Pure and Applied Functional Analysis
Volume 8, Number 4, 2023, 1135–1157

Yokohama Publishers
ISSN 2189-3764 ONLINE JOURNAL

PROOF VERIFICATION IN REAL NUMBER COMPLEXITY

KLAUS MEER

Dedicated to Hubertus Theodor Jongen on the occasion of his 75th birthday

Abstract. The question how much of a potential proof of a mathematical statement has to be read in order to be (almost) convinced that the proof is correct or false is of fundamental interest in all branches of mathematics. It led to the concept of probabilistically checkable proofs relating theoretical computer science, complexity theory and discrete algorithmic mathematics, culminating in the famous PCP-theorem characterizing the complexity class NP. The related concepts and questions make perfect sense as well in a more algebraic computational framework, as it was formalized by Blum, Shub, and Smale for computations and complexity theory over the real numbers. In this article we survey work over the last years on PCPs over the real numbers, leading to a real number version of the PCP-theorem, and its connections to questions in polynomial optimization.

1. Introduction

Let us start with addressing some general problems involving polynomials as data. Given a multivariate polynomial $f : \mathbb{R}^n \to \mathbb{R}$, in optimization one is interested in finding $\min_{x\in\mathbb{R}^n} f(x)$. The problem could as well be constrained by polynomial (in-)equalities. As a *decision problem*, a class of problems typically considered in structural complexity theory, the input data might include an additional bound $M \in \mathbb{R}$ and the question then is whether there exists an $x \in \mathbb{R}^n$ such that $f(x) \leq M$? A related family of problems is that of polynomial equation/inequality system solving PSS. Given multivariate polynomials $f_1, \ldots, f_s : \mathbb{R}^n \to \mathbb{R}$, is there a real point $x^* \in \mathbb{R}^n$ such that $f_i(x^*) = 0, 1 \leq i \leq s$? Yet another optimization problem related to PSS, but of a different flavour, is MAX-PSS: Given a family of polynomials as above we ask for the maximal number of polynomials which have a common real zero. Clearly, an algorithm solving the latter problem easily leads to one of same complexity solving the PSS decision problem since one only has to compare the outcome of MAX-PSS with the number s of polynomials in the family. Both problems in a structural complexity theoretic framework are conjectured to be computationally hard, see Section 2 below. A relaxation of the MAX-PSS problem which perhaps is easier could ask for only approximating (in a precise sense) the maximal number of polynomials with a common zero. The question whether this

2020 *Mathematics Subject Classification.* 68Q15, 68Q25, 68Q60, 03D78.
Key words and phrases. Real number computations, probabilistically checkable proofs, real number PCP theorem.

indeed results in an easier problem surprisingly lies at the heart of probabilistically checkable proofs.

Of course, polynomial optimization has evolved as an important field both from the theoretical side and concerning algorithmic solutions of problem classes. Here, we are interested in the structural complexity theory of such problems, in particular the concept of verifying solvability of instances. Moreover, we consider related problems under an algorithmic approach that is more algebraically oriented. We deal with reals as input data and consider the real number model of computation by Blum, Shub, and Smale, briefly BSS model [12]. It treats real numbers as underlying entities and allows the basic arithmetic operations being performed exactly and with unit cost. A short introduction into complexity theory for this model is given below.

We now briefly outline what our focus is when considering the above problems. Let us concentrate on the PSS decision problem, which in fact is representative for a huge class of similar problems. This is captured by the notion of $\mathrm{NP}_{\mathbb{R}}$-completeness, see below. In its general form, the decision problem seems to be difficult, and even designing algorithms deciding solvability of polynomial systems without taking complexity issues into account is far from being easily doable. Tarski [29] was the first who showed that such algorithms exist, and improving the complexity of his algorithm has been (and still is) an important task in research, see [16,18,26]. However, the problem has a specific structure which makes *verification* of solvability easy. If, by what reason ever, one has a *guess* about a solution $y \in \mathbb{R}^n$ of the system, then one can easily evaluate the defining polynomials in y and accept in case y is a zero for all of them. In a computational model which allows to perform basic arithmetic operations and an equality test (or inequality in case of inequality systems) exactly the verification is easily doable within a number of steps that is bounded in the size of the data instance (the given polynomials) by a polynomial. Such a time bound in complexity theory is, by good reasons, considered to characterize *efficient* algorithms. Though of course there is no hint how to find a suitable y in case it exists, and so there is no direct algorithmic use of the fast verification feature, this structural property is shared by many important problems and has led to the definition of the important class $\mathrm{NP}_{\mathbb{R}}$ of decision problems. The class $\mathrm{NP}_{\mathbb{R}}$ is a real number analogue of NP in the Turing model of computation dealing with discrete problems. It pinpoints the intuitive difference of proof verification versus proof finding. The above guess $y \in \mathbb{R}^n$ can be seen as a *proof certificate* that the system is solvable. However, computing such a y or another proof of solvability efficiently from the given data in case it exists is a widely open task. The class $\mathrm{NP}_{\mathbb{R}}$ formalizes the conjectured difference between proof verification and proof finding - a fundamental mathematical question. And since efficient verification of a given proof seems easier than finding it efficiently by oneself, it is reasonable to conjecture that the two notions capture different problem classes, the famous $\mathrm{P}_{\mathbb{R}} \neq \mathrm{NP}_{\mathbb{R}}$ conjecture in a real number version.

Here, we shall focus on such verification procedures, but under a different point of view. In the above example it is obvious that the outlined verification which evaluates given polynomial systems in potential solutions y in general has to inspect all components of y in order to give the correct answer for this certificate. For any potential verification strategy that does exclude inspection of a component

of y, one can design a system such that the strategy gives a false answer when using y as proof. This leads to a fundamental question. Of course, a verification procedure for PSS might use completely different ideas than starting from a potential solution of the system and evaluating all polynomials. So the question is: Are there verification procedures for which it is sufficient to inspect only a fraction of the certificate, nevertheless yielding the correct result? Or, more exaggerated: Is it possible to rewrite proofs (more precisely: for instances of $\mathrm{NP}_{\mathbb{R}}$ problems) in such a way that verifying its correctness can be done without inspecting major parts of the proof? This fundamental question has led to some of the most influential areas and results in theoretical computer science in the last three decades, culminating in the famous PCP-theorem [2,3]. It had tremendous impact, including results about (non-)approximability of important combinatorial optimization problems. The latter are similar to the above mentioned MAX-PPS problem. A moment of reflection makes evident that we have to make precise what we require from a verification procedure if a proof certificate is not entirely inspected; clearly, if a part that is not inspected is changed the verifier cannot realize it. This naturally leads to the introduction of randomization and the analysis of failure probabilities. Even then, the statement of the PCP-theorem is extremely surprising since it shows that the part of a suitably encoded proof that has to be inspected is constant and thus independent of the length of the underlying statement.

After this brief introduction into the world of probabilistically checkable proofs we outline the structure of this survey. The classical PCP-theorem for discrete Turing complexity theory has been developed around the 1990ies. There are dozens of surveys and we only point to the original proof [2,3], a significantly different proof by Dinur [13], and the textbook [1] as starting point for further studies. In this paper, we report on own work done in recent years when studying related questions in the Blum-Shub-Smale computational model over the real numbers. Section 2 briefly presents the model description, the definition of basic complexity classes and the concepts around probabilistically checkable proof $\mathrm{PCP}_{\mathbb{R}}$'s, property testers and $\mathrm{PCP}_{\mathbb{R}}$'s of proximity in the real number setting. The subsequent sections describe $\mathrm{PCP}_{\mathbb{R}}$ results having been obtained for the BSS setting. Since most of the proofs behind the statements are quite technical we only explain one easier result (property testers for linear real functions) in more detail to give the reader a first intuition about the format of the verification proofs necessary to obtain $\mathrm{PCP}_{\mathbb{R}}$ results. Proofs of the main results in the area are then described conceptually.

The paper is written for readers who have not heard much about the treated concepts so far. We mainly aim for raising interest for the discussed problems instead of being complete in the mathematical description of proofs. Nevertheless, we try to describe the main proof ideas as well as pointing to the differences and difficulties when studying the concepts in the real number framework in comparison to classical (discrete) complexity theory formalized by the Turing model. Notationally, whenever we refer to probabilistically checkable proofs or similar concepts in the BSS framework, we add an index $\mathbb{R}$ and write $\mathrm{PCP}_{\mathbb{R}}$.

