Tiwari (1987): an algorithm which retrieves f within polynomial complexity, moreover the algorithm makes just $2 \cdot s + 1$ calls to the black-box. More precisely, herein the complexity is measured as a function of the size of the output f (a priori unknown).

Complexity of black-box computations

So far, we considered two types of data: symbolic and numeric. Now we study a type of data intermediate between symbolic and numeric ones, namely, black-box computations. Assume that a computation contains a black-box which for a given input outputs the value of an a priori unknown function f. Then such a computation has the features of both numeric because the output of the black-box is numeric data, and on the other hand, the computation can treat the outputs of the black-box as symbols since the latter are a priori unknown.

The problem of *black-box interpolation* is to retrieve f. Of course, some information on f should be available. First, let f be a polynomial in n variables with s monomials, f is called *s-sparse*. Emphasize that the degree of f is a priori unknown, while s is given. Let f be defined over a field of characteristic zero, then Ben-Or-Tiwari (1987): an algorithm which retrieves f within polynomial complexity, moreover the algorithm makes just $2 \cdot s + 1$ calls to the black-box. More precisely, herein the complexity is measured as a function of the size of the output f (a priori unknown).

Complexity of black-box computations

So far, we considered two types of data: symbolic and numeric. Now we study a type of data intermediate between symbolic and numeric ones, namely, black-box computations. Assume that a computation contains a black-box which for a given input outputs the value of an a priori unknown function f. Then such a computation has the features of both numeric because the output of the black-box is numeric data, and on the other hand, the computation can treat the outputs of the black-box as symbols since the latter are a priori unknown.

The problem of *black-box interpolation* is to retrieve f. Of course, some information on f should be available. First, let f be a polynomial in n variables with s monomials, f is called *s-sparse*. Emphasize that the degree of f is a priori unknown, while s is given. Let f be defined over a field of characteristic zero, then Ben-Or-Tiwari (1987): an algorithm which retrieves f within polynomial complexity, moreover the algorithm makes just $2 \cdot s + 1$ calls to the black-box. More precisely, herein the complexity is measured as a function of the size of the output f (a priori unknown).

Complexity of black-box computations

So far, we considered two types of data: symbolic and numeric. Now we study a type of data intermediate between symbolic and numeric ones, namely, black-box computations. Assume that a computation contains a black-box which for a given input outputs the value of an a priori unknown function f. Then such a computation has the features of both numeric because the output of the black-box is numeric data, and on the other hand, the computation can treat the outputs of the black-box as symbols since the latter are a priori unknown.

The problem of *black-box interpolation* is to retrieve f. Of course, some information on f should be available. First, let f be a polynomial in n variables with s monomials, f is called *s-sparse*. Emphasize that the degree of f is a priori unknown, while s is given. Let f be defined over a field of characteristic zero, then Ben-Or-Tiwari (1987): an algorithm which retrieves f within polynomial complexity, moreover the algorithm makes just $2 \cdot s + 1$ calls to the black-box. More precisely, herein the complexity is measured as a function of the size of the output f (a priori unknown).

Complexity of black-box interpolation

Let an s-sparse polynomial f be defined over a finite field, then G.-Karpinski-Singer (1988): an algorithm for its retrieval within polynomial complexity.

For larger classes of functions when $f = g/h$ is a rational function where g, h are s-sparse polynomials G.-Karpinski-Singer (1989): retrieval of f within polynomial complexity. Note that this representation of a rational function can be reducible, while the irreducible representation can be non-sparse, as in the example $(x^n - 1)/(x - 1) = x^{n-1} + \cdots + 1$.

Finally, when $f(X_1, \ldots, X_n)$ is an *algebraic function* being s-sparse, i. e. f satisfies an s-sparse polynomial equation $p(X_1, \ldots, X_n, f) = 0$, one can also retrieve f within polynomial complexity (G.-Karpinski-Singer (1990)).

Complexity of black-box interpolation

Let an s-sparse polynomial f be defined over a finite field, then G.-Karpinski-Singer (1988): an algorithm for its retrieval within polynomial complexity.

