Karlsruhe Institute of Technology

Defining the Discrete Real Polynomial Hierarchy with Oracle Machines

Bachelor's Thesis of

Illia Minkin

At the Department of Computer Science
Institute of Theoretical Computer Science

Karlsruhe Institute of Technology

Reviewer:	PD Dr. Torsten Ueckerdt
Second reviewer:	T.T.-Prof. Dr. Thomas Bläsius
Advisor:	Paul Jungeblut

November 1, 2023 – March 1, 2024

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

Statutory Declaration

I hereby declare that I have developed and written the enclosed thesis completely by myself. I have not used any other than the aids that I have mentioned. I have marked all parts of the thesis that I have included from referenced literature, either in their original wording or paraphrasing their contents. I have followed the by-laws to implement scientific integrity at KIT.

Karlsruhe, March 1, 2024

.....

(Illia Minkin)

Abstract

The importance of the recently introduced complexity class $\exists\mathbb{R}$ has increased significantly because of its relevance in computational geometry. The *discrete real polynomial hierarchy* **DRPH** is a generalization of $\exists\mathbb{R}$ — similarly to the classical polynomial hierarchy, which is usually defined with oracle Turing machines. Conversely, the discrete real polynomial hierarchy is typically defined through complete problems.

We would like to use the oracle notation for the **DRPH** as well, as not only is it convenient, but also reflects the intuition behind the structure of the problems contained in the corresponding complexity classes. However, to do so, we need a suitable computational model. The goal of this thesis is to find a computational model that can be equipped with suitable oracle sets and to define the **DRPH** using it.

We review relevant results from classical complexity theory, followed by an introduction to BSS and real random access (oracle) machines. Then, we familiarize the reader with the existential theory of the reals and the corresponding class $\exists\mathbb{R}$, as well as its definition via the BSS and the real RAM models. Finally, we bring the (discrete) real polynomial hierarchy in and define it using BSS oracle machines, which is a central result of the thesis. Furthermore, we show how **DRPH** can be defined with the (oracle) real RAM. Additionally, we provide some discussion on the definition of the zeroth level of **DRPH**.

Zusammenfassung

Die kürzlich eingeführte Komplexitätsklasse $\exists\mathbb{R}$ spielt eine wichtige Rolle in der algorithmischen Geometrie. Diese Klasse kann ähnlich zur polynomiellen Hierarchie verallgemeinert werden – dadurch erhält man die sogenannte *diskrete reelle polynomielle Hierarchie* **DRPH**. Im Gegensatz zur klassischen Polynomialzeithierarchie, die üblicherweise mithilfe von Orakel-Turingmaschinen definiert wird, wird **DRPH** klassischerweise über vollständige Probleme eingeführt.

Die „Orakel-Notation“ ist nicht nur intuitiv, sondern spiegelt die Struktur von Problemen in den entsprechenden Komplexitätsklassen wider. Es ist erstrebenswert, diese Notation auch für **DRPH** zu verwenden, allerdings ist solch eine Notation ohne ein passendes Berechnungsmodell nicht wohldefiniert. Das Ziel dieser Arbeit ist es, ein passendes Orakel-Berechnungsmodell zu finden und **DRPH** damit zu definieren.

Wir fassen relevante Ergebnisse aus der klassischen Komplexitätstheorie zusammen und führen die BSS sowie die reellen RA (*random access*) Maschinen ein. Danach stellen wir die existentielle Theorie der reellen Zahlen sowie die entsprechende Komplexitätsklasse $\exists\mathbb{R}$ vor. Schlussendlich führen wir die diskrete reelle polynomielle Hierarchie ein und definieren diese mithilfe von Orakel BSS und reellen RA Maschinen – dies ist ein zentrales Ergebnis der Thesis. Überdies diskutieren wir, wie die nullte Stufe von **DRPH** definiert werden kann.

Contents

1 Introduction **1**
- 1.1 Related Work . . . 2
- 1.2 Outline . . . 3

2 Preliminaries **5**
- 2.1 The Classes **NP** and **coNP** . . . 5
- 2.2 The Classes $\mathbf{\Sigma}_2^p$ and $\mathbf{\Pi}_2^p$ . . . 6
- 2.3 The Polynomial Hierarchy . . . 8
- 2.4 Alternative Definitions of **PH** . . . 10
  - 2.4.1 Complete Problems for $\mathbf{\Sigma}_i^p$ and $\mathbf{\Pi}_i^p$ . . . 10
  - 2.4.2 Oracle Machines . . . 11
- 2.5 The Blum–Shub–Smale Model . . . 15
- 2.6 The Real RAM Model . . . 17
  - 2.6.1 Definition of the Real RAM . . . 17
  - 2.6.2 Equipping the Real RAM with Oracles and Nondeterminism . . . 20

3 The Existential Theory of the Reals and the Class $\exists\mathbb{R}$ **21**
- 3.1 The Existential Theory of the Reals . . . 21
- 3.2 The Class $\exists\mathbb{R}$ and $\exists\mathbb{R}$-completeness . . . 22
- 3.3 Special Cases of ETR . . . 23
- 3.4 Definition via BSS machines . . . 26
  - 3.4.1 Complexity Classes Over $\mathbb{R}$ . . . 26
  - 3.4.2 An Analog of the Cook–Levin Theorem for $\mathbf{NP}_{\mathbb{R}}$ . . . 27
  - 3.4.3 Defining $\exists\mathbb{R}$ . . . 32
- 3.5 Definition via Real RAM . . . 32

4 The (Discrete) Real Polynomial Hierarchy **35**
- 4.1 The Universal Theory of the Reals and the Class $\forall\mathbb{R}$ . . . 35
- 4.2 Extensions of ETR and UTR as Complete Problems . . . 36
- 4.3 Properties of **DRPH** . . . 36
- 4.4 Generalization of the Feasibility Problem . . . 38
- 4.5 The Real Polynomial Hierarchy . . . 40
- 4.6 Corresponding Versions of the Real (In-)Feasibility Problem . . . 41
- 4.7 Definition of (**D**)**RPH** Using Oracle BSS Machines . . . 43
- 4.8 Definition of **DRPH** with the (Oracle) Real RAM . . . 46
- 4.9 The Zeroth Level of the Hierarchy . . . 47

5 Conclusion **51**
- 5.1 Future Work . . . 51

Bibliography **53**

1. Introduction

About a decade ago, Schaefer and Štefankovič [SŠ17] introduced a new complexity class $\exists\mathbb{R}$ (pronounced "exists R" or "ER"), which has recently received extensive publicity because a lot of known problems from computational geometry — and even training neural networks — turned out to be $\exists\mathbb{R}$-complete. A known complete problem for $\exists\mathbb{R}$ is the existential theory of the reals (ETR for short), a problem where a system of polynomial equations and inequalities with bounded integer coefficients and real-valued variables is given. The question is whether the given system is solvable, that is to say, whether there exists an assignment for the variables satisfying the constraints. In fact, the complexity class $\exists\mathbb{R}$ is conventionally defined by making ETR a complete problem. The existential theory of the reals can be thought of as the real analog of the Boolean satisfiability problem Sat: the problem structure remains the same, but the real variables and more connectives and operators are allowed. Similarly, the class $\exists\mathbb{R}$ can be thought of as the real analog of **NP** — **NP** is even denoted as $\exists\mathbf{P}$ in some literature [ODo17].

The classical complexity classes **P**, **NP**, and **coNP** have been studied widely [AB06]. The polynomial hierarchy, which is a generalization of these classes, has been extensively explored as well [AB06]. Similarly to the "classical" polynomial hierarchy, the complexity class $\exists\mathbb{R}$ can be generalized by adding alternating quantifiers to obtain the discrete real polynomial hierarchy **DRPH**. For instance, the canonical complete problem EUTR (existential-universal theory of the reals) for the class $\exists\forall\mathbb{R}$ from the second level of **DRPH** has the following form. Just like in ETR, we are given a system of polynomial equations and inequalities; but now there are two blocks of variables: the first one is quantified existentially and the second one universally. This means that we need to find an assignment for the variables from the existential block such that the given system holds for *any* assignment of the variables from the universal block.

The "classical" polynomial hierarchy **PH** can be defined via canonical complete problems, which are analogous modifications of the Sat problem. However, the usual way to define **PH** is using so-called oracle Turing machines. An oracle Turing machine is a Turing machine with an extra "power": it has an additional oracle tape and an oracle instruction. An oracle is an efficient — that is, working in constant (!) time — black-box solver for some decision problem, e.g., the Boolean satisfiability problem. The machine can write a Sat instance onto the oracle tape and, when the oracle instruction is called, the oracle gives an answer whether it is a yes-instance in constant time.

When defining complexity classes with an oracle computational model, one usually uses the superscript notation. For instance, the second level of the polynomial hierarchy $\mathbf{\Sigma}_2^p$ becomes $\mathbf{NP}^{\mathbf{NP}}$ (or $\mathbf{NP}^{\text{Sat}}$ because Sat is **NP**-complete and can technically be replaced by any other **NP**-complete problem since they are computationally equivalent up to a polynomial-time reduction). $\mathbf{NP}^{\mathbf{NP}}$ is the complexity class containing all

languages (or decision problems) decidable in polynomial time by a nondeterministic Turing machine with access to an **NP**-complete oracle. It is crucial to note that adding oracles is an operation on *machines*, *not on languages*.

We would like to write something like $\exists\mathbb{R}^{\forall\mathbb{R}}$, in particular, because some problems lie in so-called hybrid complexity classes, where some quantifiers are Boolean [SŠ23]. For example, a problem from evolutionary game theory lies in $\exists\forall\mathbb{R}$, but we do not need real variables in the existential quantifier block — to indicate this, one could write $\mathbf{NP}^{\forall\mathbb{R}}$ [BH22]. There is another example of a problem lying in $\exists\forall\exists\mathbb{R}$, but the second quantifier is actually Boolean, meaning that we could write $\exists\mathbb{R}^{\mathbf{coNP}^{\exists\mathbb{R}}}$ [SŠ23]. However, without a suitable oracle model for $\exists\mathbb{R}$, this notation is not well-defined, and "the details of [such model] would still need to be worked out" [SŠ23]. The primary goal of this thesis is to develop a suitable oracle model for defining the discrete real polynomial hierarchy.

Furthermore, developing such a model might be relevant for the relativization barrier of the **P** vs. **NP** question, or rather its real counterpart. Baker, Gill and Solovay showed the existence of oracle sets A and B such that $\mathbf{P}^A = \mathbf{NP}^A$ and $\mathbf{P}^B \neq \mathbf{NP}^B$ [BGS75]. This means that a diagonalization argument would not be enough for proving $\mathbf{P} \neq \mathbf{NP}$, as it would also be applicable to the relativized problem $\mathbf{P}^C \neq \mathbf{NP}^C$ for an arbitrary oracle set C [BGS75]. Blum, Shub and Smale introduced complexity classes $\mathbf{P}_\mathbb{R}$ and $\mathbf{NP}_\mathbb{R}$ over the real numbers; the question $\mathbf{P}_\mathbb{R} \stackrel{?}{=} \mathbf{NP}_\mathbb{R}$ remains open [BSS89]. Having such an oracle model would allow for transferring the relativization argument to such questions as $\mathbf{NP} \stackrel{?}{=} \exists\mathbb{R}$, $\exists\mathbb{R} \stackrel{?}{=} \mathbf{PSPACE}$ or $\exists\forall\mathbb{R} \stackrel{?}{=} \mathbf{PSPACE}$.

1.1. Related Work

In 1989, Blum, Shub and Smale introduced an extension of classical Turing machines able to perform computations over an arbitrary field [BSS89]. They define complexity classes $\mathbf{P}_\mathbb{R}$ and $\mathbf{NP}_\mathbb{R}$ over the reals using this computational model and extend them to an "unrestricted" real polynomial hierarchy. Yet, the BSS model is too powerful because it is allowed to work with real-valued constants, which are forbidden in the theory of the reals (e.g., it is clear that $\exists\mathbb{R} \subseteq \mathbf{NP}_\mathbb{R}$, the reverse is not the case). Hence, to be able to define the discrete real polynomial hierarchy, one needs to restrict the BSS model additionally.

In their further work with Felipe Cucker from 1998 [BCSS98], they introduce an oracle model for additive machines — BSS machines that are only allowed to perform addition and subtraction. Furthermore, they define the additive hierarchy using these restricted machines and even provide its definition using oracle additive machines. They note that the proof can be applied to the unrestricted real polynomial hierarchy as well. What is more, they show the existence of complete problems for the levels of the unrestricted hierarchy, however, the proof is only roughly sketched. Nevertheless, to define the discrete real polynomial hierarchy, one would still need to transfer the proof to the unrestricted hierarchy first and then discretize this oracle model, so that it defines the desired complexity classes.

Oracle BSS machines are used in some later papers, but rather for algebraic and number theoretic results [MZ05] or studying the real halting problem [CKM10] — no further work in developing this computational model for the definition of **DRPH** has been done.

Erickson, van der Hoog and Miltzow study the real RAM model and show that a decision problem is in $\exists\mathbb{R}$ if and only if there is a polynomial-time verification algorithm on a real RAM [EvM19]. However, it is not mentioned whether this model can be used for higher levels of **DRPH**; moreover, it is not straightforward how the real RAM can be equipped with an oracle.

1.2. Outline

We begin by summarizing relevant results from classical complexity theory in Chapter 2, such as the polynomial hierarchy and its definition via oracle Turing machines. Then we introduce the BSS model, an extension of classical Turing machines performing computations with real numbers. Furthermore, we make the reader acquainted with the real RAM model — another suited candidate for computations with real numbers, which is an extension of the word RAM.

Chapter 3 introduces the existential theory of the reals (ETR), which can be thought of as a real analog of Sat. The corresponding complexity class $\exists\mathbb{R}$ is introduced; we also provide its alternative definitions using the BSS and the real RAM model.

In Chapter 4, we generalize the complexity class $\exists\mathbb{R}$ to obtain the (discrete) real polynomial hierarchy — the "discreteness" depends on whether one allows or forbids real-valued constants. Moreover, we define these hierarchies using BSS machines and the real RAM. At long last, we define the (discrete) real polynomial hierarchy with oracle BSS and real random access machines, which is a central result of the thesis. Additionally, we discuss how the zeroth level of the discrete real polynomial hierarchy can be reasonably defined.

2. Preliminaries

To define the real polynomial hierarchy, we first need to get acquainted with its discrete cousin, which is an extension of the well-known complexity classes **P**, **NP**, and **coNP**. In this chapter, we recall a couple of definitions from classical complexity theory and introduce the polynomial hierarchy using various definitions. We follow the notation of [AB06], as it is considered to be the standard work on theory of computation. As for prerequisites, the reader is expected to be familiar with basic notions of theoretical computer science, such as Turing machines and formal languages.

