Topics in real and complex number complexity theory

Martijn Baartse and Klaus Meer*

Computer Science Institute, BTU Cottbus
Konrad-Wachsmann-Allee 1
D-03046 Cottbus, Germany
martijnbaartse@msn.com,meer@informatik.tu-cottbus.de

Abstract. The paper intends to introduce into topics relevant in real and complex number complexity theory. This is done in a survey style. Taking as starting point the computational model introduced by Blum, Shub, and Smale the following issues are addressed: Basic results concerning decidability and NP-completeness, transfer results of open questions between different models of computation, structural complexity inside $\mathrm{NP}_{\mathbb{R}}$, computational universality, and probabilistically checkable proofs over the real and complex numbers.

1 Introduction

Complexity theory as a mathematical discipline is a relatively young subject. In a systematic way it was basically developed since the 1970's in Theoretical Computer Science based on the Turing machine as underlying model of computation. This led to a theory nowadays basically devoted to study complexity issues of discrete problems over finite structures. Problem instances are coded by sequences of bits and the complexity of algorithms is measured by counting the number of elementary bit operations necessary. It seems that Turing himself was as well interested in complexity and accuracy issues of numercial algorithms. He also addressed an idealized model in which floating-point numbers are used as kind of entities and was working on notions like the conditioning of a problem [95].

In contrast to the observation that complexity theory often is considered as a discipline in computer science mathematicians have designed and analyzed algorithms already since centuries. Some of the most important and prominent ones were developed long before computers existed. Their inventors certainly had as well an intuition about complexity issues, though often under other perspectives. Think about algorithms like Gaussian elimination, Newton's method and notions like the order of convergence in numerical analysis, or algorithms for deciding the existence of complex solutions of a polynomial system related to Hilbert's Nullstellensatz.

Algorithms located in more classical areas of mathematics usually work with objects from uncountable continuous domains like the real or complex numbers, respectively. Often the number of basic arithmetic operations and test operations reflecting the underlying structure are of major interest. One then disregards the influence of round-off errors and uses an idealized model that computes with real or complex numbers as entities. This allows to focus on algebraic properties of problems and algorithms solving them. One of the first formalizations of such a viewpoint in complexity theory has been worked with in the area of Algebraic Complexity Theory [21] and can be traced back at least to the 1950's. Models of computation

* Both authors were supported by DFG project ME 1424/7-1. We gratefully acknowledge the support. K.Meer wants to cordially thank L.M. Pardo and J.L. Montaña for the hospitality during the Santaló summer school in Santander, on which occasion this paper was written.

used there are algebraic circuits and straight line programs. In 1989, Blum, Shub and Smale introduced a model of computation now called the BSS model, see [15, 14]. It gives a general approach to computability over rings and fields with a particular emphasis on $\mathbb{R}$ and $\mathbb{C}$. When considered over the finite field $\mathbb{Z}_2$ it results in the classical Turing machine model, whereas over fields like the real or complex numbers it gives a uniform model generalizing the ones previously used in algebraic complexity theory.

Let us mention that different models of computation have become more and more interesting in recent years both in computer science and mathematics. Think about such diverse models as Neural Networks [43], Quantum Computers [73], Analog Computers [91], Biological Computers [38] to mention a few. Beside in algebraic complexity the BSS model is also used as underlying computational model in the area of Information Based Complexity in which algorithms for numerical problems without complete information are studied [99, 100]. Computational models dealing with real numbers are also studied in Recursive Analysis. Here, objects like real numbers or real functions are coded in a certain way by Cauchy sequences leading to notions like that of a computable real (already introduced by Turing) and computable real number functions. The theory arising from this approach is focussing more on stability of real number algorithms and thus different from the setting of this paper. For introduction and some additional diverse discussions on the question which model to use in what situation we refer the reader to the following literature: [12, 18, 48, 86, 98, 99].

In this paper the Blum-Shub-Smale model builds the main topic of interest. The intention is to give an introduction into problems and methods relevant in real number complexity theory. The paper is organized as follows. Section 2 starts with a motivating example from kinematics that leads to several interesting questions in complexity, both with respect to the classical Turing and the real/complex number model. These problems are outlined and lead to a formal introduction of the real number model in the following section. There, basic complexity classes as well as the concept of $\mathrm{NP}_{\mathbb{R}}$-completeness are introduced and some first results are presented. We then focus on structural complexity theory for the real and complex numbers by discussing three different topics: Transfer results between different computational models, analysis of the structure inside $\mathrm{NP}_{\mathbb{R}}$ and $\mathrm{NP}_{\mathbb{C}}$ along the lines of a classical result by Ladner in the Turing model, and recursion theory over the reals. The rest of the paper then focusses on Probabilistically Checkable Proofs PCPs. The PCP theorem by Arora et al. [3, 2] was a cornerstone in Theoretical Computer Science giving a new surprising characterization of complexity class NP and haveing tremendous applications in the area of approximation algorithms. After introducing the main ideas behind probabilistically checkable proofs we give a detailed proof of the existence of long transparent proofs for $\mathrm{NP}_{\mathbb{R}}$ and $\mathrm{NP}_{\mathbb{C}}$. Then, we outline how one can obtain a real analogue of the PCP theorem along the lines of a more recent proof of the classical PCP theorem by Dinur [33].

The paper is written in a survey kind style. We do not cover all the interesting work that has been done since introduction of the Blum-Shub-Smale model about 20 years ago. Instead, the focus will be on topics the authors have also worked on themselves. With one exception dealing with long transparent proofs we most of the time do not present full proofs of the results treated. Instead it is tried to give the reader a feeling of the ideas behind such proofs, some more detailed and some not. More interested readers will easily find all details in the cited literature. Finally, we expect the reader to have a basic knowledge of classical complexity theory and the theory of NP-completeness [39], [1]. This is not crucial for understanding the flow of ideas, but we frequently refer to the Turing model in order to pinpoint similarities and differences between real number and classical complexity theory.

2 A motivating example

A typical problem in kinematics asks for finding suitable mechanisms that fulfill given motion tasks. Having chosen a mechanism which in principle can solve the problem the dimensions of the mechanism's components have to be determined. In its mathematical formulation this often leads to solving polynomial systems. As example of such a motion synthesis task consider the following one. Construct a mechanism which is able to generate a rigid body motion such that some constraints are satisfied. Constraints, for example, could be certain positions that have to be reachable by the mechanism. Figure 1 shows a typical example in which the point P has to be guided through given positions.

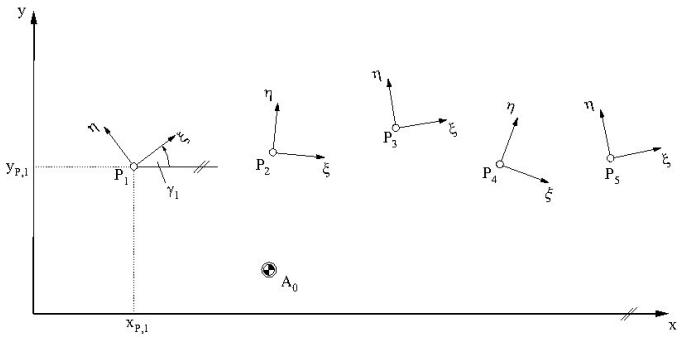

Fig. 1. Synthesis-task "Motion generation for five precision poses"

The engineer's task is to choose a suitable mechanism being able to solve the task. Then, its precise dimensions have to be determined. Here it often is desirable to perform a *complete synthesis*, i.e., to find all possible realizations of a synthesis task. This gives the engineer the possibility to choose particular mechanisms optimized with respect to additional criteria not reflected in the mathematical description, and to fine-tune. A class of mechanisms suitable for the above task are so-called Stephenson mechanisms, one example of which is shown in Figure 2.

In order to define the motion of a plane in relation to a fixed base, a $\xi - \eta$-system with origin P shall be attached to the moving plane. Similarly, a $x - y$-system shall be attached to the base A_0. Then the planar rigid body motion can be defined by certain poses of the $\xi - \eta$-system with respect to the $x - y$-system. Having chosen the kind of mechanism that is suitable to solve the problem (structural synthesis), in the dimensional synthesis step the unknown kinematic dimensions of the chosen mechanism have to be calculated. Mathematically, the problem leads to a polynomial system that has to be solved either over the real or the complex numbers depending on the formalization. Though both the number of variables and the degrees of the involved equations remain moderate, computing a complete catalogue of solutions in many cases already is demanding. Note that of course not all solutions of the resulting polynomial system are meaningful from an engineering point of view. A first complete dimensional synthesis for Stephenson mechanisms has been performed in [82], for a general introduction to solution algorithms for such kinematic problems see [90].

An important numerical technique to practically solve polynomial systems are homotopy methods. Here, the basic idea for solving $F(x) = 0$ is to start with another polynomial system G that in a certain sense has a similar structure as F. The idea then is to build a homotopy

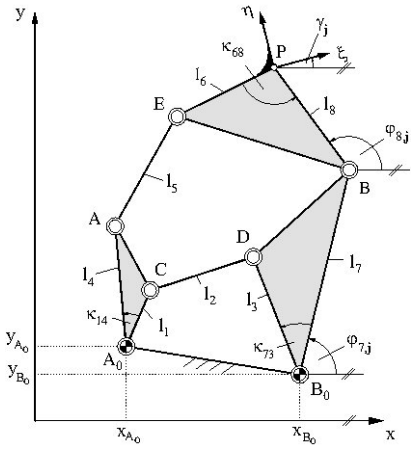

Fig. 2. Six-bar Stephenson-1I mechanism

between G and F and follow the zeros of G numerically to those of F. A typical homotopy used is the linear one $H(x,t) := (1-t)\cdot G(x) + t\cdot F(x), 0 \le t \le 1$. In order to follow this approach the zeros of the starting system should be easily computable.

Homotopy methods for solving polynomial systems are a rich source for many interesting and demanding questions in quite different areas. Their analysis has seen tremendous progress in the last 20 years and is outside the scope of this survey. There will be contributions in this volume by leading experts (which the authors of the present paper are not!) in the area, see the articles by C. Beltrán, G. Malajovich, and M. Shub. In addition, we point to some of the deep results obtained and recommend both the other contributions in this volume and the cited literature as starting point for getting deeper into homotopy methods: [84, 83, 9, 24]. Note that [84] is only the first of a series of five papers by the same authors that stands at the beginning of the many results on the above questions obtained in the last two decades.

For the purposes of the present paper we are just interested in some particular aspects arising from the above discussions. They lead into different directions of complexity theory, both with respect to the classical Turing model and real number complexity theory. In the rest of this section we discuss a problem resulting from the above approach that leads to a hard combinatorial optimization problem in classical complexity theory. The following sections then deal with the problem to decide solvability of such polynomial systems; as we shall see this is a task at the heart of real and complex number complexity theory.

For the moment let us at the moment restrict ourselves to considering polynomial systems of the form $F : \mathbb{C}^n \mapsto \mathbb{C}^n, F := (f_1, \ldots, f_n)$ over the complex numbers. Here, each component polynomial f_i is supposed to have a degree $d_i \in \mathbb{N}$. Since the system has as many equations as variables it is canonically solvable. For a successful application of homotopy methods the choice of the starting system G is of huge importance. One aspect is that if the zero structure of G significantly differs (for example, with respect to its cardinality) from that of the target system F, then many zeros from G are followed in vain, thus wasting computation time. A common idea for choosing G therefore is to get as far as possible the same number of zeros as F. There are different ways to estimate the number of complex zeros that a canonical system $F : \mathbb{C}^n \to \mathbb{C}^n$ has. A first classical result is Bézout's theorem which upper bounds the number of isolated zeros by $d := \prod_{i=1}^{n} d_i$. Though this number can be easily calculated and a system G with d many isolated zeros is easily found, the disadvantage is that it often

drastically overestimates the number of zeros of F. A prominent example is the computation of eigenvalues and -vectors of an (n,n)-matrix M, formulated via the polynomial system $Mx - \lambda x = 0, \|x\|^2 - 1 = 0$ in variables $(x,\lambda) \in \mathbb{C}^{n+1}$. The Bézout number is exponential in n, whereas clearly only n solutions exist.

To repair this disadvantage one might try to use better bounds for the number of solutions. A famous theorem by Bernstein [11] determines for *generic* systems the exact number of zeros in $(\mathbb{C}^*)^n$. Though giving the exact number applying this theorem algorithmically suffers from another aspect. In order to compute this bound one has to calculate so-called mixed volumes. The latter is a computational problem that is expected to be even much harder than solving problems in NP because in suitable formulations it leads to $\#P$-hard problems.[1] Thus at least in general one has to be careful whether to compute this bound for the target system F in order to construct G.

A third approach has been used as well, relying on so-called multi-homogeneous Bézout numbers, see [54] for more. Here, the idea is to obtain better estimates by first partitioning the problem's variables into groups and then applying Bézout's theorem to each group. In many cases like the eigenvalue problem mentioned above the resulting bound is much closer to the true number of zeros than it is the case for the Bézout number. However, the question then again is how difficult it is to find an optimal grouping of the variables such that the resulting upper bound is minimal. Though we deal with solving numerically systems of polynomials over the complex numbers, the above question leads to a typical problem about a *combinatorial* optimization problem and thus into the framework of classical complexity theory. This is due to the structure of multi-homogeneous Bézout numbers. More precisely, the optimal grouping mentioned above only depends on the support of the given system, i.e., the structure of monomials with non-zero coefficients. It is not important how these coefficients look like. As consequence, the problem changes to a purely combinatorial one. The question of how difficult it is to compute the optimal variable partitioning has been answered in [59] which gives a hardness result for the problem. It is therefore sufficient to focus on particular polynomial systems, namely systems $F := (f_1, \ldots, f_n) = 0$ in which all f_i have the same support. More precisely, consider $n \in \mathbb{N}$, a finite $A \subset \mathbb{N}^n$ and a polynomial system

$$f_1(z) \;=\; \textstyle\sum_{\alpha\in A} f_{1\alpha} z_1^{\alpha_1} z_2^{\alpha_2} \cdots z_n^{\alpha_n} \;,\; \ldots \;,\; f_n(z) \;=\; \sum_{\alpha\in A} f_{n\alpha} z_1^{\alpha_1} z_2^{\alpha_2} \cdots z_n^{\alpha_n},$$

where the $f_{i\alpha}$ are non-zero complex coefficients. Thus, all f_i have the same support A. A multi-homogeneous structure is a partition of $\{1,\ldots,n\}$ into k subsets $(I_1,\ldots,I_k)$, $I_j \subseteq \{1,\ldots,n\}$. For each such partition we define the block of variables related to I_j as $Z_j = \{z_i | i \in I_j\}$; the corresponding degree of f_i with respect to Z_j is $d_j := \max\limits_{\alpha\in A} \sum_{l\in I_j} \alpha_l$. It is the same for all polynomials f_i because all have the same support.

Definition 1. *a) The multi-homogeneous Bézout number with respect to support A and partition $(I_1,\ldots,I_k)$ is the coefficient of $\prod_{j=1}^k \zeta_j^{|I_k|}$ in the formal polynomial $(d_1\zeta_1 + \cdots + d_k\zeta_k)^n$, which is*

$$\text{Béz}(A, I_1, \ldots, I_k) = \binom{n}{|I_1|\ |I_2|\ \cdots\ |I_k|} \prod_{j=1}^k d_j^{|I_j|}$$

Here, we assume the f_i to be not yet homogeneous with respect to variable group Z_j; otherwise, replace d_j's exponent by $|I_j| - 1$.

[1] A bit more information about the class $\#P$ can be found at the end of section 4.

b) The minimal multi-homogenous Bézout number for a system having support A is

$$\min_{I \text{ partition}} \text{Béz}(A, I)$$

It is known that this minimal number bounds the number of isolated solutions in a suitable product of projective spaces and trivially is never worse than the Bézout number, see [57] for a proof. Unfortunately, as it is the case with Bernstein's bound computing such an optimial partition is a hard task. Even if one would be satisfied with only approximating the minimal multi-homogeneous Bézout number using an efficient Turing algorithm this is not likely possible. More precisely, the following holds:

Theorem 1 ([59]). *a) Given a polynomial system $F : \mathbb{C}^n \to \mathbb{C}^n$ there is no polynomial time Turing-algorithm that computes the minimal multi-homogeneous Bézout number unless $P = NP$.*

b) The same holds with respect to the task of efficiently approximating the minimal multi-homogeneous Bézout number within an arbitrary constant factor of the minimum.

Proof. As mentioned already above the task of computing the best variable partition is a purely discrete one because its definition only depends on the discrete structure of the support of the given system. The proof thus shows that an efficient algorithm for any of the two mentioned tasks would result in an efficient algorithm for the 3-coloring problem in graph theory. This problem is well known to be NP-complete in discrete complexity theory. Relating graph coloring with the problem at hand is done by assigning to a given graph G over vertex set V monomials that have the vertices of G as its variables and reflect the presence of edges and triangles in G. Doing this appropriately will result in a polynomial system whose mimimal multi-homogeneous Bézout number equals $C := \frac{(3n)!}{n!n!n!}$ in case the graph has a 3-coloring and otherwise is at least $\frac{4}{3}C$. This gives claim a). For the non-approximability result one performs a similar construction which allows to blow up the factor $\frac{4}{3}$ to an arbitrary constant. For this construction, a multiplicative structure of the multi-homogeneous Bézout numbers is exploited. □

In practice this means that one has to decide whether one would prefer a longer precomputation for getting a better starting system either by using mixed volumes or by determining a suitable multi-homogeneous structure or abstains from such a precomputation. Choosing a random starting system also in theory is an important alternative here, see [9].

Finally note that multi-homogeneous Bézout numbers also play some role outside the realm of polynomial equation solving. An example is given in [32], where the number of roots is used to bound geometrical quantities such as volume and curvature which is applied to the theory of Linear Programming.

The discussion in this section intended to show the wide range of interesting questions arising from different areas related to polynomial system solving. In engineering, many tasks can be formalized using such systems. Solving them then leads to demanding problems in many different disciplines, ranging from algebraic geometry over numerical analysis to algorithm design and complexity theory. Being the focus of the present paper we concentrate on complexity theory. Above we have seen a question arising from polynomial system solving and being located in the framework of combinatorial opimization. This is a typical area of interest in classical discrete complexity theory, where also (non-)approximability results like the one given in Theorem 1 are studied, see [4, 46, 47, 52].

However, taking into account domains like $\mathbb{R}$ and $\mathbb{C}$ over which the systems are to be solved nearby other questions arise: Can we design deterministic algorithms that decide whether a

general such system has a solution at all in the respective domain? General here in particular means that we do not longer relate the number of variables and polynomials. If such decision algorithms exist what is the intrinsic complexity of this problem, i.e., can we give good lower and upper bounds on the running time of such algorithms? Is there a way to compare related problems with respect to their complexity? These are also typical questions in classical complexity theory when dealing with a class of problems. Since we are interested in real and/or complex number instances the Turing model seems at least not appropriate to deal with all above questions. Also the homotopy methods mentioned above usually are formulated in a framework in which real numbers are considered as entities and complexity is measured, for example, in terms of Newton steps that are applied to follow the homotopy.

This led Blum, Shub, and Smale [15] to introduce a computational model formalizing algorithms over quite general domains together with a related complexity theory. This model will be the central one considered in this paper. In the next section we give a short summary of its definition, focussing on the real and complex numbers as underlying domains.

3 The real number model by Blum, Shub, and Smale

As already mentioned when dealing with algorithms over uncountable structures like $\mathbb{R}$ and $\mathbb{C}$ as they often occur in many areas of mathematics it is quite natural to formulate such algorithms in a computational model which disregards the concrete representation of objects in modern computers. Then the real or complex numbers to compute with are considered as entities and each elementary operation on such numbers is supposed to take unit time. Of course, this does not mean that issues related to such a number representation are not important in algorithm design and analysis. But if one focusses on certain aspects of a computational problem, for example, on the number of basic arithmetic operations intrinsically necessary to solve it, this abstraction makes sense. One important new aspect for the algorithmic treatment of algebraic problems is to place them into the framework of a uniform P versus NP question. This has also inspired a lot of further interesting new questions in the area of algebraic complexity, see [21, 20].

In 1989 Blum, Shub, and Smale [15] introduced a formal framework that allows to carry over important concepts from classical complexity theory in the Turing machine model to computational models over a large variety of structures. For computations over the real and complex numbers they obtained an analogue of the currently most important open question of classical complexity theory, namely the P versus NP problem. We remark that the Blum-Shub-Smale model was introduced over general ring structures; in case the underlying ring is the finite field $\mathbb{Z}_2$ it gives back the classical Turing model.

We now give a brief introduction into the model, its main complexity classes and then turn to the above mentioned version of a P versus NP question. Full details can be found in [14]. We give the basic definitions for real number computations; they easily extend to other structures.