2. Blum-Shub-Smale model over $\mathbb{R}$, complexity classes, and concepts of proof verification

2.1. **The computational model and basic complexity classes.** In the BSS-model of computation real numbers are considered as entities. An input instance x of a computational problem is a finite sequence of real numbers, i.e., x stems from the set $\mathbb{R}^\infty := \bigsqcup_{i\geq 1} \mathbb{R}^i$. Algorithms in the model are allowed to perform the basic arithmetic operations $\{+,-,\bullet,:\}$ as well as a test: 'is $a \geq b$' for real numbers a, b that have been computed already or are parts of the input. These operations are performed exactly, so the model abstracts from round-off errors. Data is stored in registers each containing a real number; there are countably many registers available, but at every specific point in time only finitely many registers contain relevant data. In order to deal with uniform algorithms that handle input instances of larger and larger size, copy instructions allow to shift data between specific registers. A BSS-program M is given by a flowchart of such instructions. A specific starting state determines the beginning of a computation and in which registers the input is stored. A halting state marks the end of a computation if ever reached. Then, the result can be found in some pre-determined registers. For computational arithmetic operations, the next instruction is uniquely determined by the flowchart, for a test instruction the program can branch according to its outcome. All instructions have unit cost and the running time $T_M(x)$ of M on input $x \in \mathbb{R}^\infty$ is the number of operations performed until M halts and ∞ if M computes forever. For studying complexity of algorithms it is important to analyse the asymptotic behavior of T_M in dependence of the *size* of instances. In view of the model's abstraction to consider reals as basic entities it is consequent to define

Definition 2.1. a) Let $x \in \mathbb{R}^n \subset \mathbb{R}^\infty$. The *(algebraic) size* of x, denoted by $size_{\mathbb{R}}(x)$, is defined as $size_{\mathbb{R}}(x) = n$.

b) Let $t : \mathbb{N} \to \mathbb{N}$ and M be a BSS algorithm. M is *t-time bounded* if $\forall x \in \mathbb{R}^\infty : T_M(x) \leq t(size_{\mathbb{R}}(x))$.

c) If M is time-bounded by a polynomial we say that M *runs in polynomial time*.

d) Conversely, a function $f : I \subseteq \mathbb{R}^\infty \to \mathbb{R}^\infty$ is *polynomial time computable*, if there is a BSS algorithm M computing f and being polynomial time bounded. The class of all such functions is denoted by $\mathrm{FP}_{\mathbb{R}}$.

Of special interest in complexity theory are decision problems. Here, for a set $L \subseteq \mathbb{R}^\infty$ the question is to decide algorithmically whether an input $x \in \mathbb{R}^\infty$ belongs to L or not, i.e., to compute the characteristic function χ_L of L in $\mathbb{R}^\infty$. [1]

Definition 2.2. The class $\mathrm{P}_{\mathbb{R}}$ of *polynomial-time decidable* real number decision problems consists of all $L \subseteq \mathbb{R}^\infty$ such that $\chi_L \in \mathrm{FP}_{\mathbb{R}}$.

[1] For many problems it makes sense to restrict the set of inputs to a subset $I \subset \mathbb{R}^\infty$ and then decide L in I. For example, below we often consider as input families of polynomials. Then, in a first step an algorithm should check whether an input coding has the desired form. Since this is an easy task for all problems considered here, i.e., deciding I in $\mathbb{R}^\infty$ is a problem in $\mathrm{P}_{\mathbb{R}}$, we disregard this technical detail and always consider inputs from $\mathbb{R}^\infty$.

Example 2.3. A decision problem in $\mathrm{P}_{\mathbb{R}}$ is the question for solvability of linear equation systems. It is straightforward that Gaussian elimination is a polynomial time real number algorithm in the algebraic size of a system (A, b), which is $n(m+1)$ for systems with m equations in n variables. This is actually easier to show than the corresponding result in the Turing model since we do not have to take the bit-size of growing intermediate results into account. Another problem in $\mathrm{P}_{\mathbb{R}}$ is the question, whether a univariate polynomial $f : \mathbb{R} \to \mathbb{R}$ of degree d, given by its $d+1$ coefficients, has a real zero. Sturm's algorithm [8] provides an efficient decision algorithm. In contrast, the algebraic complexity of Linear Programming as real number decision problem: Given $A \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^m$, is there an $x \in \mathbb{R}^n$ s.t. $Ax \leq b$ is a major open problem in optimization theory. Well-known polynomial time algorithms in the Turing framework like interior-point or the ellipsoid method are not efficient (so-called strongly polynomial time) algorithms in the algebraic setting (see [30]) and it is unclear, whether the latter exist, see also [28].

The following problems will be crucial throughout the rest of the paper.

Definition 2.4 (Quadratic Polynomial Systems QPS)**.**

a) The decision problem *QPS* is defined as follows. Input: Integers $s, n \in \mathbb{N}$, and for $1 \leq i \leq s$ polynomials $f_i : \mathbb{R}^n \to \mathbb{R}$ of degree at most 2. In addition, each f_i only depends on at most 3 among the variables. Question: Is there a $y \in \mathbb{R}^n$ such that $f_i(y) = 0\ \forall 1 \leq i \leq s$?

b) The *MAX-QPS* problem is the following optimization problem. Input: Integers s, n, polynomials $f_1, \ldots, f_s$ as above. Question: What is the maximal number of polynomials among the f_i that have a common real zero?

If above in addition to the f_i we also provide a $y \in \mathbb{R}^n$ and ask, whether y is a common zero, then the resulting decision problem easily is seen to be in $\mathrm{P}_{\mathbb{R}}$ (the interested reader should specify the input size and the complexity of the obvious decision algorithm). The aspect which makes the problems most likely difficult is asking for the existence of such a y in the uncountable search space $\mathbb{R}^n$. Clearly, MAX-QPS is at least as difficult as QPS, since MAX-QPS $\in \mathrm{FP}_{\mathbb{R}}$ would imply QPS $\in \mathrm{P}_{\mathbb{R}}$.

A natural size for a QPS instance is the sum of the number of coefficients for all f_i. Since every f_i depends on at most 3 variables there are 10 coefficients, thus the size is $O(s)$. Note that $n \geq s/3$ if the system depends on all variables, a reasonable requirement. Therefore, without loss of generality below we take n as the size of such an instance. Deciding existence of a common real zero seems to be a difficult problem, justified by its status of being $\mathrm{NP}_{\mathbb{R}}$-complete, see below. However, if we guess a solution y^*, it is quite easy to verify whether it actually is one; just evaluate all f_i in y^* and check whether $f_i(y^*) = 0$. Obviously, the corresponding verification algorithm runs in polynomial time in the size of the system (actually, in linear time). This structural property is present in many important real number decision problems and is formalized in the following definition.

Definition 2.5 (Class $\mathrm{NP}_{\mathbb{R}}$, $\mathrm{NP}_{\mathbb{R}}$-completeness)**.** a) A decision problem $L \subseteq \mathbb{R}^\infty$ 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 inputs $(x, y) \in \mathbb{R}^\infty \times \mathbb{R}^\infty$ such that

(i) $\forall x \in \mathbb{R}^\infty, y \in \mathbb{R}^\infty : \quad \Phi_M(x,y) \in \{0,1\}$.
(ii) $\forall x \in \mathbb{R}^\infty, y \in \mathbb{R}^\infty : \quad \Phi_M(x,y) = 1 \implies x \in L$
(iii) $\forall x \in L \; \exists y \in \mathbb{R}^\infty : \quad \Phi_M(x,y) = 1$ and $T_M(x,y) \leq p(size_{\mathbb{R}}(x))$

b) 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 $f : \mathbb{R}^\infty \to \mathbb{R}^\infty, f \in \mathrm{FP}_{\mathbb{R}}$ which satisfies $\forall x \in \mathbb{R}^\infty : x \in B \Leftrightarrow f(x) \in A$.

Note that in part a) above the dependence of $T_M(x,y)$ on $n := size_{\mathbb{R}}(x)$ only implies that only certificates y of size polynomially bounded in that of x are of interest. So the 'search space' for correct proofs is $\mathbb{R}^{p(n)}$. Any y will be rejected if $x \notin L$, for $x \in L$ at least one easily verifiable proof must exist. Just as in classical complexity theory the most important open question in our framework is whether $\mathrm{P}_{\mathbb{R}} \neq \mathrm{NP}_{\mathbb{R}}$. The relation $\mathrm{P}_{\mathbb{R}} \subseteq \mathrm{NP}_{\mathbb{R}}$ is obvious. Now $\mathrm{NP}_{\mathbb{R}}$-complete problems capture the difficulty of $\mathrm{NP}_{\mathbb{R}}$ in the sense that $\mathrm{P}_{\mathbb{R}} = \mathrm{NP}_{\mathbb{R}}$ iff there exists an $\mathrm{NP}_{\mathbb{R}}$-complete problem in $\mathrm{P}_{\mathbb{R}}$. This can easily be seen by noticing that the composition of a polynomial time reduction from any problem $A \in \mathrm{NP}_{\mathbb{R}}$ to an $\mathrm{NP}_{\mathbb{R}}$-complete one L with a potential efficient algorithm for L results in a polynomial time algorithm for A.

The following theorem summarizes two fundamental results in real number complexity theory. Part a) is due to [12], part b) had a long history starting with the work of Tarski [29] on quantifier elimination and the existential theory of the reals in particular. The complexity bound stated is far from trivial and has been established in various different forms independently by [16, 18, 26].