For larger classes of functions when $f = g/h$ is a rational function where g, h are s-sparse polynomials G.-Karpinski-Singer (1989): retrieval of f within polynomial complexity. Note that this representation of a rational function can be reducible, while the irreducible representation can be non-sparse, as in the example $(x^n - 1)/(x - 1) = x^{n-1} + \cdots + 1$.

Finally, when $f(X_1, \ldots, X_n)$ is an *algebraic function* being s-sparse, i. e. f satisfies an s-sparse polynomial equation $p(X_1, \ldots, X_n, f) = 0$, one can also retrieve f within polynomial complexity (G.-Karpinski-Singer (1990)).

Complexity of black-box interpolation

Let an s-sparse polynomial f be defined over a finite field, then G.-Karpinski-Singer (1988): an algorithm for its retrieval within polynomial complexity.

For larger classes of functions when $f = g/h$ is a rational function where g, h are s-sparse polynomials G.-Karpinski-Singer (1989): retrieval of f within polynomial complexity. Note that this representation of a rational function can be reducible, while the irreducible representation can be non-sparse, as in the example $(x^n - 1)/(x - 1) = x^{n-1} + \cdots + 1$.

Finally, when $f(X_1, \ldots, X_n)$ is an *algebraic function* being s-sparse, i. e. f satisfies an s-sparse polynomial equation $p(X_1, \ldots, X_n, f) = 0$, one can also retrieve f within polynomial complexity (G.-Karpinski-Singer (1990)).

Complexity of black-box interpolation

Let an s-sparse polynomial f be defined over a finite field, then G.-Karpinski-Singer (1988): an algorithm for its retrieval within polynomial complexity.

For larger classes of functions when $f = g/h$ is a rational function where g, h are s-sparse polynomials G.-Karpinski-Singer (1989): retrieval of f within polynomial complexity. Note that this representation of a rational function can be reducible, while the irreducible representation can be non-sparse, as in the example $(x^n - 1)/(x - 1) = x^{n-1} + \cdots + 1$.

Finally, when $f(X_1, \ldots, X_n)$ is an *algebraic function* being s-sparse, i. e. f satisfies an s-sparse polynomial equation $p(X_1, \ldots, X_n, f) = 0$, one can also retrieve f within polynomial complexity (G.-Karpinski-Singer (1990)).

Complexity of black-box interpolation

Let an s-sparse polynomial f be defined over a finite field, then G.-Karpinski-Singer (1988): an algorithm for its retrieval within polynomial complexity.

For larger classes of functions when $f = g/h$ is a rational function where g, h are s-sparse polynomials G.-Karpinski-Singer (1989): retrieval of f within polynomial complexity. Note that this representation of a rational function can be reducible, while the irreducible representation can be non-sparse, as in the example $(x^n - 1)/(x - 1) = x^{n-1} + \cdots + 1$.

Finally, when $f(X_1, \ldots, X_n)$ is an *algebraic function* being s-sparse, i. e. f satisfies an s-sparse polynomial equation $p(X_1, \ldots, X_n, f) = 0$, one can also retrieve f within polynomial complexity (G.-Karpinski-Singer (1990)).

Computations vs. tests

It happens sometimes that it is more difficult to compute a function $g(x)$ at a point x than to test whether for given x, y it holds $y = g(x)$.

Freiwald (1979): a probabilistic test for matrix multiplication. Namely, given $n \times n$ matrices A, B, C to test whether $A = B \cdot C$. For a randomly chosen vector v compute vector $w := C \cdot v$, then compute vector $u := B \cdot w$ and compare it with the vector $A \cdot v$. If $u = A \cdot v$ then the probabilistic test declares that $A = B \cdot C$. The algebraic complexity of this test is $O(n^2)$, while the complexity of the matrix multiplication is supposedly much bigger.