Definition 2. *A (real) Blum-Shub-Smale (shortly: BSS) machine is a Random Access Machine over $\mathbb{R}$. Such a machine has a countable number of registers each storing a real number. It is able to perform the basic arithmetic operations $\{+, -, *, :\}$ together with branch instructions of the form: is a real number $x \geq 0$? These operations are performed on the input components and the intermediate results; moreover, there is a finite number of constants from the underlying domain used by the algorithm. They are called machine constants. In addition, there are instructions for direct and indirect addressing of registers. A BSS machine M now can be defined as a directed graph. Each node of the graph corresponds to an instruction. An outgoing edge points to the next instruction to be performed; a branch node has two*

outgoing edges related to the two possible answers of the test. Such a machine handles finite sequences of real numbers as inputs, i.e., elements from the set $\mathbb{R}^\infty := \bigcup_{k\in\mathbb{N}} \mathbb{R}^k$. *Similarly, after termination of a computation it outputs an element from* $\mathbb{R}^\infty$ *as result.*

A machine does not necessarily terminate for each of the suitable inputs. For computations over other structures one has to adjust the set of operations that can be performed accordingly. For example, when computing with complex numbers there is no ordering available, therefore tests are of the form: is a complex number $z = 0$? In a more formal treatment of the model one additionally has to specify how inputs are presented to the machine in form of a start configuration and how a terminal configuration leads to the output. This can easily be done by specifying a set of registers in which an input is placed and others where the result of a computation has to be delivered. However, being almost straightforward we skip to go through the related formalism and refer instead once more to [14].

The problems we are mainly interested in are decision problems.

Definition 3. *A set* $A \subseteq \mathbb{R}^\infty$ *is called* real decision problem. *It is called* decidable *if there is a real BSS algorithm that decides it, i.e., given an input* $x \in \mathbb{R}^\infty$ *the algorithm terminates with result 1 in case* $x \in A$ *and result* 0 *otherwise.*

The problem is semi-decidable *if the algorithm stops for all* $x \in A$ *with result* 1 *but computes forever for inputs* $x \notin A$. *Similarly for complex decision problems.*

Before turning to complexity issues one natural question in computability theory is whether there exist decision problems that cannot be decided at all by an algorithm in the respective model.

Definition 4 (Real Halting Problem). *The* real Halting Problem $\mathbb{H}_\mathbb{R}$ *is the following decision problem: Given a code* $c_M \in \mathbb{R}^\infty$ *of a real BSS machine* M *and an* $x \in \mathbb{R}^\infty$, *does machine* M *stop its computation on input* x*?*

The Halting Problem was one of the first that has been shown to be undecidable in the real number model in [15]. There are further problems shown to be undecidable by simple topological arguments. Recall that the Mandelbrot set $\mathcal{M}$ is defined as the set of those $c \in \mathbb{C}$ whose iterates under the map $z \mapsto z^2 + c$ remain bounded when starting the iteration in $z = 0$.

Theorem 2 ([15]). *The following problems are undecidable in the real number model: The real Halting problem* $\mathbb{H}_\mathbb{R}$, *the problem* $\mathbb{Q}$ *to decide whether a given real number is rational, the Mandelbrot set* $\mathcal{M}$ *seen as subset of* $\mathbb{R}^2$. *Moreover,* $\mathbb{H}_\mathbb{R}, \mathbb{Q}$ *and the complement of* $\mathcal{M}$ *in* $\mathbb{R}^2$ *are semi-decidable.*

In the complex BSS model, the corresponding complex version of the Halting problem is undecidable. The same holds for deciding the integers $\mathbb{Z}$. *Both problems are semi-decidable.*

Proof. For proving undecidability of $\mathbb{H}_\mathbb{R}$ in a first step one constructs a universal BSS machine, i.e, a machine U that takes as its input pairs (c_M, x), where $c_M \in \mathbb{R}^\infty$ codes a BSS machine M as element in $\mathbb{R}^\infty$ and $x \in \mathbb{R}^\infty$ is an input for this machine M. Machine U on such an input simulates the computation of M on x. The computational model is strong enough to guarantee U's existence, though the precise construction is tedious. Now, undecidability is otained by a typical diagonalization argument in which x is taken to be c_M. Semi-decidability easily follows from performing U's computation. If the universal machine halts the input belongs to $\mathbb{H}_\mathbb{R}$ by definition, otherwise not. The argument over $\mathbb{C}$ is the same.

For the other two real number problems undecidability follows from the topological structure of the respective problems. First, every semi-decidable set in $\mathbb{R}^\infty$ is an at most countable

union of semi-algebraic sets.[2] This follows from the algebraic structure of the basic operations allowed in algorithms. Both the complement of $\mathbb{Q}$ in $\mathbb{R}$ and the Mandelbrot set are known not to be such a countable union, so it follows these that sets cannot be semi-decidable. But since decidability of a problem A is equivalent to semi-decidability of both A and its complement both problems can neither be decidable. Semi-decidability of $\mathbb{Q}$ is straightforward by enumerating $\mathbb{Q}$, that of $\mathbb{R}^2 \setminus \mathcal{M}$ follows immediately from $\mathcal{M}$'s definition: As soon as an iterate of an input $c \in \mathbb{R}^2$ in absolute value becomes larger than 2 this c belongs to $\mathcal{M}$'s complement. This condition as well characterizes the complement.

As to undecidability of the integers over $\mathbb{C}$ another frequently used topological argument is helpful. Consider a potential machine deciding the problem. Then any input x^* that is algebraically independent of the extension field obtained when joining the complex machine constants to $\mathbb{Q}$ must be branched along the not-equal-alternative of each test node. This computation path must be finite. But the set of inputs that are branched at least once along an equal-alternative is finite by the fundamental theorem of algebra. Thus there must exist integers for which the machine uses the same computation path and gives the same result as for x^*. On such integers the decision is false and the machine has to fail. □

The above statements for $\mathbb{Q}$ and $\mathbb{Z}$ are closely related to so called definability issues in real and algebraically closed fields, see [16]. We shall exploit similar arguments again below when analyzing computationally universal problems in section 4.3.

Next, algorithms should be equipped with a time measure for their execution. As usual, in order to then classify problems with respect to the running time needed to solve them one also has to define the size of an instance. The time consumption is considered as function in the input size. The intuitive approach for measuring the algebraic complexity of a problem described at the beginning of this section is now made more precise as follows.

Definition 5. *Let M be a real BSS machine. The* size *of an element $x \in \mathbb{R}^k$ is $size_{\mathbb{R}}(x) := k$. The* cost *of each basic operation is* 1. *The cost of an entire computation is the number of operations performed until the machine halts. The (partial) function from $\mathbb{R}^\infty$ to $\mathbb{R}^\infty$ computed by M is denoted by Φ_M. The cost of M's computation on input $x \in \mathbb{R}^\infty$ is also called its* running time *and denoted by $T_M(x)$. If $\Phi_M(x)$ is not defined, i.e., M does not terminate on x we assign the running time $T_M(x) := \infty$.*

Most of the well known Boolean time-complexity classes can now be defined analogously over the reals. We give a precise definition of the two main such classes.

Definition 6 (Complexity classes, completeness).

a) A problem $A \subseteq \mathbb{R}^\infty$ is in class $\mathrm{P}_{\mathbb{R}}$ *(*decidable in polynomial time over $\mathbb{R}$*) iff there exist a polynomial p and a real BSS machine M deciding A such that $T_M(x) \leq p(size_{\mathbb{R}}(x))\ \forall x \in \mathbb{R}^\infty$.*

b) A is in $\mathrm{NP}_{\mathbb{R}}$ *(*verifiable in non-deterministic polynomial time over $\mathbb{R}$ *) iff there exist a polynomial p and a real BSS machine M working on input space $\mathbb{R}^\infty \times \mathbb{R}^\infty$ such that*
(i) $\Phi_M(x,y) \in \{0,1\}\ \forall x \in \mathbb{R}^\infty, y \in \mathbb{R}^\infty$
(ii) $\Phi_M(x,y) = 1 \implies x \in A$
(iii) $\forall x \in A\ \exists y \in \mathbb{R}^\infty\ \Phi_M(x,y) = 1$ and $T_M(x,y) \leq p(size_{\mathbb{R}}(x))$

c) A problem A in $\mathrm{NP}_{\mathbb{R}}$ *is* $\mathrm{NP}_{\mathbb{R}}$-complete *iff every other problem in* $\mathrm{NP}_{\mathbb{R}}$ *can be reduced to it in polynomial time.* Polynomial time reducibility *from problem B to problem A means: There is a polynomial time computable function $f : \mathbb{R}^\infty \to \mathbb{R}^\infty$ which satisfies:*

[2] A semi-algebraic set in $\mathbb{R}^n$ is a finite Boolean combination of sets defined as solution of finitely many polynomial equalities and inequalities.

$\forall x \in \mathbb{R}^\infty : x \in B \Leftrightarrow f(x) \in A$. *This type of reduction is also called polynomial time many one reduction.*

d) The corresponding definitions over $\mathbb{C}$ *lead to classes* $P_\mathbb{C}, NP_\mathbb{C}$, *and* $NP_\mathbb{C}$*-completeness.*

When talking about a problem $A \in \mathrm{NP}_\mathbb{R}$, for an input $x \in A$ the y whose existence is required in part b,ii) above can be seen as a proof of x's membership in A. The definition then requires that correctness of this proof can be checked efficiently in the size of x. Below we often use the phrase that on input x machine M guesses a proof y for establishing $x \in A$.

The definition directly implies that $\mathrm{P}_\mathbb{K}$ is included in $\mathrm{NP}_\mathbb{K}$ for $\mathbb{K} \in \{\mathbb{R}, \mathbb{C}\}$. The currently most important open question in real and complex number complexity theory is whether these inclusions are strict. This is easily seen to be equivalent to the existence of already one single $\mathrm{NP}_\mathbb{K}$-complete problem which does not belong to the corresponding class $\mathrm{P}_\mathbb{K}$.

The following closely related two problems turn out to be extremely important for the entire theory and will occur in one or the other form throughout the rest of this paper.

Definition 7. *Let* $\mathbb{K}$ *be a field of characteristic* 0.

a) The Hilbert-Nullstellensatz problem is the problem to decide whether a given system of polynomial equations

$$p_1(x_1, \ldots, x_n) = 0 \ , \ldots , \ p_m(x_1, \ldots, x_n) = 0,$$

where all p_i *are polynomials in* $\mathbb{K}[x_1, \ldots, x_n]$ *has a common solution in* $\mathbb{K}^n$.

We denote the problem by $\mathrm{QPS}_\mathbb{K}$ *for Quadratic Polynomial Systems if, in addition, all* p_i *have a total degree bounded by* 2.

b) If $\mathbb{K} = \mathbb{R}$ *the feasibility problem* 4-$FEAS_\mathbb{R}$ *is the task to decide whether a polynomial* $f \in \mathbb{R}[x_1, \ldots, x_n]$ *of total degree at most* 4 *has a zero in* $\mathbb{R}^n$.

We shall see that the above problems in the BSS models over $\mathbb{R}$ and $\mathbb{C}$, respectively, take over the role of the famous 3-SAT problem in the Turing model.

Example 1. The following problems are easily seen to belong to the respective class NP over $\mathbb{R}$ or $\mathbb{C}$.

a) $\mathrm{QPS}_\mathbb{K}$ belongs to $\mathrm{NP}_\mathbb{K}$ for $\mathbb{K} \in \{\mathbb{R}, \mathbb{C}\}$, 4-$FEAS_\mathbb{R}$ belongs to $\mathrm{NP}_\mathbb{R}$. In all cases the verification procedure guesses a common zero of the given system or the given polynomial, respectively. The polynomials then are evaluated in this point and it is finally checked whether the guess actually was a zero. The (algebraic) size of the guess equals the number of variables the polynomials depend on. The evaluation procedure obviously only needs a number of arithmetic steps and tests that is polynomially bounded in the input size of the system. For the latter we take a dense representation, i.e., also zero-coefficients for monomials not present contribute to the size by 1. This in principle could be done differently, but here we want to avoid discussions about sparse polynomials.

As an easy example of a polynomial time reduction note that $\mathrm{QPS}_\mathbb{R}$ straightforwardly is reducible to 4-$FEAS_\mathbb{R}$ by defining an instance of the latter as the sum of the squared degree-2-polynomials of the given $\mathrm{QPS}_\mathbb{R}$ instance. Obviously, over $\mathbb{C}$ this reduction does not work correctly.

b) Another example of a problem in $\mathrm{NP}_\mathbb{R}$ is the Linear Programming problem $\mathrm{LP}_\mathbb{R}$. Here, the input is a real (m, n)-matrix A together with a vector $b \in \mathbb{R}^m$. The question to decide is whether there is a real solution $x \in \mathbb{R}^n$ satisfying $A \cdot x \leq b$. The input size is $O(mn + m)$, a verification proof once again guesses a potential solution x and then verifies whether it solves the system. The problem, when restricted to rational input data and considered in the Turing model, is known to be decidable in polynomial time. This is the well known result

implied by the ellipsoid and the interior-point methods. The running time of those algorithms, however, are polynomial in the input size only because the discrete input size is larger than the algebraic one, taking into account the bit-length necessary to represent the rational data. It is a major open question in the theory of Linear Programming whether a polynomial time algorithm also exists in the real number model [88].

By introducing slack variables and reducing the number of variables per equation to at most 3 by using additional variables, the real Linear Programming problem is polynomial time reducible to $\mathrm{QPS}_{\mathbb{R}}$. Even though it is currently open whether $\mathrm{LP}_{\mathbb{R}} \in \mathrm{P}_{\mathbb{R}}$ it is not expected that the problem becomes much harder in terms of complexity classes it belongs to, for example, becoming $\mathrm{NP}_{\mathbb{R}}$-complete. This would have strange consequences [62]. So $\mathrm{LP}_{\mathbb{R}}$ might well be a kind of intermediate problem between $\mathrm{P}_{\mathbb{R}}$ and $\mathrm{NP}_{\mathbb{R}}$-complete ones. However, even the theoretical existence of such 'intermediate' problems is currently open. We comment on this point once again below after Theorem 18.

In order to justify the importance of the $\mathrm{NP}_{\mathbb{R}}$-completeness notion first it has to be shown that such problems exist.

Theorem 3 ([15]). *For $\mathbb{K} \in \{\mathbb{R}, \mathbb{C}\}$ the Hilbert-Nullstellensatz problem $\mathrm{QPS}_{\mathbb{K}}$ is $\mathrm{NP}_{\mathbb{K}}$-complete. Over the reals the same holds for 4-$FEAS$.*

Proof. The proof in principle follows the structure of the one for Cook's theorem, i.e., the proof of NP-completeness of the 3-SAT problem in the Turing model. However, certain adjustments to the framework are needed. Given a problem $A \in \mathrm{NP}_{\mathbb{R}}$, a BSS machine M witnessing this membership, and an input $x \in \mathbb{R}^{\infty}$ a quadratic polynomial system has to be constructed that is solvable if and only if $x \in A$. The system is obtained by representing M's possible computation on x and a suitable guess y by a rectangular matrix. Rows represent the time steps of M during computations, columns represent the registers in which input, guess and intermediate results are stored. Given the polynomial running time of M this matrix has polynomial size only. Now for each assignment to the matrix with real numbers one tries to express by polynomial equations that the first row's assignment corresponds to the input configuration for M on x, each row's assignment implies that of the next row by applying a computational step of M, and the last row-assignment indicates that M's computation is accepting. Then $x \in A$ if and only if a suitable assignment of the matrix exists if and only if the generated $\mathrm{QPS}_{\mathbb{R}}$ instance is solvable. □

An important difference between $\mathrm{NP}_{\mathbb{R}}$ and its discrete counterpart NP is the fact that the guess y in the above definition of $\mathrm{NP}_{\mathbb{R}}$ is taken from an uncountable space. This is very much different to the classical setting where the search space for a correct membership proof is an element in $\{0,1\}^*$, i.e., a finite bitstring. Since the length of the guess is polynomially bounded in the length of the input, over finite alphabets the search space is finite, implying that each problem in NP is decidable in simply exponential time. Over the reals and the complex numbers this turns out to be true as well relying on much deeper results.

Theorem 4. *Let $d \in \mathbb{N}$ and let A be a (basic) semi-algebraic set that is given as*

$$A := \{x \in \mathbb{R}^n | p_i(x) \Delta_i 0, 1 \leq i \leq s\} \ ,$$

where each p_i is a polynomial of degree at most d with real coefficients and $\Delta_i \in \{=, \neq, \geq, >\}$.

Then emptiness of A can be decided by an algorithm in the BSS model that runs in $O\left((s \cdot d)^{O(n)}\right)$ arithmetic steps, i.e., in simply exponential time.

A similar statement is true for the complex numbers. It follows that both all problems in $\mathrm{NP}_{\mathbb{R}}$ and in $NP_{\mathbb{C}}$ can be decided by a BSS algorithm of the respective model in simply exponential time.

The proof of this theorem is out of the scope of this paper. It is a special case of Quantifier Elimination (the existential quantifiers are removed by the decision procedure), a question having a long tradition for real and algebraically closed fields. The first procedure for general quantifier elimination in these structures was given by Tarski [92]. Then, startin in the 1970'ies research has focussed on getting better complexity estimates, until finally for existentially quantified problems simply exponential complexity bounds could be established. Significant contributions to the complexity of such methods have been made, for example, in [7, 42, 44, 79]. The above result in this particular form if taken from [79]. Once again, with M. Giusti and J. Heintz two experts in complexity aspects of elimination theory will contribute with related issues to this volume, so the interested reader should consult their articles.

Thus we arrived at some first cornerstones in BSS complexity theory. There are problems that are algorithmically undecidable in the model. And there is a reasonable theory of efficient algorithms and hard problems in form of the $\mathrm{P}_{\mathbb{R}}$ versus $\mathrm{NP}_{\mathbb{R}}$ question. Its importance is justified by the existence of natural $\mathrm{NP}_{\mathbb{R}}$-complete problems, and all such problems can be decided within the algorithmic framework in simply exponential time. It is currently not known whether more efficient algorithms exist. Similar statements are true in the complex number model. In the following sections we shall discuss several structural questions taking their starting points in the above results.

4 Structural Complexity

In this section we exemplify typical methods and questions analysed in structural complexity theory for the BSS model over $\mathbb{R}$ and $\mathbb{C}$ on the basis of three thematic areas. These topics include transfer theorems, the structure inside $\mathrm{NP}_{\mathbb{R}}$, and recursion theory on $\mathbb{R}$.

4.1 Transfer principles for P versus NP

One of the research lines from the beginning of real and complex number complexity theory was to study similarities and differences between classical complexity theory for the Turing model dealing with finite alphabets and complexity theory in alternative than discrete structures. Defining literally the main problem whether the classes P and NP coincide is an easy task once the underlying model has been specified, but this of course does not automatically make the question interesting. Nevertheless, it fortunately turned out to be the case that several non-trivial problems arise in such alternative models. The most prominent one certainly again is the P versus NP question, this time over $\mathbb{R}$ and $\mathbb{C}$.

It is natural to wonder whether the answer(s) to this question are related for different structures. More generally, it seems interesting to combine major open complexity theoretic question in one computational model with related questions in another. Ideally, this enlarges the tools of methods that could be used to attack such open problems. Results following this guideline are usually called transfer theorems. This subsection intends to highlight some such transfer results relating different versions of the BSS model with classical complexity theory.

Of course in general it is not clear what difficulties one meets when studying a problem over continuous domains which literally is similar to the respective question in the Turing model. Sometimes, an easy solution over the reals is possible due to other arguments that can be exploited. Sometimes, a deep analysis combines different models, and sometimes, new interesting questions arise which do not have a significant counterpart in the discrete world. All of that has been observed, and the present section tries to outline some results into these directions.

Note that we saw already one good example where a literally similar question needs much more efforts to be solved in the real number model. The decidability of all problems in class $\mathrm{NP}_{\mathbb{R}}$ mentioned in the previous section relies on the non-trivial task to perform quantifier elimination in real closed fields. And obtaining a comparable time bound to the discrete world, where NP trivially can be decided in simple exponential time, becomes even more difficult.

An opposite example where a question being extremely difficult in the Turing setting turned out to be much easier in the BSS framework is the following early result by Cucker on non-parallelizability of the class $\mathrm{P}_{\mathbb{R}}$. The real number class $\mathrm{NC}_{\mathbb{R}}$ used in the statement intuitively is defined as those decision problems in $\mathrm{P}_{\mathbb{R}}$ that can be parallelized. This means they can be solved using a polynomial number of BSS machines working in parallel which but all running in poly-logarithmic time only. The theorem states that not all problems in $\mathrm{P}_{\mathbb{R}}$ can be parallelized that way.

Theorem 5 ([29]). *$NC_{\mathbb{R}} \subsetneq P_{\mathbb{R}}$*

Proof. The proof relies on an irreducibility argument by defining a decision problem whose solution requires to compute an irreducible polynomial of exponential degree. It is then shown that this computation basically cannot be split into different parallel branches. □

The above argument relies on the structure of irreducible varieties over $\mathbb{R}$. Thus, it cannot be applied to the analogue question for the Turing model. It is still is one of the major open problems in classical complexity theory.

Comparing different models is most interesting when dealing with problems that somehow can be considered in both models. This often can be achieved by restricting the set of input instances. An important example is the Hilbert Nullstellensatz problem. Given a system of polynomial equations as input on the one hand side can be considered as problem for both $\mathrm{NP}_{\mathbb{R}}$ and $\mathrm{NP}_{\mathbb{C}}$, depending on the set of coefficients. Solvability accordingly can be required either over the reals or the complex numbers. If we restrict coefficients to be rationals or integers the question as well makes sense in the Turing model, even when asking for real solutions. Moreover, if one is interested in $\{0,1\}$-solutions this can be forced by binding each single variable x through an additional equation $x \cdot (x-1) = 0$. Note that also in the Turing model it is an NP-complete task to decide whether a system of quadratic equations with integer coefficients has a real or complex solution, respectively.