Theorem 2.6. a) *The QPS problem is* $\mathrm{NP}_{\mathbb{R}}$*-complete.*
b) *Every decision problem in* $\mathrm{NP}_{\mathbb{R}}$ *can be decided by a BSS algorithm whose running time is bounded by a single-exponential function of the input size.*

$\mathrm{NP}_{\mathbb{R}}$-completeness is the justification why below we can concentrate on the QPS problem. Note that the approach can literally be transformed to a model of computation over the complex numbers where, as only difference, only equality tests are allowed. Then complexity classes are defined similarly and the previous theorem holds analogously. This in fact holds as well for the results presented below, but we shall only concentrate on the reals as underlying structure. Note that a discrete version of QPS denoted by 0-1-QPS considers quadratic polynomials over $\mathbb{Z}_2$ and asks for a common zero $x \in \{0,1\}^n$. It is well known to be NP-complete in the classical Turing-sense. Actually, here already the question of 0-1 solvability of linear systems with rational data is NP-complete [20].

2.2. **Probabilistically checkable proofs and related concepts.** As already indicated, our focus now changes. We shall from now on be interested in how much of a certificate has to be read by a verification algorithm still getting a result being true with high probability. Clearly, if we consider QPS and the most natural verification algorithm guessing a potenial common root y, each component of y has to be inspected, when the verification should be a uniform algorithm working for all QPS instances and certificates in the same way. But is it possible to read a significantly smaller part of y only? Clearly, the certificate then has to code

something else than a common zero, or has to code such a zero in a significantly different form. Similarly, as soon as parts of the proof are not read there is a chance of errors; just suppose a correct proof is changed at one component, then with high probability this component is not seen by the verifier if it only inspects a small fraction of a certificate. The concept of probabilistically checkable proofs, PCPs for short, formalizes such ideas as follows.

Definition 2.7. Let $r, q : \mathbb{N} \mapsto \mathbb{N}$ be two functions. An $(r(n), q(n))$-restricted verifier V in the BSS model over $\mathbb{R}$ is a randomized BSS algorithm working as follows. For an input $x \in \mathbb{R}^\infty$ of algebraic size n and another vector $y \in \mathbb{R}^\infty$ representing a potential membership proof of x in a certain set $L \subseteq \mathbb{R}^\infty$, 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. This again is done non-adaptively, i.e., the choice of components does not depend on previously seen values of other components. 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 (result 1) or rejects (result 0) the pair (x, y). 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 is also called its *decision time*. It has to be polynomially bounded in the size of x.

Remark 2.8. 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 $\mathrm{PCP}_{\mathbb{R}}$-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 3.1 below.

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

Definition 2.9. ($\mathrm{PCP}_{\mathbb{R}}$-classes) Let $r, q : \mathbb{N} \mapsto \mathbb{N}$; a decision problem $L \subseteq \mathbb{R}^\infty$ is in class $\mathrm{PCP}_{\mathbb{R}}(r(n), q(n))$ 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 $y \in \mathbb{R}^\infty$ such that for all randomly generated strings $\rho \in \{0,1\}^{O(r(size_{\mathbb{R}}(x)))}$ the verifier accepts. In other words: $\Pr_\rho\{V(x, y, \rho) = 1\} = 1$.

b) If $x \notin L$, then for all $y \in \mathbb{R}^\infty$: $\Pr_\rho\{V(x, y, \rho) = 0\} \geq \frac{3}{4}$.

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

In classical complexity theory the famous PCP-theorem, proved in the 1990ies by several authors, states the surprising fact that for all problems in class NP a verification is possible which uses a logarithmic amount of randomness and only inspects a constant number of proof components:

Theorem 2.10 ([2,3]). $\mathrm{PCP}(\log n, 1) = \mathrm{NP}$.

The theorem has been given an alternative proof by Dinur [13]. Below, we shall outline how both proofs can be adapted to show the analogue in the BSS model:

Theorem 2.11 ([5,6]). $\mathrm{PCP}_{\mathbb{R}}(\log n, 1) = \mathrm{NP}_{\mathbb{R}}$.

Example 2.12. Usually, the design of verifiers for interesting problems having small resource parameters is far from trivial. We therefore just present one easy example of such a construction in a very special situation related to the MAX-QPS problem already mentioned before. As we shall later explain, this example is much closer related to the $\mathrm{PCP}_{\mathbb{R}}$-theorem than one might expect at a first glance. It serves as a starting point for the second proof of Theorem 2.11, adapting the ideas of Dinur to the real number world. For $\epsilon > 0$ let GAP-QPS(ϵ) be a subclass of QPS instances defined as follows: We only consider systems $\{f_1, \ldots, f_s\}$ as above, which in addition either have a common zero or, otherwise, at most a fraction of $1 - \epsilon$ polynomials among the f_i have a common zero. Thus, in this case for each $y \in \mathbb{R}^n$ at least ϵs many f_i satisfy $f_i(y) \neq 0$. Now we show GAP-QPS(ϵ) $\in \mathrm{PCP}_{\mathbb{R}}(\log n, 1)$: The verifier V independently repeats $O(1/\epsilon)$ rounds the following. It randomly picks one f_i using $\log s = O(\log n)$ bits to code index i. Then, it interprets the certificate y as a potential zero (just as in the 'natural' verification proof) and checks whether $f_i(y) = 0$. If at least one equality test fails V rejects. Now in the special situation of instances from GAP-QPS(ϵ) this verifier needs the restricted resources from the theorem. First, if $\{f_1, \ldots, f_s\}$ has a common zero y^*, V will clearly accept it with probability 1 since each round shows $f_i(y^*) = 0$ for the chosen i. Secondly, suppose the system has no common zero. Then for every certificate y the chosen f_i satisfies $f_i(y) \neq 0$ with probability $\geq \epsilon$, thus implying rejection. The probability of not detecting an error in k independent rounds thus is at most $(1 - \epsilon)^k$, which (for small enough ϵ) is at most $\frac{1}{4}$ for $k = O(1/\epsilon)$. Since ϵ is a fixed constant in the above reasoning it follows that V in $k = O(1)$ rounds uses $O(\log n)$ random bits. Furthermore, since each f_i in the instance only depends on at most 3 variables, in each round V only inspects 3 components of the certificate. It follows that V is $(\log n, 1)$-restricted.

The example is not very interesting unless we get a feeling for how strict the restriction to the subclass GAP-QPS(ϵ) is from a complexity theoretic point of view. The easier the restricted problem is, the less interesting is the design of a good verifier (note that $\mathrm{PCP}_{\mathbb{R}}(0,0) = \mathrm{P}_{\mathbb{R}}$). And contrary, if we were able to show that GAP-QPS(ϵ) is an $\mathrm{NP}_{\mathbb{R}}$-complete problem, then the above verifier, when composed with a polynomial time reduction from any problem $A \in \mathrm{NP}_{\mathbb{R}}$ to GAP-QPS(ϵ), would establish Theorem 2.11. This is the starting point for one of the

proofs of Theorem 2.11 below: Show, that there is a polynomial time reduction from the complete QPS problem to GAP-QPS(ϵ) for a fixed ϵ (or, more precisely, suitable variants of those problems). This also shows that the question of computing efficiently good approximations to MAX-QPS instances lies at the heart of the $\mathrm{PCP}_{\mathbb{R}}$-theorem, see Theorem 3.9.

In the mentioned classical proofs further concepts varying PCPs and called property testing and PCPs of proximity play an important role. Especially, property testing has become an own branch of complexity theory, see [15]. As before, these concepts can easily be formalized in the BSS setting as well. On an intuitive level, they deal with a kind of approximate decision making for real number decision problems.

Definition 2.13. a) A *property tester* for a decision problem $L \subseteq \mathbb{R}^\infty$ is a randomized algorithm V as above, which we again call verifier. Given an input x and an error bound $\epsilon > 0$, the verification should confirm with probability 1 if $x \in L$; and if x is ϵ-far away from elements in L (where distance is measured by the number of different components), V rejects with probability at least $\frac{3}{4}$. Here, V accesses x via oracle calls, i.e., it queries a black box for components of x.

b) Let $\epsilon > 0$ be fixed. A verifier is a $\mathrm{PCP}_{\mathbb{R}}$ *of proximity* (with respect to ϵ) for L, if it accesses both x and an additional certificate y via oracle calls. For $x \in L$ there should exist a y such that $V(x, y)$ ='accept' with probability 1. And if x has distance at least ϵ to L, then V rejects (x, y) for all y with probability at least $\frac{3}{4}$.

Below we shall outline how to design a property test for linear real valued functions on a finite subset of $\mathbb{R}^n$ and a $\mathrm{PCP}_{\mathbb{R}}$ of proximity for various kinds of polynomials defined on finite domains and suitably small values of ϵ. It is then of major importance how much randomization V needs and what its query complexity to the oracle is. Both constructions are ingredients of showing the $\mathrm{PCP}_{\mathbb{R}}$-theorem.

3. Proofs of the real number PCP-theorem

In this section we shall describe two ways how to establish Theorem 2.11. Since both existing proofs conceptually take their starting point from the famous discrete counterparts, i.e., [6] follows [2,3] and [5] follows [13], we shall in particular elaborate where new difficulties arise when moving from discrete computations over binary strings and the Turing model to the BSS model over $\mathbb{R}$.