2.1. The Classes NP and coNP

The class **NP** is defined as the set of all problems with an efficiently — that is, in polynomial time — verifiable solution. This can be formalized in the following definition.

Definition 2.1 (Complexity class **NP**)**.** *Let $L \subseteq \{0,1\}^*$. Then $L \in \mathbf{NP}$ if there exists a polynomial time Turing machine M and a polynomial p, such that for all $x \in \{0,1\}^*$:*

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} : \ M(x,u) = 1.$$

In that case, u is said to be a *certificate* (or a *witness*) for x (with respect to the language L and machine M). Formulated in words, this means that for every word x in the language L, there must exist a certificate u (one can think of it as a solution to the instance x of the problem L), such that Turing machine M accepts the tuple (x, u).

An equivalent definition can be given using nondeterministic Turing Machines.

Definition 2.2 (Complexity class **NP**, alternative definition)**.** **NP** *is the set of all languages decidable by a nondeterministic Turing Machine (NTM) in polynomial time.*

The equivalence follows since a solution can be guessed using the nondeterminism and then verified by the deterministic part. In fact, it was the original definition — **NP** stands for *nondeterministic polynomial time*.

Another way to define this class is via a well-known **NP**-complete problem, the Boolean satisfiability problem (SAT).

Definition 2.3 (Complexity class **NP**, second alternative definition)**.** **NP** *is the set of all problems reducible to SAT in polynomial time.*

$$\mathbf{NP} := \{L \subseteq \{0,1\}^* \mid L \leq_p \textsc{Sat}\}.$$

A problem is said to be **NP**-hard if every problem in **NP** can be reduced to it in polynomial time. **NP**-complete problems are the hardest ones in **NP** — they are the intersection of **NP**-hard problems and of those in **NP**. The Cook–Levin theorem states that SAT is **NP**-complete. This is a central result in the complexity theory, which

facilitates showing the **NP**-completeness of other problems, since we only need to reduce a SAT-instance to an instance of the given problem. It is thus not hard to see that **NP** can be defined as the set of all problems reducible to SAT in polynomial time. This definition is rather unusual but is quite handy when working with the polynomial hierarchy.

Definition 2.4 (Complexity class **coNP**)**.** *The class* **coNP** *is defined as the set of all languages having their complement in* **NP***.*

$$\mathbf{coNP} := \{L \subseteq \{0,1\}^* \mid \overline{L} \in \mathbf{NP}\}.$$

Alternatively, this class can be defined analogously to **NP**, using Turing machines.

Definition 2.5 (Complexity class **coNP**, alternative definition)**.** *Let* $L \subseteq \{0,1\}^*$*. Then* $L \in$ **coNP** *if there exists a polynomial time Turing machine* M *and a polynomial* p*, such that for all* $x \in \{0,1\}^*$*:*

$$x \in L \iff \forall u \in \{0,1\}^{p(|x|)} : \; M(x,u) = 1.$$

The only difference is that the universal quantifier replaced the existential one. The latter definition is often preferred because it is less prone to the misconception of $\overline{\text{SAT}} = \{\varphi \mid \varphi \text{ is not satisfiable}\}$ being in **NP**, which is presumably *not* the case. It is also important to note that **coNP** is *not* the complement of **NP** — in fact, the intersection of the both classes is non-empty since it contains **P**.

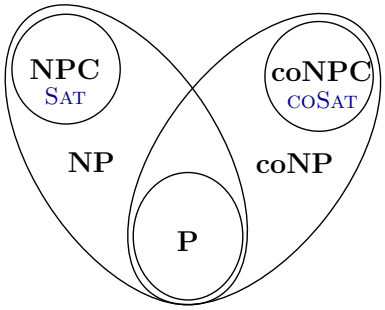

Figure 2.1: The connection between **P**, **NP**, **coNP**, **NP**- and **coNP**-complete classes.

2.2. The Classes Σ_2^p and Π_2^p

It is intuitively clear that some **NP**-hard problems are more complex than others. Recall the CLIQUE problem:

$$\text{CLIQUE} = \{\langle G, k\rangle \mid \text{graph } G \text{ has a clique of size at least } k\}.$$

This problem is known to be **NP**-complete. Now, let us enlarge upon a slightly modified version of this problem, called EXACTCLIQUE.

$$\text{EXACTCLIQUE} = \{\langle G, k\rangle \mid G \text{ is graph with the largest clique of size exactly } k\}.$$

As the name suggests, the clique size must now be *exactly* k. To grasp that this version is indeed harder, we formulate it in a slightly different way:

$$\langle G, k\rangle \in \textsc{ExactClique}$$
$$\iff \exists S \subseteq V_G, |S| = k : S \text{ is clique and } \forall S' \subseteq V_G, |S'| = k+1 : S' \text{ is not a clique.}$$

Now there is no short certificate for **NP**-membership, since we would not only need to check that S is a clique of size k, but also to make sure that *any* other vertex subset S' of larger cardinality is not a clique. Checking the subsets of size $k+1$ is sufficient because a subset of a clique is again a clique, so the check would imply that there are no larger cliques.

The polynomial hierarchy tries to distinguish between the hardness of the problems. The introduced classes **NP** and **coNP** form the first level of the hierarchy. We now generalize those classes, beginning with the second level.

Definition 2.6 (Complexity class $\mathbf{\Sigma}_2^p$)**.** *Let $L \subseteq \{0,1\}^*$. Then $L \in \mathbf{\Sigma}_2^p$ if there exists a polynomial time Turing machine M and a polynomial p, such that for all $x \in \{0,1\}^*$:*

$$x \in L \iff \exists u_1 \in \{0,1\}^{p(|x|)}\ \forall u_2 \in \{0,1\}^{p(|x|)} :\ M(x, u_1, u_2) = 1.$$

Definition 2.7 (Complexity class $\mathbf{\Pi}_2^p$)**.** *Let $L \subseteq \{0,1\}^*$. Then $L \in \mathbf{\Pi}_2^p$ if there exists a polynomial time Turing machine M and a polynomial p, such that for all $x \in \{0,1\}^*$:*

$$x \in L \iff \forall u_1 \in \{0,1\}^{p(|x|)}\ \exists u_2 \in \{0,1\}^{p(|x|)} :\ M(x, u_1, u_2) = 1.$$

Observation 2.8. $\mathbf{\Pi}_2^p = \mathbf{co\Sigma}_2^p$.

The observation follows because the definition of $\mathbf{\Pi}_2^p$ is the negation of the $\mathbf{\Sigma}_2^p$ definition statement (strictly speaking, one would need to redefine the acceptance behavior of the TM in the definition of $\mathbf{\Pi}_2^p$). The only difference in the definitions is therefore the order of the quantifiers. Furthermore, if one compares the definitions to those of **NP** and **coNP**, one can notice that a universal and an existential quantifier were added, respectively. This suggests that adding quantifiers means going up one level in the hierarchy, thereby making problems harder.

Claim 2.9. *ExactClique* $\in \mathbf{\Sigma}_2^p$.

Proof sketch. We need TM M and polynomial p satisfying the definition of $\mathbf{\Sigma}_2^p$. Choose $p(n) = n$. Let $x \in \{0,1\}^*$. Define M as follows.

Algorithm 2.1: Turing machine for ExactClique [ODo17].

Input: $x \in \{0,1\}^*$; encodings of G subgraphs $u_1, u_2 \in \{0,1\}^{p(|x|)}$.
Output: 1 if M accepts, 0 otherwise.

1 **if** $x \neq \langle G, k\rangle$, REJECT
2 **if** u_1 is not a k-clique in G, REJECT
3 **if** u_2 corresponds to a $(k+1)$-clique in G, REJECT
4 ACCEPT

If $x \in L$, then M accepts for all u_2, when u_1 corresponds to a k-clique — and such u_1 exists because x is a yes-instance. If $x \notin L$, we consider three possible cases.

Case 1: $x \neq \langle G, k\rangle$, that is, x is not a valid encoding of an ExactClique-instance. Then the algorithm always rejects in line 1, independently from u_1 and u_2.

Case 2: The size of the largest clique is greater than k. Then for all u_1, there exists an encoding of a $(k+1)$-clique u_2, which leads to rejection in the third step.

Case 3: The size of the largest clique is less than k. Then independently from the choice of u_1, it cannot be an encoding of a k-clique. Hence, the Turing machine rejects in the second step. ❑

Now consider the circuit minimization problem

$$\textsc{SmallestCircuit} = \{\langle C\rangle \mid C \text{ is the smallest circuit computing } C_f\},$$

where C_f denotes the function that C computes. Similarly, the problem can be reformulated using quantifiers:

$$\langle C\rangle \in \textsc{SmallestCircuit} \iff \forall C', \text{size}(C') < \text{size}(C) : \exists x : C_f(x) \neq C'_f(x),$$

where the size of a circuit is the number of (non-input) gates it contains. There is now a "$\forall\exists$"-pattern, suggesting that the problem lies in $\mathbf{\Pi}_2^p$. One can indeed construct a Turing machine satisfying the definition of this complexity class, — following the idea of Algorithm 2.1 — thereby showing that $\textsc{SmallestCircuit} \in \mathbf{\Pi}_2^p$.

To show that the intersection of $\mathbf{\Sigma}_2^p$ and $\mathbf{\Pi}_2^p$ is non-empty, let us consider another decision problem from graph theory.

$$\textsc{ExactIndSet} = \{\langle G, k\rangle \mid \text{the largest independent set in } G \text{ has size exactly } k\}.$$

On the one hand, it is clear that

$$\langle G, k\rangle \in \textsc{ExactIndSet} \iff \exists S \subseteq V_G, |S| = k : S \text{ is an independent set and } \forall S' \subseteq V_G, |S'| = k+1 : S' \text{ is not an independent set.}$$

Hence, $\textsc{ExactIndSet} \in \mathbf{\Sigma}_2^p$. On the other hand, a pair $\langle G, k\rangle$ is in $\textsc{ExactIndSet}$ if and only if

$$\forall S' \subseteq V_G, |S'| \geq k+1 : S' \text{ is not an independent set, but } \exists S \subseteq V_G, |S| = k : \text{S is an independent set.}$$

I.e., we can change the order of the quantifiers. This formulation implies that $\textsc{ExactIndSet}$ lies in $\mathbf{\Pi}_2^p$.

2.3. The Polynomial Hierarchy

The introduced classes can be generalized to an arbitrary (but finite) number of quantifiers and a polynomial-time computable predicate. The number of quantifiers defines the hierarchy level.

Definition 2.10 (Polynomial hierarchy)**.** *Let $i \in \mathbb{N}, L \subseteq \{0,1\}^*$. $L \in \mathbf{\Sigma}_i^p$ if there exists a polynomial time TM M and a polynomial p such that:*

$$x \in L \Leftrightarrow \exists u_1 \in \{0,1\}^{p(|x|)}\, \forall u_2 \in \{0,1\}^{p(|x|)} \ldots Q_i u_i \in \{0,1\}^{p(|x|)} : M(x, u_1, \ldots, u_i) = 1,$$

where $Q_i = \begin{cases} \forall, & i \text{ is even,} \\ \exists, & i \text{ is odd.} \end{cases}$

Similarly, $L \in \mathbf{\Pi}_i^p$ if there exists a polynomial time TM M and a polynomial p such that:

$$x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{p(|x|)}\ \exists u_2 \in \{0,1\}^{p(|x|)} \dots Q_i u_i \in \{0,1\}^{p(|x|)} : M(x, u_1, \dots, u_i) = 1,$$

$$\text{where } Q_i = \begin{cases} \exists, & i \text{ is even,} \\ \forall, & i \text{ is odd.} \end{cases}$$

*The **polynomial hierarchy** is the union over the hierarchy levels ($\mathbf{\Sigma}_0^p = \mathbf{\Pi}_0^p = \mathbf{P}$):*

$$\mathbf{PH} := \bigcup_{i \in \mathbb{N}_0} \mathbf{\Sigma}_i^p.$$

Observation 2.11. *Note that for $i \in \mathbb{N}$:*

(i) $\mathbf{\Sigma}_1^p = \mathbf{NP}$ *and* $\mathbf{\Pi}_1^p = \mathbf{coNP}$,

(ii) $\mathbf{\Pi}_i^p = \mathbf{co\Sigma}_i^p$,

(iii) $\mathbf{\Sigma}_i^p \subseteq \mathbf{\Sigma}_{i+1}^p$ *and* $\mathbf{\Sigma}_i^p \subseteq \mathbf{\Pi}_{i+1}^p$,

(iv) $\mathbf{\Pi}_i^p \subseteq \mathbf{\Pi}_{i+1}^p$ *and* $\mathbf{\Pi}_i^p \subseteq \mathbf{\Sigma}_{i+1}^p$,

(v) $\mathbf{PH} = \bigcup_{i \in \mathbb{N}_0} \mathbf{\Pi}_i^p = \bigcup_{i \in \mathbb{N}_0} \mathbf{\Sigma}_i^p$.

We introduce some notation that reflects the intuition behind the definitions of the hierarchy levels, as well as behind the proofs of some properties of **PH**.

Notation [ODo17]**.**

$\mathbf{\Sigma}_0^p = \mathbf{\Pi}_0^p = \mathbf{P}$;

$\mathbf{\Sigma}_1^p = \mathbf{NP} = \exists\mathbf{P}$,	$\mathbf{\Pi}_1^p = \mathbf{coNP} = \forall\mathbf{P}$;
$\mathbf{\Sigma}_2^p = \exists\forall\mathbf{P}$,	$\mathbf{\Pi}_2^p = \forall\exists\mathbf{P}$;
$\mathbf{\Sigma}_3^p = \exists\forall\exists\mathbf{P}$,	$\mathbf{\Pi}_3^p = \forall\exists\forall\mathbf{P}$;
$\dots$	$\dots$
$\mathbf{\Sigma}_i^p = \exists\mathbf{\Pi}_{i-1}^p$,	$\mathbf{\Pi}_i^p = \forall\mathbf{\Sigma}_{i-1}^p$.

The notation will be useful when establishing the connection to the real polynomial hierarchy. It also helps to understand Observation 2.11 (iii) and (iv) better. Adding more quantifiers gives us more power since one can always ignore one "dummy" quantifier, thereby going down one level in the hierarchy.

Figure 2.2 illustrates the structure of the polynomial hierarchy. Note that the hierarchy resides within **PSPACE**, since one can reuse the space for "verifying the quantifiers" and we are dealing with a finite number thereof.

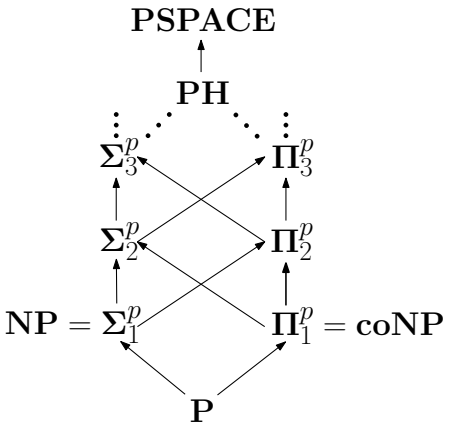

Figure 2.2: The polynomial hierarchy. The arrows denote inclusion.