Thus, for a comparison of these models an important question to solve is: Suppose, the real (or complex) QPS problem could be decided by an efficient algorithm. Could this algorithm somehow be turned into a Turing algorithm such that the changed algorithm turns out to be efficient as well for the discrete variant of QPS? Clearly, a main aspect for attacking this question is whether the potentially real or complex machine constants which the given algorithm uses could be replaced somehow in order to obtain a Turing algorithm. This topic of replacement of machine constants has turned out to be extremely interesting and demanding. In the results below such a replacement in one or the other way always is crucial. The resulting effects can be quite different. For some tasks constants can be replaced without much harm to complexity aspects, for some others such a replacement introduces new aspects like non-uniformity of algorithms, and there are situations where it remains an open question whether a replacement is possible.

A first deep result dealing with such issues was given in [13]. Here, the QPS problem is studied over arbitrary algebraically closed fields of characteristic 0. The proof of Theorem 3 shows that QPS defined accordingly is complete for the corresponding class NP in all such fields. Thus it is natural to ask whether an answer to any of the related P versus NP problems would have implications for the other fields as well. In fact, this is true.

Theorem 6 ([13]). *For all algebraically closed fields* $\mathbb{K}$ *of characteristic* 0 *the* P *versus* NP *question has the same answer in the BSS model over* $\mathbb{K}$*, i.e., either in all these fields* $\mathrm{NP}_{\mathbb{K}} = \mathrm{P}_{\mathbb{K}}$ *or in all such fields* $\mathrm{P}_{\mathbb{K}}$ *is strictly contained in* $\mathrm{NP}_{\mathbb{K}}$.

Proof. The theorem's first proof in [13] performs the elimination of constants using number theoretic arguments. We outline an alternative proof given by Koiran [51] which is based on results on quantifier elimination. A first observation using the introductory remarks before the statement of the theorem shows that a central problem to consider is QPS over the algebraic closure $\bar{\mathbb{Q}}$ of the rational number field. Since each field $\mathbb{K}$ under consideration has to contain $\bar{\mathbb{Q}}$ it suffices to analyse the QPS problem over $\mathbb{K}$ and over $\bar{\mathbb{Q}}$. There are then two directions to prove; the first asserts that the existence of an efficient algorithm for a hard problem over $\mathbb{K}$ implies the same for a hard problem over $\bar{\mathbb{Q}}$. This is the more difficult statement. The converse direction states that an efficient algorithm for the Hilbert Nullstellensatz problem in $\bar{\mathbb{Q}}$ can be lifted to one for the same problem over $\mathbb{K}$. It is true by well known results from model theory, basically applying the so called strong transfer principle for the theory of algebraically closed fields. This was first done by Michaux [72]. Since its proof does not rely on techniques for eliminating machine constants we do not go into more details here.

Let us thus focus on the other direction. Note that a QPS instance with coefficients from $\bar{\mathbb{Q}}$ has a solution over $\mathbb{K}$ if and only if it has a solution over $\bar{\mathbb{Q}}$. This follows from Hilbert's Nullstellensatz. Below we choose $\mathbb{K} := \mathbb{C}$, but the arguments remain basically the same for any other $\mathbb{K}$.

Suppose then there were an efficient algorithm solving QPS over $\mathbb{C}$, i.e., proving $\mathrm{P}_{\mathbb{C}} = \mathrm{NP}_{\mathbb{C}}$. This algorithm also solves QPS over $\bar{\mathbb{Q}}$ efficiently, but in order to conclude $\mathrm{P}_{\bar{\mathbb{Q}}} = \mathrm{NP}_{\bar{\mathbb{Q}}}$ the algorithm is not allowed to use constants from $\mathbb{K} \setminus \bar{\mathbb{Q}}$. Suppose the potential decision algorithm uses transcendental constants; with a moderate technical effort one can additionally assume without loss of generality that all these machine constants are algebraically independent. For the computation on a fixed input from $\bar{\mathbb{Q}}$ one can view each equality test performed by the algorithm as a polynomial with coefficients in $\bar{\mathbb{Q}}$ that is evaluated in the set of transcendental machine constants. Thus, no such algebraic equality test is answered positively in a reasonably small neighborhood of the set of machine constants. Consequently, for all such points the machine computes the same yes-no answers. The task is then to find a rational point in such a neighborhood. A clever application of the complexity statements behind Theorem 4 for $\mathbb{C}$ guarantees such points to exist and being not too large. 'Not too large' here means that they can be computed fast enough starting from the constant 1 by a machine working over $\bar{\mathbb{Q}}$. Thus, replacement of transcendental constants by efficiently computable algebraic ones can be accomplished and an efficient algorithm for the $\mathrm{NP}_{\bar{\mathbb{Q}}}$-complete problem QPS is found. □

The theorem unfortunately does not solve the P versus NP problem in any of those structures but just guarantees the currently unknown answer to be the same for all related fields. The next transfer theorem discussed relates the P versus NP question in the complex BSS model with randomized complexity classes in classical complexity. Here, the well known class BPP denotes decision problems L that can be solved by randomized polynomial time algorithms allowing a small two-sided error, i.e., the procedure might fail both on elements in L and its complement with a small constant probability. BPP thus stands for bounded error probability polynomial time. The precise placement of class BPP in discrete complexity theory with respect to its relations to P and NP is a major open problem. Though recent results have led to the reasonable possibility that BPP equals P, it is not even known whether BPP is a proper subset of the set of problems that can be solved in non-deterministic exponential time, see [1]. The next result shows an interesting connection between the complex BSS model and BPP. The main ingredients for its proof were independently given by Koiran [49] and Smale [87], though in the cited sources the theorem seems not outspoken explicitly.

Theorem 7 ([49, 87]). *Suppose $P_{\mathbb{C}} = NP_{\mathbb{C}}$ in the complex number BSS model, then* NP $\subseteq$ BPP *in the Turing model.*

Proof. Once again, the main idea is to extract from an efficient complex algorithm for $\mathrm{QPS}_{\mathbb{C}}$ a randomized Turing algorithm for a suitable NP-complete variant of QPS. In order to replace non-rational constants used by the given algorithm randomization enters at two places. First, relying once more on arguments like those used in the previous theorem one tries to find small rational constants that could be used instead of the original ones. These constants are chosen by random from a large enough set and their appropriateness with high probability is established by using the famous Schwartz-Zippel Lemma [103]. However, even if the new rational coefficients work fine, it might be the case that intermediate results produced in the initial $\mathrm{P}_{\mathbb{C}}$ algorithm get too large when counting bit-operations in the Turing model. This is solved by doing all computations modulo randomly chosen integers located in a suitable set. For most of them the computation then still works correctly, but now running in polynomial time as well in the Turing model. □

Even though the relation between BPP and NP is currently unknown nobody expects NP $\subseteq$ BPP to be true. The inclusion would have dramatical consequences concerning complxeity classes above NP, and here foremost the collapse of the so called polynomial hierarchy. So if one could prove that this hierarchy does not collapse in classical complexity theory it would follow $\mathrm{P}_{\mathbb{C}} \neq \mathrm{NP}_{\mathbb{C}}$ in the complex number model.

The two previous results are not known to hold for the real numbers as well. The attempt to obtain transfer results here seems to meet more obstacles. We shall encounter this phenomenon once again in the next subsection. It is then natural to first consider restrictions of the real number model in order to figure out whether more could be said for such restricted models. In addition, this might shed more light on where the difficulties lie.

The first such restriction considered here is called additive BSS model. The difference with the full real model is that only additions and subtractions are allowed as arithmetic operations. For the following discussion we also restrict ourselves to so called *constant-free* additive algorithms, i.e., there are no other machine constants used than 0 and 1. Nevertheless note that for an $\mathrm{NP}_{\mathbb{R}}^{add}$ verification algorithm it is still allowed to work with real guesses. Similar results as those described below can be obtained as well if arbitrary constants are allowed. We comment on that at the end of this subsection.

In the additive model classes $\mathrm{P}_{\mathbb{R}}^{add}$ and $\mathrm{NP}_{\mathbb{R}}^{add}$ are defined analogously to the full model.[3] Algorithms in the additive model still can work with inputs being vectors of real numbers. However, when inputs are restricted to stem from $\{0,1\}^*$, each additive computation can be simulated with only polynomial slow down by a Turing machine. This is true because the sizes of intermediate results in such a computation cannot grow too much, in contrats to the case in the full model when repeated squaring of a number is performed. The following theorem by Fournier and Koiran shows that proving lower bounds in the additive model is of the same difficulty as in classical complexity theory.

Theorem 8 ([36]). *It is* P = NP *in the Turing model if and only if it holds $P_{\mathbb{R}}^{add} = NP_{\mathbb{R}}^{add}$ in the additive (constant-free) model over* $\mathbb{R}$.

Proof. In a first step one analyzes the power of additive machines on discrete languages. For $L \subseteq \mathbb{R}^*$ one denotes it Boolean (i.e., discrete) part as $BP(L) := L \cap \{0,1\}^*$, and similarly for entire complexity classes. The above argument on a moderate growing of intermediate results

[3] In literature the constant-free classes usually are denoted with an additional superscript 0. We skip that here in order to minimize the notational overhead.

implies the equality $BP(\mathrm{P}_{\mathbb{R}}^{add}) = \mathrm{P}$. The analogue equality $BP(\mathrm{NP}_{\mathbb{R}}^{add}) = \mathrm{NP}$ is true as well, though for proving it one first has to show that guessing real components in a verification proof can be replaced by guessing small rational components. This is only known to be true in the additive model, for the full BSS model it is an open question and conjectured to be false. These observations suffice to show the easier direction, namely that $\mathrm{P}_{\mathbb{R}}^{add} = \mathrm{NP}_{\mathbb{R}}^{add}$ implies P = NP. The difficult one is the converse, and as usual we only outline the main proof ingredients. Suppose that P = NP. The idea is to show that any problem in $\mathrm{NP}_{\mathbb{R}}^{add}$ can be efficiently decided by an additive machine which has access to a discrete oracle for NP. The latter means that the algorithm is allowed to generate questions to a classical NP-complete problem and gets a correct answer at unit cost.[4] Since we assume P = NP such an oracle device can be replaced by an efficient algorithm in the Turing model, which of course is efficient as well in the additive model. This would yield the assertion. The design of this oracle algorithm is the heart of the proof. It relies on a deep result by Meyer auf der Heide [70, 71] on point location for arrangements of hyperplanes. This result establishes how to construct non-uniformly a so called linear decision tree for solving the point location problem. The proof shows how this algorithm can be made uniform if an NP oracle is available. We only outline it very roughly here. Given a problem $L \in \mathrm{NP}_{\mathbb{R}}^{add}$ the non-deterministic additive algorithm generates an exponential family of hyperplanes describing membership in L. These hyperplanes arise from accepting computations, and since it suffices to guess small rational numbers only in non-deterministic algorithms the coefficients of those hyperplanes remain small rational numbers. Moreover, the set of hyperplanes decomposes the respective part of the input space $\mathbb{R}^n$ into regions each of which either belongs to L or its complement. The main part of the proof now shows that an additive machine which is allowed to use a classical NP oracle can solve the following task: For an input $x \in \mathbb{R}^n$ it computes a set S described by few affine inequalities with small coefficients such that $x \in S$ and S is either completely contained in L or in its complement. The construction of S needs Meyer auf der Heide's results in a clever way, using at several stages the oracle. The final decision whether $S \subseteq L$ or not again is decided by means of the NP oracle. □

The theorem shows that there are deep relations between major open questions in classical complexity theory and real number models. If additive machines are allowed to use real constants similar results have been proved in the same paper [36] relying on results from [50, 30]. Basically the use of such constants introduces non-uniformity for discrete problems, that is Boolean parts of the class $\mathrm{P}_{\mathbb{R}}^{add}$ when constants can be used turn out to equal the class $\mathrm{P}/poly$ in the Turing model; the latter defines problems that can be decided efficiently by additional use of a moderate non-uniformity, see also below. The same is true for $\mathrm{NP}_{\mathbb{R}}^{add}$ and leads to a corresponding version of the previous theorem. Note that further restricting the model, for example by only allowing equality branches and therefore considering $\mathbb{R}$ as unordered vector space does not lead to a similar transfer result. In such a model the corresponding class P provably is a proper subclass of NP, see [61].

Of course, it is challenging to extend connections to discrete complexity like the ones shown above to the full real number model as well.

4.2 Inside $\mathbf{NP}_{\mathbb{R}}$ and $\mathbf{NP}_{\mathbb{C}}$

The problems to be considered in this subsection as well require to deal with the machine constants of algorithms and how to replace them by more suitable ones. This time, however, the goal will not be to replace arbitrary constants by rational ones. Instead, a family of

[4] Oracle algorithms will be considered once more in Section 4.3.

constants used non-uniformly should be replaced by a single fixed set of machine constants. Before understanding the task and how the replacement in some situations can be achieved we introduce the problem to be studied now.

Starting point of the investigations is the following classical result by Ladner [53], which in the Turing model analyzes the internal structure of complexity class NP in case P $\neq$ NP is supposed to be true:

Theorem 9 ([53]). *Suppose* NP $\neq$ P. *Then there are problems in* NP $\setminus$ P *which are not* NP*-complete under polynomial time many-one reductions.*

Proof. The proof relies intrinsically on the countability of both the family $\{P_1, P_2, \ldots\}$ of polynomial time Turing machines and the family $\{R_1, R_2, \ldots\}$ of polynomial time reduction machines in the Turing model. A diagonalization argument is performed to fool one after the other each machine in the two sets. This is briefly done as follows. Given an NP-complete problem L one constructs a problem $\tilde{L} \in$ NP such that all machines R_i fail to reduce L to $\tilde{L}$ on some input and all machines P_i fail to decide $\tilde{L}$ correctly on some input. Towards this aim the definition of $\tilde{L}$ proceeds dimension-wise while intending to fool step by step $P_1, R_1, P_2, R_2, \ldots$. In order to fool an P_i the language $\tilde{L}$ is taken to look like L for inputs of sufficiently large size. Conversely, in order to fool reduction algorithm R_i for sufficiently many of the following input-sizes $\tilde{L}$ is defined to look like an easy problem. Both steps together imply that none of the machines P_i, R_i works correctly for the new language $\tilde{L}$. Finally, a typical padding argument guarantees $\tilde{L} \in NP$.

Extensions of Ladner's result can be found, for example, in [81].

Considering computational models over uncountable structures like $\mathbb{R}$ and $\mathbb{C}$ the above diagonalization argument - at least at a first sight - fails since the corresponding algorithm classes become uncountable. So it is not obvious whether similar statements hold for $\mathrm{NP}_{\mathbb{R}}$ and/or $\mathrm{NP}_{\mathbb{C}}$. We shall now see that studying this question in the extended framework leads to interesting insights and new open problems.

As it was the case in the previous subsection also for Ladner's problem the complex BSS model is easier to handle than the real model. The first Ladner like result in the BSS framework in [58] was shown for the complex classes $\mathrm{P}_{\mathbb{C}}$ and $\mathrm{NP}_{\mathbb{C}}$:

Theorem 10 ([58]). *Suppose* $NP_{\mathbb{C}} \neq P_{\mathbb{C}}$. *Then there are problems in* $NP_{\mathbb{C}} \setminus P_{\mathbb{C}}$ *which are not* $NP_{\mathbb{C}}$*-complete under polynomial time many-one reductions in the complex number BSS model.*

Proof. The proof relies on Theorem 6 from the previous subsection. It will be crucial to transfer the question from the uncountable structure $\mathbb{C}$ of complex numbers to the countable one $\bar{\mathbb{Q}}$, the algebraic closure of $\mathbb{Q}$ in $\mathbb{C}$.

In a first step we answer Ladner's problem positively in the BSS model over $\bar{\mathbb{Q}}$. This can be done along the lines of the classical proof sketched above since both families of algorithms mentioned therein are countable. Let $\tilde{L}$ be the diagonal problem constructed.

In order to apply Theorem 6 some observations are necessary. They all are immediate consequences of Theorem 3 and Tarski's Quantifier Elimination for algebraically closed fields of characteristic 0. First, the Hilbert Nullstellensatz decision problem is $\mathrm{NP}_{\mathbb{K}}$-complete in the BSS model over $\mathbb{K}$ for $\mathbb{K} \in \{\bar{\mathbb{Q}}, \mathbb{C}\}$. The strong transfer principle mentioned already in the proof of Theorem 6 implies that an instance over $\bar{\mathbb{Q}}$ is solvable over $\mathbb{C}$ if and only if it is as well solvable already over $\bar{\mathbb{Q}}$. Since the Hilbert Nullstellensatz problem can be defined without additional complex constants Theorem 6 can be applied. This allows to lift the diagonal problem $\tilde{L}$ from $\bar{\mathbb{Q}}$ to $\mathbb{C}$ such that the lifted problem has the same properties there. Thus, Ladner's theorem holds as well over the complex numbers. □

Since Theorem 6 is not known to be true for the real number model the above proof cannot be applied to show Ladner's result for $\mathrm{NP}_{\mathbb{R}}$. Thus a new idea is necessary. If we could group an uncountable set of real algorithms into a countable partition, then may be one could at least construct diagonal problems for such a partition. But how should a reasonable partition look like?

This idea was first considered by Michaux who introduced the notion of *basic machines* in [72].

Definition 8. *A basic machine over $\mathbb{R}$ in the BSS-setting is a BSS-machine M with rational constants and with two blocks of parameters. One block x stands for a concrete input instance and takes values in $\mathbb{R}^\infty$, the other block c represents real constants used by the machine and has values in some $\mathbb{R}^k$ ($k \in \mathbb{N}$ fixed for M).*

Basic machines for variants of the BSS model are defined similarly.

Basic machines split the discrete skeleton of an original BSS machine from its real machine constants. That is done by regarding those constants as a second block of parameters. Fixing c we get back a usual BSS machine $M(\bullet, c)$ that uses the same c as its constants for all input instances x. Below, when we speak about the machine's constants we refer to the potentially real ones only.

Basic machines give rise to define a non-uniform complexity class $\mathrm{P}/\mathrm{const}$ for the different model variants we consider. The non-uniformity is literally weaker than the well-known *P/poly* class from classical complexity theory since the non-uniform advice has fixed dimension for all inputs. In *P/poly* it can grow polynomially with the input size.

Definition 9 ([72]). *A problem L is in class $\mathrm{P}_{\mathbb{R}}/\mathrm{const}$ if and only if there exists a polynomial time basic BSS machine M and for every $n \in \mathbb{N}$ a tuple $c^{(n)} \in [-1,1]^k \subset \mathbb{R}^k$ of real constants for M such that $M(\bullet, c^{(n)})$ decides L for inputs up to size n.*

Similarly for other models.

Note that in the definition $c^{(n)}$ works for all dimensions $\leq n$. The reason for this becomes obvious below. Note as well that assuming all machine constants to be bounded in absolute value is no severe restriction; if a larger constant should be used it can be split into the sum of its integer part and its non-integral part. The integer part then is taken as rational machine constant, thus belonging to the discrete skeleton.

The class $\mathrm{P}/\mathrm{const}$ turned out to be important in unifying Ladner like results in different models and to get as well a (weaker) real version. The class of basic machines clearly is countable as long as the particular choice of machine constants is not fixed. Thus, in principle we can diagonalize over $\mathrm{P}/\mathrm{const}$ decision and reduction machines in the different models.

Theorem 11 ([10]). *Suppose $\mathrm{NP}_{\mathbb{R}} \not\subseteq \mathrm{P}_{\mathbb{R}}/\mathrm{const}$. Then there exist problems in $\mathrm{NP}_{\mathbb{R}} \backslash \mathrm{P}_{\mathbb{R}}/\mathrm{const}$ not being $\mathrm{NP}_{\mathbb{R}}$-complete under $\mathrm{P}_{\mathbb{R}}/\mathrm{const}$ reductions.*

Similarly for the other model variants.

Proof. The proof again uses the usual padding argument along the classical line. The main new aspect, however, is the necessity to establish that for each basic machine M which is supposed to decide the intended diagonal problem $\tilde{L}$ an input-dimension where M's result disagrees with $\tilde{L}$'s definition can be computed effectively. The condition that M disagrees with $\tilde{L}$ for all possible choices of machine constants can be expressed via a quantified first-order formula. Deciding the latter then is possible due to the existence of quantifier elimination algorithms in the respective structures. □

Since the assumption of Theorem 11 deals with $\mathrm{P}_{\mathbb{R}}/\mathrm{const}$ instead of $\mathrm{P}_{\mathbb{R}}$ it gives a non-uniform version of Ladner's result. Note that because of $\mathrm{P}_{\mathbb{R}} \subseteq \mathrm{P}_{\mathbb{R}}/\mathrm{const}$ the theorem's implication also holds for uniform reductions. In order to achieve stronger versions one next has to study the relation between the classes P and P/const. If both are equal, then a uniform version of the theorem follows.

At this point some model theory enters. Very roughly, a structure is called recursively saturated if for each recursive family of first-oder formulas $\{\varphi_n(c)|n \in \mathbb{N}\}$ with free variables c the following holds: if each finite subset of formulas can be commonly satisfied by a suitable choice for c, then the entire family is satisfiable.[5]

Theorem 12 ([72],[10]). *In recursively saturated structures it is* $\mathrm{P} = \mathrm{P}/\mathrm{const}$.