For n-bit integer multiplication the complexity bound $O(n \cdot \log n \cdot \exp(\log^* n))$ is due to Fürer (2007) who has improved the well known algorithm of Schönhage-Strassen (1971), where $\log^*$ is the function inverse to the function $m \mapsto exp^{(m)}(2)$. (The function $\exp(\log^* n)$ grows very slowly.) G.-Tenenbaum (2009): a probabilistic test for integer multiplication, so for given integers a, b, c to test whether $a = b \cdot c$, within complexity $O(n \cdot \log \log n \cdot \exp(\log^* n))$.

Computations vs. tests

It happens sometimes that it is more difficult to compute a function $g(x)$ at a point x than to test whether for given x, y it holds $y = g(x)$.

Freiwald (1979): a probabilistic test for matrix multiplication. Namely, given $n \times n$ matrices A, B, C to test whether $A = B \cdot C$. For a randomly chosen vector v compute vector $w := C \cdot v$, then compute vector $u := B \cdot w$ and compare it with the vector $A \cdot v$. If $u = A \cdot v$ then the probabilistic test declares that $A = B \cdot C$. The algebraic complexity of this test is $O(n^2)$, while the complexity of the matrix multiplication is supposedly much bigger.

For n-bit integer multiplication the complexity bound $O(n \cdot \log n \cdot \exp(\log^* n))$ is due to Fürer (2007) who has improved the well known algorithm of Schönhage-Strassen (1971), where $\log^*$ is the function inverse to the function $m \mapsto exp^{(m)}(2)$. (The function $\exp(\log^* n)$ grows very slowly.) G.-Tenenbaum (2009): a probabilistic test for integer multiplication, so for given integers a, b, c to test whether $a = b \cdot c$, within complexity $O(n \cdot \log\log n \cdot \exp(\log^* n))$.

Computations vs. tests

It happens sometimes that it is more difficult to compute a function $g(x)$ at a point x than to test whether for given x, y it holds $y = g(x)$.

Freiwald (1979): a probabilistic test for matrix multiplication. Namely, given $n \times n$ matrices A, B, C to test whether $A = B \cdot C$. For a randomly chosen vector v compute vector $w := C \cdot v$, then compute vector $u := B \cdot w$ and compare it with the vector $A \cdot v$. If $u = A \cdot v$ then the probabilistic test declares that $A = B \cdot C$. The algebraic complexity of this test is $O(n^2)$, while the complexity of the matrix multiplication is supposedly much bigger.

For n-bit integer multiplication the complexity bound $O(n \cdot \log n \cdot \exp(\log^* n))$ is due to Fürer (2007) who has improved the well known algorithm of Schönhage-Strassen (1971), where $\log^*$ is the function inverse to the function $m \mapsto exp^{(m)}(2)$. (The function $\exp(\log^* n)$ grows very slowly.) G.-Tenenbaum (2009): a probabilistic test for integer multiplication, so for given integers a, b, c to test whether $a = b \cdot c$, within complexity $O(n \cdot \log \log n \cdot \exp(\log^* n))$.

Computations vs. tests

It happens sometimes that it is more difficult to compute a function $g(x)$ at a point x than to test whether for given x, y it holds $y = g(x)$.

Freiwald (1979): a probabilistic test for matrix multiplication. Namely, given $n \times n$ matrices A, B, C to test whether $A = B \cdot C$. For a randomly chosen vector v compute vector $w := C \cdot v$, then compute vector $u := B \cdot w$ and compare it with the vector $A \cdot v$. If $u = A \cdot v$ then the probabilistic test declares that $A = B \cdot C$. The algebraic complexity of this test is $O(n^2)$, while the complexity of the matrix multiplication is supposedly much bigger.

For n-bit integer multiplication the complexity bound $O(n \cdot \log n \cdot \exp(\log^* n))$ is due to Fürer (2007) who has improved the well known algorithm of Schönhage-Strassen (1971), where $\log^*$ is the function inverse to the function $m \mapsto exp^{(m)}(2)$. (The function $\exp(\log^* n)$ grows very slowly.) G.-Tenenbaum (2009): a probabilistic test for integer multiplication, so for given integers a, b, c to test whether $a = b \cdot c$, within complexity $O(n \cdot \log\log n \cdot \exp(\log^* n))$.