Let us start with the more algebraic proof [6]. The original discrete proof consists of three different major parts. As explained above, a verifier better suited with respect to the parameters addressed in the PCP-theorem than the natural one must code a suitable certificate proving solvability of a 0-1-QPS instance in a completely different way than just taking a zero as certificate. Towards this aim, the following verifiers are designed: a property tester for linear Boolean functions given via a function value table, and a PCP of proximity for algebraic polynomials of a certain 'low' degree over suitable finite fields. Whereas the former property tester uses too a high amount of randomness, but inspects only a constant number of values in the table, the PCP of proximity for algebraic polynomials has logarithmic randomness,

but inspects polylogarithmically many components of the certificate. The final step towards proving Theorem 2.10 was to compose such verifiers by using a newly developed technique to obtain a new verifier sharing the better parameter bounds of the involved verifiers. For this composition to work it is important that the initial verifiers have a special segmented structure, see below.

Especially the design of a segmented PCP of proximity for polynomials on finite fields, the perhaps hardest part in the entire proof, uses a lot of algebraic techniques. Given $\mathrm{NP}_{\mathbb{R}}$-completeness of QPS, a problem of quite an algebraic flavour, it might seem reasonable to expect an algebraic proof to work as well for showing the $\mathrm{PCP}_{\mathbb{R}}$-theorem. However, it turns out that in our algebraic setting performing an algebraically oriented proof seems to be much harder than following the more combinatorial proof ideas from [13].

We now first describe the algebraic approach and point out, at which places severe differences and difficulties arise. Note that the $\mathrm{PCP}_{\mathbb{R}}$-classes are closed under deterministic polynomial time reductions, so in the sequel it is justified to consider the $\mathrm{NP}_{\mathbb{R}}$-complete problem QPS solely.

3.1. **(Very) Long transparent proofs.** As a first step we describe the design of a verifier for QPS which only inspects a constant number of proof components, but from a (very) large proof certificate. The latter is related to a high amount of randomness the verifier uses. We describe the construction in more detail than the remaining ones in order to give the reader a good idea of how the verifiers to be constructed later on for QPS that are crucial to prove the $\mathrm{PCP}_{\mathbb{R}}$-theorem might look like. Since all these constructions use a lot of technical details, later on we just give an outline of the underlying ideas.

Let $(n, s, f_1, \ldots, f_s)$ be a QPS instance. The idea for designing a verifier reading much less of a proof certificate is to code a potential common zero $a \in \mathbb{R}^n$ via the linear function $A : x \to a^T \cdot x$ generated by a. The function is provided by a table f_A of its values for a suitable finite domain D_A. Then the idea of verifying whether a is a zero of the polynomials is as follows: Consider the function $F(a, r) := \sum_{i=1}^{s} f_i(a) \cdot r_i$ for $a \in \mathbb{R}^n, r \in \{0,1\}^s$. This is a first easy step introducing randomization. For a fixed a and an r chosen uniformly by random it is $F(a, r) = 0$ with probability 1 in case a is a common zero and if not, then $Pr_r(F(a, r) = 0) \leq \frac{1}{2}$. Of course, evaluating $F(a, r)$ in a direct way does not help us since one still would have to access a entirely. Here, another way of expressing F helps. In fact, it is not hard to see that F can be decomposed in the form

$$(*) \quad F(a, r) = E(r) + A\left(L_a(r)\right) + B\left(L_B(r)\right),$$

where $E : \{0,1\}^s \to \mathbb{R}, L_A : \{0,1\}^s \to \mathbb{R}^n, L_B : \{0,1\}^s \to \mathbb{R}^{n^2}$ are linear maps that can be deterministically computed from the QPS instance, i.e., from the given polynomials f_i. The interested reader might compute this representation for an easy example of few polynomials. Consequently, E, L_A and L_B can be efficiently and deterministically evaluated in arguments r. The function $A : \mathbb{R}^n \to \mathbb{R}$ is the above introduced scalar product generated by a potential common zero $a, B : \mathbb{R}^{n^2} \to \mathbb{R}$ is as well generated by a being the linear function given by

$B(y_{11}, y_{12}, \ldots, y_{nn}) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j y_{ij}$ for all $y = (y_{11}, y_{12}, \ldots, y_{nn}) \in \mathbb{R}^{n^2}$. What is the advantage of representing the evaluation of $F(a, r)$ in this seemingly much more involved format? Recall that our goal is not to inspect a in total. Formula $(*)$ instead says: $F(a, r)$ for a random r can be computed by computing (deterministically) $E(r), L_A(r)$, and $L_B(r)$ and then querying a single value from the function value table for A (namely the one in argument $L_A(r)$) and another value from the table for B (in argument $L_B(r)$). This is structurally precisely what we want. If the proof certificate consists of two tables, one for A and one for B, for a single evaluation of $F(a, r)$ in a random r only two components of the proof are read. Repeating this a constant number of times still requires $O(1)$ queries into the certificate only. And the probability to detect an error if a is not a common zero can be pushed arbitrarily close to 1 (with constant distance depending on the number of independent repetitions). Though in principle this describes a verifier which has much better query complexity, the problems only start now. The alternative way to do an evaluation introduces two additional tasks to be settled by the verifier within the allowed resources. The first of these new problems is the same as in the Turing model; both functions A and B are given via tables f_A, f_B of function values. But starting from those tables the verifier has to test that the given tables indeed represent linear functions and that both functions are consistent, i.e., the tables arise from one particular $a \in \mathbb{R}^n$ in the manner described above. This first task (except verifying consistency) leads to *property testing* for linear functions. In the Turing framework for functions from $\mathbb{Z}_2^n \to \mathbb{Z}_2$ this has been done in [11]. Here, a natural idea is to pick random x, y in the domain $\mathbb{Z}_2^n$ and compare the three values of f_A at the positions for x, y, and $x + y$. It can be shown that by repeating the test whether $f_A(x) + f_A(y) = f_A(x + y)$ a constant number of times the conditions of being a property tester for linear Boolean functions are met. Note that the size of the domain $\mathbb{Z}_2^n$ is 2^n, so the table f_A has exponential size in the size $O(n)$ of the 0-1-QPS instance and the amount of randomization needed to address points in the domain is polynomial instead of logarithmic in n. In this sense the table f_A provides (parts of) a long proof. The main new difficulty when following this idea in the BSS setting is the question, on which domain A should be defined as a real number function. Whereas $\mathbb{Z}_2^n$ as domain has the structure of a vector space, if we look for a suitable finite set D_A on which the table f_A should provide the values of A, we do not even have closeness of addition and scalar multiplication. Also, for the latter it is far from obvious which scalars should be considered; since the table f_A is the input of the verification task it has to be finite and so has the set of real scalars for which the linearity condition might be checked. Beside this, D_A clearly must satisfy the following elementary properties: If the values on D_A correspond to a linear function, it must be unique. So D_A should contain a basis of $\mathbb{R}^n$. And in order to evaluate $(*)$ for random r all points $L_A(r), r \in \mathbb{Z}_2^s$ have to be in D_A. Similar conditions must be met by the domain D_B considered for f_B.

We now describe the idea behind defining D_A and how then linearity can be verified. First, define a finite set $\mathcal{A}_0 \subset \mathbb{R}^n$ which contains $\mathbb{Z}_2^n$ (and thus a basis of $\mathbb{R}^n$ as real vectors space), all values in the range of $L_A(\mathbb{Z}_2^s)$ together with some additional elements of no concern for outlining the basic idea. The cardinality of

$\mathcal{A}_0$ will be of order $O(n^n)$. Similarly, define a set Λ_0 of real scalars with cardinality of order $O(n)$. The goal is to perform a test which with high probability figures out whether the function value table f_A on $\mathcal{A}_0$ with scalars from Λ_0 is a linear function.This is basically done using tests of the form: pick x, y, λ randomly from $\mathcal{A}_0 \times \mathcal{A}_0$ and Λ_0, respectively, and check whether $f_A(x) + f_A(y) = f_A(x + y)$ and $f_A(\lambda x) = \lambda f_A(x)$. The test should figure out with high probability if f_A in a certain sense is far away from a function satisfying linearity on $\mathcal{A}_0$ with respect to scalars from Λ_0. The major problem is that sums of elements in $\mathcal{A}_0$ as well as multiplication with $\lambda \in \Lambda_0$ in general does not give a point in $\mathcal{A}_0$. We therefore extend the 'safe' domain $\mathcal{A}_0$ to a much larger 'test' domain $\mathcal{A}_1$ of double exponential size in n in such a way that $\mathcal{A}_1$ is almost close under addition with points from $\mathcal{A}_0$ and under scalar multiplication with $\lambda \in \Lambda_0$. And similarly, Λ_0 is enlarged to some Λ_1 such that the latter is almost closed under multiplication with scalars from Λ_0. The domain D_A on which the verifier would like f_A to be given then is $D_A := \{x + y | x, y \in \mathcal{A}_1\} \cup \{\alpha \cdot x | \alpha \in \Lambda_1, x \in \mathcal{A}_1\}$ and as well has a size being double exponential in that of the input system. Now the verifier for property testing linearity on $\mathcal{A}_0$ works as follows. It expects a table f_A of real values on $\mathcal{A}_1$. Then it performs $O(1)$ rounds the following: Pick random $x, y \in \mathcal{A}_1$ and random $\alpha, \beta \in \Lambda_1$ and check whether $f_A(x+y) = f_A(\alpha \cdot x)/\alpha + A(\beta \cdot y)/\beta$. The verifier accepts f_A if all arising equalities in all rounds are satisfied, otherwise it rejects.