Proof. Let L be a language in P/const and M the respective basic machine. The proof basically is a combination of the definition of saturation with a reasonable description of M's behaviour on instances up to a given dimension. This description, being folklore in BSS theory, gives the recursive family $\{\varphi_n(c)\}_n$ of formulas required, where n stands for the input dimension and the free variables c for the machine constants taken for the basic machine M. Saturation then implies that a single choice for the machine constants can be made which works for all dimensions. This choice turns M into a uniform polynomial time algorithm for L. □

As a consequence, Ladner's results holds uniformly over structures like $\{0, 1\}$ and $\mathbb{C}$ which are well known to be recursively saturated. Thus, Ladner's original result as well as Theorem 10 are reproved.

However, since $\mathbb{R}$ is not recursively saturated – take as family $\varphi_n(c) \equiv c > n$ for $c \in \mathbb{R}$ – the theorem's consequence does not apply to $\mathbb{R}$. So once again the above technique does not give a uniform analogue of Ladner's result over the reals and additional ideas seem necessary. Due to its importance for the above questions Chapuis and Koiran in [26] have undertaken a deep model-theoretic analysis of P/const and related classes. They argue that for the full real model already the equality $\mathrm{P}_{\mathbb{R}} = \mathrm{P}_{\mathbb{R}}/1$ is highly unlikely unless some major complexity theoretic conjecture is violated. Here, $\mathrm{P}_{\mathbb{R}}/1$ is defined by means of basic machines which use a finite number of uniform and a single non-uniform machine constant only. Nevertheless, for the reals with addition and order (additive model) they were able to show once again $\mathrm{P}_{\mathbb{R}}^{add} = \mathrm{P}_{\mathbb{R}}^{add}/\mathrm{const}$ and thus

Theorem 13 ([26]). *Suppose* $NP_{\mathbb{R}}^{add} \neq P_{\mathbb{R}}^{add}$. *Then there are problems in* $NP_{\mathbb{R}}^{add} \setminus P_{\mathbb{R}}^{add}$ *which are not* $NP_{\mathbb{R}}^{add}$*-complete.*

Their proof for showing the inclusion $\mathrm{P}_{\mathbb{R}}^{add}/\mathrm{const} \subseteq \mathrm{P}_{\mathbb{R}}^{\mathrm{add}}$ once more makes use of the moderate growth of intermediate results in an additive computation. This allows to bound the size of and compute efficiently and uniformly for each input dimension n a set of rational machine constants $c^{(n)}$ such that the given $\mathrm{P}_{\mathbb{R}}^{add}/\mathrm{const}$-machine works correctly on $\mathbb{R}^{\leq n}$ if $c^{(n)}$ is taken as vector of constants.

This idea is one of the starting points to extend the result to yet another variant of the full real number model named restricted model in [66]. In this model, the use of machine constants is restricted in that all intermediate results computed by a restricted algorithm should only

[5] Recursiveness here is understood in the Turing sense and just requires that one should be able to enumerate the formulas without using additional machine constants. In the present applications the formulas of the family always represent computations of certain basic machines up to a certain dimension. By 'hiding' constants from the underlying computational structure as variables it follows that such a family satisfies the recursiveness assumption. For more details see [72]

depend linearly on the machine constants. In contrast to additive machines input variables can be used without limitation, i.e., they can be multiplied with each other. The motivation of considering this model is that it is closer to the original full real BSS model than the additive one. As one indication for this fact note that the $\mathrm{NP}_{\mathbb{R}}$-complete feasibility problem QPS over $\mathbb{R}$ is $\mathrm{NP}^{\mathrm{rc}}_{\mathbb{R}}$-complete as well in the restricted model, where the superscript rc is used to denote respective complexity classes in the restricted model. Since Theorem 11 holds as well here the main task once more is to analyze the relation between $\mathrm{P}^{\mathrm{rc}}_{\mathbb{R}}$ and $\mathrm{P}^{\mathrm{rc}}_{\mathbb{R}}/\mathrm{const}$.

Theorem 14 ([66]). *It is $P^{\mathrm{rc}}_{\mathbb{R}} = P^{\mathrm{rc}}_{\mathbb{R}}/\mathrm{const}$. As a consequence, supposing $\mathrm{QPS} \notin NP^{\mathrm{rc}}_{\mathbb{R}}$ there exist non-complete problems in $NP^{\mathrm{rc}}_{\mathbb{R}} \setminus P^{\mathrm{rc}}_{\mathbb{R}}$.*

Proof. Crucial for showing $\mathrm{P}^{\mathrm{rc}}_{\mathbb{R}} = \mathrm{P}^{\mathrm{rc}}_{\mathbb{R}}/\mathrm{const}$ is a certain convex structure underlying the set of suitable machine constants. Given a problem $L \in \mathrm{P}^{\mathrm{rc}}_{\mathbb{R}}/\mathrm{const}$ and a corresponding basic machine M using k constants define $E_n \subset \mathbb{R}^k$ as set of constants that can be used by M in order to decide $L \cap \mathbb{R}^{\leq n}$ correctly. It can be shown that without loss of generality the $\{E_n\}_n$ build a nested sequence of bounded convex sets. If the intersection of all E_n is non-empty any point in it can be taken as uniform set of machine constants and we are done. Thus suppose the intersection to be empty. The main point now is to establish by a limit argument in affine geometry the following: There exist three vectors $c^*, d^*, e^* \in \mathbb{R}^k$ such that for all $n \in \mathbb{N}$ and small enough $\mu_1 > 0, \mu_2 > 0$ (μ_2 depending on μ_1 and both depending on n) machine M correctly decides $L \cap \mathbb{R}^{\leq n}$ when using $c^* + \mu_1 \cdot d^* + \mu_2 \cdot e^*$ as its constants. This is sufficient to change M into a polynomial time restricted machine that decides L and uses c^*, d^*, e^* as its *uniform* machine constants. □

Let us summarize the methods described so far in view of the main open problem in this context, namely Ladner's result for $\mathrm{NP}_{\mathbb{R}}$. The diagonalization technique used above allows some degree of freedom as to how to define $\mathrm{P}_{\mathbb{R}}/\mathrm{const}$. This means that we can put some additional conditions onto the set of constants that we allow for a fixed dimension to work. To make the diagonalization work there are basically two aspects that have to be taken into account. First, the resulting class has to contain $\mathrm{P}_{\mathbb{R}}$. Secondly, the conditions we pose on the constants have to be semi-algebraically definable without additional real constants. Playing around with suitable definitions might be a way to attack Ladner's problem as well in the full real number model. However, for a problem L in $\mathrm{P}_{\mathbb{R}}/\mathrm{const}$ the topological structure of the set of suitable constants is more complicated since now each branch results in a (potentially infinite) intersection of semi-algebraic conditions. Then one has to study how the topology of the sets $\bigcap_{i=1}^{N} E_i$ evolves for increasing N. For example, could one guarantee the existence of say a semi-algebraic limit curve along which one could move from a point c^* into an E_n? In that case, a point on the curve might only be given by a semi-algebraic condition. As consequence, though one would likely not be able to show $\mathrm{P}_{\mathbb{R}}/\mathrm{const} \subseteq \mathrm{P}_{\mathbb{R}}$ may be at least a weaker uniform version of Ladner's result could be settled.

To finish this subsection let us refer the interested reader to [19], where similar questions concerning Ladner like results are studied in Valiant's model of computation.

4.3 Recursion theory

Whereas so far the focus was on decidable problems, in this subsection we consider problems of increased computational difficulty, i.e., undecidable ones in the BSS model. Recursion theory which deals with degrees of undecidability certainly was one of the main topics that at the beginning stimulated research in classical computability theory, see [75]. For alternative models it is in particular interesting with respect to the so called area of *hypercomputation*,

i.e., whether there are (natural) computational devices that are more powerful than Turing machines and thus violate the famous Church-Turing hypothesis. For an introduction to hypercomputation and an extended list of references see [91], and [102] for a particular focus on real hypercomputation.

The real Halting Problem $\mathbb{H}_\mathbb{R}$ was already mentioned earlier. We consider it here in the following version: Given a code $c_M \in \mathbb{R}^\infty$ of a real BSS machine M, does this machine stop its computation on input 0? The problem was the first that has been shown to be undecidable in the real number model in [15]. We now deal with the following question: Are there problems which in a reasonable sense are strictly easier than $\mathbb{H}_\mathbb{R}$ yet undecidable? In the Turing model this was a famous question asked by Post in 1944 and solved about 15 years later independently by Friedberg and Muchnik, see [89]. Nevertheless, until today there is no natural problem with this properties known in the Turing model. We shall see that the question turns out to be much easier (though not trivial) in our framework. A second question to be discussed then is that of finding as well more natural problems that are equivalent to $\mathbb{H}_\mathbb{R}$, i.e., have the same degree of undecidability. Finally, aspects of bounded query computation are treated briefly.

Before explaining some of the results obtained we have to be more specific on what should be understood under terms like *easier* and *equivalent* if we deal with computability issues. As above with $\mathrm{NP}_\mathbb{R}$-completeness this again is formalized using special reductions, this time focussing on computability only instead of complexity.

Definition 10. *A real decision problem $A \subseteq \mathbb{R}^\infty$ is* Turing reducible *to another problem $B \subseteq \mathbb{R}^\infty$ iff there exists an oracle BSS machine M working as follows. For inputs $x \in \mathbb{R}^\infty$ M works like a normal BSS machine except that it additionally has repeatedly access to an oracle for B. In such an oracle state the machine queries the oracle whether a previously computed $y \in \mathbb{R}^\infty$ belongs to B and gets the correct answer in one step. After finitely many steps (normal and oracle) M stops and gives the correct answer whether x belongs to A or not.*

Turing reducibility gives a straightforward way to compare undecidable problems. If A can be decided by an oracle machine using B as oracle but not vice versa, then A is strictly easier than B. If both are Turing reducible to each other they are said to be equivalent. Note that all problems below (i.e., easier than) or equivalent to $\mathbb{H}_\mathbb{R}$ at least are semi-decidable: there is an algorithm which halts exactly for inputs from the problem under consideration. This follows from the existence of a Turing reduction and semi-decidability of $\mathbb{H}_\mathbb{R}$. Problems equivalent to $\mathbb{H}_\mathbb{R}$ are also called *computationally complete* for the real BSS model.

Now our first question, the real version of Post's problem reads: Is there a semi-decidable problem A which is neither decidable nor reducible from $\mathbb{H}_\mathbb{R}$?

Theorem 15 ([68]). *The rational numbers $\mathbb{Q}$ represent an undecidable decision problem which is strictly easier than $\mathbb{H}_\mathbb{R}$. Thus, there is no real BSS oracle machine that decides $\mathbb{H}_\mathbb{R}$ by means of accessing $\mathbb{Q}$ as oracle.*

Proof. Undecidability of $\mathbb{Q}$ was already shown in Theorem 2. The same arguments give undecidability of the real algebraic numbers $\mathbb{A}$. The main step now is to show that $\mathbb{Q}$ is strictly easier than $\mathbb{A}$. Note that this implies the result because $\mathbb{A}$ easily can be decided by a machine accessing $\mathbb{H}_\mathbb{R}$ as oracle. So if the statement was false, i.e., $\mathbb{H}_\mathbb{R}$ would be Turing reducible to $\mathbb{Q}$, transitivity of the reduction notion implies that $\mathbb{A}$ should also be decidable using $\mathbb{Q}$ as oracle.

Assume to the contrary that M is an oracle algorithm deciding $\mathbb{A}$ by means of accessing $\mathbb{Q}$. The arguments used to get a contradiction combine some elementary topology and number theory similar to those used in the proof of Theorem 2. Topology enters for dealing with inequality branches of M, whereas number theory is used for branches caused by queries

to the $\mathbb{Q}$-oracle. Since all intermediate results computed by M are rational functions in the input it turns out to be crucial for analysing the outcome of oracle queries to see how a rational function maps algebraic numbers to rationals. The main observation is the following: Suppose f is a rational function computed by M as oracle query for an input $x \in \mathbb{R}$, i.e., M asks whether $f(x) \in \mathbb{Q}$. If f maps a large enough yet finite set of algebraic numbers to $\mathbb{Q}$, then f will map all algebraic numbers of a large enough degree to a non-rational real. Consequently, such an oracle query is not able to distinguish any algebraic number of large enough degree from a transcendental number. Using this fact together with basic continuity and counting arguments one can conclude that an oracle machine for $\mathbb{A}$ accessing $\mathbb{Q}$ will always fail on certain algbraic numbers. Thus, M cannot work correctly. □

The above proof actually can be extended to get an infinite family of problems that are all strictly easier than $\mathbb{H}_{\mathbb{R}}$ but pairwise incomparable with respect to Turing reductions. Thus there is a rich structure between decidable problems and computationally complete ones in the real BSS model. Similar results have been obtained in [40] for the additive BSS model, whereas [28] studies related questions for higher levels of undecidability above $\mathbb{H}_{\mathbb{R}}$. Degrees of undecidability in the BSS model are as well studied in [101] and [25].

Having solved Post's problem we turn to the question whether there are other problems beside $\mathbb{H}_{\mathbb{R}}$ being computationally universal in the BSS model. In the Turing model several very different problems turned out to be such examples. To mention some of the most prominent ones there is Post's Correspondence Problem [77] which asks for matching a finite set of strings according to some rules, Hilbert's 10th problem [60] which asks for solvability of diophantine equations, and the word problem in finitely presented groups [17, 74]. The problems considered so far in BSS theory naturally have a very strong connection to semi-algebraic geometry because of the underlying set of operations implying that all intermediate results in an algorithm are related to rational functions. So it is demanding to find other significant problems in the theory which basically are not problems in semi-algebraic geometry. We shall now discuss that a suitable variant of the discrete word problem is such an example. Note that the first two of the above mentioned problems do not provide such examples. The Post Correspondence Problem by nature has strong discrete aspects as a kind of matching problem, whereas a real analogue of Hilbert's 10th problem, i.e., deciding real solvability of a real polynomial system is decidable by quantifier elimination.

To understand the word problem let us start with an easy discrete example. Suppose we are given a formal string bab^2ab^2aba in a free group $\langle\{a, b\}\rangle$ generated by the two generators a, b. Here, x^i denotes the i-fold repetition of element x, and concatenation represents the group operation. Now we add some relations between certain elements of the freely generated group. That way a quotient group of the original free group is obtained. For example, assume the equation $ab = 1$ to hold. It is then easy to see that the given element in the resulting quotient group represents b^2. But it cannot be reduced to 1 in this group. However, if as well the relation $a^4 = a^2$ holds, then the given word in the resulting new quotient group does represent the neutral element 1.

This leads to the definition of the word problem.

Definition 11. *a) Let X denote a set. The* free group *generated by X, denoted by $(\langle X\rangle, \circ)$, is the set $(X \cup X^{-1})^*$ of all finite sequences $\bar{w} = x_1^{\alpha_1} \cdots x_n^{\alpha_n}$ with $n \in \mathbb{N}$, $x_i \in X$, $\alpha_i \in \{-1, +1\}$, equipped with concatenation $\circ$ as group operation subject to the rules*

$$x \circ x^{-1} \quad = \quad 1 \quad = \quad x^{-1} \circ x, \qquad x \in X \ , \tag{1}$$

where $x^1 := x$ and 1 denotes the empty word, that is, the unit element.

b) *A group* $(G, \bullet)$ *is called* finitely presented *if* $G \cong \langle X\rangle/\langle R\rangle_{\langle X\rangle} =: \langle X|R\rangle$ *is (isomorphic to) the quotient of a free group* $\langle X\rangle$ *with finite set of generators* X *and the normal subgroup* $\langle R\rangle_{\langle X\rangle}$ *of* $\langle X\rangle$ *generated by the finite set* $R \subseteq \langle X\rangle$.

c) *The* word problem for $\langle X|R\rangle$ *is the task of deciding, given* $\bar{w} \in \langle X\rangle$, *whether* $\bar{w} = 1$ *holds in* $\langle X|R\rangle$.

Intuitively, R describes finitely many rules "$\bar{r} = 1$", $\bar{r} \in R$ additional to those necessarily satisfied in a group. The famous work of Novikov and, independently, Boone establishes the existence of a finitely presented group $\langle X|R\rangle$ whose associated word problem is many-one reducible by a Turing Machine from the discrete Halting Problem H and thus computationally complete in the Turing model. The result is interesting in linking a purely algebraic problem with recursion theory. Since algebra of course is not restricted to discrete groups it is natural to ask whether similar relations can be established between other groups and BSS recursion theory. In the following we shall outline that this indeed is possible.

A natural generalization of Definition 11 to the real number setting is obtained by allowing the sets X and R to become uncountable. Formally, this is expressed by considering sets $X := \{x_r\}_r$ of abstract generators indexed with real vectors r ranging over some subset of $\mathbb{R}^\infty$, and similarly for the relations R. Then interesting word problems arise by putting restrictions on these sets in $\mathbb{R}^\infty$. For sake of notational simplicity we identify X with the sets in $\mathbb{R}^\infty$ the corresponding r's belong to, and similarly for R and the rules.

Definition 12. *Let* $X \subseteq \mathbb{R}^\infty$ *and* $R \subseteq \langle X\rangle \subseteq \mathbb{R}^\infty$. *The elements in* $\langle X\rangle$ *are coded as elements in* $\mathbb{R}^\infty$. *The tuple* (X, R) *is called a* presentation *of the* real group $G = \langle X|R\rangle$. *This presentation is* algebraically generated *if* X *is BSS-decidable and* $X \subseteq \mathbb{R}^N$ *for some* $N \in \mathbb{N}$. G *is termed* algebraically enumerated *if* R *in addition is BSS semi-decidable; and if* R *is BSS-decidable we call* G algebraically presented. *The* word problem *for the presented real group* $G = \langle X|R\rangle$ *is the task of BSS-deciding, given* $\bar{w} \in \langle X\rangle$, *whether* $\bar{w} = 1$ *holds in* G.

Example 2 ([69]). The following three examples should clarify the above notions. The first two give different presentations $\langle X|R\rangle$ of the additive group $(\mathbb{Q}, +)$ of rational numbers with decidable word problem, whereas the third has an undecidable word problem due to its connection to deciding $\mathbb{Q}$ in $\mathbb{R}$.

i) $X = \{x_r : r \in \mathbb{Q}\}, \quad R = \{x_r x_s = x_{r+s} : r, s \in \mathbb{Q}\}$;

ii) $X = \{x_{p,q} : p, q \in \mathbb{Z}, q \neq 0\}$,
$R = \{x_{p,q}x_{a,b} = x_{(pb+aq,qb)} : p, q, a, b \in \mathbb{Z}\} \cup \{x_{p,q} = x_{(np,nq)} : p, q, n \in \mathbb{Z}, n \neq 0\}$;

iii) $X = \{x_r : r \in \mathbb{R}\}, \quad R = \{x_{nr} = x_r, x_{r+k} = x_r : r \in \mathbb{R}, n \in \mathbb{N}, k \in \mathbb{Z}\}$.

Case ii) yields an algebraic presentation, i) is not even algebraically generated, but iii) is algebraically presented. The word problem is trivially decidable for i) because after embedding the task into $(\mathbb{R}, +)$ one can simply compute on the indexes and check whether the result is 0. Also for ii) it is decidable by a similar argument. For iii) the word problem is undecidable because it holds $x_r = x_0 \Leftrightarrow r \in \mathbb{Q}$. Note, however, that case iii) by means of Theorem 15 does not provide a group for which the word problem is computationally universal.

It is not hard to establish that for all algebraically enumerated groups the corresponding word problem is semi-decidable in the BSS model. This just requires a folklore argument based on quantifier elimination. The more interesting result is

Theorem 16 ([69]). *There exists an algebraically presented real group* $\mathcal{H} = \langle X|R\rangle$ *such that the real Halting problem* $\mathbb{H}_\mathbb{R}$ *is reducible to the word problem in* $\mathcal{H}$. *This word problem thus is computationally universal for the real BSS model.*

The proof in a first step embeds the membership problem for any set in $\mathbb{R}^\infty$ to the word problem in a suitable group. Then, it proceeds showing that for $\mathbb{H}_\mathbb{R}$ this embedding can be arranged such that the resulting group is algebraically presented. We skip further details because they rely on a lot of classical techniques in combinatorial group theory such as HNN extensions and Britton's lemma. For more on that see [56] and the full proof in [69].

The theorem is interesting in that it gives a problem computationally significant in the BSS model over $\mathbb{R}$ yet only indirectly related to semi-algebraic features. The list of such problems at the moment is much smaller than in classical recursion theory and it seems an interesting topic for future research to find more such problems. Open questions related immediately to the above theorem are the following. Can the corresponding universality result be established for algebraically presented groups for which as well the set R of rules comes from a finite dimensional space $\mathbb{R}^k$? In the construction of the proof it turns out to be crucial that R lives in $\mathbb{R}^\infty$, i.e., there have to be included rules for vectors of arbitrarily large dimension. Recall that in the original result by Boone and Novikov both X and R are finite, and it seems that a suitable analogue of finiteness in the discrete setting is finite dimensionality of these sets in the real number framework. Another interesting question is that of finding particularly structured groups whose respective word problems are universal for complexity classes. One such task thus would be to find particular algebraically generated groups for which the word problem is $\mathrm{NP}_\mathbb{R}$-complete.