Computations vs. tests

It happens sometimes that it is more difficult to compute a function $g(x)$ at a point x than to test whether for given x, y it holds $y = g(x)$.

Freiwald (1979): a probabilistic test for matrix multiplication. Namely, given $n \times n$ matrices A, B, C to test whether $A = B \cdot C$. For a randomly chosen vector v compute vector $w := C \cdot v$, then compute vector $u := B \cdot w$ and compare it with the vector $A \cdot v$. If $u = A \cdot v$ then the probabilistic test declares that $A = B \cdot C$. The algebraic complexity of this test is $O(n^2)$, while the complexity of the matrix multiplication is supposedly much bigger.

For n-bit integer multiplication the complexity bound $O(n \cdot \log n \cdot \exp(\log^* n))$ is due to Fürer (2007) who has improved the well known algorithm of Schönhage-Strassen (1971), where $\log^*$ is the function inverse to the function $m \mapsto exp^{(m)}(2)$. (The function $\exp(\log^* n)$ grows very slowly.) G.-Tenenbaum (2009): a probabilistic test for integer multiplication, so for given integers a, b, c to test whether $a = b \cdot c$, within complexity $O(n \cdot \log\log n \cdot \exp(\log^* n))$.

Computations vs. tests

It happens sometimes that it is more difficult to compute a function $g(x)$ at a point x than to test whether for given x, y it holds $y = g(x)$.

Freiwald (1979): a probabilistic test for matrix multiplication. Namely, given $n \times n$ matrices A, B, C to test whether $A = B \cdot C$. For a randomly chosen vector v compute vector $w := C \cdot v$, then compute vector $u := B \cdot w$ and compare it with the vector $A \cdot v$. If $u = A \cdot v$ then the probabilistic test declares that $A = B \cdot C$. The algebraic complexity of this test is $O(n^2)$, while the complexity of the matrix multiplication is supposedly much bigger.

For n-bit integer multiplication the complexity bound $O(n \cdot \log n \cdot \exp(\log^* n))$ is due to Fürer (2007) who has improved the well known algorithm of Schönhage-Strassen (1971), where $\log^*$ is the function inverse to the function $m \mapsto exp^{(m)}(2)$. (The function $\exp(\log^* n)$ grows very slowly.) G.-Tenenbaum (2009): a probabilistic test for integer multiplication, so for given integers a, b, c to test whether $a = b \cdot c$, within complexity $O(n \cdot \log \log n \cdot \exp(\log^* n))$.

Computations vs. tests

It happens sometimes that it is more difficult to compute a function $g(x)$ at a point x than to test whether for given x, y it holds $y = g(x)$.

Freiwald (1979): a probabilistic test for matrix multiplication. Namely, given $n \times n$ matrices A, B, C to test whether $A = B \cdot C$. For a randomly chosen vector v compute vector $w := C \cdot v$, then compute vector $u := B \cdot w$ and compare it with the vector $A \cdot v$. If $u = A \cdot v$ then the probabilistic test declares that $A = B \cdot C$. The algebraic complexity of this test is $O(n^2)$, while the complexity of the matrix multiplication is supposedly much bigger.

For n-bit integer multiplication the complexity bound $O(n \cdot \log n \cdot \exp(\log^* n))$ is due to Fürer (2007) who has improved the well known algorithm of Schönhage-Strassen (1971), where $\log^*$ is the function inverse to the function $m \mapsto exp^{(m)}(2)$. (The function $\exp(\log^* n)$ grows very slowly.) G.-Tenenbaum (2009): a probabilistic test for integer multiplication, so for given integers a, b, c to test whether $a = b \cdot c$, within complexity $O(n \cdot \log \log n \cdot \exp(\log^* n))$.

Computations vs. tests

It happens sometimes that it is more difficult to compute a function $g(x)$ at a point x than to test whether for given x, y it holds $y = g(x)$.