If a table f_A passes this test it can be shown that with high probability one can extract from f_A a function A which is linear on $\mathcal{A}_0$ with scalars from Λ_0. With high probability means that each particular value of A can be computed correctly with high probability using values in the table f_A. Note that even if f_A passes the above test, if we want to evaluate A in one specific point (because we want to evaluate $(*)$), it might still be the case that f_A gives a false value in this point. The previous remark indicates that there is a way around this problem, called self-correction, which allows even in such a case to compute the correct value with high probability. Doing the same for the linear function B and designing a verifier checking consistency of A and B (this is much easier) this gives a way to verify whether the given QPS instance is satisfiable by checking constantly many proof components only. More precisely, we arrive at the following theorem originally proved in [22]; our above representation is close to [4]:

Theorem 3.1. *For every problem $L \in \mathrm{NP}_{\mathbb{R}}$ there is a verifier working as follows: Given an instance x of size n the verifier expects a proof of double exponential size in n. The verifier generates uniformly a finite number of random strings. Using those strings it computes the addresses of finitely many proof-components it wants to read. This computation is done without reading the input x, i.e., the components to be seen only depend on the random strings generated. In its decision phase the verifier uses input x together with the finitely many components and accepts L according to the requirements of Definition 2.7. It has a decision time that is polynomially bounded in the input size n. The verifier does not use any additional (real) constants.*

Let us add a more technical comment on the theorem in view of Remark 2.8 above. Recall that the size of D_A in our proof is double exponential in the input size n. Therefore, the random strings used in the proof above are exponential in length. In

the verification procedure they are used to compute the proof-components which the verifier wants to see. In contrast to Definition 2.7 these components are computed independently of the concrete input x (but dependent on n). The reason to require this is that we want to forbid the verifier to potentially use exponential time in the query phase in order to decide the input. After having read the values of the finitely many components the verifier uses the input and the values of those components (and not any longer the random string) in order to make its decision after a running time being polynomial in the size of the input. The verifier constructed above thus is more restricted than general verifiers because it is limited with respect to how it computes the components to be seen. Note, however, that the decisive point behind Theorem 3.1 is the structure of the verification proof. In the next section we shall see that for the $\mathrm{PCP}_{\mathbb{R}}$-theorem the transparent (very) long proofs from Theorem 3.1 are invoked in a situation where inputs are of constant size. In this situation of course also the length of each random string remains constant. Then the structure of the verification procedure is more important than the parameter values; the latter automatically are constant. Therefore, when used in the framework of the full $\mathrm{PCP}_{\mathbb{R}}$-theorem the verifier in Theorem 3.1 can again be chosen according to Definition 2.7.

3.2. **PCPs of proximity for trigonometric polynomials.** The verifier from Theorem 3.1 of course does not realize the resource requirements stated in the $\mathrm{PCP}_{\mathbb{R}}$-theorem. Its certificates are (far) too long and so the necessary randomization is too high. The first proof of the classical PCP-theorem proceeds as follows. It constructs a second verifier for 0-1-QPS which has (better) $O(\log n)$ randomness, but reads $polylog(n)$ positions in the certificate. This proof is based on the use of (algebraic) polynomials having a moderate degree and defined on large finite fields. Such a 'low-degree' polynomial is coding a potential zero of a 0-1-QPS instance similarly as we did it above for linear functions. However, the test procedure for verifying that a function represented by a table of values is close to such a polynomial is much more involved both from the conceptual and the technical side. Beside the table of values the certificate contains further information about the ideally coded polynomial, among other things information about univariate restrictions of the coded polynomial. This leads to the task of designing a PCP of proximity for polynomials in the Turing setting. The third important step in the original proof is to compose the verifiers constructed in the first two steps in a clever way. The underlying technique has been developed as well in relation with the proof; an important property of verifier composition is that it inherits in a precise sense the better parameters of the verifiers involved in the composition. In order to obtain the parameters of the PCP-theorem, the low-degree verifier is composed once with itself and the resulting verifier then is composed with the long transparent one resulting from property testing linearity. One important structural feature to apply verifier composition is *segmentation* of the involved verifiers. Segmentation requires that a verifier reads the information from the certificate in a very structured form.

Definition 3.2. Let $r, q : \mathbb{N} \to \mathbb{N}$ be two resource functions. A real probabilistic $(r(n), q(n))$-restricted verifier V (both of a $\mathrm{PCP}_{\mathbb{R}}$ or a $\mathrm{PCP}_{\mathbb{R}}$ of proximity) is called *segmented* if the queries are structured such that V asks $O(1)$ many segments of

length at most $O(q(n))$ from the certificate (in case of a $\mathrm{PCP}_{\mathbb{R}}$) or both from the instance and the certificate (in case of a $\mathrm{PCP}_{\mathbb{R}}$ of proximity). Here, a segment of a certificate from $\mathbb{R}^{\infty}$ is a consecutive block of components.

Let us sketch the importance of segmentation in relation with the technique of verifier composition. Suppose we have two verifiers V_1, V_2 for QPS using different resources concerning randomization and query complexity. Suppose V_1 during its computation on an input of size n and certificate y_1 wants to inspect $q_1(n)$ many components of y_1. It reads the components and then performs in its decision phase a deterministic polynomial time computation to finally accept or reject. Starting point of verifier composition is to consider this final deterministic decision phase as verification task of another problem; the decision phase of V_1 can be seen as $\mathrm{P}_{\mathbb{R}}$-problem with instance being the string v_1 composed of the $q_1(n)$ many components V_1 wants to see from certificate y_1. As a $\mathrm{P}_{\mathbb{R}}$-problem it is as well in $\mathrm{NP}_{\mathbb{R}}$. So instead of letting V_1 perform the deterministic computation on v_1 one can use the second verifier V_2 to probabilistically verify whether v_1 satisfies the $\mathrm{P}_{\mathbb{R}}$-condition. This way, in principle the 'outer' verifier V_1 can be composed with the 'inner' verifier V_2 to obtain a new one V. The randomization V uses is $r_1(n) + r_2(q_1(n))$, since it starts with simulating V_1 and then replaces V_1's decision phase by simulation of V_2 on inputs of length $q_1(n)$. And its query complexity is that of V_2, again on inputs of size $q_1(n)$. So ideally, V inherits the better parameters from both V_1 and V_2, depending on which outer and inner verifiers are chosen. However, this very short description once more hides major technical problems behind composition. Take as example the previously explained long transparent verifier as V_2. A satisfying assignment (i.e., v_1 above as satisfying assignment for a $\mathrm{P}_{\mathbb{R}}$-property) is encoded as a linear function on a special domain. When V_1 is running several rounds it might be the case that it wants to see several times same components from its certificate, but in different contexts. Then, V has to guarantee that the encodings used by V_2 for instances sharing identical components of y_1 are consistent. Achieving this is obviously heavily depending on which encodings are used to code an assignment. It is here where segmentation together with using suitable encodings plays a crucial role. Verifiers just inspecting $O(1)$ many components automatically are segmented. But for the mentioned PCPs of proximity for low-degree polynomials it was a major difficult task to achieve the segmented form.

Next, we outline new problems arising when one tries to follow a similar approach in the BSS framework. Given that QPS is an $\mathrm{NP}_{\mathbb{R}}$-complete problem dealing with a fundamental question about polynomial system solving it seems natural that the algebraic approach of the proofs in [2, 3] is as well suitable to show the $\mathrm{PCP}_{\mathbb{R}}$-theorem. So the next step could be to code a common zero $y \in \mathbb{R}^n$ of a QPS instance via a table of function values $f_{A(y)}$ on a suitable finite domain $A(y) \subset \mathbb{R}^n$ depending on y. Then, one would have to design a $\mathrm{PCP}_{\mathbb{R}}$ of proximity for verifying that $f_{A(y)}$ on A is close enough to a unique low-degree polynomial on $\mathbb{R}^n$. And in a second step the verifier should manage to use the information given in this $\mathrm{PCP}_{\mathbb{R}}$ of proximity to evaluate with high probability correctly the given QPS instance in y. This outline corresponds to the steps we made in the previous subsection in relation with using linear maps as codes, but this time giving a $\mathrm{PCP}_{\mathbb{R}}$ with resource parameters $(\log n, \mathit{polylog}(n))$. A verifier proving $\mathrm{QPS} \in \mathrm{PCP}_{\mathbb{R}}(\log n, \mathit{polylog}(n))$

based on the use of real algebraic polynomials as codes actually was designed already in [23]. However, as a major drawback it lacks segmentation and thus could not be used in the composition framework. In the Turing model, algebraic polynomials of arity $k = O(\frac{\log n}{\log\log n})$ and low degree $O(\log n)$ are considered on a finite field $\mathbb{F}$ of suitable size. A polynomial p is given via its table of values on $\mathbb{F}^k$ and a PCP of proximity is designed that uses as certificate information about univariate restrictions of p along all lines through $\mathbb{F}^k$. Then (very briefly) closeness of the function represented by the value table to a polynomial is verified by comparing in randomly chosen points the value in the table with that of certain univariate restrictions. Beside the considerable technical difficulties that have to be managed to follow this approach, one important feature is the advantage of working over a finite field as very structured domain. This in particular implies that the restrictions to one-dimensional lines again are low-degree polynomials defined on $\mathbb{F}$.