To close this section we briefly mention yet another area of recursion theory which has intensively been studied in the Turing model and only seen some initial considerations in our framework, namely bounded query computations. Here, the interest is shifted from the direct consideration of decision problems, i.e., computing the characteristic function χ_A of an $A \subseteq \mathbb{R}^\infty$ to the following type of questions: Given an $n \in \mathbb{N}$ how many oracle queries to a set $B \subseteq \mathbb{R}^\infty$ are needed in order to compute the n-fold characteristic function of A on n many inputs $x_i \in \mathbb{R}^\infty, 1 \leq i \leq n$. Different choices of A and B, where also $A = B$ is possible, give quite different results. An easy example shows that by using binary search for each semi-decidable set A the n-fold characteristic function can be computed by $\lceil \log_2 n + 1 \rceil$ many calls to an oracle for $\mathbb{H}_\mathbb{R}$. The following result is much less obvious

Theorem 17 ([67]). *Let $n \in \mathbb{N}$ and consider the n-fold characteristic function $\chi_n^\mathbb{Q}$ on $\mathbb{Q}$. Let $B \subseteq \mathbb{R}$ be an arbitrary subset of the reals. Then no BSS oracle machine having access to B as oracle can compute $\chi_n^\mathbb{Q}$ with less many than n queries.*

Proof. Suppose such an oracle machine exists it must in a certain way reduce n questions about $\mathbb{Q}$ to at most $n-1$ many questions about the arbitrary real set B. Now the main idea is to arrange the situation for an application of the implicit function theorem for functions from $\mathbb{R}^n \mapsto \mathbb{R}^{n-1}$. Along the one-dimensional solution curve which the theorem guarantees to exist the oracle machine then can be shown to necessarily err. □

The application of classical tools from analysis like the implicit function theorem shows a significant difference to proofs in the Turing framework. So we expect a lot of interesting problems to exist in this area which need other methods not available in discrete recursion theory.

This section aimed to present some challenging questions and techniques in structural complexity theory for real or complex number computations. The problems treated just reflect a small fraction of topics studied in the last two decades in this area. We close by pointing to some more literature.

A prominent class of problems that have been studied intensively in classical complexity theory are counting problems. This has lead to the definition of a counting analogue of NP

denoted by #P and the search for complete problems in that class. Roughly speaking, #P captures functions that count the number of accepting computations of an NP-algorithm. Assuming P $\neq$ NP this counting class contains much harder problems than those in NP. This is justified by Toda's famous result [93] which says that using an oracle from #P in deterministic polynomial time computations captures all problems in the so called polynomial hierarchy, a set conjectured to be much larger than NP. A prominent result by Valiant [96] shows that the computation of the permanent for a matrix with $\{0,1\}$-entries reflects the difficulty of this class, i.e., is a #P-complete problem.

In the real number framework counting problems have been extensively studied in several papers by Bürgisser, Cucker and co-authors. Many of the relevant problems have a strong algebraic flavour, for example tasks like computing Betti numbers of algebraic varieties. As a starting point for readers being interested in such questions we just refer to [22, 23]. Analogues of Toda's theorem both in real and complex number complexity theory were recently obtained in [8, 6].

Another branch of complexity theory that was studied in the real number framework is decsriptive complexity. Here the goal is to describe complexity classes independently of the underlying computational model. Instead, the logical shape in which a problem can be expressed reflects the algorithmic complexity sufficient to solve it. The first result into this direction can be found in [41], where both for $\mathrm{P}_{\mathbb{R}}$ and $\mathrm{NP}_{\mathbb{R}}$ such logical characterizations are given. [31] contains further such results, [63] deals with counting problems from a logical point of view.

Transfer results, one of the main topics in this section, have as well been analyzed with respect to other algebraic approaches to complexity, and here foremost Valiant's complexity classes VP and VNP, see [97]. This approach focusses on families of polynomials over a field that have a polynomially bounded degree in the number of their variables and can be computed by a non-uniform family of small circuits. This constitutes class VP, whereas VNP essentially is the family of polynomials whose coefficients are functions in VP, though there might be exponentially many monomials. A notion of reduction then is introduced by using a projection operator and once again the permanent polynomials turn out to be a complete family for VNP. This gives another algebraic variant of a P versus NP problem, this time for VP versus VNP and it is nearby to ask whether this question as well is related to some of the other problems of that style mentioned before. Readers interested in learning more about progress being made into this direction are refered to [20] as starting point.

5 Probabilistically checkable proofs over $\mathbb{R}$

For the rest of this paper we shall now turn to the area of probabilistically checkable proofs, for short PCPs. The PCP theorem first shown by Arora et al. [3, 2] certainly is one of the landmark results in Theoretical Computer Science in the last two decades. It gives a new characterization of class NP in the Turing model and had tremendous impact on obtaining non-approximability results in combinatorial optimization. More recently, an alternative proof of the theorem was given by Dinur [33].

The first subsection below briefly surveys the classical PCP theorem and its currently existing proofs. The main part of this section is then devoted to studying PCPs in the BSS model. We shall give a complete proof of the existence of so-called long transparent proofs for both $\mathrm{NP}_{\mathbb{R}}$ and $\mathrm{NP}_{\mathbb{C}}$, see [64]. Then, we outline how the full PCP theorem can be shown to hold as well in these two models.

5.1 The classical PCP theorem: A short outline

The PCP theorem gives a surprising alternative characterization of the class NP. It is based on a new point of view concerning the verification procedure necessary to establish membership of a problem L in class NP. Recall that according to the definition of NP verifying that an input x belongs to L can be done by guessing a suitable proof y and then verifying by a deterministic polynomial time algorithm in the size of x that the pair (x, y) satisfies the property defining L. For example, verifying satisfiability of a given Boolean formula $x := \phi$ in conjunctive normal form can be done by guessing a satisfying assignment y and then evaluating $\phi(y)$. Clearly such a verification algorithm in general must read all components of y in order to work correctly. Note that for $x \in L$ at least one such y has to exist, whereas for $x \notin L$ all potential proofs y have to be rejected.

In the PCP theorem the requirements for the verification procedure are changed. Here is a brief outline of these new aspects, precise definitions are given in the next subsection. Suppose membership of x in L should be verified using proof y. The verifier is randomized in that it first generates a random string. This string and input x are then used to determine a number of proof components in y it wants to read. This number is intended to be dramatically smaller than the size of y, actually only constant in the PCP theorem. Finally, using the input, the random string and those particular proof components the verifier makes its decision whether to accept or rejct the input. This way there will be a possibility that the verifier comes to a wrong conclusion, but as long as this probability is not too big this is allowed. $\mathrm{PCP}(r(n), q(n))$ denotes the class of those languages that have a verifier using $r(n)$ random bits and inspecting $q(n)$ components of the given proof y for inputs x of size n. The PCP theorem states that $\mathrm{PCP}(O(\log(n)), O(1)) = \mathrm{NP}$. It thus shows that there exists a format of verification proofs for languages in NP which is stable in the following sense: If $x \in L$, then there is a proof y that is always accepted (just as in the original definition of NP); and if $x \notin L$ for each proof y the verifier detects a fault in that proof with high probability by reading a constant number of components only. The number of components of y to be seen in particular is independent of the length of the input!

The PCP theorem implies lots of inapproximability results. One of the first such results states that given a propositional Boolean formula ϕ in conjunctive normal form having m clauses, there is no polynomial time algorithm in size ϕ which for an arbitrary given $\epsilon > 0$ computes a value k such the maximum number $\max(\phi)$ of commonly satisfiable clauses of ϕ lies within a constant factor of at most $1 + \epsilon$, i.e., satisfies $\max(\phi)/k \leq 1 + \epsilon$.

Recall that we saw a similar negative result in Theorem 1, part b). However, that result was much easier to obtain than the one above which could only be shown as an application after the PCP theorem was proven. The close relation between PCPs and approximability is the starting point of Dinur's proof and will also be important for studying such questions in the real number setting.

Let us shortly outline the two existing proofs of the PCP theorem. The original one by Arora et al. is very algebraic in nature. Here, different verifiers are constructed which are then combined to a single verifier with the desired properties. One of the verifiers used for the composition needs a large amount of randomness but inspects constantly many proof components only. It is based on coding a satisfying assignment of a Boolean formula via certain linear functions. The second verifier uses logarithmic randomness but needs to read more components. Here the used coding of an assignment is done via multivariate polynomials of not too high degree. Both verifiers are then cleverly combined by a newly invented technique called verifier composition to yield a third verifier with the required resources.

The second proof of the PCP theorem given by Dinur [33] in 2005 is more combinatorial in structure. The basic idea of this proof is to exploit the strong relation between PCPs and

(non-)approximability results. More precisely, Dinur's proof uses an NP-complete problem called CSP which stands for constraint satisfiability problem; such problems are extensions of the Boolean satisfiability problem. An instance of the CSP problem consists of a number of constraints in a finite number of variables taking values in a finite alphabet. The question is again whether there exists an assignment of the variables that satisfies all constraints. Instead of directly constructing a verifier for this problem one considers the following approximation problem: Is there an efficient algorithm which for any given $\epsilon > 0$ approximates the maximal number of constraints that are commonly satisfiable within a factor at most $1+\epsilon$. Clearly, since the decision problem is NP-complete computing the maximal number exactly is an NP-hard problem as well. But it is not clear whether the above optimization task can be accomplished more easily. This question is intimately related to the PCP theorem as follows. Suppose there exists a polynomial time reduction from CSP instances to CSP instances such that a satisfiable CSP instance is mapped to a satisfiable CSP instance and a non-satisfiable CSP instance is mapped to a CSP instance for which no assignment satisfies more than a certain fixed fraction of the constraints. Then the PCP theorem would follow from the existence of that reduction. A verifier for CSP first performs the reduction on an input instance. It then expects the proof to give an assignment to the variables of the instance resulting from the reduction. Now if this resulting instance is not satisfiable, then the assignment the proof encodes violates at least a fixed fraction of the constraints. So the verifier can check the proof by selecting a constant number of constraints, reading the constantly many values that the proof assigns to the variables occuring in these constraints, and checking if one of these constraints is violated by the assignment. Due to the existence of the fixed fraction repeating this test constantly many times will guarantee that the verifier respects the necessary error bounds.

Dinur's proof constructs such a polynomial time reduction between CSP instances. There are two major steps involved in the construction. Given an unsatisfiable set of constraints at the beginning we only know that at least one among the constraints is not satisfiable together with the remaining ones. Thus at the beginning we have no constant fraction of unsatisfied constraints. The first step is an amplification step that increases this fraction by a constant factor. Repeating it logarithmically many times would yield a constant fraction. However, the amplification also increases the size of the finite alphabet used. To control this a second step called alphabet reduction is necessary. This second step as well heavily relies on the existence of long transparent proofs, i.e., verifiers that accept CSP using a large (super-logarithmic) amount of randomness and inspecting constantly many proof components. Note that for using the corresponding verifier, in both proofs its structure is much more important than the values of the parameters r and q. This is due to the fact that the long-transparent-proof verifier is applied to instances of constant size only. This as well will be important below in the real number setting.

This short outline of the classical proof structures should be sufficient here. Similar ideas will be described much more explicitly in the next subsections in relation to PCPs for the real and complex BSS model. Complete descriptions of the two classical proofs can be found in the already cited original papers as well as in [45, 1, 78].

5.2 Verifiers in BSS setting; long transparent proofs

For $\mathbb{K} \in \{\mathbb{R}, \mathbb{C}\}$ consider once again the Hilbert Nullstellensatz decision problem $\mathrm{QPS}_{\mathbb{K}}$ studied in previous sections. To show its membership in $\mathrm{NP}_{\mathbb{K}}$ one can guess a potential solution $y \in \mathbb{K}^n$, plug it into the polynomials of the system and verify whether all equations are satisfied by y. Clearly, this verification algorithm in general has to inspect all components of y. So the above question for the discrete satisfiability problem as well makes perfect sense

here: Can we give another verification proof for solvability of such a system that is much more stable in the sense of detecting errors with high probability by inspecting only a small amount of proof components?

This kind of question is made more precise by defining the corresponding verification procedures as well as the languages in $\mathbb{K}^*$ which are accepted by such verifiers.

Definition 13. *Let $r, q : \mathbb{N} \mapsto \mathbb{N}$ be two functions. An $(r(n), q(n))$-restricted verifier V in the BSS model over $\mathbb{K}, \mathbb{K} \in \{\mathbb{R}, \mathbb{C}\}$ is a randomized BSS algorithm over $\mathbb{K}$ working as follows. For an input $x \in \mathbb{K}^*$ of algebraic size n and another vector $y \in \mathbb{K}^*$ representing a potential membership proof of x in a certain set $L \subseteq \mathbb{K}^*$, the verifier in a first phase generates non-adaptively a sequence of $O(r(n))$ many random bits (under the uniform distribution on $\{0,1\}^{O(r(n))}$). Given x and these $O(r(n))$ many random bits V in the next phase computes in a deterministic manner the indices of $O(q(n))$ many components of y. Finally, in the decision phase V uses the input x together with the random string and the values of the chosen components of y in order to perform a deterministic polynomial time algorithm in the BSS model. At the end of this algorithm V either accepts or rejects x. For an input x, a guess y and a sequence of random bits ρ we denote by $V(x, y, \rho) \in \{0, 1\}$ the result of V in case the random sequence generated for (x, y) was ρ.*

The time used by the verifier in the decision phase 3 is also called its decision-time. It should be polynomially bounded in the size of x.

Remark 1. Concerning the running time of a verifier the following has to be pointed out. In general, generating a random bit is assumed to take one time unit, and the same applies when the verifier asks for the value of a proof component. Below in relation to long transparent proofs we need more than polynomially many random bits. In such a situation the time for generating a random string would be superpolynomial. We then assume that the entire random string can be generated at unit cost. Note however that this is of no concern since the existence of long transparent proofs will be used in the proof of the full PCP theorem only for instances of constant size and thus the number of random bits is constant as well. We comment on this point once more after Theorem 18 below.

Using the above notion of a verifier it is immediate to define the languages accepted by verifiers.

Definition 14. *($\mathrm{PCP}_{\mathbb{K}}$-classes) Let $\mathbb{K} \in \{\mathbb{R}, \mathbb{C}\}$ and let $r, q : \mathbb{N} \mapsto \mathbb{N}$; a decision problem $L \subseteq \mathbb{K}^*$ is in class $\mathrm{PCP}_{\mathbb{K}}(r(n), q(n))$ iff there exists an $(r(n), q(n))$-restricted verifier V such that conditions a) and b) below hold:*

a) For all $x \in L$ there exists a $y \in \mathbb{K}^*$ such that for all randomly generated strings $\rho \in \{0,1\}^{O(r(size_{\mathbb{K}}(x)))}$ the verifier accepts. In other words:

$$\Pr_{\rho}\{V(x, y, \rho) = \ 'accept'\} = 1 \quad .$$

b) If $x \notin L$, then for all $y \in \mathbb{K}^*$

$$\Pr_{\rho}\{V(x, y, \rho) = \ 'reject'\} \geq \frac{3}{4} \quad .$$

In both cases the probability is chosen uniformly over all strings $\rho \in \{0,1\}^{O(r(size_{\mathbb{K}}(x)))}$.

In this section we will discuss the existence of transparent long proofs for problems in $\mathrm{NP}_{\mathbb{K}}$ in detail. Our exposition and the given proof below basically follow [64], where this existence

was shown for $\mathrm{NP}_{\mathbb{R}}$. Note however that though the almost same analysis is used it seems that a longer verification proof is needed than the one given there; so we adapt the required arguments accordingly. This change nevertheless is of no concern with respect to the role long transparent proofs play in the full PCP Theorem 21 below. There, they are applied to constant size inputs only, so the length of a long transparent proof a verifier wants to inspect is constant anyway. The more important aspect is the structure of the verification proof, see below.

We shall construct such a verifier for the $\mathrm{NP}_{\mathbb{K}}$-complete problem $\mathrm{QPS}_{\mathbb{K}}$. The construction is described for $\mathbb{K} := \mathbb{R}$, always pointing out where some care has to be taken when $\mathbb{K} = \mathbb{C}$ is considered instead.

The system's coefficients can be arbitrary real numbers. The verifier will receive the following three objects as input: A family of degree two polynomials, a (possibly incorrect) proof of the existence of an assignment under which the polynomials evaluate to zero, and a sequence of random bits. The verifier outputs either "accept" if it believes the proof to be correct or "reject" otherwise. The polynomials and the proof will be in the form of a sequence of real numbers whereas the random string is a sequence over $\{0,1\}$. Randomness is used to decide which locations in the proof to query. Since the corresponding addresses can be coded discretely only discrete randomness is needed.

Throughout this subsection let n denote the number of variables of the input polynomials. Let $\mathcal{P} := \{p_1, \ldots, p_m\}$ denote the system. All polynomials p_i are of degree at most two and depend on at most three variables. For $r \in \{0,1\}^m$ define $P(x,r) := \sum_{i=1}^{m} p_i(x) \cdot r_i$. The following is easy to see: Let $x \in \mathbb{R}^n$ be fixed. If x is a common zero of all $p_i(x)$, then $P(x,r) = 0$ for all r. And if x is no common zero the probability for uniformly taken r that $P(x,r) = 0$ is at most $\frac{1}{2}$. We work with $P(x,r)$ in order to capture both the real and the complex case in common.

Of course, if we want to verify whether an $a \in \mathbb{R}^n$ solves the system we can neither plug it into $P(a,r)$ and evaluate because this requires again reading all components of a. We therefore rewrite $P(a,r)$ as follows:

$$P(a,r) = E(r) + A \circ L_A(r) + B \circ L_B(r), \tag{1}$$

where functions E, A, B, L_A, and L_B have the following properties. A and B are linear functions with n and n^2 many inputs, respectively. The coefficient vectors that represent these mappings depend on the chosen a only. More precisely,

$$A : \mathbb{R}^n \mapsto \mathbb{R} \text{ such that } A(x_1, \ldots, x_n) = \sum_{i=1}^{n} a_i \cdot x_i \ \forall\ x \in \mathbb{R}^n;$$
$$B : \mathbb{R}^{n^2} \mapsto \mathbb{R} \text{ such that } B(y_{11}, \ldots, y_{nn}) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_i \cdot a_j \cdot y_{ij} \ \forall\ y \in \mathbb{R}^{n^2}$$

The functions E, L_A and L_B are linear as well. They take as arguments inputs from $\mathbb{Z}_2^m := \{0,1\}^m$ and give results in the spaces $\mathbb{R}, \mathbb{R}^n$ and $\mathbb{R}^{n^2}$, respectively. It is important to note that these mappings do only depend on the coefficients of the polynomials $p_1, \ldots, p_m$ but not on a. Therefore, given the system and a random vector $r \in \mathbb{Z}_2^m$ these functions can be evaluated deterministically without inspecting a component of the verification proof. As an immediate consequence of equation (1), for evaluating $P(a,r)$ it is sufficient to know two function values of certain linear functions, namely the value of A in $L_A(r) \in \mathbb{Z}_2^n$ and that of B in $L_B(r)$. The verifier expects from the verification proof to contain these two real values.

More precisely, the proof is expected to contain so-called linear function encodings of the coefficient vectors defining A and B. This means that instead of expecting the proof to just

write down those vectors we do the following. We define a finite subset $\mathfrak{D}$ of $\mathbb{R}^n$ and require the proof to contain all values of $A(x) := a^t \cdot x$ for all $x \in \mathfrak{D}$; similarly for B and a subset of $\mathbb{R}^{n^2}$. In order to work out this idea several problems have to be handled. First, though A in principle is a linear function over all $\mathbb{R}^n$ the verification proof must be finite. It can only contain finitely many components representing values of A. Among these components we of course must find those values in arguments that arise as images $L_A(r)$ for $r \in \mathbb{Z}_2^m$. Secondly, the verifier cannot trust the proof to represent a linear function which maps $\mathfrak{D}$ to $\mathbb{R}$. All it can do is to interpret the proof as giving just a function A from $\mathfrak{D}$ to $\mathbb{R}$ and try to find out if it is linear. Thirdly, even if the functions A and B are indeed linear on their corresponding domains and encode coefficient vectors a and b the verifier has to find out whether b is consistent with a, i.e., whether the coefficient vector $\{b_{ij}\}$ defining B satisfies $b_{ij} = a_i \cdot a_j$.

To verify all requirements within the necessary resources and error bounds the verifier tries to realize the following tasks: It expects the proof to provide two function value tables representing A and B on suitable domains (to be specified). Then first it checks whether both tables with high probability represent a linear function on the respective domains and if 'yes' how to compute the correct values of those functions in a given argument with high probability. In a second part the verifier checks consistency of the two involved coefficient vectors with high probability. Finally, it evaluates (1) to check whether the result equals 0.

A correct proof will provide the tables of two linear functions on the appropriate domains of form $A(x) = a^t \cdot x$ and $B(x) = b^t \cdot x$ with vectors $a \in \mathbb{R}^n, b \in \mathbb{R}^{n^2}$ such that $b_{ij} = a_i \cdot a_j, 1 \leq i, j \leq n$. In this ideal case, equation (1) can be evaluated by reading only two components of the entire proof, namely one value of A and one of B. If the proof is correct the verifier will always accept.

Suppose then that the given $\mathrm{QPS}_{\mathbb{R}}$ instance has no solution. The verifier has to detect this for any proof with high probability. There are different cases to consider where in the proof errors can occur. The first such case is the one in which one of the two functions which the proof provides is in a certain sense far from being linear. The verifier will be able to detect this with high probability by making only a few queries into the function value table and then reject. A more difficult situation occurs when the given function is not linear but close to linear. In this case the verifier's information about the proof is not sufficient to conclude that it is not completely correct. To get around this problem a proceeding that aims to self-correct the values which the proof gives is invoked. For A and B as given in the tables we shall define self-corrections f_A, f_B. Assuming that the function value tables are almost linear will guarantee that these self-corrected functions are linear on the part of the domain which is important for us. Furthermore, the values of these self-corrected functions can be computed correctly with high probability at *any* argument in this part of the domain by making use of constantly many other values in the table only. In case A is linear f_A equals A on the domain on which it is defined.