2.4. Alternative Definitions of PH

In this section, we consider two alternative possibilities to define the polynomial hierarchy. It can be done with the help of complete problems for each level or using so-called oracle Turing machines — the latter option is a quite common way to define **PH**.

2.4.1. Complete Problems for Σ_i^p and Π_i^p

Similarly to **NP**-completeness, one can define the completeness for each level of the hierarchy. For $i \in \mathbb{N}$, a language L is Σ_i^p-complete if it lies in Σ_i^p and every problem in Σ_i^p can be reduced to L in polynomial time. Π_i^p- and **PH**-completeness are defined in the same way.

To obtain complete problems for the levels of the polynomial hierarchy, one can generalize the well-known SAT problem.

Definition 2.12 (Σ_i-SAT)**.** *Let $\varphi(y_1, \ldots, y_i)$ be a Boolean formula, where each y_j is a vector of Boolean variables: $y_j \in \{0,1\}^{n_j}$ ($j \in [i], n_j \in \mathbb{N}$).*

$$\Sigma_i\text{-SAT} := \{\langle\varphi(y_1, \ldots, y_i)\rangle \mid \exists z_1 \forall z_2 \ldots Q_i z_i : \varphi(z_1, \ldots, z_i) \equiv \text{TRUE}\},$$

where $Q_i = \begin{cases} \forall, & i \text{ is even}, \\ \exists, & i \text{ is odd}. \end{cases}$

Lemma 2.13. *Σ_i-SAT is Σ_i^p-complete for $i \in \mathbb{N}$.*

It is not hard to see that the problem has the corresponding quantifier structure. To reduce any problem in Σ_i^p to Σ_i-SAT, one can use the proof idea of the Cook–Levin theorem, with slight modifications. The problem Π_i-SAT is defined analogously (starting with a universal quantifier) and is Π_i^p-complete.

Conversely, it is believed that **PH** itself does not have complete problems, as it would imply that the hierarchy collapses to the i^{th} level for some constant $i \in \mathbb{N}_0$.

The polynomial hierarchy levels can be defined similarly to Definition 2.3, as the set of all languages reducible to the corresponding complete problem in polynomial time.

Definition 2.14 (Polynomial hierarchy, alternative definition via complete problems)**.** *For* $i \in \mathbb{N}$, $\mathbf{\Sigma}_i^p = \{L \subseteq \{0,1\}^* \mid L \leq_p \mathbf{\Sigma}_i\text{-SAT}\}$, $\mathbf{\Pi}_i^p = \{L \subseteq \{0,1\}^* \mid L \leq_p \mathbf{\Pi}_i\text{-SAT}\}$.

The equivalence follows immediately from Lemma 2.13.

2.4.2. Oracle Machines

We have already seen that nondeterminism does not always suffice, which means we need a more powerful computational model to be able to solve some problems. Now imagine that we have an efficient black box SAT-solver. This would not only allow us to efficiently solve **NP**-complete problems such as — clearly — SAT, HAMILTONPATH, 4COL or 3COL, but also $\overline{\text{SAT}}$, $\overline{\text{3COL}}$ and, in fact, all of **coNP** problems: we can simply return the negation of the SAT-solver result. Moreover, this would also allow us to solve problems that are believed to be neither in **NP** nor in **coNP** classes.

Consider the problem of deciding whether a given graph has chromatic number 4:

$$\text{CHROMNO4} := \text{4COL} \cap \overline{\text{3COL}},$$

i.e., whether the graph is 4-colorable and *not* 3-colorable. We can solve this problem as follows.

Algorithm 2.2: Solving CHROMNO4 with a SAT-solver [ODo17].

Input: Graph G.
Output: 1 if $\chi(G) = 4$, 0 otherwise.

```
φ3 = R_{3COL→SAT}(G) // reduce the 3COL-instance of G to a SAT-instance
φ4 = R_{4COL→SAT}(G) // reduce the 4COL-instance of G to a SAT-instance
b3 = satSolver(φ3)
b4 = satSolver(φ4)
return b4 ∧ ¬b3
```

However, such a black box does not seem to help much if we were to solve the SMALLESTCIRCUIT problem — or any problem in the second level of **PH**. This means that having a black box solving a problem is not as good as having the actual algorithm for it.

One can formalize this approach with so-called oracle machines, where such a black-box is called an oracle and is able to solve the corresponding problem in constant time.

Definition 2.15. *A* SAT-***oracle Turing machine*** *is a Turing machine with an extra "oracle tape" and an extra* ORACLE *instruction. Whenever the* ORACLE *instruction is called, the content* y *of the oracle tape is replaced by 1 if* $y \in$ SAT *and by 0 otherwise. This takes time in* $\mathcal{O}(1)$*.*

Remark 2.16. *More generally, one can define a* ***B-oracle TM*** *for any language* B*. The set of all languages accepted by a deterministic (nondeterministic)* B*-oracle TM is denoted by* $\mathbf{P}^B$ ($\mathbf{NP}^B$)*.*

It is important to note that this notation makes sense only in combination with the corresponding computational model — in our case this is a (non-)deterministic Turing machine.

The polynomial hierarchy can be defined using this computational model. We first take a look at an intermediate level of the hierarchy.

$\mathbf{P}^{\text{SAT}} := \{L \mid L \text{ is decidable in polynomial time by a (deterministic) SAT-oracle TM}\}$.

Note that $\mathbf{P}^B \subseteq \mathbf{P}^{\text{SAT}}$ if $B \in \mathbf{NP}$ and $\mathbf{P}^B = \mathbf{P}^{\text{SAT}}$ if B is **NP**- or **coNP**-complete. The class $\mathbf{P}^{\text{SAT}}$ is therefore denoted as $\mathbf{P}^{\mathbf{NP}}$ in some literature.

Observation 2.17. *(i)* $\mathbf{NP} \subseteq \mathbf{P}^{\mathbf{NP}}$ *and* $\mathbf{coNP} \subseteq \mathbf{P}^{\mathbf{NP}}$,

(ii) $\textsc{ChromNo4} \in \mathbf{P}^{\mathbf{NP}}$,

(iii) $\textsc{SmallestCircuit} \notin \mathbf{P}^{\mathbf{NP}}$ *(under the assumption* $\mathbf{P} \neq \mathbf{NP}$*).*

It can be shown that $\mathbf{P}^{\mathbf{NP}} \subseteq \mathbf{\Sigma}_2^p$ and, since $\mathbf{coP}^{\mathbf{NP}} = \mathbf{P}^{\mathbf{NP}}$, it follows that $\mathbf{P}^{\mathbf{NP}} \subseteq \mathbf{\Pi}_2^p$. For a detailed proof, we refer the reader to [Sto76].

Definition 2.18 (Polynomial hierarchy, alternative definition via oracle machines)**.** *Having* $\mathbf{\Delta}_0^p = \mathbf{\Delta}_1^p = \mathbf{\Sigma}_0^p = \mathbf{\Pi}_0^p = \mathbf{P}$*, we define for* $i \in \mathbb{N}$*:*

$$\begin{aligned}\mathbf{\Delta}_{i+1}^p &= \mathbf{P}^{\mathbf{\Sigma}_i^p},\\ \mathbf{\Sigma}_{i+1}^p &= \mathbf{NP}^{\mathbf{\Sigma}_i^p},\\ \mathbf{\Pi}_{i+1}^p &= \mathbf{coNP}^{\mathbf{\Sigma}_i^p}.\end{aligned}$$

The following theorem shows the equivalence of the definitions.

Theorem 2.19. *For* $i \geq 2$*,* $\mathbf{\Sigma}_i^p = \mathbf{NP}^{\mathbf{\Sigma}_{i-1}\text{-SAT}}$*. That is,* $\mathbf{\Sigma}_i^p$ *is the set of all languages decidable by a polynomial time NTM with access to a* $\mathbf{\Sigma}_{i-1}$*-SAT-oracle.*

Proof. We proceed by induction on i. We could also formulate the theorem for $i \in \mathbb{N}$ — the base case would be $i = 1$, which would facilitate our work. Nevertheless, we stick to $i = 2$ as the base case for educational purposes, as it helps to understand the intuition behind the induction step better.

Base case ($i = 2$). We need to show that $\mathbf{\Sigma}_2^p = \mathbf{NP}^{\text{SAT}}$.

"$\subseteq$": For the left inclusion, we follow the proof of [AB06]. Suppose $L \in \mathbf{\Sigma}_2^p$, i.e., there exists a polynomial time TM M and a polynomial p such that:

$$x \in L \iff \exists u_1 \in \{0,1\}^{p(|x|)}\ \forall u_2 \in \{0,1\}^{p(|x|)} :\ M(x, u_1, u_2) = 1.$$

We observe that for fixed x and u_1, the remaining statement is a **coNP**-statement and can thus be determined by a SAT-oracle. This means that we can construct an NTM N with a SAT-oracle deciding L: on the input x, we guess u_1 nondeterministically and then use the oracle to check if $\forall u_2 \in \{0,1\}^{p(|x|)} :\ M(x, u_1, u_2) = 1$. This works since $x \in L$ if and only if there exists such u_1 that makes N accept.

"$\supseteq$": Suppose $L \in \mathbf{NP}^{\text{SAT}}$, meaning that L is decidable by a polynomial time NTM with a SAT-oracle. Since the nondeterminism is used for guessing the solutions, this is equivalent to the existence of a witness y verifiable by a *deterministic* TM N with a SAT-oracle:

$$x \in L \iff \exists y \in \{0,1\}^{p(|x|)} : N^{\text{SAT}}(x, y) = 1$$

for some polynomial p.

The idea is to simulate the oracle using the additional $\forall$ quantifier. On each query q_i, if $q_i \in$ SAT, then there exists a witness y' containing a satisfying assignment for q_i. Conversely, if $q_i \notin$ SAT, then q_i is unsatisfiable for *every* assignment.

We construct a TM M without an oracle to simulate N. The new witness y' contains the original witness y and, additionally, the answers $a_i \in \{0,1\}$ to oracle queries q_i. If the answer is positive ($a_i = 1$), then y' also contains a satisfying assignment s_i for q_i. If the answer is negative ($a_i = 0$), then the "co-witness" z' should contain an unsatisfiable assignment for the query. Thus, the "co-witness" z' represents (all) possible assignments for the queries with a negative answer. We can now construct the Turing machine M formally.

Algorithm 2.3: Simulating an oracle TM with a co-witness.

Input: $x \in \{0,1\}^*$, witness y', co-witness z', encoding of N.
Output: 1 if M accepts, 0 otherwise.

simulate N (copy every step till an oracle query)
if N makes oracle query q_i **then**
  if $a_i = 1$, verify that s_i is contained in y' and is a satisfying assignment for q_i
  if $a_i = 0$, REJECT if u_i in z' is a satisfying assignment for q_i
continue the simulation of N (step 1)
if all verifications are successful **and** N accepts, ACCEPT
else REJECT

We have:

$$x \in L \iff \exists y' \in \{0,1\}^{p(|x|)}\ \forall z' \in \{0,1\}^{p(|x|)} : M(x, y', z') = 1$$

for some polynomial p.

It remains to substantiate that the witness y' and the co-witness z' have polynomial length with respect to $|x|$. Firstly, note that $\text{TIME}(N) = \text{poly}(|x|)$, meaning that there are at most polynomially many oracle queries q_i. Secondly, an assignment contains one value for each variable of the formula encoded in $|x|$, which means its length does not exceed the input length: $|s_i| = |u_i| \leq |x|$. Since a composition of polynomials is again a polynomial, we get $|y'|, |z'| \leq \text{poly}(|x|)$.

Putting everything together, we have that $L \in \mathbf{\Sigma}_2^p$.

Induction assumption. $\mathbf{\Sigma}_{i-1}^p = \mathbf{NP}^{\mathbf{\Sigma}_{i-2}\text{-SAT}}$ for an arbitrary but fixed $i > 2$, i.e., any language in $\mathbf{NP}^{\mathbf{\Sigma}_{i-2}\text{-SAT}}$ can be written with $(i-1)$ alternating quantifiers, starting with an existential one.

Induction step. We need to show that $\mathbf{\Sigma}_i^p = \mathbf{NP}^{\mathbf{\Sigma}_{i-1}\text{-SAT}}$.

"$\subseteq$": The left inclusion is again easy. Let L be in $\mathbf{\Sigma}_i^p$ and recall that

$$x \in L \Leftrightarrow \exists u_1 \in \{0,1\}^{p(|x|)}\ \forall u_2 \in \{0,1\}^{p(|x|)} \ldots\ Q_i u_i \in \{0,1\}^{p(|x|)} : M(x, u_1, \ldots, u_i) = 1.$$

Just like in the base case, for fixed x and u_1, the remaining statement is a $\mathbf{co\Sigma}_{i-1}^p$-statement and can thus be verified with a $\mathbf{\Sigma}_{i-1}$-SAT-oracle.

"$\supseteq$": The right inclusion requires a little bit more work, but we follow the same idea as in the base case. Suppose $L \in \mathbf{NP}^{\mathbf{\Sigma}_{i-1}\text{-SAT}}$, that is, L is decidable by a polynomial time $\mathbf{\Sigma}_{i-1}$-SAT-oracle NTM, which is equivalent to the existence of a witness y verifiable by a deterministic TM N with a $\mathbf{\Sigma}_{i-1}$-SAT-oracle:

$$x \in L \iff \exists y \in \{0,1\}^{p(|x|)} : N^{\mathbf{\Sigma}_{i-1}\text{-SAT}}(x, y) = 1$$

for some polynomial p.

We want to construct a TM M without an oracle for simulating N. The new witness y' contains the original witness y and, additionally, the answer $a_j \in \{0,1\}$ for each query q_j. If $a_j = 1$, — that is, $q_j \in \mathbf{\Sigma}_{i-1}$-SAT — then

$$\exists y'_{j,1} \in \{0,1\}^{p(|x|)} \ \forall y'_{j,2} \in \{0,1\}^{p(|x|)} : T_1^{\mathbf{\Sigma}_{i-2}\text{-SAT}}(q_j, y'_{j,1}, y'_{j,2}) = 1$$

for some TM T_1 with a $\mathbf{\Sigma}_{i-2}$-SAT-oracle and some polynomial p.