We have seen above the serious problems arising in the real number setting already when defining a suitable domain for encoding a potential zero of a QPS instance as a linear map. When we try to extend these ideas to multivariate polynomials on some finite domain $A \subset \mathbb{R}^n$ problems become even harder. The structure of being a finite field as domain is lost. Moreover, it is neither clear which directions of lines to which a polynomial is restricted one should consider (there are uncountably many), nor are such lines in general defined on the set A. As consequence, an idea similar to that of Section 3.1 defining a polynomial on a larger test domain and choosing only a finite set of lines on which restrictions are considered seems not to work. In [6] therefore other coding objects are taken. The idea is to somehow combine the structural advantage of using finite fields as domains with the real number objects that are encoded. The set $\mathbb{F}_q := \{0, 1, \ldots, q-1\}$ with large enough prime q is considered together with *trigonometric* polynomials mapping $\mathbb{F}_q^k$ to $\mathbb{R}$.

Definition 3.3. Let $\mathbb{F}_q$ be a finite field as above. For $k, d \in \mathbb{N}$ a trigonometric polynomial $f : \mathbb{F}_q^k \mapsto \mathbb{R}$ of max-degree d is given as $f(x_1, \ldots, x_k) = \sum_t c_t \cdot \exp(\frac{2\pi i}{q} \sum_{j=1}^{k} x_j t_j)$, where the sum is taken over all $t := (t_1, \ldots, t_k) \in \mathbb{Z}^k$ with $|t_1| \leq d, \ldots, |t_k| \leq d$ and $c_t \in \mathbb{C}$ satisfy $c_t = \overline{c_{-t}}$ for all such t.

Periodicity of trigonometric polynomials allows to use $\mathbb{F}_q$ as a finite field; in particular, lines can be considered being still defined on $\mathbb{F}_q$. This partially reduces the problems mentioned above for algebraic polynomials. However, new difficulties occur by the following reason. When a trigonometric polynomial of maximal degree d is restricted to a straight line given as $t \to u + t \cdot v, u, v, \in \mathbb{F}_q^k, t \in \mathbb{F}_q$ the degree of the resulting univariate polynomial depends on the components defining the directional vector v. In a $\mathrm{PCP}_{\mathbb{R}}$ of proximity for trigonometric polynomials the certificate will contain such restrictions in an explicit form, therefore the size of certificates depends on the degree of such restrictions. This leads to the problem that we cannot use all univariate restrictions along lines in $\mathbb{F}_q^k$ to design a $\mathrm{PCP}_{\mathbb{R}}$ of proximity. A major part of the mathematical work in [6] is devoted to characterizing polynomials on $\mathbb{F}_q^k$ via univariate polynomials defined on a suitable small set of lines through $\mathbb{F}_q^k$. In principle, one would like to work with a randomly chosen such line; due to the

problems described above, one tries to approach such a random line by a random walk using only a restricted set of lines defined by sufficiently small components.

This investigation leads to the following

Theorem 3.4. *(*$\mathrm{PCP}_{\mathbb{R}}$ *of proximity for trigonometric polynomials) Let* $d \in \mathbb{N}$, $h := 10^{15}$, $k \geq \frac{3}{2}(2h+1)$, $\tilde{d} := 2hkd$, *and let* $\mathbb{F}_q$ *be a finite field with* q *being a prime number larger than* $10^4(2hkd+1)^3$. *Let* $f : \mathbb{F}_q^k \to \mathbb{R}$ *be a function given by a table of its values.*

a) *There exists a probabilistic verification algorithm in the BSS-model of computation over the reals with the following properties:*
   i) *The verifier as input gets the table for* f *together with a proof string consisting of at most* q^{2k} *segments. Each segment has at most* $2hkd + k+1$ *many real components. The verifier uniformly generates* $O(k \log q)$ *random bits and has a running time that is polynomially bounded in the quantity* $k \log q$, *i.e., polylogarithmic in the input size* $O(q^k)$.
   ii) *For every table representing a trigonometric max-degree* d *polynomial on* $\mathbb{F}_q^k$ *there exists a proof such that the verifier accepts with probability* 1.
   iii) *For any* $0 < \epsilon < 10^{-19}$ *and for every function value table whose distance to a closest max-degree* $\tilde{d} := 2hkd$ *trigonometric polynomial is at least* 2ϵ, *the probability that the verifier rejects is at least* ϵ, *no matter which proof is given.*

b) *Suppose the verifier under a) has accepted* f *and the closest trigonometric polynomial of max-degree* $\leq \tilde{d}$ *is* $\tilde{f}$ *with a distance at most* δ *for arbitrary fixed and small enough* $\delta > 0$. *There exists another segmented verifier working as follows:*
   i) *It gets as input the table for* f, *a point* $x \in \mathbb{F}_q^k$ *and an additional proof certificate with at most* $O(\sqrt{q}^{k-1})$ *segments of length* $2\sqrt{q}kd + 1$ *each. The verifier uniformly generates* $O(k \log q)$ *random bits and has a running time that is polynomially bounded in the quantity* $k\sqrt{q}d$.
   ii) *If* $f \equiv \tilde{f}$ *is a trigonometric max-degree* d *polynomial, there is a certificate such that the verifier accepts with probability* 1.
   iii) *If* $f(x) \neq \tilde{f}(x)$, *the verifier rejects every certificate with probability* $\geq \frac{3}{4}$.

The above result is one of two main building blocks to design a $(\log n, polylog(n))$-restricted and segmented verifier for QPS. The second important ingredient is to evaluate a given QPS instance by another segmented verifier in the point $y \in \mathbb{R}^n$ that is coded as a low-degree trigonometric polynomial. To do so, the verifier expects a potential zero y of a QPS instance depending on n variables to be coded via a k-variate trigonometric polynomial t_y of max-degree $O(\log n)$ on a large enough finite field $\mathbb{F}_q$. Here, k is taken as $\lceil \frac{\log n}{\log \log n} \rceil$ to guarantee, that t_y can code more than n reals. The components of point y then can be found among the values of t_y. Similarly as it was the case in Section 3.1 with equation $(*)$, the requirement that y is a zero of the polynomials defining the QPS instance is expressed as a

condition of the form

$$(**) \qquad \sum_{z_1,\ldots,z_k=0}^{\lfloor \log n \rfloor} g(z_1,\ldots,z_k) \;=\; 0.$$

Here, g is a k-variate trigonometric polynomial of degree $O(\log n)$ defined by means of the input polynomials of the QPS instance. The problem is that $(**)$ is a sum with too many (exponential in $\log n$) terms. Since evaluation of a single term requires to read components in the table for t_y, a straightforward evaluation of the sum leads to too a high query complexity. This problem in the discrete framework has been solved in [21] by a so-called randomized sum-check algorithm. Though the principle ideas can be used, an additional new difficulty in the BSS model again is due to the requirement of segmenting this sum-check. The use of trigonometric polynomials does not allow a direct application of the discrete proof. We abstain from explaining the technical solution of this since it does not increase an overall understanding of ideas. This way it can be shown

Theorem 3.5. *For every problem $L \in \mathrm{NP}_{\mathbb{R}}$ there is a $(\log n, \mathit{poly}\log n)$-restricted real verifier accepting L. The verifier is segmented, i.e., it reads $O(1)$ many blocks of length $\mathit{poly}\log n$ from the proof certificate.*

The final step towards obtaining Theorem 2.11 now is following the discrete roadmap. Since the verifier from the previous theorem is segmented, verifier composition can be applied using it both as outer and inner verifier. For the resulting verifier we apply composition for a second time, again with the verifier from Theorem 3.5 as inner one.[2] A third composition then uses the verifier from Theorem 3.1 as inner one; since its input size has been reduced by the first two compositions far enough, the obtained composed verifier is $(\log n, O(1))$-restricted, thus proving Theorem 2.11.