We will now carry out the following plan:

1. Define the domains on which we want the verification proof to define functions A and B;
2. check linearity of these functions such that if they are far from linear it will be discovered with high probability;
3. assuming no contradiction to linearity has been detected so far define the self-corrections f_A and f_B; use these to detect with high probability an error if consistency between the coefficient vectors of the two linear functions is violated;
4. for random $r \in \mathbb{Z}_2^m$ obtain the correct values of $f_A(L_A(r))$ and $f_B(L_B(r))$ with high probability and use these values together with $E(r)$ to evaluate $P(a, r)$. Check whether the result is zero.

Appropriate domains for linearity. We will now describe the domain $\mathfrak{D}$ on which the values of A should be provided by the proof. The domain on which we want the proof to define the function B will be constructed analogously.

The function $L_A : \mathbb{Z}_2^m \to \mathbb{R}^n$ which generates the arguments in which A potentially has to be evaluated has a simple structure depending on the input coefficients of the polynomials p_i. Written as a matrix its entries are either 0 or such coefficients, i.e., real numbers that constitute the $\mathrm{QPS}_{\mathbb{R}}$ instance. Let $\Lambda := \{\lambda_1, \ldots, \lambda_K\}$ denote this set of entries in L_A, considered as a multiset. Since each p_i depends on at most 3 variables it is $K = O(m)$. In order to simplify some of the calculations below we assume without loss of generality that $m = O(n)$; if not we can add a polynomial number of dummy variables to the inital instance. Thus $K = O(n)$. Without loss of generality we also assume $\lambda_1 = 1$. The components of any vector occuring as argument of A now are 0-1 linear combinations of elements in Λ. We therefore define

$$\mathcal{X}_0 := \{\sum_{i=1}^{K} s_i \cdot \lambda_i \mid s_i \in \{0,1\}\}^n.$$

This set contains $\mathbb{Z}_2^n$ and thus a basis of $\mathbb{R}^n$. If we could guarantee additivity on pairs taken from $\mathcal{X}_0$ as well as scalar multiplicativity with respect to all scalars taken from Λ we could be sure to work with a correct linear function for our purposes.

Here a first problem occurs: For getting almost surely a linear function A on $\mathcal{X}_0$ from a table for A we need to know and test values of A on a much larger domain $\mathcal{X}_1$. So a larger test domain is needed in order to get a much smaller safe domain, compare [80]. The idea behind constructing $\mathcal{X}_1$ is as follows: We want $\mathcal{X}_1$ to be almost closed under addition of elements from $\mathcal{X}_0$. With this we mean that for every fixed $x \in \mathcal{X}_0$, picking a random $y \in \mathcal{X}_1$ and adding x to it results with high probability again in an element in $\mathcal{X}_1$. Similarly, $\mathcal{X}_1$ should be almost closed under scalar multiplication with a factor $\lambda \in \Lambda$. These properties of $\mathcal{X}_1$ will be important in proving linearity of f_A on $\mathcal{X}_0$ if A satisfies the tests on $\mathcal{X}_1$ to be designed. We remark that these requirements are more difficult to be satisfied than in the corresponding construction of a long transparent proof in the Turing model. There, all domains are subsets of some $\mathbb{Z}_2^N$ and thus arguments are performed on a highly structured set with a lot of invariance properties of the uniform distribution. Secondly, there are no scalars other than 0 and 1, so additivity implies linearity. In the BSS setting some difficulties arise because some of the elements in Λ can be algebraically independent.

The above motivates the following definition. Let $M := \{\prod_{i=1}^{K} \lambda_i^{t_i} | t_i \in \{0, \ldots, n^2\}\}$, $M^+ := \{\prod_{i=1}^{K} \lambda_i^{t_i} | t_i \in \{0, \ldots, n^2+1\}\}$ and

$$\mathcal{X}_1 := \left\{ \frac{1}{\alpha} \sum_{\beta \in M^+} s_\beta \cdot \beta \mid s_\beta \in \{0, \ldots, n^3\}, \alpha \in M \right\}^n .$$

We now prove that $\mathcal{X}_1$ does indeed have the desired properties. To keep things simple we will think of elements in $\mathcal{X}_0$, $\mathcal{X}_1$ (and later also in $\mathfrak{D}$) as formal sums of products defining M^+. This means for example that we distinguish elements in $\mathcal{X}_1$ which have the same numerical value because some λ_i's in Λ could be the same, but arise from formally different sums. Such elements are counted twice below when talking about the uniform distribution on the respective domains. Doing it this way simplifies some counting arguments because we don't have to take algebraic dependencies between the λ_i's into account.

Lemma 1. *Let $\epsilon > 0$ and $n \in \mathbb{N}$ such that $n \geq c/\epsilon$ for a suitable constant $c > 0$, then the following holds:*

a) For every fixed $x \in \mathcal{X}_0$ it is $\Pr_{y\in\mathcal{X}_1}\{y + x \in \mathcal{X}_1\} \geq 1 - \epsilon$.

Here, the probability distribution is the uniform one on $\mathcal{X}_1$, taking into account the above mentioned way how to count elements in $\mathcal{X}_1$.

b) Similarly, for fixed $\lambda_s \in \Lambda$ it is $\Pr_{y\in\mathcal{X}_1}\{\lambda_s \cdot y \in \mathcal{X}_1\} \geq 1 - \epsilon$.

c) For fixed $\lambda \in \Lambda$ it is $\Pr_{\alpha\in M}\{\alpha/\lambda \in M\} \geq 1 - \epsilon$.

Proof. For part a) let us focus on a single coordinate j. Then x_j is a 0-1 sum of the λ_i's. We have y_j of the form $\frac{1}{\alpha}\sum_{\beta\in M^+} s_\beta \cdot \beta$ with $\alpha \in M$ and $s_\beta \leq n^3$ for $\beta \in M^+$. If the sum for x_j contains a term $1 \cdot \lambda_i$ and the corresponding coefficient of monomial λ_i in y_j is $< n^3$, then $y_j + x_j$ also has the required form. Thus for each of the at most $K = O(n)$ many addends in x_j there are n^3 out of $n^3 + 1$ choices for the coefficient of the corresponding monomial in y_j that imply $x_j + y_j$ to be of the required form with respect to this monomial. Since this argument applies for all n components one obtains

$$\Pr_{y\in\mathcal{X}_1}\{y+x \in \mathcal{X}_1\} = \left(\frac{n^3}{n^3+1}\right)^{K\cdot n} = \left(1 - \frac{1}{n^3+1}\right)^{O(n^2)} \underbrace{\geq}_{Bernoulli} 1 - \frac{O(n^2)}{n^3+1} \geq 1 - \frac{c}{n} \geq 1 - \epsilon.$$

For part b) consider an arbitrary fixed $\lambda_s \in \Lambda$ together with a random $y \in \mathcal{X}_1$. Consider again a fixed component j of y. The α in the representation of this y_j has the form $\prod_{i=1}^{K} \lambda_i^{t_i}$ with $t_i \in \{0, \ldots, n^2\}$. If the particular exponent t_s of λ_s in this α satisfies $t_s > 0$, then $\lambda_s \cdot y$ will belong to $\mathcal{X}_1$ (and for some cases with $t_s = 0$ as well). The probability that $t_s > 0$ and thus $\lambda_s \cdot y \in \mathcal{X}_1$ is therefore bounded from below by

$$\Pr_{y\in\mathcal{X}_1}\{\lambda_s \cdot y \in \mathcal{X}_1\} = \left(\frac{n^2}{n^2+1}\right)^{n} \geq 1 - \frac{c}{n} \geq 1 - \epsilon.$$

Part c) is trivial. □

In order to verify (almost) linearity of A on $\mathcal{X}_0$ with respect to scalars from Λ a test is designed that works on arguments of the forms $x + y$, where $x, y \in \mathcal{X}_1$ and $\alpha \cdot x$ with $\alpha \in M, x \in \mathcal{X}_1$. The function value table expected from a proof therefore must contain values in all arguments from the set $\mathfrak{D} := \{x + y | x, y \in \mathcal{X}_1\} \cup \{\alpha \cdot x | \alpha \in M, x \in \mathcal{X}_1\}$. In the next subsection a test is designed on $\mathfrak{D}$ that verifies with high probability linearity of A on $\mathcal{X}_0$.

The linearity test and self-correction. As in the previous section we will only describe how things work for the function $A : \mathfrak{D} \to \mathbb{R}$. In the ideal case this function A is linear and thus uniquely encodes the coefficient vector $a \in \mathbb{R}^n$ of the related linear function.

In order to make the formulas look a bit simpler we define the abbreviation $A_\alpha(x) := A(\alpha \cdot x)/\alpha$. We repeat the following test a constant number of times:

Linearity test:

- Uniformly and independently choose random x, y from $\mathcal{X}_1$ and random α, β from M;
- check if $A(x + y) = A_\alpha(x) + A_\beta(y)$.

If all checks were correct the test accepts. Otherwise the test rejects.

Each round will inspect at most three different proof components, namely $A(x+y)$, $A(\alpha\cdot x)$ and $A(\beta \cdot y)$. Thus in finitely many rounds $O(1)$ components will be inspected.

Clearly the linearity test accepts any linear function A with probability 1. For any $\delta > 0$ and $\epsilon > 0$ we can choose the number of repetitions of the linearity test so large that if

$$\Pr_{x,y \in \mathcal{X}_1, \alpha, \beta \in M} \{A(x+y) = A_\alpha(x) + A_\beta(y)\} > 1 - \delta \tag{2}$$

does not hold, then the test rejects with probability $1 - \epsilon$. The following cases have to be analyzed. If the linearity test rejects the verifier rejects the proof and nothing more is required. So suppose the linearity test does not give an error. If (2) is not satisfied, i.e., in particular the function value table does not come from a linear function, the verifier would err. Luckily it is easy to show that the probability for this to happen is small. And according to the definition of the $\mathrm{PCP}_{\mathbb{R}}$ classes we are allowed to accept incorrect proofs with small probability. It remains to deal with the only more difficult situation: The linearity test accepts and (2) holds. This of course does not mean that all values in the table necessarily are the correct ones. If the verifier asks for a particular such value we must therefore guarantee that at least with high probability we can extract the correct one from the table. One can get around this problem by defining a so-called self-correction f_A on $\mathcal{X}_1$ which can be shown to be linear on $\mathcal{X}_0$. This self-correction looks as follows: For $x \in \mathcal{X}_1$ define

$$f_A(x) = \mathrm{Majority}_{y \in \mathcal{X}_1, \alpha \in M} \{A_\alpha(x+y) - A_\alpha(x)\}.$$

Hence $f_A(x)$ is the value that occurs most often in the multiset $\{A_\alpha(x+y) - A_\alpha(x) | y \in \mathcal{X}_1, \alpha \in M\}$. It could be the case that $A_\alpha(x+y)$ is not defined. If this happens we just do not count this 'value'.

Lemma 2. *Under the above assumptions the function f_A is linear on $\mathcal{X}_0$ with scalars from Λ, i.e., for all $v, w \in \mathcal{X}_0$ we have $f_A(v+w) = f_A(v) + f_A(w)$ and for all $x \in \mathcal{X}_0, \lambda \in \Lambda$ we have $f_A(\lambda \cdot x) = \lambda \cdot f_A(x)$.*

Proof. For arbitrary fixed $v \in \mathcal{X}_0$ and random $x \in \mathcal{X}_1$ by Lemma 1 it is $x + v \in \mathcal{X}_1$ with probability $\geq 1 - \epsilon$ assuming n is large enough. Since $x \mapsto x + v$ is injective and due to the use of the uniform distribution in (2) replacing x by $x + v$ in (2) gives

$$\Pr_{x,y \in \mathcal{X}_1, \alpha, \beta \in M} \{A(x+v+y) = A_\alpha(x+v) + A_\beta(y)\} > 1 - \delta - \epsilon.$$

Doing the same with y instead of x yields

$$\Pr_{x,y \in \mathcal{X}_1, \alpha, \beta \in M} \{A(x+v+y) = A_\alpha(x) + A_\beta(v+y)\} > 1 - \delta - \epsilon$$

and combining these two inequalities results in

$$\Pr_{x,y \in \mathcal{X}_1, \alpha, \beta \in M} \{A_\alpha(x+v) - A_\alpha(x) = A_\beta(v+y) - A_\beta(y)\} > 1 - 2\delta - 2\epsilon.$$

From this it follows that

$$\Pr_{x \in \mathcal{X}_1, \alpha \in M} \{f_A(v) = A_\alpha(x+v) - A_\alpha(x)\} \geq 1 - 2\delta - 2\epsilon. \tag{3}$$

Similarly, for a fixed $w \in \mathcal{X}_0$ one obtains

$$\Pr_{x \in \mathcal{X}_1, \alpha \in M} \{f_A(w) = A_\alpha(x+w) - A_\alpha(x)\} \geq 1 - 2\delta - 2\epsilon$$

and using again the fact that shifting a random $x \in \mathcal{X}_1$ by a fixed $v \in \mathcal{X}_0$ does not change the distribution too much we obtain

$$\Pr_{x \in \mathcal{X}_1, \alpha \in M} \{f_A(w) = A_\alpha(x+v+w) - A_\alpha(x+v)\} \geq 1 - 2\delta - 3\epsilon. \tag{4}$$

Using the above argument a third time, now with $v + w$ instead of v (and thus 2ϵ instead of ϵ) we get

$$\Pr_{x \in \mathcal{X}_1, \alpha \in M}\{f_A(v+w) = A_\alpha(x+v+w) - A_\alpha(x)\} \geq 1 - 2\delta - 4\epsilon. \tag{5}$$

Combining (3), (4) and (5) it follows

$$\Pr_{x \in \mathcal{X}_1, \alpha \in M}\{f_A(v+w) = f_A(v) + f_A(w)\} \geq 1 - 6\delta - 9\epsilon.$$

This is independent of both x and α, so the probability is either 0 or 1. Hence, choosing δ and ϵ small enough it will be 1 and the first part of the linearity condition is proved.

Concerning scalar multiplicativity let $e_i \in \mathbb{R}^n$ be a unit vector and $\lambda \in \Lambda$. Since $\lambda \cdot e_i \in \mathcal{X}_0$ one can apply Lemma 1 together with (3) to get

$$\Pr_{x \in \mathcal{X}_1, \alpha \in M}\{f_A(\lambda \cdot e_i) = A_{\alpha/\lambda}(\lambda \cdot e_i + \lambda \cdot x) - A_{\alpha/\lambda}(\lambda \cdot x)\} \geq 1 - 2\delta - 4\epsilon.$$

Since $A_{\alpha/\lambda}(\lambda \cdot e_i + \lambda \cdot x) - A_{\alpha/\lambda}(\lambda \cdot x) = \lambda(A_\alpha(e_i + x) - A_\alpha(x))$ and by (3)

$$\Pr_{x \in \mathcal{X}_1, \alpha \in M}\{f_A(e_i) = A_\alpha(e_i + x) - A_\alpha(x)\} \geq 1 - 2\delta - 2\epsilon$$

it follows that

$$\Pr_{x \in \mathcal{X}_1, \alpha \in M}\{f_A(\lambda \cdot e_i) = \lambda f_A(e_i)\} \geq 1 - 4\delta - 6\epsilon.$$

This is again independent of x and α, so choosing δ and ϵ small enough yields $f_A(\lambda \cdot e_i) = \lambda f_A(e_i)$. Finally, given additivity on $\mathcal{X}_0$ and scalar multiplicativity for scalars $\lambda \in \Lambda$ on the standard basis the claim follows. □

Checking consistency. If the function value tables for both A and B have been tested with high probability to be close to unique linear functions f_A and f_B it remains to deal with consistency of these two functions. If $a \in \mathbb{R}^n, b \in \mathbb{R}^{n^2}$ are the corresponding coefficient vectors consistency means that $b_{ij} = a_i \cdot a_j$. In this subsection it is outlined how to test it.

For any $x \in \mathcal{X}_0$ and $\epsilon > 0$ it has been shown how to compute the correct value of $f_A(x)$ with probability $1 - \epsilon$ by making only a constant number of queries. We can therefore from now on pretend to simply get the correct values of $f_A(x)$ and $f_B(z)$. The probabilities of obtaining an incorrect value at the places where these functions are used are added to the small probability with which we are allowed to accept incorrect proofs.

For $x \in \mathbb{R}^n$, let $x \otimes x$ denote the vector $y \in \mathbb{R}^{n(n+1)/2}$ for which $y_{i,j} = x_i \cdot x_j$, $1 \leq i \leq j \leq n$. Now a is consistent with b if and only if for all $x \in \mathbb{Z}_2^n$ it is the case that $f_A(x)^2 = f_B(x \otimes x)$.[6] This is the property that will be tested. Repeat the following consistency test a constant number of times:

Consistency test:

- Uniformly choose random x from $\mathbb{Z}_2^n$;
- check if $f_A(x)^2 = f_B(x \otimes x)$.

If in every round of the test the check is correct the verifier accepts, otherwise it rejects.

As with the linearity test the interesting case to deal with is when the verifier accepts the consistency test with high probability.

[6] The appropriate domain on which f_B can be shown to be linear in particular contains $x \otimes x$ for all $x \in \mathbb{Z}_2^n$.

Lemma 3. *With the above notations if the linearity test accepts with probability* $> \frac{3}{4}$*, then consistency of* a *and* b *holds.*

Proof. The proof basically relies on the fact that if two vectors in some $\mathbb{R}^N$ are different multiplying both with a random $x \in \mathbb{Z}_2^N$ will give different results with probability at least $\frac{1}{2}$. This is applied to the two linear functions on $\mathbb{R}^{n^2}$ resulting from $a \otimes a$ and b. The same is true over $\mathbb{C}$. For details see [64].

Putting everything together. The linearity and consistency tests together ensure that any proof for which the self-corrections f_A and f_B are not linear on $\mathcal{X}_0$ or are not consistent are rejected with high probability. So the only thing left to do is to verify whether a is indeed a zero of the polynomial system. This is done by evaluating equation (1). If it evaluates to zero the verifier accepts, otherwise not.

Summarizing the results of this section we finally get

Theorem 18. *For every problem* $L \in \mathrm{NP}_{\mathbb{R}}$ *there is a verifier working as follows: Given an instance* w *of size* n *the verifier expects a proof of length* $f(n)$*, where* f *is doubly exponential in* n*. The verifier inspects a constant number of components and accepts* L *according to the requirements of Definition 13. It has a decision time that is polynomially bounded in* n*.*

The according statement holds for $\mathrm{NP}_{\mathbb{C}}$*.*

The proof can be adapted word by word for the complex number BSS model. There is no argument involved that uses the presence of an ordering, except that in the definition of $P(x, r)$ we avoided to use instead the sum of the squared single polynomials of the system as could be done over the reals. This would save some small amount of randomness, introducing at the same time a more complicated polynomial of degree 4.

Recall that the size of $\mathfrak{D}$ in our proof is doubly exponential in the input size. That was the reason for including Remark 1 above. Note however that the decisive point behind Theorem 18 is the structure of the verification proof. In the next section we shall see that for the full $\mathrm{PCP}_{\mathbb{R}}$ theorem transparent long proofs are invoked in a situation where inputs are of constant size. Then their structure is more important than the parameter values since the latter automatically are constant.

5.3 The full PCP theorem

In 2005 Dinur [33] gave an alternative proof for the classical PCP theorem. This proof was much more combinatorial than the original one by Arora et al. In this subsection we outline how to transform Dinur's proof to the BSS model both over $\mathbb{R}$ and $\mathbb{C}$. We only sketch the main ideas and refer to [5] for full proof details. The next subsection then discusses our current knowledge concerning a potential proof that closer follows the lines of the original one by Arora et al.

Central aspect of Dinur's proof is the design of a very particular reduction between instances of a Constraint Satisfiability Problem CSP. The latter is a generalizitaion of the 3-Satisfiability problem. An instance of CSP consists of a collection of constraints over a finite alphabet and the question is whether all can be satisfied in common by an assignment for variables taken from the underlying alphabet. The reduction longed for creates a gap in the following sense. If a given CSP instance is satisfiable so is the one generated by the reduction. But if the given instance is not satisfiable, then for the one obtained by the reduction at least a constant fraction of constraints cannot be satisfied in common. This constant fraction

is the gap. It has been well known early that the existence of such an efficient gap-creating reduction is equivalent to the PCP theorem. However, before Dinur's proof it could only be designed using the PCP theorem.

It is easy to see that for a suitable variant of the QPS problem the existence of a gap reduction as well would imply the PCP theorem over both $\mathbb{R}$ and $\mathbb{C}$. So the strategy is to adapt Dinur's proof to the BSS framework. This in fact turns out to be possible. Below we describe the main ideas for the real number model. Over $\mathbb{C}$ nothing changes significantly.

The problem to consider. A problem in the real number model which is similar to the above mentioned CSP problem is the following variant of the $\mathrm{QPS}_{\mathbb{R}}$ problem. Here, the way to look upon a system of polynomial equations is slightly changed.