By induction assumption, we can simulate T_1 with alternating quantifiers and a TM without an oracle, meaning we can "unfold" the above statement — we omit the variables' domain $\{0,1\}^{p(|x|)}$ for better readability:

$$\exists y'_{j,1} \ \forall y'_{j,2} \ \ldots \ Q_{i-1} y'_{j,i-1} : y'_{j,1}, \ldots, y'_{j,i-1} \text{ is a satisfying assignment for } q_j.$$

Conversely, if $a_j = 0$ ($q_j \notin \mathbf{\Sigma}_{i-1}$-SAT), then

$$\forall z'_{j,2} \in \{0,1\}^{p(|x|)} \ \exists z'_{j,3} \in \{0,1\}^{p(|x|)} : T_2^{\mathbf{\Sigma}_{i-2}\text{-SAT}}(q_j, z'_{j,2}, z'_{j,3}) = 1$$

for some TM T_2 with a $\mathbf{\Sigma}_{i-2}$-SAT-oracle and some polynomial p.

Again, by induction assumption, we can simulate T_2 with an oracle-free TM and thereby unfold the above statement (the domain is omitted):

$$\forall z'_{j,2} \ \exists z'_{j,3} \ \ldots \ Q_i z'_{j,i} : q_j \text{ is not satisfied with the assignment } z'_{j,2}, \ldots, z'_{j,i}.$$

Note that in the latter case we numerate the quantifiers from 2 to i — this is done for convenience. One can merge the two statements by reformulating:

$$\begin{aligned}
&\exists y'_{j,1} \ \forall (y'_{j,2}, z'_{j,2}) \ \ldots \ Q_{i-1}(y'_{j,i-1}, z'_{j,i-1}) \ Q_i z'_{j,i} : \\
&\qquad (a_j = 1 \text{ AND } y'_{j,1}, \ldots, y'_{j,i-1} \text{ is a satisfying assignment for the query } q_j) \\
&\qquad \text{OR} \\
&\qquad (a_j = 0 \text{ AND } z'_{j,2}, \ldots, z'_{j,i} \text{ is not a satisfying assignment for the query } q_j).
\end{aligned}$$

This means that we have the desired quantifiers structure. We can now construct the machine M for simulating the oracle formally.

Algorithm 2.4: Simulating a $\mathbf{\Sigma}_{i-1}$-SAT-oracle TM.

Input: $x \in \{0,1\}^*$, (co-)witnesses y' and z', encoding of N.
Output: 1 if M accepts, 0 otherwise.

simulate N (copy every step till an oracle query)
if N makes oracle query q_j **then**
  if $a_j = 1$, verify that $y'_{j,1}, \ldots, y'_{j,i-1}$ in y' is a satisfying assignment for q_j
  if $a_j = 0$, REJECT if $z'_{j,2}, \ldots, z'_{j,i}$ in z' is a satisfying assignment for q_j
continue the simulation of N (step 1)
if all verifications are successful **and** N accepts, ACCEPT
else REJECT

This gives us

$$x \in L \iff \exists y'_1 \ \forall (y'_2, z'_2) \ \ldots \ Q_{i-1}(y'_{i-1}, z'_{i-1}) \ Q_i z'_i : M(x, y'_1, \ldots, z'_i) = 1.$$

The argumentation for the polynomial length of the (co-)witnesses is the same as in the base case. We thus get $L \in \mathbf{\Sigma}_i^p$, which finishes the proof. ∎

Figure 2.3 summarizes the introduced definitions of the polynomial hierarchy with various notations and shows the placement of the intermediate levels $\mathbf{\Delta}_i^p$.

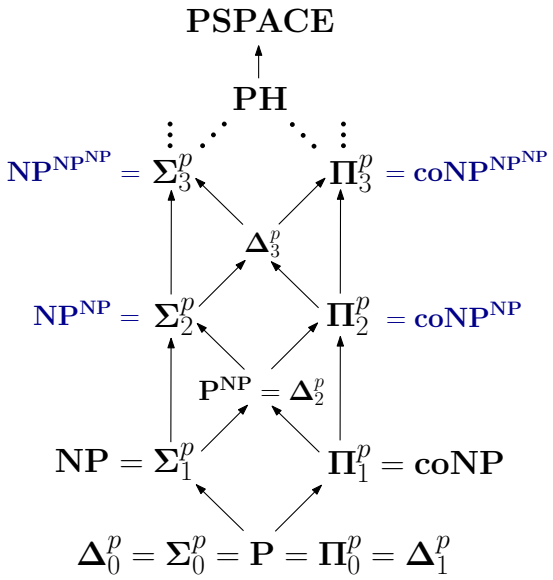

Figure 2.3: The polynomial hierarchy. The arrows denote inclusion.

2.5. The Blum–Shub–Smale Model

In order to deal not only with discrete problems, but also with problems containing real numbers, we need a more powerful computational model. In 1989, Blum, Shub and Smale introduced a generalization of Turing machines that performs computations over some arbitrary field K [BSS89]. Classical Turing machines usually operate over the residue class ring $\mathbb{Z}_2$ (but can operate over any *finite* alphabet). We are particularly

interested in such BSS machines over the reals, that is, $K = \mathbb{R}$ (the band alphabet is not finite anymore). It is hard to find a "standard" definition of a BSS machine. Our definition is based on the original paper, with slight modifications introduced by Meer and Ziegler [MZ05].

Definition 2.20 (BSS machine, [BSS89], [MZ05])**.** *Let $Y \subseteq \mathbb{R}^\infty := \bigcup_{k\in\mathbb{N}} \mathbb{R}^k$, i.e., the set of all finite sequences over the real numbers.*

*(i) A **BSS machine** M **over** $\mathbb{R}$ **with admissible input set** Y is given by a finite set I of **instructions** (**nodes**) labeled by $1, \ldots, N$. Y is also called the input space, $O \subseteq \mathbb{R}^\infty$ is the output space.*

*The **state space** is given by $S = \mathbb{R}_\infty := \{(\ldots, x_{-1}, x_0, x_1, \ldots) \mid x_i \in \mathbb{R} \cup \{\sqcup\}, i \in \mathbb{Z}\}$. It models the tape of a classical Turing machine and has infinitely many registers, with x_0 being the analogy of the position of the head. The first instruction is called the **input node** and is a linear map from the input to the state space ($in\colon Y \to S$), which takes the input $y \in Y$ and places it on the tape such that the first entry is located under the machine's head (in the registry x_0); all other registers are initialized with the blank symbol. The N^{th} instruction is called the **output node** and is a linear map from the state space to the output space ($out\colon S \to O$), which takes the tape content $x \in \mathbb{R}_\infty$ and removes the blank symbols.*

*A **configuration** of M is a quadruple $(n, i, j, x) \in I \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{R}_\infty$. Here, n denotes the currently executed instruction, i and j are used as addresses (copy-registers), and x is the tape content of M. The initial configuration of M's computation on input $y \in Y$ is $(1, 0, 0, in(y))$. If $n = N$ and the actual configuration is (N, i, j, x), the computation stops with output $out(x)$.*

The instructions M is allowed to perform (except for the input and the output nodes) are of the following types:

- ***computation:** apply a rational function $g\colon \mathbb{R}_\infty \to \mathbb{R}_\infty$, which depends on and changes a finite number of registers. The n^{th} instruction is $x_0 \leftarrow x_i \circ_n x_j$, where $\circ_n \in \{+, -, \times, \div\}$, or $x_0 \leftarrow c$ for some constant $c \in \mathbb{R}$. The register x_0 will get the value $x_i \circ_n x_j$ (i and j are the addresses from the current configuration) or c, respectively. All other register entries remain unchanged. The next instruction will be $n + 1$; moreover, the copy-register i is either incremented by one, replaced by 0, or remains unchanged. The same holds for copy-register j.*
- ***shift:** shift the contents of the tape one register to the left or to the right (one can think of it as of moving the head of a TM).*
- ***branch:** compute a polynomial $p\colon \mathbb{R}_\infty \to \mathbb{R}$ (depending on a finite number of registers) and branch to different nodes depending on the result. The polynomial p is fixed (it is not part of the corresponding branch-instruction, but is rather part of the BSS machine), meaning that it depends only on the current tape content.*
- ***copy:** $x_i \leftarrow x_j$, i.e., the content of the register j is copied into the register i. The next instruction is $n + 1$; all other registers remain unchanged.*

(ii) The size of an $x \in \mathbb{R}^k$ is $size_\mathbb{R}(x) = k$. The cost of any of the above operations is 1. The cost of a computation is the number of operations performed until the machine halts.

(iii) A set $A \subseteq \mathbb{R}^\infty$ is called a ***decision problem*** *or a* ***language*** *over $\mathbb{R}^\infty$. We call a function $f\colon A \to \mathbb{R}^\infty$* ***(BSS-)computable*** *if it is realized by a BSS machine over admissible input set A. Similarly, a set $A \subseteq \mathbb{R}^\infty$ is* ***decidable*** *in $\mathbb{R}^\infty$ if its characteristic function is computable.*

Recall that the characteristic function of A is defined as

$$\mathbb{1}_A\colon \mathbb{R}^\infty \to \{0,1\};\ \mathbb{1}_A(x) = \begin{cases} 1, & x \in A, \\ 0, & x \notin A. \end{cases}$$

A is said to be ***semi-decidable*** *if there is a BSS machine that takes inputs from $\mathbb{R}^\infty$ and accepts precisely the elements belonging to A — on the elements not in A, the machine either rejects or does not halt.*

(iv) A ***BSS oracle machine*** *using an oracle set $B \subseteq \mathbb{R}^\infty$ is a BSS machine with an additional type of node called an oracle node. Entering such a node, the machine can ask the oracle whether the current tape content $out(x) \in \mathbb{R}^\infty$ belongs to B. The oracle gives the correct answer at unit cost.*

In contrast to classical Turing machines, different instructions (e.g., computation and head movements) are not performed simultaneously. There is no particular reason for it in terms of computational power, but it allows for a more refined complexity analysis and helps to distinguish between the "real" and discrete worlds.

Complexity classes, polynomial-time reductions, and universal machines can be defined analogously to Turing machines.

Additionally, we are going to need a notion of a nondeterministic BSS machine. We do not go into details of the formal definition of such a machine, but it works similarly to the nondeterministic Turing machines. The nondeterminism can be used for guessing the certificates $y \in \mathbb{R}^\infty$ before a BSS machine starts its deterministic computational process. This means that guessing the witnesses can be "outsorced" to the nondeterministic mode: similarly to Turing machines, if there is a possibility for a nondeterministic machine to accept, it will be used — i.e., if there *exists* a valid solution $y \in \mathbb{R}^\infty$ for a given problem instance, it will be written on the band in the nondeterministic mode.

2.6. The Real RAM Model

Another suited computational model for dealing with real numbers is the real RAM model, which is an extension of the word RAM model that can store real values and perform (infinite precision) computations on them in constant space and time [EvM19]. The word RAM is a more realistic computational model resembling actual computers: the memory is divided into blocks (registers) of a fixed size, it has a finite set of instructions and an integer program counter [Mou02]. Moreover, computations take more time on longer inputs [EvM19].

2.6.1. Definition of the Real RAM

The real RAM — or *random access machine* — has been the standard model in computational geometry since late 1970s, however, the first formal definition of it was given by Erickson, van der Hoog and Miltzow — our definition is thus strongly based on it [EvM19].

The memory of a real RAM consists of two sets of registers $W[0..2^w - 1]$ and $R[0..2^w - 1]$, which are arrays with index access. The parameter w is called the *word size* — it determines the number of registers and the maximum integer size (integers are represented as sequences of w bits and are stored in the word registers $W[i]$):

$$W[i] \in \{0, \ldots, 2^w - 1\}, i \in \{0, \ldots, 2^w - 1\}.$$

The real registers $R[i]$ can store real numbers with infinite precision: $R[i] \in \mathbb{R}$.

A *real RAM algorithm* consists of a finite set of instructions. Furthermore, the real RA machine has an integer program counter storing the number of the current instruction, which is initially set to 1.

There is a *central processing unit* (CPU) performing operations on registers in constant time. We differentiate between the instructions on the word and real registers. The instructions such a machine is allowed to perform are of the following types (each instruction can be parameterized by integers of valid size $i, j, k \in \mathbb{N}$ and a valid instruction number $l \in \mathbb{N}$).

Table 2.1: Allowed constant-time instructions for the real RAM model. Table taken from [EvM19].

Type	**Word**	**Real**
Constants	$W[i] \leftarrow j$	$R[i] \leftarrow \{0, 1\}$
Memory	$W[i] \leftarrow W[j]$ $W[W[i]] \leftarrow W[j]$ $W[i] \leftarrow W[W[j]]$	$R[i] \leftarrow R[j]$ $R[W[i]] \leftarrow R[j]$ $R[i] \leftarrow R[W[j]]$
Casting	— —	$R[i] \leftarrow j$ $R[i] \leftarrow W[j]$
Arithmetic and Boolean	$W[i] \leftarrow W[j] \boxplus W[k]$	$R[i] \leftarrow R[j] \oplus R[k]$
Comparisons	`if` $W[i] = W[j]$ `goto` l `if` $W[i] < W[j]$ `goto` l	`if` $R[i] = 0$ `goto` l `if` $R[i] > 0$ `goto` l
Control flow	`goto` l / `halt` / `accept` / `reject`	

A key difference to the BSS model is indirect memory access (cf. memory instructions), which facilitates the construction of some algorithms.

The model supports following arithmetic and Boolean operations $\boxplus$ on the word registers ($x, y, z \in W$):

- **addition:**
$$x \leftarrow y + z \mod 2^w;$$
- **subtraction:**
$$x \leftarrow y - z \mod 2^w;$$
- **lower multiplication:**
$$x \leftarrow y \cdot z \mod 2^w;$$
- **upper multiplication:**
$$x \leftarrow \left\lfloor \frac{yz}{2^w} \right\rfloor;$$

- **rounded division:**
$$x \leftarrow \left\lfloor \frac{y}{z} \right\rceil, \quad \text{where } z \neq 0;$$
- **remainder:**
$$x \leftarrow y \bmod z, \quad \text{where } z \neq 0;$$
- **bitwise NAND:**
$$x \leftarrow y \barwedge z \quad (\text{i.e., } x_i \leftarrow y_i \barwedge z_i \text{ for every bit-index } i).$$

The real RAM model supports following exact operations $\oplus$ with real numbers $(x, y, z \in R)$:

- **addition:**
$$x \leftarrow y + z;$$
- **subtraction:**
$$x \leftarrow y - z;$$
- **multiplication:**
$$x \leftarrow y \cdot z;$$
- **exact division:**
$$x \leftarrow \frac{y}{z}, \quad \text{where } z \neq 0;$$
- **(optional) exact square root:**
$$x \leftarrow \sqrt{y}, \quad \text{where } y \geq 0.$$

It is important to note that including rounding operations on real registers would allow constant-time integer arithmetic with infinite precision. This, in turn, would allow solving any problem from **PSPACE** in polynomial time — the model would be too powerful [Sch79]. To avoid this, we explicitly forbid rounding ($\lfloor \cdot \rfloor$), ceiling ($\lceil \cdot \rceil$), trigonometric and logarithmic functions.