3.3. **The second proof.** In [13] Dinur presented a new proof of the discrete PCP-theorem. It is based on an close relation between PCPs and gap-producing reductions for so-called Constraint-Satisfaction-Problems CSP. The latter generalizes the well known satisfiability problem 3SAT for propositional formulas in conjunctive normal form. In this problem, an instance consists of a finite collection of clauses which are disjunctions of at most three literals per clause. A literal is a Boolean variable x_i or its negation $\bar{x}_i$. Now the question is to decide whether there exists an assignment $x^* \in \{0,1\}^n$ for the involved propositional variables which makes all clauses true. Note that each such instance easily can be transformed into an (with respect to satisfiability) equivalent 0-1-QPS instance, the reason why the latter is NP-complete as well. A CSP problem now considers general finite alphabets as variable domains (not only the Boolean $\{0,1\}$) and a collection of more general constraints than just clauses. To each such CSP decision problem there corresponds a MAX-CSP optimization problem, similar to the above Example 2.12 for MAX-QPS. It has been well known that if one succeeds in constructing a gap-creating

[2]This additional round is necessary because of parameter reduction. Since the domain for the verifier checking linearity is larger than exponential, we first have to reduce randomization of the outer verifier so far that even a double exponential size of the certificate does not harm.

polynomial time reduction between CSP instances of one particular NP-complete CSP problem, the PCP-theorem would follow. Dinur's ingenious work was precisely doing that. The decisive steps are twofold. In several alternating rounds (not too many) there are amplification steps that significantly increase the gap, i.e., the fraction of unsatisfiable constraints in case a CSP instance is not satisfiable in total. These steps are done at the expense of increasing the cardinality of the underlying finite alphabet of the CSP considered. This cardinality would become too large if amplification is applied sufficiently many times to reach a constant-fraction gap. Therefore, after each amplification step a so-called alphabet reduction step is performed. It worsens respectively decreases the gap a bit, but reduces the alphabet size back to an absolute constant. Alphabet reduction involves the (discrete original of) long transparent proofs that was used to show NP = PCP($poly(n), 1$). The mathematics of Dinur's proof at a first glance seems tailored for the discrete framework. Especially, the amplification step applies combinatorial mathematics with heavy use of so-called expander graphs, a class of well-structured graphs with a lot of applications especially in complexity theory. Moreover, the cardinalities of the finite(!) domains underlying the different CSPs play a crucial role. From all this, it seems unclear whether the techniques could be used to deal with real number problems like QPS. Here, the underlying alphabet is the fixed and uncountable set $\mathbb{R}$. Nevertheless, it turned out that this first impression is false and a proof of Theorem 2.11 along Dinur's ideas is at least easier than the algebraically oriented one outlined in the previous subsection. A key observation is that in order to better adapt the techniques from [13] a more detailed view of the structure of QPS instances is beneficial. This refers to a more subtle way to group equations and variables in polynomial equation systems. In particular, a (single) constraint now can be a collection of polynomial equations that all have to be satisfied in order to fulfill the constraint.

Definition 3.6. Let $m, k, q, s \in \mathbb{N}$. An instance of problem QPS(m, k, q, s) 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., an array ranges over $\mathbb{R}^s$ and the polynomials are maps from $\mathbb{R}^{qs} \to \mathbb{R}$.

Hence, a single constraint in a QPS(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. And the previous QPS problem corresponds to QPS$(m, 1, 3, 1)$ seeing each variable as its own array. 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 bounded by a 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 3.7. A QPS(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(ϕ).

With a gap reduction we mean an algorithm which in polynomial time transforms a QPS(m, k, q, s) instance ϕ into a QPS(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 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 importantly, ϵ is a fixed constant not depending on the size of the given instances.

The following can be established like Example 2.12 and shows the importance of gap-reductions for the $\mathrm{PCP}_{\mathbb{R}}$-theorem:

Lemma 3.8. *Suppose there exists a gap reduction for an* $\mathrm{NP}_{\mathbb{R}}$*-complete QPS*(m, k, q, s) *with a fixed* $\epsilon > 0$*. Then the* $\mathrm{PCP}_{\mathbb{R}}$*-theorem follows.*

We now sketch how the $\mathrm{PCP}_{\mathbb{R}}$-theorem in [5] is proved by designing such a gap reduction. It turns out that the dimension s of variable arrays in QPS(m, k, q, s) instances can play the same role as the alphabet size in discrete CSPs. Thus, after a preprocessing step, a gap reduction once again is obtained by alternating amplification steps with steps decreasing the value of the array dimension. Starting from a QPS$(m, k, 2, s)$ instance ψ_{in} with parameters that constitute an $\mathrm{NP}_{\mathbb{R}}$-complete problem, the single steps produce instances of various other such problems by varying the involved parameters. These problems very roughly are obtained as follows. Instances involving $q = 2$ variable arrays per constraint canonically define a *constraint graph* in which vertices correspond to variable arrays and edges indicate that the two incident arrays participate in the same constraint. The initial instance ψ obtained after preprocessing has a very special constraint graph being a so-called expander, in which a regularity parameter $d \in \mathbb{N}$ is important. Using the properties of expanders amplification constructs another QPS instance $\tilde{\psi}(t)$ (with different parameters) such that $UNSAT(\tilde{\psi}(t)) \geq c(t) \cdot UNSAT(\psi_{in})$ for a suitable $c > 1$ and an additional parameter $t \in \mathbb{N}$ that can be chosen (relatively) arbitrarily. Basic ingredient of the construction of instance $\tilde{\psi}(t)$ is the use of random walks in the expander graph defined by ψ_{in}, just as this is done in [13] using graphs attached to CSP instances. Random walks here are used to define new constraints of a larger dimension depending on the chosen t and the regularity parameter d. Satisfiable initial instances are mapped to satisfiable new ones. More difficult, the structure of expanders guarantees that whenever an unsatisfiable constraint in ψ_{in} is violated under a specific assignment, in the new instance a significantly larger amount of constraints are violated under any assignment. Whereas classically the new CSP instance is defined over alphabets of a larger cardinality, here $\tilde{\psi}(t)$ uses variable arrays of larger size than the s we started with. Since the dimension of the new instance grows with t this amplification step cannot be applied arbitrarily often in a row. The dimension in the end is related to the query complexity of the verifier behind Lemma 3.8 and thus has to remain constant. So in a next round a reduction of the dimension has to be achieved. Analogously to the ongoing in the discrete setting, the (very) long transparent proofs from Section 3.1 can be used to reach this goal. Then, a next round is applied until finally (after $\log m$ rounds) an

instance ψ is obtained whose UNSAT-value is above a given constant. The $\mathrm{PCP}_{\mathbb{R}}$-theorem follows once again, given the above remarks concerning its relation to gap reductions.

An immediate consequence of the tight relation between gap reductions and the $\mathrm{PCP}_{\mathbb{R}}$-theorem is the following negative result concerning the MAX-QPS (and thus also the more general MAX-PSS) optimization problem discussed in the introduction. We mentioned already the easy observation that computing the maximum number of polynomials in such a system that have a common zero is at least as hard as solving $\mathrm{NP}_{\mathbb{R}}$-complete decision problems. A consequence of the $\mathrm{PCP}_{\mathbb{R}}$-theorem shows that even approximating this maximal number arbitrarily well with a polynomial time algorithm is likely impossible. More precisely it holds

Theorem 3.9. *Given a set of real polynomials and an arbitrary $\epsilon > 0$, unless $\mathrm{P}_{\mathbb{R}} = \mathrm{NP}_{\mathbb{R}}$ there is no real number algorithm running in polynomial time in the system's size which approximates the maximal number of polynomials having a common real zero within a relative factor $1 + \epsilon$. The latter means that the ratio of the maximal number of polynomials having a common zero and the algorithm's output is at most $1 + \epsilon$.*

The theorem follows from the existence of a gap reduction for the different variants of the QPS problem discussed above. If such an approximation algorithm would exist, the $\mathrm{NP}_{\mathbb{R}}$-complete version of $\mathrm{QPS}(m, k, q, s)$ used above could be decided efficiently by computing a sufficiently good approximation of the maximum of the related optimization version $\text{MAX-QPS}(m, k, q, s)$. The interested reader might fill in the details her-/himself.

4. Further research and open problems

We finish this survey by mentioning a few more results on related questions and collecting some further problems that seem interesting to us for future investigations.

The algebraic proof of Theorem 2.11 as one major ingredient uses the design of a $\mathrm{PCP}_{\mathbb{R}}$ of proximity for trigonometric polynomials. Of course, one could ask for a lot of other objects in the BSS framework whether there exist property testers and/or $\mathrm{PCP}_{\mathbb{R}}$'s of proximity in the sense of Definition 2.13. For multivariate algebraic polynomials defined on a finite subset of $\mathbb{R}$ this has been done recently:

Theorem 4.1 ([24])**.** *Let $\epsilon > 0$ be fixed; let q be a prime and let $q, k, d \in \mathbb{N}$ be such that $q \in \Omega(\frac{k^2 d^3}{\epsilon^2})$ and $\mathbb{F}_q := \{0, 1, \ldots, q-1\} \subset \mathbb{R}$. There is a $(O(k \log q), O(1))$-restricted $\mathrm{PCP}_{\mathbb{R}}$ of proximity for testing whether a given $f : \mathbb{F}_q^k \mapsto \mathbb{R}$ is an algebraic polynomial of maximal degree d in each of its variables. The verifier gets as input a table of the function values of f together with a proof certificate of length $O((kq)^{O(k)})$. Its running time is polynomial in kq.*

There are several open questions with respect to the proofs of Theorem 2.11. First, the constants hidden behind the different constructed verifiers, such as the length of the required certificate, the amount of randomization and the constant behind the $O(1)$ query complexity of the $\mathrm{PCP}_{\mathbb{R}}$-verifier are huge. Since proof details (in both variants) are quite technical we do not see an obvious way to reduce theses

constants significantly. This holds for both approaches. Another more structural question is whether the use of trigonometric polynomials can be avoided by designing a more direct proof using algebraic ones. Though Theorem 4.1 seems to indicate that this is possible, it is not clear. Its proof heavily relies once again on the use of trigonometric polynomials as coding objects being necessary to design the $\mathrm{PCP}_{\mathbb{R}}$ of proximity. So it does not provide an alternative way to show the $\mathrm{PCP}_{\mathbb{R}}$-theorem. The nearby question then is: Does there exist a more 'direct' $\mathrm{PCP}_{\mathbb{R}}$ of proximity for real algebraic polynomials? Similarly, one can ask for the existence of property testers for the two kind of polynomials, i.e., verifiers that only access a function value table as oracle, but no additional certificate. For algebraic polynomials on certain finite sets in the Turing setting such a test has been developed in [14], but with non-constant query complexity. More generally, it seems interesting to find more function classes or properties, respectively, for which property tests or $\mathrm{PCP}_{\mathbb{R}}$'s of proximity can be constructed in the BSS model. These problems are also closely related to the field of software testing, i.e., the question of checking whether a given BSS-program fulfills the computational task it was designed for, f.e., computing a polynomial of a certain structure.