Definition 15. *Let $m, k, q, s \in \mathbb{N}$. An instance of the $\mathrm{QPS}_{\mathbb{R}}(\mathrm{m}, \mathrm{k}, \mathrm{q}, \mathrm{s})$ problem is a set of m constraints. Each constraint consists of at most k polynomial equations each of degree at most two. The polynomials in a single constraint depend on at most q variable arrays which have dimension s, i.e., they range over $\mathbb{R}^s$.*

Hence, a single constraint in a $\mathrm{QPS}_{\mathbb{R}}(m, k, q, s)$-instance depends on at most qs variables in $\mathbb{R}$. So if there are m constraints the whole instance contains at most qm arrays and at most qsm variables. For what follows parameters q and s are most important; q will be chosen to be 2, i.e., each constraint will depend on 2 variable arrays. Controlling s so that it remains constant is a crucial goal during the different design steps of the gap reduction. Note that the problem is $\mathrm{NP}_{\mathbb{R}}$-complete for most values of (q, s), for example if $q \geq 2, s \geq 3$.

Definition 16. *A $QPS_{\mathbb{R}}(m, k, q, s)$-instance ϕ is satisfiable if there exists an assignment in $\mathbb{R}^{mqs}$ which satisfies all of its constraints. A constraint is satisfied by an assignment if all polynomials occurring in it evaluate to zero. The minimum fraction of unsatisfied constraints, where the minimum is taken over all possible assignments, is denoted by $UNSAT(\phi)$. So if ϕ is satisfiable $UNSAT(\phi) = 0$ and if ϕ is unsatisfiable, then $UNSAT(\phi) \geq 1/m$.*

With a gap reduction we mean an algorithm which in polynomial time transforms a $\mathrm{QPS}_{\mathbb{R}}(m, k, q, s)$-instance ϕ into a $\mathrm{QPS}_{\mathbb{R}}(m', k', q, s)$-instance ψ such that there exists a fixed constant $\epsilon > 0$ and

- if ϕ is satisfiable, then ψ is satisfiable and
- if ϕ is not satisfiable, then $\mathrm{UNSAT}(\psi) \geq \epsilon$.

Thus either all constraints in the output instance ψ are satisfiable or at least an ϵ-fraction is violated, no matter which values are assigned to the variables. Most important, ϵ is a fixed constant not depending on the size of the given instances.

The following easy lemma shows the importance of gap-reductions for the PCP theorem:

Lemma 4. *Suppose there exists a gap reduction for an $\mathrm{NP}_{\mathbb{R}}$-complete $\mathrm{QPS}_{\mathbb{R}}(\mathrm{m}, \mathrm{k}, \mathrm{q}, \mathrm{s})$ with a fixed $\epsilon > 0$. Then the $\mathrm{PCP}_{\mathbb{R}}$ theorem holds, i.e., $\mathrm{NP}_{\mathbb{R}} = \mathrm{PCP}_{\mathbb{R}}(O(\log n), O(1))$.*

Proof. The task is to construct a $(O(\log n), O(1))$-verifier V for $\mathrm{QPS}_{\mathbb{R}}(m, k, q, s)$. Supposing the existence of a gap-reduction the verifier works as follows on an instance ϕ. First, it applies the reduction and computes ψ. As proof of satisfiability of ψ (and thus of ϕ) it expects an assignment for the variables of ψ. Then, finitely many times the following is repeated: V selects at random a constraint in ψ and evaluates it in the given assignment. Since each constraint of ψ depends on at most qs variables this bounds the number of proof components V reads in a single round. In case that ϕ is not satisfiable each assignment violates an ϵ-fraction of clauses in ψ. Thus V randomly picks with probability $\geq \frac{1}{\epsilon}$ such a constraint for the assignment given by the proof. Repeating this procedure constantly many times the error probability can be made arbitrarily small, thus proving the $\mathrm{PCP}_{\mathbb{R}}$ theorem. □

Outline for creating a gap reduction. The goal now is to design such a gap reduction following Dinur's original construction. This is done in a number of rounds. Each of them increases the gap by a factor of at least 2 if the instance on which the round was performed still had a small gap below a suitable constant ϵ_{final}. Since for the original instance ϕ it holds that either ϕ is satisfiable or at least a $\frac{1}{m}$-fraction of its constraints is always unsatisfied (i.e., one constraint), in principle it suffices to perform a logarithmic in m number of such rounds in order to create an instance which has a gap of at least ϵ_{final}. The number of rounds not being constant it is important that in a single round the size of the instance grows linearly only. That way a logarithmic number of rounds creates a polynomial growth in size only.

Each round consists of three steps, a preprocessing, a gap amplification step and a dimension reduction step. These steps use $\mathrm{QPS}_{\mathbb{R}}$-instances with different values for the number q of variable arrays the constraints of an instance depend on. The two important values for this q are 2 in the amplification step and a constant Q coming from the constant query complexity of long transparent proofs.

A round starts on an instance of the problem $\mathrm{QPS}_{\mathbb{R}}(m, k, Q, 1)$, where Q denotes the number of queries the verifier of Section 5.2 needs to verify the long transparent proofs. So in these instances every constraint depends on at most Q arrays of dimension 1, i.e., variables ranging over $\mathbb{R}^1$. The first main step in a single round is the amplification step which amplifies the gap. However, this step requires the array dimension to be $q = 2$ as well as some nice structure on its input instances. To fulfill these requirements a preprocessing step is necessary. Though amplification increases the gap it has the disadvantage of enlarging the dimension of variable arrays. To get finally back arrays of dimension 1 it is necessary to continue after amplification with a dimension reduction step. For this step long transparent proofs are crucial.

Both the preprocessing step and the dimension reduction step decrease the gap. Since the amplification factor can be taken large enough in comparison to the two factors by which the other steps reduce the gap, in total there will be a sufficient increase of the gap after logarithmically many rounds.

Preprocessing consists of a number of relatively simple constructions, so we omit this technical step and just summarize its outcome.

Proposition 1. *There exist a constant $d \in \mathbb{N}$ and a polytime computable reduction from* $\mathrm{QPS}_{\mathbb{R}}$ *instances to* $\mathrm{QPS}_{\mathbb{R}}$ *instances such that the following holds. The reduction maps an instance ϕ in* $\mathrm{QPS}_{\mathbb{R}}(\mathrm{m}, \mathrm{k}, \mathrm{q}, \mathrm{s})$*-instance to a* nice *instance ψ in* $\mathrm{QPS}_{\mathbb{R}}(3\mathrm{qd}^2\mathrm{m}, \mathrm{k} + \mathrm{qs}, 2, \mathrm{qs})$ *such that*

- *if ϕ is satisfiable, then ψ is satisfiable;*
- *if ϕ is not satisfiable, then $UNSAT(\psi) \geq UNSAT(\phi)/(240qd^2)$.*

The term *nice* in the above statement refers to a special structure the resulting instances exhibit. This structure is related to so-called expander graphs which are heavily used in the amplification step. Expanders in particular are regular graphs and the parameter d used in the statement denotes this regularity. Without being too technical below it will be pointed out what kind of properties of expanders are needed.

The amplification step. As already mentioned amplficiation requires a $\mathrm{QPS}_{\mathbb{R}}(m, k, 2, s)$-instance ψ as input. To such an instance one can canonically attach a constraint graph in which vertices correspond to variable arrays and edges correspond to the constraints depending on at most two arrays each. Constraints depending on a single array only give loops in the constraint graph.

Starting with an instance ψ obtained at the end of preprocessing a new instance ψ^t is constructed as follows. The ultimate goal is to amplify the occurence of a constraint in ψ in such a way that it influences much more constraints in ψ^t. Since constraints correspond to edges in the constraint graph this amplification is obtained by Dinur invoking deep results about expander graphs. Very roughly, constraints of the new instance collect several constraints (edges) of ψ in such a way that each violated old constraint forces violation of many of the new constraints in which it occurs. In order to make this idea working the structure of d-regular expander graphs is important.

Here is a brief outline of how the new instance ψ^t is obtained. Its construction depends both on the regularity d and an additional freely choosable constant parameter $t \in \mathbb{N}$. Both determine the factor with which the gap is amplified.

The new instance ψ^t will have the same number of variable arrays but they will be of larger dimension. For every vertex in the constraint graph of the input instance ψ a new array will be defined. The dimension of these new arrays will be so large that they can claim values for all old arrays which can be reached in the constraint graph within at most $t + \sqrt{t}$ steps. Since one of the conditions on the input instance is that its constraint graph is regular of degree d for some constant d the size of the new arrays is bounded by the constant $d^{t+\sqrt{t}+1} \cdot s$. So for every array in the old instance ψ there will be lots of arrays in the new instance ψ^t that claim a value for it. Of course these claimed values can be different. In the proof for the classical case there is the guarantee that at least a certain fraction of the claims will be equal because of the finite alphabet. In the real number case we do not have this guarantee because there the "alphabet" is of course infinite. However, this technical problem can easily be circumvented by adding some consistency requirements to the constraints in the new instance ψ^t.

The constraints in ψ^t will be sets of constraints of the old instance ψ together with consistency requirements as mentioned above. For every path of length $2t+1$ in the constraint graph of ψ a constraint will be added to the new instance ψ^t. This constraint will depend on the two arrays corresponding to the endpoints of the path. The new arrays claim values for all old arrays in a $t + \sqrt{t} + 1$-neighborhood of the vertex. Therefore, all old arrays related to vertices in a certain middle segment of such a path get values from the new arrays corresponding to both end-points. The constraint that we add will express that these claimed values are consistent and that they satisfy the constraints of the old instance ψ. This finishes the rough description of how to construct the new instance.

It is easy to see that the construction transfers satisfiability from ψ to ψ^t. The hard part of Dinur's proof is to show that if the input instance ψ is not satisfiable, then the fraction of unsatisfied constraints in the new instance ψ^t is increased by a constant factor depending on t. This is shown by Dinur as follows and basically can be done similarly in the real and complex number framework. Take any assignment for the new arrays. From this assignment a plurality assignment is defined for the old arrays. More precisely, perform a random walk of t steps on the constraint graph starting in the vertex of the old array. Its plurality value is the assignment that most frequently is claimed for the old array by those new arrays that occur as an endpoint of such a walk. The further analysis now exploits both the expander properties of the constraint graph together with an additional structural requirement called niceness before. It basically addresses the number of loops each vertex in the constraint graph has. This in a suitable way makes random walks of length t basically look like walks of bit shorter or longer length. It finally guarantees the quantity $UNSAT(\psi^t)$ to grow proportionally in $\sqrt{t} \cdot UNSAT(\psi)$.

The formal statement resulting from the above ideas reads

Theorem 19. *There exists an algorithm which works in polynomial time that maps a nice* $\mathrm{QPS}_{\mathbb{R}}(\mathrm{m},\mathrm{k},2,\mathrm{s})$*-instance* ψ *to a* $\mathrm{QPS}_{\mathbb{R}}(\mathrm{d}^{2\mathrm{t}}\mathrm{m}, 2\sqrt{\mathrm{t}}\mathrm{k}+(2\sqrt{\mathrm{t}}+1)\mathrm{s}, 2, \mathrm{d}^{\mathrm{t}+\sqrt{\mathrm{t}}+1}\mathrm{s})$*-instance* ψ^t *and has the following properties:*

- *If* ψ *is satisfiable, then* ψ^t *is satisfiable.*
- *If* ψ *is not satisfiable and* $UNSAT(\psi) < \frac{1}{d\sqrt{t}}$*, then* $UNSAT(\psi^t) \geq \frac{\sqrt{t}}{3520d} \cdot UNSAT(\psi)$*.*

Dimension reduction. Amplification increases the gap but also the dimension of variable arrays. However, the latter in the end has to remain constant because it is directly related to the query complexity. Thus in the final step of a single round the array dimension has to be reduced. Actually, it can be put down to 1 before the next round starts.

To achieve this aim the structure of transparent long proofs as explained above plays the decisive role. First, there is a natural close relation between the computation of verifiers and sets of constraints. To each string of random bits a verifier generates one can associate a constraint. This constraint expresses the verifier's computation after the random string has been generated. It is satisfied if the verifier accepts the proof with the corresponding random bits. If the input instance of the verifier is satisfiable, then there exists an assignment (i.e., a proof) which satisfies all of these constraints; and if the input instance is not satisfiable every assignment violates at least half of the constraints.

Now we apply this viewpoint to the instance ψ^t generated after amplification. The idea is to view every constraint in ψ^t as an input instance for the long transparent proof verifier and replace this constraint with the set of constraints which we described in the lines above. Note that a single constraint in ψ^t still has constant size. It depends on two arrays of dimension $s(t) := d^{t+\sqrt{t}+1} \cdot s$. Since the verifier checks this for a concrete assignment within a time bound depending on the constraint size, all of the derived constraints described above also are constant in size. It therefore is of no concern that the verifier needs long transparent proofs; they all still have constant size. More important is the structure of the verifier.

In order to realize this idea for dimension reduction first parts of the preprocessing step are applied once more; to do so the original constraints in ψ^t have to be decoupled. This means that different constraints have to depend on different arrays. It is achieved by using formal copies. Of course, the intended reduction must carry over satisfiability, so the decoupling of variables has to be repaired afterwards by introducing consistency constraints.

Now the particular structure of the long transparent proof guarantees that both the original constraints in ψ and the consistency constraints can be replaced by constraints that depend on at most Q variable arrays of dimension 1 each. Here, Q is the query complexity of the long transparent verifier.

Dimension reduction thus gives

Theorem 20. *There exists a reduction which works in polynomial time and maps a* $\mathrm{QPS}_{\mathbb{R}}(\mathrm{m}(\mathrm{t}),\mathrm{k}(\mathrm{t}),2,\mathrm{s}(\mathrm{t}))$*-instance* ψ^t *to a* $\mathrm{QPS}_{\mathbb{R}}(\widehat{\mathrm{m}}(\mathrm{t}),\mathrm{C},\mathrm{Q},1)$*-instance* $\widehat{\psi^t}$*, where* C, Q *are constants,* $\widehat{m}(t)$ *is linear in* $m(t)$ *(the multiplication factor being double exponential in* $s(t)$*) and the following holds:*

- *If* ψ^t *is satisfiable, then so is* $\widehat{\psi^t}$ *and*
- *if* ψ^t *is unsatisfiable, then* $UNSAT(\widehat{\psi^t}) \geq UNSAT(\psi^t)/(160(d+1)^2)$*.*

The final argument is to apply the above steps a logarithmic in m number of times for a given $\mathrm{QPS}_{\mathbb{R}}(m,k,q,s)$-instance. A suitable choice of t guarantees that the amplification factor in each round is at least 2. So starting with a fraction of $\frac{1}{m}$ unsatisfied constraints the gap is increased to a constant fraction. We finally obtain

Theorem 21 (PCP theorem for $\mathrm{NP}_{\mathbb{R}}$, [5]). *It holds* $\mathrm{NP}_{\mathbb{R}} = \mathrm{PCP}_{\mathbb{R}}(O(\log n), O(1))$.

All proof details are given in [5]. None of the arguments rely on the ordering available over the real numbers and so the statement holds as well for the complex number BSS model.

5.4 Almost transparent short proofs

Though we have seen in the previous sections that Dinur's proof of the PCP theorem can be adapted to the BSS model another interesting question remains open. Can the real number PCP theorem be proved as well along the lines of the classical proof by Arora et al.? In this final section we briefly indicate what is currently known concerning this problem.

The classical proof uses long transparent proofs as well as two additional constructions. Another verifier is designed that uses a logarithmic amount of randomness and inspects a polylogarithmic number of components. The 'almost transparent short' proof that this verifier requires in addition must obey a certain structure in order to make the final step applicable. This is a composition step of the two verifiers resulting in the final verifier whose existence implies the PCP theorem. So far it is possible to construct an almost transparent short proof for $\mathrm{NP}_{\mathbb{R}}$. However, at the time being it is not clear to the authors how to put this verifier into a more specific structure in order to make the final step working. We comment on this point at the end.

Let us shortly explain the main ideas in designing this verifier following [65]. Instead of using tables of linear functions as coding objects for a zero of a polynomial system now multivariate polynomials of a not too large degree are employed. They are usually called low-degree polynomials in this framework.

The problem setting. Starting point once again is the $\mathrm{QPS}_{\mathbb{R}}$ problem in the version of Definition 7. For it the verifier is constructed. Given the $\mathrm{NP}_{\mathbb{R}}$-completeness proof of $\mathrm{QPS}_{\mathbb{R}}$ in [15] an instance system $\mathcal{P}$ in variables $x_1, \ldots, x_n$ can be further assumed to be of the following particular form. Each polynomial has one the types below:

Type 1: $x_{i_1} - c_\ell$, where c_ℓ is one among finitely many fixed real constants,
Type 2: $x_{i_1} - (x_{i_2} - x_{i_3})$,
Type 3: $x_{i_1} - (x_{i_2} + x_{i_3})$ or
Type 4: $x_{i_1} - (x_{i_2} \cdot x_{i_3})$.

Here the i_1, i_2 and i_3 do not have to be different. As with the linear encodings used before we change a bit the viewpoint on an assignment for the system's variables.

To do this the index set of the variables in $\mathcal{P}$ is coded differently. Choose integers h, k such that $h^k \geq n$ and set $H := \{1, \ldots, h\}$. Now H^k is used as index set instead of $\{1, \ldots, n\}$. Thus a real assignment $a \in \mathbb{R}^n$ to the variables is a function $f_a : H^k \to \mathbb{R}$. Next, the way to look upon the system $\mathcal{P}$ is altered. More precisely, for each polynomial in $\mathcal{P}$ its type is extracted by means of using certain characteristic functions for the types.

Towards this end $\mathcal{P}$ is seen as a subset of some universe U to be specified; now identify $\mathcal{P}$ with the function $\chi : U \to \{0, 1\}$ which maps elements, i.e., polynomials in $\mathcal{P}$ to 1 and elements outside $\mathcal{P}$ to 0. Actually, we will first split $\mathcal{P}$ in four parts $\mathcal{P}^1$, $\mathcal{P}^2$, $\mathcal{P}^3$ and $\mathcal{P}^4$. Part $\mathcal{P}^1$ further splits into finitely many parts $\mathcal{P}^1_\ell$, one for each real coefficient c_ℓ of the system introduced via a polynomial of type 1. A triple $(i_1, i_2, i_3) \in H^{3k}$ uniquely identifies a polynomial in each part. For example, in part $\mathcal{P}^1_\ell$ it identifies the polynomial $x_{i_1} - c_\ell$, in part $\mathcal{P}^2$ it identifies the polynomial $x_{i_1} - (x_{i_2} - x_{i_3})$, and so on. Hence, each part of $\mathcal{P}$ can

be identified with a subset of H^{3k}. The characteristic functions of the respective parts are denoted by χ^1_ℓ, χ^2, χ^3, and χ^4, respectively.

The solvability question this way is transformed into the question of the existence of a function $f_a : H^k \to \mathbb{R}$ such that for all $(i_1, i_2, i_3) \in H^{3k}$ the following equations hold:

$$\begin{aligned}
\chi^1_\ell(i_1, i_2, i_3) \cdot (f(i_1) - c_\ell) &= 0 \quad \text{for all } \ell, \\
\chi^2(i_1, i_2, i_3) \cdot (f(i_1) - (f(i_2) - f(i_3))) &= 0, \\
\chi^3(i_1, i_2, i_3) \cdot (f(i_1) - (f(i_2) + f(i_3))) &= 0, \\
\chi^4(i_1, i_2, i_3) \cdot (f(i_1) - (f(i_2) \cdot f(i_3))) &= 0.
\end{aligned}$$

Squaring and adding lead to $\sum_{(i_1,i_2,i_3)\in H^{3k}} g(i_1, i_2, i_3) = 0$, where $g : H^{3k} \to \mathbb{R}$ is defined as

$$g(i_1, i_2, i_3) := \sum_\ell \left[\chi^{(1)}_\ell(i_1, i_2, i_3) \cdot (f(i_1) - c_\ell)\right]^2 + \left[\chi^{(2)}(i_1, i_2, i_3) \cdot (f(i_1) - (f(i_2) - f(i_3)))\right]^2 +$$
$$\left[\chi^{(3)}(i_1, i_2, i_3) \cdot (f(i_1) - (f(i_2) + f(i_3)))\right]^2 + \left[\chi^{(4)}(i_1, i_2, i_3) \cdot (f(i_1) - (f(i_2) \cdot f(i_3)))\right]^2.$$

Summarizing, an assignment $a \in \mathbb{R}^n$ is a zero of the given system $\mathcal{P}$ if and only if the sum $\sum_{(i_1,i_2,i_3)\in H^{3k}} g(i_1, i_2, i_3) = 0$, where g is defined as above using an encoding $f_a : H^k \to \mathbb{R}$ for a. The degree of g in each of its variables is $d := O(h)$.

Sum check and low degree extensions. At the moment not much is gained. If the sum is evaluated term by term there are at least $|H|^k \geq n$ many terms that depend on at least one value of f_a, thus such a direct evaluation would inspect too many proof components. However, the particular form allows to proceed differently in order to circumvent this problem. The first step is to apply a well-known technique called sum-check procedure [55] in order to evaluate the above huge sum more efficiently using randomization. The idea is to express the sum recursively as iterated sum of univariate polynomials and including an encoding of those univariate polynomials in the verification proof. More precisely, for $1 \leq i \leq 3k$ one defines partial-sum polynomials of g as

$$g_i(x_1, \ldots, x_i) := \sum_{y_{i+1}\in H} \sum_{y_{i+2}\in H} \cdots \sum_{y_{3k}\in H} g(x_1, \ldots, x_i, y_{i+1}, \ldots, y_{3k}).$$

Note that $\sum_{r\in H^{3k}} g(r) = \sum_{x_1\in H} g_1(x_1)$ and $g_i(x_1, \ldots, x_i) = \sum_{y\in H} g_{i+1}(x_1, \ldots, x_i, y)$ for all i.