The *input* of a real RAM has the form $(a, b) \in \mathbb{Z}^m \times \mathbb{R}^n$ for some $m, n \in \mathbb{N}$; we require that the integer part of the input consists of the words of size w: $a_i \leq 2^w - 1$ for $i \in [m]$. The input is encoded into the memory registers before the machine starts its computation. Furthermore, we assume that there are enough registers to hold the input: $w \geq \log_2(n + m)$ — this is so-called *transdichotomous* assumption. However, to preserve uniformity, we need to require that the input sizes n and m, and the word size w are *not* known to any algorithm at "compile time".

The *output* is the memory content when the program executes the `halt` instruction. The *running time* of a real RAM is the number of executed instructions before the program halts.

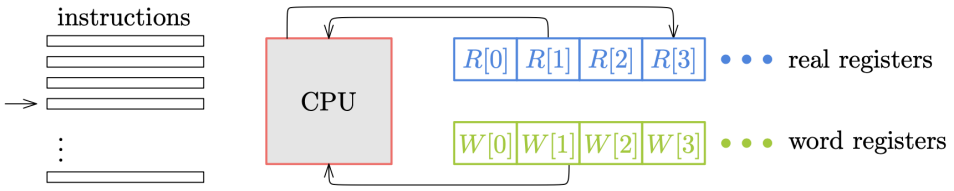

Figure 2.4: The real RAM model. Figure taken from [EvM19].

2.6.2. Equipping the Real RAM with Oracles and Nondeterminism

When it comes to extending the real RAM with oracles and nondeterminism, the most straightforward way to do it would be analogously to the word RAM model. The problem is, we are unaware of any published formal definition of such a model that would suit our interests. The reason for that might be that the word RAM model is more "realistic" — adding oracles and nondeterministic computation are abstract and theoretical concepts.

Some papers implicitly use these concepts in the context of a word RAM, without explicitly defining it [Cou23]. McKay and Williams [MW19] give a definition of a word RAM with a *random* oracle, but their model has an oracle *tape*, an oracle head, and a write-once output tape, which does not really fit into the concept of this computational model. We therefore give our own definition of an oracle real RAM.

Definition 2.21 (Oracle real RAM)**.** *Let O be a language (or decision problem) over $\{0,1\}^*$. An **oracle real RAM** is a real RAM with an additional* ORACLE *instruction and a Boolean register. An algorithm can query the oracle set O with the current content of the word registers W by calling the* ORACLE *instruction, i.e., ask whether the word registers' content is contained in O. The oracle gives an answer in constant time; the answer is written to the dedicated Boolean register.*

It suffices to query the word registers' content because the instances can be encoded as integer vectors: real numbers are only needed for the variables' *values* — variables themselves contained in a given instance can be encoded using integers.

As for nondeterminism, we stay on a rather intuitive level and assume that it works similarly to Turing machines. The nondeterminism can be used for guessing the certificates $y \in \mathbb{R}^\infty$ before a real RAM starts its deterministic computational process. That is to say, a real RAM is allowed to write arbitrary real numbers into the real registers in the nondeterministic mode. We say that the machine accepts if there is at least one accepting execution path.

Note that if we are working with a nondeterministic real RAM, the input can be restricted to an integer vector for encoding the instances (solutions are guessed in the nondeterministic mode).

3. The Existential Theory of the Reals and the Class $\exists\mathbb{R}$

In this chapter, we introduce the *existential theory of the reals* (ETR for brevity) with the corresponding complexity class $\exists\mathbb{R}$, as well as provide some motivation on why studying $\exists\mathbb{R}$-complete problems is of scientific relevance. The notation used throughout the sections mostly follows that of [Mat14].

3.1. The Existential Theory of the Reals

So far, we have dealt with discrete problems — the variables were Boolean or integers. But what happens if we allow variables to be real numbers? The theory of the reals tries to find an answer to this question by generalizing classical complexity theory.

The *first-order theory of the reals* is the set of all true sentences with polynomial equations and inequalities as atoms (i.e., there are no free variables) over the real numbers. It is denoted by $\mathrm{Th}(\mathbb{R})$. First-order means that we quantify over individual elements — in our case, these are real numbers. $\mathrm{Th}(\mathbb{R})$ was proven to be decidable by Tarski [Hen49] and was shown to lie in **EXPSPACE** [BKR86].

An example of such a sentence would be

$$\exists x \in \mathbb{R}\ \forall y \in \mathbb{R} : x \cdot x + y \cdot y > 0,$$

as the statement is true for every $x \neq 0$.

The *existential theory of the reals* — denoted by ETR or $\mathrm{Th}_{\exists}(\mathbb{R})$ — is the existential fragment of $\mathrm{Th}(\mathbb{R})$, that is, the set of all true existential sentences with polynomial equations and inequalities as atoms.

Definition 3.1 (Existential theory of the reals, ETR)**.** *The problem ETR is defined as*

$$ETR = \{\langle\varphi(y_1, \ldots, y_n)\rangle \mid \exists x_1, \ldots, x_n \in \mathbb{R} : \varphi(x_1, \ldots, x_n) \equiv \textsc{True}\},$$

where φ is a quantifier-free Boolean formula consisting of constants $0, 1$, binary operators $+, -, \times$, relations $<, \leq, \geq, >, =, \neq$ and using $\wedge, \vee, \neg, \leftrightarrow$ as connectives.

Remark 3.2. *Later on, we say that a formula is from the* ***theory of the reals*** *if it has the same signature as in the aforementioned definition.*

In other words, ETR is a decision problem where we are given an existential sentence on the input, and the output is yes if and only if the given sentence is true, meaning that there exist variables satisfying φ, which is essentially a system of polynomial equations and inequalities. Note that only constants 0 and 1 are allowed, implying that

polynomials have integer coefficients (they can be written using the binary expansion), although the variables range over the real numbers. One could also only allow one direction of inequalities, e.g., $<$ and $\leq$ (the expressions on the both sides can simply be flipped), but we allow both directions for convenience.

Consider the following example of an ETR instance:

$$\exists x, y \in \mathbb{R} : x + y \leq 1 \wedge x \geq 0 \wedge y \geq 0.$$

The formula is satisfiable by any point within the triangle with vertices $(0, 0), (0, 1), (1, 0)$ in a Cartesian plane and is therefore a yes-instance.

ETR is often called a real-valued analog of Sat. It is not hard to see that the existential theory of the reals is **NP**-hard because one can easily reduce a Sat-instance to an ETR one [Sho90]. Furthermore, Canny showed the placement of ETR within **PSPACE** in 1988 [Can88].

Since we are going to deal with polynomial reductions, a notion of formula size is needed. The input size is measured by the length of the formula, i.e., the number of symbols therein multiplied by a logarithmic factor. Although not explicitly allowed, we write integer constants in decimal for convenience (e.g., 2 instead of $1 + 1$), since it does not increase the length crucially — one needs $\mathcal{O}(\log k)$ bits for encoding $k \in \mathbb{Z}$ using the binary expansion. For better readability, we use exponents as an abbreviation for repeated multiplication, but the power operation is explicitly not allowed in the theory of the reals. Moreover, one needs to be careful when working with polynomials. For instance, the length of the polynomial $(1 + x_1) \dots (1 + x_n)$ is linear in n, but if we were to multiply it out, the length would become exponential $(\mathcal{O}(2^n))$ [Mat14].

3.2. The Class ∃ℝ and ∃ℝ-completeness

Just like ETR is called a real analog of Sat, the complexity class ∃ℝ can be thought of as a real analog of **NP**. In contrast to classical complexity theory, where **NP**-completeness of Sat is proven, ∃ℝ is usually defined directly by making ETR a complete problem.

Definition 3.3 (Complexity class ∃ℝ)**.** *The complexity class* ∃ℝ *is the set of all problems reducible to ETR in polynomial time.*

$$\exists\mathbb{R} := \{L \subseteq \{0, 1\}^* \mid L \leq_p ETR\}.$$

Note that languages in ∃ℝ can still be encoded using the alphabet $\{0, 1\}$, which is why this class is rather a *discretized* real analog of **NP**. The **NP**-hardness of ETR gives us $\mathbf{NP} \subseteq \exists\mathbb{R}$; together with Canny's result [Can88] we have

$$\mathbf{NP} \subseteq \exists\mathbb{R} \subseteq \mathbf{PSPACE}.$$

But why are ∃ℝ-complete problems so important? For Sat, we have "simple" exponential algorithms — if instances are small enough, one can often solve problems in **NP** "fast". Conversely, the existential theory of the reals is deeply intertwined with algebraic geometry — the best algorithms for solving this problem require knowledge from this field of mathematics. Furthermore, we can only solve small instances of ∃ℝ-complete problems optimally in practice. Even exponential time algorithms are hard to find, and they (almost) always rely on algebraic geometry.

Consider, for instance, the two-dimensional packing problem Pack, where we are given a container, which is a square in $\mathbb{R}^2$, and polygonal items in $\mathbb{R}^2$. The question is: can one fit the given items into the container by using rotation and translation?

Pack was shown to be $\exists\mathbb{R}$-complete [AMS22]. Moreover, given an instance with eleven (!) unit squares, we do not know how to fit them into a larger square container optimally. Figure 3.1 illustrates the best-known solution for this instance.

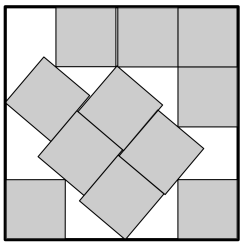

Figure 3.1: The currently best-known packing of eleven unit squares [AMS22].

Euclidean distances are often irrational and require infinite precision — thus, real-valued variables are needed and the corresponding problems are believed to be not in **NP** anymore.

A lot of geometrical problems turn out to be $\exists\mathbb{R}$-complete. One well-known example is the art gallery problem [AAM17], a visibility problem from computational geometry. It tries to minimize the number of guards (points in $\mathbb{R}^2$) who together can observe a whole art gallery (a polygon in $\mathbb{R}^2$). A point is said to be guarded (or observed) if there is a line segment connecting it with a guard and the segment stays within the polygon representing the gallery.

Some other examples include various graph drawing problems [Jun23a], matrix decomposition [Shi17], and even training neural networks. The latter result was shown by Abrahamsen et al. [AKM21] and improved by Bertschinger et al. [Ber+22]; the corollary being that if we could, for example, optimally solve the packing problem, then we would also be able to optimally train neural networks since both problems are $\exists\mathbb{R}$-complete.

3.3. Special Cases of ETR

To show the $\exists\mathbb{R}$-completeness of other problems, we need polynomial reductions from known $\exists\mathbb{R}$-complete problems. When reducing directly from ETR, the systems of polynomial equations and inequalities can become quite large, but we would like to keep them in a — more or less — standard form. In this section, we take a look at some special cases of the existential theory of the reals with additional restrictions retaining $\exists\mathbb{R}$-completeness.

We begin with Ineq, a variation of ETR where φ is a conjunction of (not necessarily strict) polynomial inequalities in standard form, i.e., written as a sum of monomials. Since we allow non-strict inequalities, equations can be modeled as well ($a = b$ is equivalent to $a \leq b \wedge b \leq a$). The connectives $\vee, \neg, \leftrightarrow$ are not allowed. StrictIneq is a further restriction of Ineq, where only strict inequalities are allowed; $\leq, \geq$, and $=$ are forbidden.

The feasibility problem $\text{FEAS}_{\mathbb{R}}^{\mathbb{Z}}$ is another special case of INEQ asking whether a single polynomial $p \in \mathbb{Z}[X_1, \ldots, X_n]$ has a zero. The $\mathbb{Z}$ in the superscript indicates that the polynomial has integer coefficients; the $\mathbb{R}$ in the subscript that variables range over the real numbers.

QUAD is a restriction of ETR asking whether a set of quadratic polynomials has a common zero. 4-$\text{FEAS}_{\mathbb{R}}^{\mathbb{Z}}$ is a special case of $\text{FEAS}_{\mathbb{R}}^{\mathbb{Z}}$, where the degree of the given polynomial does not exceed 4.

Another interesting variation of the existential theory of the reals is ETR-INV. In this problem, the question is if there exist variables $x_1, \ldots, x_n \in [\frac{1}{2}, 2]$ such that constraints $C_1, \ldots, C_n$ are fulfilled. Each constraint C_i is either $x + y = z$ or $xy = 1$, where x, y, z are some variables. The idea of proving the $\exists\mathbb{R}$-completeness of this problem is to simplify the formulae, scale the solutions, and to replace multiplication by inversion [AM19]. This result can be used to show that PACK is $\exists\mathbb{R}$-complete [AMS22]. We do not go into details of these proofs since we do not need these problems for further work.

However, we show the $\exists\mathbb{R}$-completeness of (4-)$\text{FEAS}_{\mathbb{R}}^{\mathbb{Z}}$, as this result is needed for defining $\exists\mathbb{R}$ with a machine model. What is more, this fact implies that solving a system of polynomial equations and inequalities over the real numbers is — up to a polynomial-time reduction — computationally equivalent to solving a single polynomial equation in many variables.

Definition 3.4 (k-feasibility problem, k-$\text{FEAS}_{\mathbb{R}}^{\mathbb{Z}}$)**.** *For $k \in \mathbb{N}$, the k-$\text{FEAS}_{\mathbb{R}}^{\mathbb{Z}}$ problem is*

$$k\text{-}\textit{FEAS}_{\mathbb{R}}^{\mathbb{Z}} = \{\langle p(y_1, \ldots, y_n)\rangle \mid \exists x_1, \ldots, x_n \in \mathbb{R} : p(x_1, \ldots, x_n) = 0\},$$

where p is a polynomial in n variables with integer coefficients of degree at most k:

$$p \in \mathbb{Z}[X_1, \ldots, X_n],\ \deg p \leq k.$$

Definition 3.5 (Prenex normal form)**.** *A formula is in **prenex normal form** if it has the following structure:*

$$Q_1 x_1\ Q_2 x_2\ \ldots\ Q_n x_n : \varphi(x_1, \ldots, x_n),$$

where $Q_1, \ldots, Q_n$ are quantifiers and φ is a quantifier-free formula.

In order to convert an arbitrary formula to prenex form, one can simply push the quantifiers outside. Additionally, one needs to "flip" the quantifier when pushing through negation.