Freiwald (1979): a probabilistic test for matrix multiplication. Namely, given $n \times n$ matrices A, B, C to test whether $A = B \cdot C$. For a randomly chosen vector v compute vector $w := C \cdot v$, then compute vector $u := B \cdot w$ and compare it with the vector $A \cdot v$. If $u = A \cdot v$ then the probabilistic test declares that $A = B \cdot C$. The algebraic complexity of this test is $O(n^2)$, while the complexity of the matrix multiplication is supposedly much bigger.

For n-bit integer multiplication the complexity bound $O(n \cdot \log n \cdot \exp(\log^* n))$ is due to Fürer (2007) who has improved the well known algorithm of Schönhage-Strassen (1971), where $\log^*$ is the function inverse to the function $m \mapsto exp^{(m)}(2)$. (The function $\exp(\log^* n)$ grows very slowly.) G.-Tenenbaum (2009): a probabilistic test for integer multiplication, so for given integers a, b, c to test whether $a = b \cdot c$, within complexity $O(n \cdot \log\log n \cdot \exp(\log^* n))$.

Computations vs. tests

It happens sometimes that it is more difficult to compute a function $g(x)$ at a point x than to test whether for given x, y it holds $y = g(x)$.

Freiwald (1979): a probabilistic test for matrix multiplication. Namely, given $n \times n$ matrices A, B, C to test whether $A = B \cdot C$. For a randomly chosen vector v compute vector $w := C \cdot v$, then compute vector $u := B \cdot w$ and compare it with the vector $A \cdot v$. If $u = A \cdot v$ then the probabilistic test declares that $A = B \cdot C$. The algebraic complexity of this test is $O(n^2)$, while the complexity of the matrix multiplication is supposedly much bigger.

For n-bit integer multiplication the complexity bound $O(n \cdot \log n \cdot \exp(\log^* n))$ is due to Fürer (2007) who has improved the well known algorithm of Schönhage-Strassen (1971), where $\log^*$ is the function inverse to the function $m \mapsto exp^{(m)}(2)$. (The function $\exp(\log^* n)$ grows very slowly.) G.-Tenenbaum (2009): a probabilistic test for integer multiplication, so for given integers a, b, c to test whether $a = b \cdot c$, within complexity $O(n \cdot \log\log n \cdot \exp(\log^* n))$.

Computations vs. tests

It happens sometimes that it is more difficult to compute a function $g(x)$ at a point x than to test whether for given x, y it holds $y = g(x)$.

Freiwald (1979): a probabilistic test for matrix multiplication. Namely, given $n \times n$ matrices A, B, C to test whether $A = B \cdot C$. For a randomly chosen vector v compute vector $w := C \cdot v$, then compute vector $u := B \cdot w$ and compare it with the vector $A \cdot v$. If $u = A \cdot v$ then the probabilistic test declares that $A = B \cdot C$. The algebraic complexity of this test is $O(n^2)$, while the complexity of the matrix multiplication is supposedly much bigger.

For n-bit integer multiplication the complexity bound $O(n \cdot \log n \cdot \exp(\log^* n))$ is due to Fürer (2007) who has improved the well known algorithm of Schönhage-Strassen (1971), where $\log^*$ is the function inverse to the function $m \mapsto exp^{(m)}(2)$. (The function $\exp(\log^* n)$ grows very slowly.) G.-Tenenbaum (2009): a probabilistic test for integer multiplication, so for given integers a, b, c to test whether $a = b \cdot c$, within complexity $O(n \cdot \log\log n \cdot \exp(\log^* n))$.

Computations vs. tests

It happens sometimes that it is more difficult to compute a function $g(x)$ at a point x than to test whether for given x, y it holds $y = g(x)$.

Freiwald (1979): a probabilistic test for matrix multiplication. Namely, given $n \times n$ matrices A, B, C to test whether $A = B \cdot C$. For a randomly chosen vector v compute vector $w := C \cdot v$, then compute vector $u := B \cdot w$ and compare it with the vector $A \cdot v$. If $u = A \cdot v$ then the probabilistic test declares that $A = B \cdot C$. The algebraic complexity of this test is $O(n^2)$, while the complexity of the matrix multiplication is supposedly much bigger.