Coming back to the proofs of Theorem 2.11 and the question for easier verification algorithms we remark that in the classical PCP literature, the important initial role played by segmented verifiers has been subsumed by that of so-called robust verifiers, see [9] and also [17]. Since the main technical problem to overcome is segmentation, it is of course interesting to ask whether robust verifiers in the real-number framework would lead to easier proofs as well, including a possibility to avoid trigonometric polynomials. Another impact of classical robust verifiers is to obtain property testers instead of PCPs of proximity. Once again we can ask whether this would be possible both for algebraic and trigonometric polynomials? We do not have an educated guess at the moment.

Theorem 3.9 reminds a class of problems that has been studied intensively in discrete complexity theory, namely the approximation of (combinatorial) optimization problems. Of course, in our setting there is an unusual mix of the uncountable underlying alphabet $\mathbb{R}$ and a discrete value to optimize in form of the number of commonly satisfiable constraints. Given the huge importance of approximating combinatorial problems in classical complexity theory it might be interesting to investigate whether a similar theory can be developed here.

As a final area related to the concepts presented here we mention a generalization of PCPs. In the above setting, the verifier inspects a certificate presented to it performing a randomized algorithm. As a more complex framework one can also allow a kind of communication between the verifier and a so-called prover which is assumed to have unlimited computational power. For a PCP, the prover would just present the certificate to the verifier and then does not interfere further. In *interactive proofs* there are several rounds of communication and computation between the verifier and the prover. As before, the verifier performs random computations and generates questions for the prover which the latter then answers. The verifier continues the computation using the given answers. The process is repeated until finally the verifier has to decide whether it wants to accept this interactive communication as proof for a statement or not. Like for the $\mathrm{PCP}_{\mathbb{R}}$-classes further conditions

are imposed on the probabilities of accepting and rejecting computations. This way, in the Turing model one obtains a class of decision problems denoted by IP for interactive proofs and an analogue class $\mathrm{IP}_{\mathbb{R}}$ in the BSS model. The languages that can be accepted by such a protocol most likely build a much larger class than NP. A famous theorem by Shamir [27] characterizes IP as PSPACE, the class of all decision problems decidable by a Turing algorithm that uses polynomial space. The latter is conjectured to be much larger than NP. In the BSS setting the first study of such interactive protocols was done in [19] for a restricted class of additive BSS-algorithms. In [7] a class $\mathrm{IP}_{\mathbb{R}}$ for the (full) BSS model was studied and it was shown that it contains a class of real number decision problems conjectured to be much larger than $\mathrm{NP}_{\mathbb{R}}$. Note, however, that space resources alone don't make much sense in the BSS setting due to certain easy fundamental coding abilities [25]. As a consequence, in the real BSS model there are several (provably different) complexity classes that can be seen as real number substitutes for the discrete class PSPACE. The main result in [7] shows that the class $\mathrm{IP}_{\mathbb{R}}$ is strictly larger than one of those classes and contained in another. One open question now is to find a characterization of $\mathrm{IP}_{\mathbb{R}}$ by a natural other complexity class, a second is to figure out whether $\mathrm{IP}_{\mathbb{R}}$ is closed under complementation, a property the discrete class IP has as consequence of Shamir's theorem.

Acknowledgement

My thanks go to an anonymous referee for helpful comments.

References

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

[2] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy, *Proof verification and hardness of approximation problems*, Journal of the ACM **45** (1998), 501–555. Preliminary version: Proc. 33rd Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, 1992, pp. 14–23.

[3] S. Arora and S. Safra, *Probabilistic checking proofs: A new characterization of NP,* Journal of the ACM **45** (1998), 70–122. Preliminary version: Proc. of the 33rd Annual IEEE Symposium on the Foundations of Computer Science, 1992, pp. 2–13.

[4] M. Baartse and K. Meer, *Topics in real and complex number complexity theory*, in: Recent Advances in Real Complexity and Computation, Contemporary Mathematics vol. 604, J. L. Montaña, L. M. Pardo (eds.), American Mathematical Society, 2013, pp. 1–53.

[5] M. Baartse and K. Meer, *The PCP theorem for NP over the reals*, Foundations of Computational Mathematics, **15** (2015), 651–680.

[6] M. Baartse and K. Meer, *An algebraic proof of the real number PCP theorem*, Journal of Complexity **40** (2017), 34–77.

[7] M. Baartse and K. Meer, *Interactive proofs and a Shamir-like result for real number computations*, Computational Complexity **28** (2019), 437–469.

[8] R. Benedetti and J. J. Risler, *Real Algebraic and Semi-algebraic Sets*, Hermann, 1990.

[9] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan and S. P. Vadhan, *Robust PCPs of Proximity, Shorter PCPs, and Applications to Coding*, SIAM J. Comput. **36** (2006), 889–974.

[10] L. Blum, F. Cucker, M. Shub and S. Smale, *Complexity and Real Computation*, Springer, 1998.

[11] M. Blum, M. Luby and R. Rubinfeld, *Self-testing/correcting with applications to numerical problems*, J. Comput. Syst. Sci. **47** (1993), 549–595.

[12] L. Blum, M. Shub and S. Smale, *On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines*, Bull. Amer. Math. Soc. **21** (1989), 1–46.
[13] I. Dinur, *The PCP theorem by gap amplification*, Journal of the ACM **54** (2007).
[14] K. Friedl, Z. Hátsági and A. Shen, *Low-degree tests*, in: Proc. SODA, 1994, pp. 57–64.
[15] O. Goldreich, *Introduction into Property Testing*, Cambridge Univ. Press, 2017.
[16] D. Y. Grigoriev, *Complexity of deciding Tarski algebra*, Journal of Symbolic Computation **5** (1988), 65–108.
[17] P. Harsha, *Robust PCPs of Proximity and Shorter PCPs*, Massachusetts Institute of Technology, Cambridge, MA, USA, 2004. http://www.tcs.tifr.res.in/~prahladh/papers/thesis/
[18] J. Heintz, M. F. Roy and P. Solerno, *Sur la complexité du principe de Tarski-Seidenberg*, Bull. Soc. Math. France **118** (1990), 101–126.
[19] S. Ivanov and M. de Rougemont, *Interactive Protocols on the reals*, Computational Complexity, **8** (1999), 330–345.
[20] R. Karp, *Reducibility among combinatorial problems*, in: Complexity of Computer Computations, J.W. Thatcher, R.E. Miller (eds.), Plenum Press New York, 1972, pp. 85–103.
[21] C. Lund, L. Fortnow, H. Karloff and N. Nisan, *Algebraic methods for interactive proof systems*, Journal of the ACM **39** (1992), 859–868.
[22] K. Meer, *Transparent long proofs: A first PCP theorem for $NP_{\mathbb{R}}$*, Foundations of Computational Mathematics **5** (2005), 231–255.
[23] K. Meer, *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, vol. 6914, 2011, pp. 41–52.
[24] K. Meer, *A PCP of proximity for real algebraic polynomials*, in: Proc. 16th International Computer Science Symposium in Russia CSR 2021, Sochi, R. Santhanam and D. Musatov (eds.), Springer LNCS 12730, 2021, pp. 264–282.
[25] C. Michaux, *Une remarque à propos des machines sur $\mathbb{R}$ introduites par Blum, Shub et Smale*, C.R. Acad. Sci. Paris, t. 309, série I (1989), 435–437.
[26] J. Renegar, *On the computational complexity and geometry of the first-order theory of the reals, I - III*, Journal of Symbolic Computation **13** (1992), 255–352.
[27] A. Shamir, *IP = PSPACE*, Journal of the ACM **39** (1992), 869–877.
[28] S. Smale, *Mathematical problems for the next century*, in: V.I. Arnold, M. Atiyah, P. Lax, B. Mazur, B. (eds.). Mathematics: Frontiers and Perspectives. American Mathematical Society, 1999, pp. 271–294.
[29] A. Tarski, *A Decision Method for Elementary Algebra and Geometry*, 2nd edition, Univ. Calif. Press, Berkeley, 1951.
[30] J. F. Traub and H. Woźniakowski, *Complexity of linear programming*, Operations Research Letters **1** (1982), 59–62.

Manuscript received October 5 2021
revised February 17 2022

K. Meer
Computer Science Institute, BTU Cottbus-Senftenberg, Platz der Deutschen Einheit 1 D-03046 Cottbus, Germany
E-mail address: meer@b-tu.de