Theorem 3.6. *4-$\text{FEAS}_{\mathbb{R}}^{\mathbb{Z}}$ and QUAD are $\exists\mathbb{R}$-complete.*

Proof sketch. We follow the idea of [Mat14] and [SŠ17] to show the $\exists\mathbb{R}$-completeness of 4-$\text{FEAS}_{\mathbb{R}}^{\mathbb{Z}}$. We reduce from ETR and show

$$\text{ETR} \leq_p \text{QUAD} \leq_p \text{4-FEAS}_{\mathbb{R}}^{\mathbb{Z}}.$$

Let $\exists x_1, \ldots, x_n \in \mathbb{R} : \varphi(x_1, \ldots, x_n)$ be an existential sentence in prenex normal form of length L from an ETR-instance. The goal is to transform it to an equivalent 4-$\text{FEAS}_{\mathbb{R}}^{\mathbb{Z}}$-instance of length $\mathcal{O}(L)$ in polynomial time. One can use the idea of the so-called Tseitin transformation [Tse68] to construct quadratic polynomials p_j with new variables such that all of these polynomials are feasible if and only if the original formula is satisifable:

$$\exists x \in \mathbb{R}^n : \varphi(x) \equiv \textsc{True} \iff \exists x \in \mathbb{R}^n, y \in \mathbb{R}^m : \bigwedge_{j \in [k]} p_j(x, y) = 0$$

for some $m, k \in \mathbb{N}$.

For each subformula γ, we add a new existentially quantified real-valued variable y_γ — or multiple variables y_{γ_i}, if needed — representing the value of γ. We process the subformulae from the inner- to outermost and use various tricks to write them as polynomial equations. As we proceed, a new equation is added for every subformula to the constraints set Γ, which is initially empty.

We do not go into the details of the precise definition of a subformula or a subterm and keep it on a rather intuitive level. Moreover, we index the polynomials with the name of corresponding subformulae for convenience.

Any subterm s of φ can easily be replaced by a polynomial equation: for $s = u \circ v$ or $s = a$, we add $p_s = u \circ v - s$ or $p_s = s - a$, respectively, where $\circ$ is any allowed binary operation from $\{+, -, \times\}$.

For a disjunction $\gamma \equiv \alpha \vee \beta$, the corresponding polynomial is $p_\gamma = y_\alpha + y_\beta - y_\alpha y_\beta - 1$; for a conjunction $\gamma \equiv \alpha \wedge \beta$, we add $p_\gamma = y_\alpha y_\beta - 1$. The Boolean values are represented by 1 and 0 for True and False, respectively. The negation $\gamma \equiv \neg\alpha$ is handled by $p_\gamma = \alpha$ since $\alpha = 0$ is equivalent to $\neg\alpha = 1$. The equivalence $\gamma \equiv \alpha \leftrightarrow \beta$ can be emulated by $(\alpha \wedge \beta) \vee (\neg\alpha \wedge \neg\beta)$.

For an equation $\gamma \equiv a = b$, we define $p_\gamma = y_a - y_b$. Without loss of generality, we consider only inequalities of the form $a < b$. For strict inequalities of the form $a > b$, one can simply swap the sides; a non-strict inequality $a \leq b$ is equivalent to the disjunction $(a < b) \vee (a = b)$. Let $\gamma \equiv a < b$. We define two polynomials:

$$p_{\gamma_1} = y_{\gamma_1} - y_{\gamma_2}^2, \tag{3.1}$$

$$p_{\gamma_2} = (y_b - y_a) y_{\gamma_1} - 1. \tag{3.2}$$

If $a < b$, then $y_b - y_a > 0$ and we can choose $y_{\gamma_1} = y_{\gamma_2}^2 > 0$ such that $(y_b - y_a) y_{\gamma_1} = 1$, which yields $p_{\gamma_1} = p_{\gamma_2} = 0$. Conversely, if $p_{\gamma_1} = p_{\gamma_2} = 0$, then we have:

$$\begin{cases} y_{\gamma_1} = y_{\gamma_2}^2, \\ (y_b - y_a) y_{\gamma_1} = 1. \end{cases} \tag{3.3}$$

$$\Rightarrow y_b - y_a = \frac{1}{y_{\gamma_1}} = \frac{1}{y_{\gamma_2}^2} > 0, \tag{3.4}$$

implying that $y_a < y_b$.

Eventually, we add $p_\varphi = y_\varphi - 1$ to Γ. By construction, if $\varphi(x)$ is true for a fixed x, than there exist a variable assignment for y such that $p_j(x, y) = 0$ for every $p_j \in \Gamma$. On the other hand, if $p_j(x, y) = 0$ for all equations in Γ, then, in particular, $y_\varphi = 1$, meaning that $\varphi(x)$ is true.

The equations in Γ can be constructed in polynomial time. Furthermore, note that we added no more than three variables and two polynomials for each subformula; the length of the coefficients remained the same. Because φ has at most L subformulae (the number cannot exceed the length of the formula), it follows that $m \leq 3L$ and $k \leq 2L$ — the size of the new instance is linear in L.

Since the degree of the polynomials in Γ does not exceed two, we have constructed an equivalent Quad instance, thereby showing that Quad is ∃ℝ-hard. Moreover, Quad is a special case of ETR — a Quad-instance is trivially an ETR one — and therefore lies in ∃ℝ. This gives us ∃ℝ-completeness of Quad.

Finally, observe that a system of equations $p = 0 \wedge q = 0$ is equivalent to $p^2 + q^2 = 0$. To obtain the desired 4-Feas$^{\mathbb{Z}}_{\mathbb{R}}$-instance, we replace the equations in Γ by a single polynomial equation $p_1^2 + \cdots + p_k^2 = 0$. This at most doubles the degree of polynomials and the formula length, which means that we indeed get an equivalent 4-Feas$^{\mathbb{Z}}_{\mathbb{R}}$-instance.

Again, 4-Feas$^{\mathbb{Z}}_{\mathbb{R}}$ is a restriction of ETR. Thus, 4-Feas$^{\mathbb{Z}}_{\mathbb{R}}$ is both ∃ℝ-hard and lies in ∃ℝ, meaning that this problem is ∃ℝ-complete. ❑

Corollary 3.7. *Feas$^{\mathbb{Z}}_{\mathbb{R}}$ and Ineq are ∃ℝ-complete.*

The corollary follows immediately from Theorem 3.6. Since 4-Feas$^{\mathbb{Z}}_{\mathbb{R}}$ and Quad are special cases of Feas$^{\mathbb{Z}}_{\mathbb{R}}$ and Ineq, respectively, the reductions in the above proof can be used to construct the instances of the corresponding problems as well, giving us ∃ℝ-hardness. On top of that, these problems are both restrictions of ETR, meaning that they lie in ∃ℝ and thus are ∃ℝ-complete.

Remark 3.8. *StrictIneq is ∃ℝ-complete.*

It suggests that solving a system of strict polynomial inequalities and a system of arbitrary ones are computationally equivalent up to a polynomial time reduction. This statement is not trivial and its proof requires a "reasonably difficult result" from real algebraic geometry [Mat14].

3.4. Definition via BSS machines

Adding oracles is an operation on machines, not on languages. This means that if we want to expand the class ∃ℝ by giving it "oracle powers", we need a definition using a machine model. We begin by exploring the complexity classes defined via BSS machines. The notation in this section is largely based on [BSS89].

3.4.1. Complexity Classes Over ℝ

Definition 3.9 (Complexity class $\mathbf{P}_{\mathbb{R}}$)**.** *The complexity class $\mathbf{P}_{\mathbb{R}}$ is the set of all languages $L \subseteq \mathbb{R}^\infty$ decidable by a deterministic BSS machine in polynomial time.*

$\mathbf{NP}_{\mathbb{R}}$ contains all languages whose witness-checking problem lies in $\mathbf{P}_{\mathbb{R}}$, that is, there exists a certificate that can be verified by some BSS machine in polynomial time.

Definition 3.10 (Complexity class $\mathbf{NP}_{\mathbb{R}}$)**.** *Let $L \subseteq \mathbb{R}^\infty$. Then $L \in \mathbf{NP}_{\mathbb{R}}$ if there exists a polynomial time BSS machine M and a polynomial p, such that for all $x \in \mathbb{R}^\infty$:*

$$x \in L \iff \exists u \in \mathbb{R}^{p(|x|)} : M(x, u) = 1.$$

Using nondeterminism, an alternative definition of $\mathbf{NP}_{\mathbb{R}}$ can be given.

Definition 3.11 (Complexity class $\mathbf{NP}_{\mathbb{R}}$, alternative definition)**.** *The complexity class $\mathbf{NP}_{\mathbb{R}}$ is the set of all languages decidable by a nondeterministic BSS machine in polynomial time.*

Note that the languages in $\mathbf{NP}_{\mathbb{R}}$ are over $\mathbb{R}^\infty$, but the languages in $\exists\mathbb{R}$ are over $\{0,1\}^*$. We would like to discretize the classes defined by BSS machines in order to obtain $\exists\mathbb{R}$. This can be done in two steps.

Firstly, we allow BSS machines to work only with constants 0 and 1 (the constants could also be bounded integers, the main restriction is that irrational or transcendental constants are forbidden).

Definition 3.12 (Complexity classes $\mathbf{P}^0_{\mathbb{R}}$ and $\mathbf{NP}^0_{\mathbb{R}}$)**.** *The complexity class $\mathbf{P}^0_{\mathbb{R}}$ is the constant-free fragment of $\mathbf{P}_{\mathbb{R}}$, that is, the set of all languages $L \subseteq \mathbb{R}^\infty$ decidable by a deterministic BSS machine with constants 0 and 1 in polynomial time.*

Similarly, the complexity class $\mathbf{NP}^0_{\mathbb{R}}$ is the set of all languages $L \subseteq \mathbb{R}^\infty$ whose witnesses are verifiable by a BSS machine with constants 0 and 1 in polynomial time.

We are on a good path because the existential theory of the reals only allows constants 0 and 1 as well. Nevertheless, the languages in $\mathbf{P}^0_{\mathbb{R}}$ and $\mathbf{NP}^0_{\mathbb{R}}$ are still defined over the infinite alphabet of the real numbers. This is where the second step comes into play: we take the so-called Boolean part of the languages.

Definition 3.13 (Boolean part)**.** *The Boolean part of a language $L \subseteq \mathbb{R}^\infty$ is its intersection with the set $\{0,1\}^*$:*

$$\mathbf{BP}(L) := L \cap \{0,1\}^*.$$

The Boolean part of a complexity class $\mathcal{C}$ over $\mathbb{R}$ consists of the Boolean parts of the languages therein:

$$\mathbf{BP}(\mathcal{C}) := \{L \cap \{0,1\}^* \mid L \in \mathcal{C}\}.$$

In the context of the BSS model, this means that the admissible input set of the corresponding BSS machine must be a subset of $\{0,1\}^* \times \mathbb{R}^\infty$. In other words, the instances of the problems in $\mathbf{BP}(L)$ can be encoded in binary (or any other finite alphabet); the witnesses are still from $\mathbb{R}^\infty$.

When working with nondeterministic BSS machines, the admissible input set is simply $\{0,1\}^*$ for encoding the instances, but we need to allow access to real-valued constants in the nondeterministic mode to be able to guess the witnesses.

One can write $\mathbf{BP}^0(L)$ to indicate that both discretization steps are applied simultaneously. Now, consider the language $\mathbf{BP}(\mathbf{NP}^0_{\mathbb{R}})$. It seems to do the trick: it only allows the constants 0 and 1 and is defined over the binary alphabet $\{0,1\}$. Indeed, $\mathbf{BP}(\mathbf{NP}^0_{\mathbb{R}})$ corresponds exactly to the complexity class $\exists\mathbb{R}$. Nonetheless, this result is nontrivial and will be shown in this section. First, it requires some theoretical preparation.

3.4.2. An Analog of the Cook–Levin Theorem for $\mathbf{NP}_{\mathbb{R}}$

In their paper from 1989, where they introduced BSS machines, Blum, Shub, and Smale also prove an analog of the Cook–Levin Theorem [BSS89]. The theorem states that the 4-feasibility problem over $\mathbb{R}$ (4-$\textsc{Feas}_{\mathbb{R}}$), where the polynomials are allowed to have real coefficients, is $\mathbf{NP}_{\mathbb{R}}$-complete. The proof is quite complicated and technical; we therefore need to take a closer look at the Blum–Shub–Smale model first.

3.4.2.1. Under the Hood of a BSS Machine

For proving the real analog of the Cook–Levin theorem, we would like to express the computational process of a BSS machine as a system of polynomial equations because it allows an easier construction of a feasibility instance. For that purpose, some additional definitions, elaborations, and technicalities are needed.

Recall Definition 2.20, where we introduced BSS machines. Let M be a BSS machine. The Cartesian product of the instruction set with the state space $I \times S$ is called the *full state space* of M. The computational process of M is described by a so-called *computational endomorphism* H, defined by

$$\begin{aligned} H &\colon I \times S \to I \times S, \\ H(n, x) &= \big(\beta(n, \operatorname{sgn} x_0), g_n(x)\big), \end{aligned} \tag{3.5}$$

where β describes the next instruction and g_n the next state (i.e., the tape content). One can think of it as of the transition function of a Turing machine.

For a non-computational node n, the function g_n leaves the tape content unchanged. Otherwise, $g_n(x)$ is the computation result.

The signum function simply gives the sign of the element in the registry x_0:

$$\begin{aligned} \operatorname{sgn} &\colon \mathbb{R} \to \{0, \pm 1\}, \\ \operatorname{sgn} y &= \begin{cases} -1, & y < 0, \\ 0, & y = 0, \\ 1, & y > 0. \end{cases} \end{aligned} \tag{3.6}$$

The next instruction is defined as follows:

$$\begin{aligned} \beta &\colon I \times \{0, \pm 1\} \to I, \\ \beta(n, \sigma) &= \begin{cases} N, & n = N, \\ n + 1, & n < N \text{ and } n \text{ is nonbranching}, \\ \beta^+(n), & \text{n is branching and } \sigma \geq 0, \\ \beta^-(n), & \text{n is branching and } \sigma < 0, \end{cases} \end{aligned} \tag{3.7}$$

where $(n + 1)$ denotes the instruction from I with the corresponding label; $\beta^{\pm}(n)$ are some nodes from the instruction set I and are specified in the corresponding branching node n.