For n-bit integer multiplication the complexity bound $O(n \cdot \log n \cdot \exp(\log^* n))$ is due to Fürer (2007) who has improved the well known algorithm of Schönhage-Strassen (1971), where $\log^*$ is the function inverse to the function $m \mapsto exp^{(m)}(2)$. (The function $\exp(\log^* n)$ grows very slowly.) G.-Tenenbaum (2009): a probabilistic test for integer multiplication, so for given integers a, b, c to test whether $a = b \cdot c$, within complexity $O(n \cdot \log\log n \cdot \exp(\log^* n))$.

Computations vs. tests

It happens sometimes that it is more difficult to compute a function $g(x)$ at a point x than to test whether for given x, y it holds $y = g(x)$.

Freiwald (1979): a probabilistic test for matrix multiplication. Namely, given $n \times n$ matrices A, B, C to test whether $A = B \cdot C$. For a randomly chosen vector v compute vector $w := C \cdot v$, then compute vector $u := B \cdot w$ and compare it with the vector $A \cdot v$. If $u = A \cdot v$ then the probabilistic test declares that $A = B \cdot C$. The algebraic complexity of this test is $O(n^2)$, while the complexity of the matrix multiplication is supposedly much bigger.

For n-bit integer multiplication the complexity bound $O(n \cdot \log n \cdot \exp(\log^* n))$ is due to Fürer (2007) who has improved the well known algorithm of Schönhage-Strassen (1971), where $\log^*$ is the function inverse to the function $m \mapsto exp^{(m)}(2)$. (The function $\exp(\log^* n)$ grows very slowly.) G.-Tenenbaum (2009): a probabilistic test for integer multiplication, so for given integers a, b, c to test whether $a = b \cdot c$, within complexity $O(n \cdot \log \log n \cdot \exp(\log^* n))$.

Computations vs. tests

It happens sometimes that it is more difficult to compute a function $g(x)$ at a point x than to test whether for given x, y it holds $y = g(x)$.

Freiwald (1979): a probabilistic test for matrix multiplication. Namely, given $n \times n$ matrices A, B, C to test whether $A = B \cdot C$. For a randomly chosen vector v compute vector $w := C \cdot v$, then compute vector $u := B \cdot w$ and compare it with the vector $A \cdot v$. If $u = A \cdot v$ then the probabilistic test declares that $A = B \cdot C$. The algebraic complexity of this test is $O(n^2)$, while the complexity of the matrix multiplication is supposedly much bigger.

For n-bit integer multiplication the complexity bound $O(n \cdot \log n \cdot \exp(\log^* n))$ is due to Fürer (2007) who has improved the well known algorithm of Schönhage-Strassen (1971), where $\log^*$ is the function inverse to the function $m \mapsto exp^{(m)}(2)$. (The function $\exp(\log^* n)$ grows very slowly.) G.-Tenenbaum (2009): a probabilistic test for integer multiplication, so for given integers a, b, c to test whether $a = b \cdot c$, within complexity $O(n \cdot \log \log n \cdot \exp(\log^* n))$.

Computations vs. tests

It happens sometimes that it is more difficult to compute a function $g(x)$ at a point x than to test whether for given x, y it holds $y = g(x)$.

Freiwald (1979): a probabilistic test for matrix multiplication. Namely, given $n \times n$ matrices A, B, C to test whether $A = B \cdot C$. For a randomly chosen vector v compute vector $w := C \cdot v$, then compute vector $u := B \cdot w$ and compare it with the vector $A \cdot v$. If $u = A \cdot v$ then the probabilistic test declares that $A = B \cdot C$. The algebraic complexity of this test is $O(n^2)$, while the complexity of the matrix multiplication is supposedly much bigger.