In order to make the probability analysis of the following procedure working it turns out that the polynomials g and g_i have to be defined on a larger set F^{3k}, where $H \subset F$ (even though the sum to be computed still ranges over H^{3k}). The verifier expects a proof to contain for each $1 \leq i \leq 3k, (r_1, \ldots, r_{i-1}) \in F^{i-1}$ a univariate polynomial $x \to g'_i(r_1, \ldots, r_{i-1}, x)$ of degree at most d. The proof is required to represent such a polynomial by specifying its $d+1$ many real coefficients. An ideal proof is supposed to use the corresponding restriction $x \to g_i(r_1, \ldots, r_{i-1}, x)$ of the partial-sum polynomial g_i as $g'_i(r_1, \ldots, r_{i-1}, x)$. The basis of the sum-check procedure now is to verify for all i the relation $g_i(x_1, \ldots, x_i) = \sum_{y\in H} g_{i+1}(x_1, \ldots, x_i, y)$ together with $\sum_{y\in H} g_1(y) = 0$. In the corresponding test this is done finitely many times using a random choice $(r_1, \ldots, r_{3k}) \in F^{3k}$ of points in F for the x_i's. It can be shown that this test when accepted guarantees with high probability that the pairs (g_i, g_{i+1}) are consistent and that the entire sum evaluates to 0. Most important, this part of the verifier needs the following resources. Choosing $3k$ many points from F randomly requires $O(k \cdot \log |F|)$ many

random bits. For each of the $O(k)$ many equations tested the verifier reads $d+1$ many proof components representing the univariate polynomial $y \mapsto g_i(r_1, \ldots, r_{i-1}, y)$. For the final check $\sum_{y \in H} g_1(y) = 0$ a constant number of values from f_a is required.

The analysis (not given here) then guarantees everything to work fine when the involved parameters are chosen according to $k = O(\log n), h = O(\log n), |F| = poly \log n$.

There is one major new problem that has been tacitly introduced in the above sum-check procedure. Its probability analysis makes it necessary to consider the g_i on a larger domain F^{3k}. But f_a originally is defined on H^k only. So if in the sum-check part a value of f_a in an argument outside H^k has to be inspected, we must first extend f_a consistently to domain F^k. Consistency of course is crucial here in order to make sure that still the same f_a, and thus the same assignment $a \in \mathbb{R}^n$, is used.

Dealing with this problem is the main task for obtaining the desired verifier. By interpolation every function $f : H^k \to \mathbb{R}$ can in a unique way be seen as a polynomial in k variables that ranges over H and with degree at most $h-1$ in each variable. This polynomial of course is defined as well on any larger set F^k since we consider both H and F as subsets of $\mathbb{R}$. It is this low-degree extension that should be used in the sum-check. Note that the above g in fact is obtained in a similar way using as well the low-degree extensions of the $\chi^{(i)}$ functions in its definition.

But then we are left with a question that is very similar to what has been discussed in relation to long transpaernt proofs. The verifier expects a function value table of a function $\tilde{f} : F^k \to \mathbb{R}$. As part of its test it first has to convince itself that the table with high probability is close to a unique low-degree polynomial $f : F^k \to \mathbb{R}$. This polynomial is identical to the low-degree extension of a function f_a. It is this a which then is expected by the verifier to solve the given polynomial system.

So it is necessary to design a test which checks such a function $\tilde{f}$ for being close to a low-degree polynomial with high probability. Once again, a problem for doing it is that the domains H and F cannot be taken to have a nice structure such as finite fields which are used in the Turing setting. However, based on work by Friedl et al. [37] it is shown in [65] that such a test can be developed and additionally respects the required resource bounds. Putting this low-degree test and the sum-check procedure together one obtains

Theorem 22 ([65]). $\mathrm{NP}_{\mathbb{R}} = \mathrm{PCP}_{\mathbb{R}}(O(\log n), poly \log n)$.

It remains open whether the full $\mathrm{PCP}_{\mathbb{R}}$ theorem can be obtained pushing the above ideas forward by combining the two verifiers using a long transparent and a short almost transparent proof, respectively. The reason why we are doubtful is that in order to apply a real version of verifier composition - a technique introduced by [3] and similar to the use of the long transparent proof in Dinur's approach - the verifier of Theorem 22 needs to obey an improved structure. In the classical proof this better structure is obtained by designing yet another low-degree test which considers the *total* degree of polynomials. In contrast, the low-degree test used above is dealing with the maximal degree of each variable. What is the problem here? It seems that designing a better structured total degree test over the reals might use a much larger domain to be tested, thus leading to a higher amount of randomness necessary. We do not know at the time being whether such a test can be designed respecting the required resourse bounds. This certainly is an interesting research question of independent interest since it deals with a typical example of property testing in real domains.

References

1. Arora, S., Barak, B.: Computational Complexity: A Modern Approach. Cambridge University Press, 2009.

2. Arora, S., Lund, C., Motwani, R., Sudan, M., Szegedy, M.: Proof verification and hardness of approximation problems. Journal of the ACM 45 (3), 501–555, 1998. Preliminary version: Proc. 33rd Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, 14–23, 1992.
3. Arora, S., Safra, S.: Probabilistic checking proofs: A new characterization of NP. Journal of the ACM 45 (1), 70–122, 1998. Preliminary version: Proc. of the 33rd Annual IEEE Symposium on the Foundations of Computer Science, 2–13, 1992.
4. Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., Protasi, M.: Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties. Springer, 1999.
5. Baartse, M., Meer, K.: The PCP theorem for NP over the reals. Preprint available from the authors, 2012.
6. Basu, S.: A complex analogue of Toda's theorem. Foundations of Computational Mathematics, vol. 12, 327–362, 2012.
7. Basu, S., Pollack, R., Roy, M.F.: On the combinatorial and algebraic complexity of quantifier elimination. Journal of the ACM, 43(6), 1002–1045, 1996.
8. Basu, S., Zell, T.: Polynomial hierarchy, Betti numbers, and a real analogue of Toda's theorem. Foundations of Computational Mathematics, 10(4), 429–454, 2010.
9. Beltrán, C., Pardo, L.M.: On Smale's 17th problem: a probabilistic positive solution. Found. Comput. Math., 8(1), 1–43, 2008.
10. Ben-David, S., Meer, K., Michaux, C. : A note on non-complete problems in $NP_{\mathbb{R}}$. Journal of Complexity vol. 16, no. 1, 324–332, 2000.
11. Bernstein, D.N.: The number of roots of a system of equations. Funct. Anal. Appl.,), 183–185, 1975.
12. Blum, L.: Computing over the reals: Where Turing meets Newton. Notices of the AMS, vol. 51(9), 1024–1034, 2004.
13. Blum, L., Cucker, F., Shub, M., Smale, S.: Algebraic settings for the problem P $\neq$ NP. In: Proceedings of the AMS Summer Seminar on "Mathematics of Numerical Analysis: Real Number Algorithms", Park City 1995, Lectures in Applied Mathematics, eds.: J. Renegar, M. Shub, S. Smale, 125–144, 1996.
14. Blum, L., Cucker, F., Shub, M., Smale, S.: Complexity and Real Computation. Springer, 1998.
15. Blum, L., Shub, M., Smale, S.: On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. Bull. Amer. Math. Soc., vol. 21, 1–46, 1989.
16. Blum, L., Smale, S.: The Gödel Incompleteness Theorem and Decidability over a Ring. In : From Topology to Computation , Proceedings of the Smalefest, Springer, 321–339, 1993.
17. Boone, W.W.: The word problem. Proc. Nat. Acad. Sci. USA, vol. 44, 265–269, 1958.
18. Braverman, M., Cook, S.A.: Computing over the Reals: Foundations for Scientific Computing. Notices of the AMS, 53(3), 318–329, 2006.
19. Bürgisser, P.: On the Structure of Valiant's Complexity Classes. Discrete Mathematics & Theoretical Computer Science 3(3), 73–94, 1999.
20. Bürgisser, P.: Completeness and Reduction in Algebraic Complexity Theory. Volume 7 of Algorithms and Computation in Mathematics, Springer, 2000.
21. Bürgisser, P., Clausen, M., Shokrollahi, M.A.: Algebraic Complexity Theory, volume 315 of *Grundlehren*. Springer, 1997.
22. Bürgisser, P., Cucker, F.: Variations by complexity theorists on three themes of Euler, Bézout, Betti, and Poincaré. In: Complexity of computations and proofs, J. Krajicek (ed.), Quaderni di Matematica 13, 73–152, 2005.
23. Bürgisser, P., Cucker, F.: Counting complexity classes for numeric computations. II. Algebaric and semialgebraic sets. Journal of Complexity, vol. 22, no.2 , 147–191, 2006.
24. Bürgisser, P., Cucker, F.: On a Problem Posed by Steve Smale Annals of Mathematics 174(3), 1785–1836, 2011.
25. Calvert, W., Kramer, K., Miller, R.: Noncomputable functions in the Blum-Shub-Smale model Logical Methods in Computer Science, vol. 7(2), DOI: 10.2168/LMCS-7(2:15)2011, 2011.

26. Chapuis, O., Koiran, P.: Saturation and stability in the theory of computation over the reals. Annals of Pure and Applied Logic, 99, 1–49, 1999.
27. Collins, G.E.: Quantifier Elimination for real closed fields by Cylindrical Algebraic Decomposition. Springer Lecture Notes in Computer Science vol. 33, 134–183, 1977.
28. Cucker, F.: The Arithmetical Hierarchy over the Reals. Journal of Logic and Computation, vol 2(3), 375–395, 1992.
29. Cucker, F.: $\mathrm{NC}_{\mathbb{R}} \neq \mathrm{P}_{\mathbb{R}}$. Journal of Complexity, vol. 8, 230–238, 1992.
30. Cucker, F., Koiran, P.: Computing over the reals with addition and order: Higher complexity classes. Journal of Complexity, vol. 11, 358–376, 1995.
31. Cucker, F., Meer, K. : Logics which capture complexity classes over the reals. Journal of Symbolic Logic, Vol. 64, No.1, 363–390, 1999.
32. Dedieu, J.P., Malajovich, G., Shub, M.: On the curvature of the central path of lineare programming theory. Found. Comput. Mathematics, vol. 5(2), 145–171, 2005.
33. Dinur, I.: The PCP theorem by gap amplification. Journal of the ACM Vol. 54 (3), 2007.
34. Fitchas, N., Galligo, A., Morgenstern, J.: Precise sequential and parallel complexity bound for quantifier elimination over algebraically closed fields. Journal of Pure and Applied Algebra 67, 1–14, 1990.
35. Fournier, H., Koiran, P.: Are lower bounds easier over the reals? Proc. of the 30^{th} Annual ACM Symposium on Theory of Computing, 507–513, 1998.
36. Fournier, H., Koiran, P.: Lower Bounds Are not easier over the reals: Inside PH. 27th Proc. International Colloquium Automata, Languages and Programming ICALP, Springer Lecture Notes in Computer Science 1853, 832–843, 2000.
37. Friedl, K., Hátsági, Z., Shen, A.: Low-degree tests. Proc. SODA, 57–64, 1994.
38. Frisco, P.: Computing with Cells - Advances in Membrane Computing. Oxford University Press, 2009.
39. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman, San Francisco, 1979.
40. Gaßner, C.: A Hierarchy below the Halting Problem for Additive Machines. Theor. Comp. Systems, vol. 43 (3-4), 464–470, 2008.
41. Grädel, E., Meer, K.: Descriptive Complexity Theory over the Real Numbers. In: Proceedings of the AMS Summer Seminar on "Mathematics of Numerical Analysis: Real Number Algorithms", Park City 1995, Lectures in Applied Mathematics, eds.: J. Renegar, M. Shub, S. Smale, 381–404, 1996.
42. Grigoriev, D.Y.: Complexity of Deciding Tarski Algebra. Journal of Symbolic Computation, 5, 65–108, 1988.
43. Haykin, S.: Neural Networks - A Comprehensive Foundation. Prentice Hall, 2nd edition, 1999.
44. Heintz, J., Roy, M.F., Solerno, P.: *On the complexity of semialgebraic sets.* Proceedings IFIP 1989, San Fracisco, North-Holland 293–298 (1989).
45. Hougardy, D., Prömel, H.J., Steger, A.: Probabilistically checkable proofs and their consequences for approximation algorithms. Discrete Mathematics 136, 175–223, 1994.
46. Jansen, K., Margraf, M.: Approximative Algorithmen und Nichtapproximierbarkeit. de Gruyter, 2008.
47. Jongen, H.Th., Meer, K., Triesch, E.: Optimization Theory. Springer, 2004.
48. Ko, K.: Complexity Theory of Real Functions. Progress in Theoretical Computer Science. Birkhäuser Boston, 1991.
49. Koiran, P.: A weak version of the Blum-Shub-Smale model. Proceedings of the ACM Conference on Foundations of Computer Science FOCS'93, 486–495, 1993.
50. Koiran, P.: Computing over the reals with addition and order. Theoretical Computer Science, 133(1), 35–48, 1994.
51. Koiran, P.: Eliminations of Constants from Machines over Algebraically Closed Fields. Journal of Complexity 13, 65–82, 1997.
52. Korte, B., Vygen, J.: Combinatorial Optimization: Theory and Algorithms. Algorithms and Combinatorics 21, 5th edition, Springer, 2012.
53. Ladner, L.: On the structure of polynomial time reducibility. J. of the ACM, Vol. 22, 155-171, 1975.

54. Li, T.Y.: Numerical solution of multivariate polynomial systems by homotopy continuation methods. Acta Numer., 6, Cambridge Univ. Press, 399–436, 1997.
55. Lund, C., Fortnow, L., Karloff, H., Nisan, N.: Algebraic methods for interactive proof systems. Journal of the ACM 39 (4), 859-868, 1992.
56. Lyndon, R.C., Schupp, P.E.: Combinatorial Group Theory, Springer, 1977.
57. Malajovich, G.: Nonlinear equations. $28^{\underline{0}}$ Colóquio Brasileiro de Matematicá, Publicações Matemáticas, IMPA Rio de Janeiro, ISBN:978-85-244-329-3, 2011.
58. Malajovich, G. and Meer, K.: On the structure of $\mathrm{NP}_{\mathbb{C}}$, SIAM Journal on Computing, vol. 28 (1), 27–35, 1999.
59. Malajovich, G., Meer, K.: Computing Multi-homogeneous Bézout numbers is hard. Theory of Computing Systems, vol. 40, 553–570, 2007.
60. Matijasevich, Y.: Enumerable sets are diophantine. Dokl. Acad. Nauk 191, 279–282, 1970.
61. Meer, K.: A note on a P $\neq$ NP$-$ result in a restricted class of real machines. Journal of Complexity 8, 451–453, 1992.
62. Meer, K.: On the complexity of Quadratic Programming in real number models of computation. Theoretical Computer Science, vol. 133(1), 85–94, 1994.
63. Meer, K.: Counting Problems over $\mathbb{R}$. Theor. Computer Science, vol. 242, no. 1-2, 41–58, 2000.
64. Meer, K.: Transparent long proofs: A first PCP theorem for $\mathrm{NP}_{\mathbb{R}}$. Foundations of Computational Mathematics, Springer, Vol. 5, Nr. 3, 231–255, 2005.
65. Meer, K.: Almost transparent short proofs for $\mathrm{NP}_{\mathbb{R}}$. Extended abstract in: Proc. 18th International Symposium on Fundamentals of Computation Theory FCT 2011, Oslo, Lecture Notes in Computer Science 6914, 41–52, 2011.
66. Meer, K.: On Ladner's result for a class of real machines with restricted use of constants. Information and Computation, Vol. 210, 13–20, 2012.
67. Meer, K.: Some initial thoughts on bounded query computations over the reals. To appear in: Special Issue on 'Frontier between Decidability and Undecidability and Related Problem', International Journal of Foundations of Computer Science.
68. Meer, K., Ziegler, M.: An explicit solution to Post's problem over the reals. Journal of Complexity, Vol. 24 (1), 3–15, 2008.
69. Meer, K., Ziegler, M.: Real Computational Universality: The Word Problem for a Class of Groups with Infinite Presentation. Foundations of Computational Mathematics, Vol. 9, 599–609, 2009.
70. Meyer auf der Heide, F.: A Polynomial Linear Search Algorithm for the n-Dimensional Knapsack Problem. Journal of the ACM, Vol. 31 (3), 668–676, 1984.
71. Meyer auf der Heide, F.: Fast Algorithms for N-Dimensional Restrictions of Hard Problems. Journal of the ACM, vol. 35 (3), 740–747, 1988.
72. Michaux, C.: $P \neq NP$ over the nonstandard reals implies $P \neq NP$ over $\mathbb{R}$. Theoretical Computer Science, 133, 95–104, 1994.
73. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information. Cambridge University Press, 2000.
74. Novikov, P.S.: On the algorithmic unsolvability of the word problem in group theory. Trudy Mat. Inst. Steklov, vol. 44, 1–143, 1959.
75. Odifreddi, P.: Classical Recursion Theory. North-Holland Publishing Co, 1989.
76. Post, E.: Recursively enumerable sets of positive integers and their decision problems. Bulletin of the American Mathematical Society, vol. 50, 284–316, 1944.
77. Post, E.: A variant of a recursively unsolvable problem. Bulletin of the American Mathematical Society, vol. 52, 264–268, 1946.
78. Radhakrishnan, J., Sudan, M.: On Dinur's proof of the PCP theorem. Bulletin of the American Mathematical Society, vol. 44(1), 19–61, 2007.
79. Renegar, J.: On the computational Complexity and Geometry of the first-order Theory of the Reals , I - III. Journal of Symbolic Computation, 13, 255–352, 1992.
80. Rubinfeld, R., Sudan, M.: Self-testing polynomial functions efficiently and over rational domains. Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms (Orlando, FL, 1992), ACM, 23–32, 1992.
81. Schöning, U.: A uniform approach to obtain diagonal sets in complexity classes. Theoretical Computer Science, vol. 18, 95-103, 1982.

82. Schreiber, H., Meer, K., Schmitt, B.: Dimensional Synthesis of Planar Stephenson-Mechanisms for Motion Generation by Circlepoint Search and Homotopy Methods. Mechanism and Machine Theory, Vol. 37, Nr. 7, 717–737, 2002.
83. Shub, M.: Complexity of Bézout's theorem VI: Geodesics in the condition (number) metric. Found. Comput. Math., 9(2), 171–178, 2009.
84. Shub, M., Smale, S.: Complexity of Bezout's Theorem I : Geometric aspects. Journal of the AMS, 6, 459–501, 1993.
85. Shub, M., Smale, S.: On the intractibility of Hilbert's Nullstellensatz and an algebraic version of $P = NP$. Duke Math. J., 81, 47–54, 1995.
86. Smale, S.: Some Remarks on the foundations of numerical analysis. SIAM Review, vol. 32 (2), 211–220, 1990.
87. Smale, S.: Complexity Theory and Numerical Analysis. Acta Numerica, 6, 523–551, 1997.
88. Smale, S.: Mathematical problems for the next century. Mathematics: frontiers and perspectives, Amer. Math. Soc., Providence, RI, 271–294, 2000.
89. Soare, R.: Recursively enumerable sets and degrees. Springer, 1987.
90. Sommese, A.J., Wampler, C.W.: The numerical solution of systems of polynomials arising in engineering and science. World Scientific, 2005.
91. Syropoulos, A.: Hypercomputation: Computation beyond the Church-Turing barrier. Springer, 2008.
92. Tarski, A.: A decision method for elementary algebra and geometry, 2nd edition, Univ. Calif. Press, Berkeley, 1951.
93. Toda, S.: PP is hard as hard as the polynomial-time hierarchy. SIAM J. Comput., vol. 20(5), 865 – 877, 1991.
94. Traub, J.F., Wasilkowski, G.W., Woźniakowski, H.: Information-based complexity. Academic Press, 1988.
95. Turing, A.M.: Rounding-off errors in matrix processes. Quart. J. Mech. Appl. Math 1, 287–308, 1948.
96. Valiant, L.G.: The complexity of computing the permanent. Theoretical Computer Science, vol. 8(2), 189–201, 1979.
97. Valiant, L.G.: Completeness classes in algebra. In: Proc. 11th ACM Symposium on Theory of Computing 1979, ACM New York, 249–261, 1979.
98. Weihrauch, K.: Computable Analysis: An Introduction. Springer, 2000.
99. Woźniakowski, H.: Why does information-based complexity use the real number model? Theoretical Computer Science, 219 (1-2), 451–465, 1999.
100. Woźniakowski, H., Novak, E.: Tractability of multivariate problems. Volume 1: Linear information. European Mathematical Society, Tracts in Mathematics 6, 2008.
101. Yonezawa, Y.: The Turing degrees for some computation model with the real parameter. J. Math. Soc. Japan, vol. 60 (2), 311–324, 2008.
102. Ziegler, M.: (Short) Survey of real hypercomputation. Proc. 3rd Conf. on Computability in Europe CiE, Siena, Springer LNCS vol. 4497, 809–824, 2007.
103. Zippel,R: Probabilistic algorithms for sparse polynomials. Proc. Int.Symposium on Symbolic and Algebraic Computation, Lecture Notes in Computer Science vol. 72, Springer, 216–226, 1979.