Yet, in our definition of a BSS machine, we say that a branching node computes some polynomial and the branching occurs according to the result. Indeed, β can be written as a polynomial function. For a node $n \in I$ and the current instruction y, we define an auxiliary polynomial

$$a_n(y) := \prod_{j \in I \setminus \{n\}} \frac{y - j}{n - j} = \begin{cases} 1, & y = n, \\ 0, & y \neq n. \end{cases} \tag{3.8}$$

Thus, the next instruction $\beta(y, \sigma)$ depending on the current instruction y and the sign of the zeroth registry $\sigma = \operatorname{sgn} x_0$ is given by:

$$\begin{aligned}\beta(y,\sigma) = &\sum_{n\in I\setminus B} a_n(y)(n+1)\\ &+ \underbrace{\Big(\frac{\sigma(\sigma+1)}{2} + (1+\sigma)(1-\sigma)\Big)}_{\gamma} \sum_{n\in B} a_n(y)\beta^+(n)\\ &+ \underbrace{\frac{\sigma(\sigma-1)}{2}}_{\delta} \sum_{n\in B} a_n(y)\beta^-(n),\end{aligned} \tag{3.9}$$

where $B \subseteq I$ denotes the set of all branching nodes. Because $a_n(y) = 0$ for all nodes distinct from y, all but one summands are equal to zero. To differentiate between β^+ and β^- depending on the sign of the zeroth registry x_0, we introduce the terms γ and δ, which are placed before the corresponding sums. Consider the following case distinction.

$$\begin{aligned}\sigma = 1\colon\ \gamma &= 1 + 2\cdot 0 = 1,\\ \delta &= 0;\\ \sigma = 0\colon\ \gamma &= 0 + 1\cdot 1 = 1,\\ \delta &= 0;\\ \sigma = -1\colon\ \gamma &= 0 + 0\cdot 2 = 0,\\ \delta &= \frac{-1\cdot(-2)}{2} = 1.\end{aligned}$$

Thus, we have

$$\gamma = \begin{cases}1, & \sigma \geq 0,\\ 0, & \sigma < 0;\end{cases} \text{ and } \delta = \begin{cases}1, & \sigma < 0,\\ 0, & \sigma \geq 0.\end{cases} \tag{3.10}$$

That is to say, $\beta(y, \sigma)$ is either the next instruction $(y + 1)$ (if y is a non-branching node) or $\beta^{\pm}(y)$ depending on $\sigma = \operatorname{sgn} x_0$ otherwise. This gives us the equivalence of (3.7) and (3.9).

To bring β into a pure polynomial form, we need to get rid of the signum function. This can be done by introducing a new existentially quantified variable $u \neq 0$ and adding the equation

$$x_0(x_0u^2 + 1)(x_0u^2 - 1) = 0. \tag{3.11}$$

One can observe that $x_0 = 0$ leads to $x_0u^2 = 0$ and $x_0 > 0$ ($x_0 < 0$) enforces $x_0u^2 = 1$ ($x_0u^2 = -1$). Conversely, if $x_0u^2 = 0$, then x_0 must be 0 since $u \neq 0$. Moreover, $x_0u^2 = 1$ implies $x_0 = \frac{1}{u^2} > 0$. Analogously, $x_0u^2 = -1$ enforces $x_0 = -\frac{1}{u^2} < 0$. Hence, x_0u^2 corresponds to $\operatorname{sgn} x_0$.

For a BSS machine M, we denote the function that M computes by f_M. Let $n_k \in I$ denote the k^{th} instruction of M. If $n_T = N$ for some $T < \infty$, we call T the *halting time*. More precisely, on the input $y \in Y \subseteq \mathbb{R}^\infty$, we define

$$T_M(y) := T(y) := \min\{T \in \mathbb{N} \mid n_T = N\}$$

as the *halting time for M to compute $f_M(y)$*.

We can now describe the computational process of BSS machines by a system of polynomial equations. The computation of a machine M with input $y \in Y$ is described by a sequence $(z_k)_{k\in\mathbb{N}_0}$, $z_k = (n_k, x^{(k)}) \in I \times S$, $z_0 = \big(1, \mathrm{in}(y)\big)$, $z_k = H(z_{k-1}), k \in \mathbb{N}$, where $x^{(k)}$ denotes the tape content at the time point k. The system

$$\begin{cases} n_0 = 1, \\ x^{(0)} = \mathrm{in}(y), \\ (n_k, x^{(k)}) = H(n_{k-1}, x^{(k-1)}), k \in \mathbb{N}; \end{cases} \tag{3.12}$$

is called the *registry equations* of M. The third equation means that the current instruction n_k and the tape content $x^{(k)}$ are given by the computational endomorphism H (3.5) with the previous instruction and tape content as input.

The *time T halting equations* is the system

$$\begin{cases} n_0 = 1, \\ x^{(0)} = \mathrm{in}(y), \\ (n_k, x^{(k)}) = H(n_{k-1}, x^{(k-1)}), k \in [T], \\ n_T = N; \end{cases} \tag{3.13}$$

that is, the machine M reaches the final instruction N after T steps. In this case, the equation

$$\mathrm{out}\big(x^{(T)}\big) = f_M(y) \tag{3.14}$$

is satisfied. Using the preparation above, one can rewrite these equations as follows:

$$\begin{cases} n_0 = 1, \\ x^{(0)} = \mathrm{in}(y), \\ n_T = N, \\ x_0^{(k-1)}(x_0^{(k-1)}u_{k-1}^2 + 1)(x_0^{(k-1)}u_{k-1}^2 - 1) \overset{(3.11)}{=} 0, \\ n_k \overset{(3.11)}{=} \beta(n_{k-1}, x_0^{(k-1)}u_{k-1}^2), \\ x^{(k)} \overset{(3.5)}{=} g_{n_{k-1}}(x^{(k-1)}),\ k \in [T]. \end{cases} \tag{3.15}$$

We call the last three equations the *transition equations* of M.

3.4.2.2. The Blum–Shub–Smale Theorem

Definition 3.14 (Real k-feasibility problem, k-Feas$_\mathbb{R}$)**.** *For $k \in \mathbb{N}$, the k-Feas$_\mathbb{R}$ problem is*

$$k\text{-Feas}_\mathbb{R} = \{\langle p(y_1, \ldots, y_n)\rangle \mid \exists x_1, \ldots, x_n \in \mathbb{R} : p(x_1, \ldots, x_n) = 0\},$$

where p is a polynomial in n variables with real coefficients of degree at most k:

$$p \in \mathbb{R}[X_1, \ldots, X_n],\ \deg p \le k.$$

Theorem 3.15 (Blum, Shub, Smale. Analog of the Cook–Levin theorem for $\mathbf{NP}_\mathbb{R}$ [BSS89])**.** *4-Feas$_\mathbb{R}$ is $\mathbf{NP}_\mathbb{R}$-complete.*

Proof sketch. Let $A \subseteq \mathbb{R}^\infty$ be an arbitrary decision problem in $\mathbf{NP}_\mathbb{R}$. We need a polynomial-time reduction to 4-Feas$_\mathbb{R}$. The idea is similar to the one in the proof of the Cook–Levin theorem: we abstract from the concrete problem and consider the corresponding BSS machine M that can verify solutions to A in polynomial time. Recall that $A \in \mathbf{NP}_\mathbb{R}$ if there exists a polynomial time BSS machine M and a polynomial p, such that for all $y \in \mathbb{R}^\infty$:

$$y \in A \iff \exists y' \in \mathbb{R}^{p(|y|)} : M(y, y') = 1.$$

The admissible input set Y of M is $Y_A \times \mathbb{R}^m$ for some $m \in \mathbb{N}$, Y_A is the set of all correctly encoded instances of A. The halting time T of M is polynomial in the input size: $T = \text{poly}(|y|)$.

We want to describe the computational process of M by a system of polynomial equations, which can then be transformed to construct a 4-Feas$_\mathbb{R}$-instance. In the previous subsection, we discussed some aspects of how to do it.

Consider the time T halting equations (3.15) of M (in our case, $x^{(0)} = \text{in}(y, y')$). We extend this system by the equation

$$\text{out}\big(x^{(T)}\big) = 1 \text{ (yes)}, \tag{3.16}$$

which is equivalent to

$$x_0^{(T)} - 1 = 0. \tag{3.17}$$

The new system consisting of the time T halting equations of M and the equation (3.17) has a solution if and only if M accepts the input y, i.e., y is a yes-instance of A ($y \in A$). This system of polynomial equations is easily convertible to a single polynomial equation of degree at most four — one can simply follow the proof idea of Theorem 3.6 and use the Tseitin transformation. We thus get an equivalent 4-Feas$_\mathbb{R}$-instance. We denote the transformed instance by $\psi(y)$:

$$\psi(y) \in \text{4-Feas}_\mathbb{R} \iff y \in A.$$

The original system has polynomially many equations in T, the transformed instance is linear in the size of the original system, thus, $|\psi(y)| = \text{poly}(T)$. Since $T = \text{poly}(|y|)$, we get $|\psi(y)| = \text{poly}(|y|)$. Furthermore, the transformation ψ can be performed in polynomial time, inferring that it is indeed a polynomial-time reduction and 4-Feas$_\mathbb{R}$ is $\mathbf{NP}_\mathbb{R}$-hard.

It remains to show that 4-Feas$_\mathbb{R} \in \mathbf{NP}_\mathbb{R}$. Given a solution $y' \in \mathbb{R}^\infty$, we need to test if $p(y') = 0$ for a polynomial $p \in \mathbb{R}[X_1, \ldots, X_n]$ of degree at most four. Since $\deg p \leq 4$, we can evaluate $p(y')$ in polynomial time with a BSS machine. Such a polynomial can be encoded in $\mathbb{R}^\infty$ using the so-called *power-free representation*. The encoding starts with $(4, n)$, followed by a sequence of pairs $(\alpha, c_\alpha)_{\alpha \in \Lambda}$, where $\alpha = (\alpha_1, \ldots, \alpha_4)$, $\alpha_i \in \{0, \ldots, n\}$, $\alpha_i \leq \alpha_{i+1}$, $c_\alpha \in \mathbb{R}$, and Λ is an index set for enumerating the sequences. The pair (α, c_α) represents the monomial $c_\alpha x_{\alpha_1} x_{\alpha_2} x_{\alpha_3} x_{\alpha_4}$, with c_α being the coefficient,

x_{α_i} representing one of the n variables and $x_0 = 1$ to allow for monomials of degree less than four. The pairs are ordered lexicographically on α. The polynomial p can be thus written as

$$p(x) = \sum_{\alpha \in \Lambda} c_\alpha x_{\alpha_1} x_{\alpha_2} x_{\alpha_3} x_{\alpha_4}.$$

The length of the encoding is polynomial in the length of p written in the standard form. ▢

Remark 3.16 ([Tri90])**.** *k-Feas$_\mathbb{R}$ $\in \mathbf{P}_\mathbb{R}$ for $k \leq 3$.*

In 1990, Triesch introduced a polynomial-time algorithm for solving k-Feas$_\mathbb{R}$ for $k \leq 3$ [Tri90]. This means that the minimal degree k for the $\mathbf{NP}_\mathbb{R}$-completeness of the k-feasibility problem is four.

3.4.3. Defining ∃ℝ

After a quite cumbersome preparation, we can finally define the complexity class ∃ℝ using BSS machines as the computational model. Namely, ∃ℝ corresponds exactly to the discretized version of $\mathbf{NP}_\mathbb{R}$.

Theorem 3.17 ([BC06], [SŠ23])**.** $\exists\mathbb{R} = \mathbf{BP}(\mathbf{NP}^0_\mathbb{R})$.

Proof sketch. We follow the idea of [SŠ23] and show that 4-Feas$^\mathbb{Z}_\mathbb{R}$ is a complete problem for $\mathbf{BP}(\mathbf{NP}^0_\mathbb{R})$. Since the same problem is also ∃ℝ-complete (Theorem 3.6), it implies that $\exists\mathbb{R} = \mathbf{BP}(\mathbf{NP}^0_\mathbb{R})$.

For the $\mathbf{BP}(\mathbf{NP}^0_\mathbb{R})$-completeness of 4-Feas$^\mathbb{Z}_\mathbb{R}$, we follow the proof of Bürgisser and Cucker [BC06]. To obtain a polynomial-time reduction from an arbitrary problem in $\mathbf{BP}(\mathbf{NP}^0_\mathbb{R})$ to 4-Feas$^\mathbb{Z}_\mathbb{R}$, one can use the same idea as in the proof of the real analog of Cook–Levin theorem (Theorem 3.15). We take the corresponding BSS machine with constants 0 and 1 verifying the solutions to the given problem, express its computational process with a system of polynomial equations and construct a 4-Feas$^\mathbb{Z}_\mathbb{R}$-instance. It works because the reduction in the mentioned proof does not use any other constants than 0 and 1. Hence, 4-Feas$^\mathbb{Z}_\mathbb{R}$ is $\mathbf{BP}(\mathbf{NP}^0_\mathbb{R})$-hard.

It remains to justify that 4-Feas$^\mathbb{Z}_\mathbb{R}$ $\in \mathbf{BP}(\mathbf{NP}^0_\mathbb{R})$. Given a solution to a 4-Feas$^\mathbb{Z}_\mathbb{R}$-instance, we can verify it with a BSS machine with constants 0 and 1 in polynomial time. We can use the power-free representation to encode polynomials as in Theorem 3.15. The computational power suffices because the polynomial coefficients are now integers: they can be encoded with logarithmic length using binary expansion.

Putting everything together yields the $\mathbf{BP}(\mathbf{NP}^0_\mathbb{R})$-completeness of 4-Feas$^\mathbb{Z}_\mathbb{R}$. ▢

3.5. Definition via Real RAM

Even though BSS machines are a decent candidate for defining ∃ℝ with a computational model, the ∃ℝ-membership of the most problems from computational geometry is proven using the real RAM model. The reason is that BSS machines do not support the *integer* operations necessary to implement some simple algorithms [EvM19]. Moreover, the BSS model — in contrast to the real RAM — does not support indirect memory access, which complicates constructing a verification algorithm using it [EvM19].

Erickson, van der Hoog and Miltzow show an analog of the Cook–Levin theorem for the real RAM, which allows us to give another alternative definition of the class $\exists\mathbb{R}$ [EvM19].

Definition 3.18 (Real verification algorithm)**.** *Let $L \subseteq \{0,1\}^*$ be a decision problem. We say that L has a **real verification algorithm** if for every $x \in \{0,1\}^*$:*

$$x \in L \iff \exists \text{ polynomial-time real RAM } R, \text{ witness } y \in \mathbb{R}^{p(|x|)} : R(x,y) = 1$$

for some polynomial p.

Definition 3.19 (Complexity class $\exists\mathbb{R}$, alternative definition with real RAM)**.** *The complexity class $\exists\mathbb{R}$ is the set of all decision problems over $\{0,1\}^*$ with a real verification algorithm.*

The equivalence of this and the previous definitions follows from the following theorem.

Theorem 3.20 (Erickson et al. [EvM19])**.** *Let $L \subseteq \{0,1\}^*$ be a decision problem. Then*

$$L \in \exists\mathbb{R} \iff L \text{ has a real verification algorithm.}$$

Proof sketch. We follow the proof of [EvM19] and show that ETR is complete for the complexity class containing all problems with a real verification algorithm.

"⇒": Let $L \in \exists\mathbb{R}$, that is, $L \leq_p$ ETR per definition. We therefore need to show that ETR has a real verification algorithm: an instance of L can be transformed to an ETR-instance in polynomial time and then verified using the algorithm. Let

$$\exists x_1, \ldots, x_n \in \mathbb{R} : \varphi(x_1, \ldots, x_n)$$

be a sentence from the existential theory of the reals. Note that such an instance uses a finite alphabet of size $n + \mathcal{O}(1)$ (n variables and a fixed amount of connectors, operators, and constants), meaning that we can encode the instance as an integer vector $a \in \mathbb{Z}^m$ for some $m \in \mathbb{N}$. Having a solution $y = (y_1, \ldots, y_n) \in \mathbb{R}^n$, we can verify that $\varphi(y_1, \ldots, y_n) \equiv \textsc{True}$ in polynomial time — with respect to the formula length — on a real RAM, e.g., by using a standard recursive-descent parser (i.e., we process the subformulae from the outer- to innermost).