For n-bit integer multiplication the complexity bound $O(n \cdot \log n \cdot \exp(\log^* n))$ is due to Fürer (2007) who has improved the well known algorithm of Schönhage-Strassen (1971), where $\log^*$ is the function inverse to the function $m \mapsto exp^{(m)}(2)$. (The function $\exp(\log^* n)$ grows very slowly.) G.-Tenenbaum (2009): a probabilistic test for integer multiplication, so for given integers a, b, c to test whether $a = b \cdot c$, within complexity $O(n \cdot \log \log n \cdot \exp(\log^* n))$.

Computations vs. tests

It happens sometimes that it is more difficult to compute a function $g(x)$ at a point x than to test whether for given x, y it holds $y = g(x)$.

Freiwald (1979): a probabilistic test for matrix multiplication. Namely, given $n \times n$ matrices A, B, C to test whether $A = B \cdot C$. For a randomly chosen vector v compute vector $w := C \cdot v$, then compute vector $u := B \cdot w$ and compare it with the vector $A \cdot v$. If $u = A \cdot v$ then the probabilistic test declares that $A = B \cdot C$. The algebraic complexity of this test is $O(n^2)$, while the complexity of the matrix multiplication is supposedly much bigger.

For n-bit integer multiplication the complexity bound $O(n \cdot \log n \cdot \exp(\log^* n))$ is due to Fürer (2007) who has improved the well known algorithm of Schönhage-Strassen (1971), where $\log^*$ is the function inverse to the function $m \mapsto exp^{(m)}(2)$. (The function $\exp(\log^* n)$ grows very slowly.) G.-Tenenbaum (2009): a probabilistic test for integer multiplication, so for given integers a, b, c to test whether $a = b \cdot c$, within complexity $O(n \cdot \log \log n \cdot \exp(\log^* n))$.

Computations vs. tests

It happens sometimes that it is more difficult to compute a function $g(x)$ at a point x than to test whether for given x, y it holds $y = g(x)$.

Freiwald (1979): a probabilistic test for matrix multiplication. Namely, given $n \times n$ matrices A, B, C to test whether $A = B \cdot C$. For a randomly chosen vector v compute vector $w := C \cdot v$, then compute vector $u := B \cdot w$ and compare it with the vector $A \cdot v$. If $u = A \cdot v$ then the probabilistic test declares that $A = B \cdot C$. The algebraic complexity of this test is $O(n^2)$, while the complexity of the matrix multiplication is supposedly much bigger.

For n-bit integer multiplication the complexity bound $O(n \cdot \log n \cdot \exp(\log^* n))$ is due to Fürer (2007) who has improved the well known algorithm of Schönhage-Strassen (1971), where $\log^*$ is the function inverse to the function $m \mapsto exp^{(m)}(2)$. (The function $\exp(\log^* n)$ grows very slowly.) G.-Tenenbaum (2009): a probabilistic test for integer multiplication, so for given integers a, b, c to test whether $a = b \cdot c$, within complexity $O(n \cdot \log\log n \cdot \exp(\log^* n))$.

Computations vs. tests

It happens sometimes that it is more difficult to compute a function $g(x)$ at a point x than to test whether for given x, y it holds $y = g(x)$.

Freiwald (1979): a probabilistic test for matrix multiplication. Namely, given $n \times n$ matrices A, B, C to test whether $A = B \cdot C$. For a randomly chosen vector v compute vector $w := C \cdot v$, then compute vector $u := B \cdot w$ and compare it with the vector $A \cdot v$. If $u = A \cdot v$ then the probabilistic test declares that $A = B \cdot C$. The algebraic complexity of this test is $O(n^2)$, while the complexity of the matrix multiplication is supposedly much bigger.

For n-bit integer multiplication the complexity bound $O(n \cdot \log n \cdot \exp(\log^* n))$ is due to Fürer (2007) who has improved the well known algorithm of Schönhage-Strassen (1971), where $\log^*$ is the function inverse to the function $m \mapsto exp^{(m)}(2)$. (The function $\exp(\log^* n)$ grows very slowly.) G.-Tenenbaum (2009): a probabilistic test for integer multiplication, so for given integers a, b, c to test whether $a = b \cdot c$, within complexity $O(n \cdot \log \log n \cdot \exp(\log^* n))$.