"⇐": Let $L \subseteq \{0,1\}^*$ be a decision problem with a real verification algorithm, that is, for every $x \in \{0,1\}^*$:

$$x \in L \iff \exists \text{ real RAM } R, \text{ witness } y \in \mathbb{R}^{p(|x|)} : R(x,y) = 1$$

for some polynomial p.

The idea mirrors the one from the Cook–Levin theorem: we take the real RAM R verifying the solutions for L and express its computational process with a system of polynomial equations and Boolean formulae. We thus obtain an ETR instance which is a yes-instance if and only if the original instance is a yes-instance. Furthermore, the newly constructed instance is polynomial in the length of the original one and the construction can be performed in polynomial time on a word RAM (or a Turing machine). For the construction details of such ETR instance, we refer the reader to [EvM19]. This shows that $L \leq_p$ ETR, hence, $L \in \exists\mathbb{R}$. ❑

4. The (Discrete) Real Polynomial Hierarchy

In this chapter, we introduce the real polynomial hierarchy and its discretized version. Furthermore, we define it using oracle BSS and real random access machines, which is a central result of the thesis. The notation used throughout the chapter is quite common, although it does not follow any source precisely — it was rather adapted to fit the context of the work.

Just like the Boolean satisfiability problem can be extended for the different levels of the classical polynomial hierarchy, the existential theory of the reals can be extended to obtain the real polynomial hierarchy. We begin by exploring the co-class of $\exists\mathbb{R}$ — the complexity class $\forall\mathbb{R}$.

4.1. The Universal Theory of the Reals and the Class $\forall\mathbb{R}$

Recall the existential theory of the reals

$$\text{ETR} = \{\langle\varphi(y_1, \ldots, y_n)\rangle \mid \exists x_1, \ldots, x_n \in \mathbb{R} : \varphi(x_1, \ldots, x_n) \equiv \textsc{True}\},$$

containing all solvable systems of polynomial equations and inequalities with integer coefficients. To obtain its complement — the universal theory of the reals — one simply negates the condition.

Definition 4.1 (Universal theory of the reals, UTR)**.** *The problem UTR is defined as*

$$UTR := \{\langle\varphi(y_1, \ldots, y_n)\rangle \mid \forall x_1, \ldots, x_n \in \mathbb{R} : \varphi(x_1, \ldots, x_n) \equiv \textit{True}\},$$

where φ is a quantifier-free Boolean formula from the theory of the reals.

In other words, the universal theory of the reals consists of all systems of polynomial equations and inequalities with integer coefficients that are true independently from the choice of the variables.

Definition 4.2 (Complexity class $\forall\mathbb{R}$)**.** *The complexity class $\forall\mathbb{R}$ is the set of all problems reducible to UTR in polynomial time.*

$$\forall\mathbb{R} := \{L \subseteq \{0,1\}^* \mid L \leq_p UTR\}.$$

4.2. Extensions of ETR and UTR as Complete Problems

To obtain the discretized real polynomial hierarchy, one can extend the existential and universal theory of the reals by adding quantifiers.

Definition 4.3 ($\mathbf{\Sigma}_i$-ETR, $\mathbf{\Pi}_i$-UTR)**.** *Let $\varphi(y_1, \ldots, y_i)$ be a formula from the theory of the reals (cf. Remark 3.2). Moreover, each y_k is a vector of real-valued variables: $y_j \in \mathbb{R}^{n_j}$ ($j \in [i], n_j \in \mathbb{N}$).*

For $i \in \mathbb{N}_0$, we define

$$\mathbf{\Sigma}_i\text{-}ETR := \{\langle\varphi(y_1, \ldots, y_i)\rangle \mid \exists z_1 \forall z_2 \ldots Q_i z_i : \varphi(z_1, \ldots, z_i) \equiv \textsc{True}\},$$

$$\mathbf{\Pi}_i\text{-}UTR := \{\langle\varphi(y_1, \ldots, y_i)\rangle \mid \forall z_1 \exists z_2 \ldots P_i z_i : \varphi(z_1, \ldots, z_i) \equiv \textsc{True}\},$$

where $Q_i = \begin{cases} \forall, & i \text{ is even}, \\ \exists, & i \text{ is odd}; \end{cases}$ *and* $P_i = \begin{cases} \exists, & i \text{ is even}, \\ \forall, & i \text{ is odd}. \end{cases}$

We could also define $\mathbf{\Pi}_i$-UTR as the set of all *false* formulae over the same quantifiers structures as in the definition above for the sake of consistency with the $\overline{\textsc{Sat}}$ problem, which is defined to contain all *unsatisfiable* Boolean formulae. Technically, it would not make any difference, since one could simply define a new formula $\psi \equiv \neg\varphi$, which flips the truth values — thus, the equivalence of the definitions follows. Nevertheless, the universal theory of the reals is conventionally defined as the set of all *true* sentences.

The discrete real polynomial hierarchy is defined by making the corresponding versions of ETR and UTR complete problems.

Definition 4.4 (Discrete real polynomial hierarchy, **DRPH**)**.** *For $i \in \mathbb{N}_0$, we define*

(i) $\mathbf{\Sigma}_i\mathbb{R}_\mathbb{Z} := \{L \subseteq \{0,1\}^* \mid L \leq_p \mathbf{\Sigma}_i\text{-}ETR\}$,

(ii) $\mathbf{\Pi}_i\mathbb{R}_\mathbb{Z} := \{L \subseteq \{0,1\}^* \mid L \leq_p \mathbf{\Pi}_i\text{-}UTR\}$,

(iii) $\mathbf{DRPH} := \bigcup_{i\in\mathbb{N}_0} \mathbf{\Sigma}_i\mathbb{R}_\mathbb{Z}$.

That is to say, the discrete real polynomial hierarchy is the union over all hierarchy levels $\mathbf{\Sigma}_i\mathbb{R}_\mathbb{Z}$. Note that the languages therein are still defined over the binary alphabet $\{0,1\}$ — therefore the name *discrete*. This is indicated by $\mathbb{Z}$ in the subscript.

To obtain the non-discretized version, one can allow real-valued constants in the formulae, but the usual way to define the real polynomial hierarchy is via BSS machines — this is done in further sections.

4.3. Properties of DRPH

For the lower levels of the hierarchy (i.e., for small i), it is common to use the quantifier notation since it reflects the intuition behind the problem structure therein. Moreover, the discrete real polynomial hierarchy can be defined recursively using this notation.

Notation.

$\mathbf{\Sigma}_0\mathbb{R}_\mathbb{Z} = \mathbf{\Pi}_0\mathbb{R}_\mathbb{Z} = \varnothing\mathbb{R}$;

$\mathbf{\Sigma}_1\mathbb{R}_\mathbb{Z} = \exists\mathbb{R} = \mathbf{NP}^\mathbb{Z}_\mathbb{R}$, $\quad\mathbf{\Pi}_1\mathbb{R}_\mathbb{Z} = \forall\mathbb{R} = \mathbf{coNP}^\mathbb{Z}_\mathbb{R}$;

$\mathbf{\Sigma}_2\mathbb{R}_\mathbb{Z} = \exists\forall\mathbb{R}$, $\quad\mathbf{\Pi}_2\mathbb{R}_\mathbb{Z} = \forall\exists\mathbb{R}$;

$\mathbf{\Sigma}_3\mathbb{R}_\mathbb{Z} = \exists\forall\exists\mathbb{R}$, $\quad\mathbf{\Pi}_3\mathbb{R}_\mathbb{Z} = \forall\exists\forall\mathbb{R}$;

$\ldots \qquad\qquad\qquad\qquad \ldots$

$\mathbf{\Sigma}_i\mathbb{R}_\mathbb{Z} = \exists\mathbf{\Pi}_{i-1}\mathbb{R}_\mathbb{Z}$, $\quad\mathbf{\Pi}_i\mathbb{R}_\mathbb{Z} = \forall\mathbf{\Sigma}_{i-1}\mathbb{R}_\mathbb{Z}$.

With this notation, it is not hard to see that the properties of the classical polynomial hierarchy (Observation 2.11) also hold for **DRPH**.

Observation 4.5. *For $i \in \mathbb{N}$:*

(i) $\mathbf{\Pi}_i\mathbb{R}_\mathbb{Z} = \mathbf{co\Sigma}_i\mathbb{R}_\mathbb{Z}$,

(ii) $\mathbf{\Sigma}_i\mathbb{R}_\mathbb{Z} \subseteq \mathbf{\Sigma}_{i+1}\mathbb{R}_\mathbb{Z}$ *and* $\mathbf{\Sigma}_i\mathbb{R}_\mathbb{Z} \subseteq \mathbf{\Pi}_{i+1}\mathbb{R}_\mathbb{Z}$,

(iii) $\mathbf{\Pi}_i\mathbb{R}_\mathbb{Z} \subseteq \mathbf{\Pi}_{i+1}\mathbb{R}_\mathbb{Z}$ *and* $\mathbf{\Pi}_i\mathbb{R}_\mathbb{Z} \subseteq \mathbf{\Sigma}_{i+1}\mathbb{R}_\mathbb{Z}$,

(iv) $\mathbf{DRPH} = \bigcup_{i\in\mathbb{N}_0} \mathbf{\Pi}_i\mathbb{R}_\mathbb{Z} = \bigcup_{i\in\mathbb{N}_0} \mathbf{\Sigma}_i\mathbb{R}_\mathbb{Z}$.

The relation between the co-classes in different levels of the hierarchy is quite interesting. For $i \in \mathbb{N}$, the intersection $\mathbf{\Sigma}_i\mathbb{R}_\mathbb{Z} \cap \mathbf{\Pi}_i\mathbb{R}_\mathbb{Z}$ is non-empty since both classes contain the previous level of the hierarchy. Furthermore, the polynomial identity testing problem (PIT) was shown to lie in $\exists\mathbb{R} \cap \forall\mathbb{R}$ [Jun23b]. Conversely, it makes sense to believe that $\mathbf{\Sigma}_i\mathbb{R}_\mathbb{Z} \neq \mathbf{\Pi}_i\mathbb{R}_\mathbb{Z}$, as the hierarchy would collapse to the i^{th} level otherwise.

Claim 4.6. *If $\mathbf{\Sigma}_i\mathbb{R}_\mathbb{Z} = \mathbf{\Pi}_i\mathbb{R}_\mathbb{Z}$ for some $i \in \mathbb{N}$, then the hierarchy collapses to the i^{th} level, that is,*

$$\forall j > i \in \mathbb{N} : \mathbf{\Sigma}_j\mathbb{R}_\mathbb{Z} = \mathbf{\Pi}_j\mathbb{R}_\mathbb{Z} = \mathbf{\Sigma}_i\mathbb{R}_\mathbb{Z} = \mathbf{\Pi}_i\mathbb{R}_\mathbb{Z}.$$

Proof. We showcase the idea by proving the claim for the first level of the hierarchy.

Suppose that $\exists\mathbb{R} = \forall\mathbb{R}$. Then, $\exists\forall\mathbb{R} = \exists\exists\mathbb{R} = \exists\mathbb{R}$, as two consecutive existential blocks can be merged into one. Similarly, $\forall\exists\mathbb{R} = \forall\forall\mathbb{R} = \forall\mathbb{R} = \exists\mathbb{R}$. To show the equality of the higher levels, one can start by "flipping" the last quantifier and merging it with the penultimate one. By doing this iteratively, one can eventually eliminate all but one quantifiers, ending up in the first level. For instance:

$$\underbrace{\exists\forall\ldots\forall\underbrace{\exists\mathbb{R}}_{\forall\mathbb{R}}}_{\exists\mathbb{R}} = \exists\mathbb{R}.$$

The cases with different first or last quantifier are handled analogously.

If the equality $\mathbf{\Sigma}_i\mathbb{R}_\mathbb{Z} = \mathbf{\Pi}_i\mathbb{R}_\mathbb{Z}$ holds for larger i, one can use the same idea to prove the collapsing to the i^{th} level. ∎

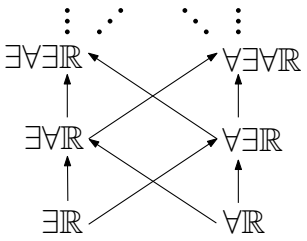

Figure 4.1: The discrete real polynomial hierarchy. The arrows denote inclusion.

4.4. Generalization of the Feasibility Problem

To define the **DRPH** with BSS machines, we would like to generalize our definition of the class $\exists\mathbb{R}$ (Theorem 3.17). The feasibility problem (Definition 3.4) can be generalized to obtain a complete problem for an arbitrary level of the hierarchy.

Recall that $\overline{\text{SAT}}$ is the set of all unsatisfiable Boolean formulae. Analogously, the complement of the feasibility problem is defined to contain all *infeasible* polynomials, that is, polynomials with no zeros.

Definition 4.7 (Generalized k-(in)feasibility problem, $\mathbf{\Sigma}_i$-k-(IN)FEAS$_{\mathbb{R}}^{\mathbb{Z}}$)**.** *Let $y_1, \ldots, y_i$ be the vectors of real-valued variables: $y_j \in \mathbb{R}^{n_j}$ ($j \in [i], n_j \in \mathbb{N}$).*

For $i, k \in \mathbb{N}$, the $\mathbf{\Sigma}_i$-k-(IN)FEAS$_{\mathbb{R}}^{\mathbb{Z}}$ problem for the i^{th} level of the hierarchy is

$$\mathbf{\Sigma}_i\text{-}k\text{-FEAS}_{\mathbb{R}}^{\mathbb{Z}} = \{\langle p(y_1, \ldots, y_i)\rangle \mid \exists z_1 \forall z_2 \ldots Q_i z_i : p(z_1, \ldots, z_i) = 0\},$$
$$\mathbf{\Sigma}_i\text{-}k\text{-INFEAS}_{\mathbb{R}}^{\mathbb{Z}} = \{\langle p(y_1, \ldots, y_i)\rangle \mid \exists z_1 \forall z_2 \ldots Q_i z_i : p(z_1, \ldots, z_i) \neq 0\}.$$

where p is a polynomial in $n := \sum_{j=1}^{i} n_j$ variables with integer coefficients of degree at most k:

$$p \in \mathbb{Z}[X_1, \ldots, X_n],\ \deg p \leq k,$$

and $Q_i = \begin{cases} \forall, & i \text{ is even,} \\ \exists, & i \text{ is odd.} \end{cases}$

One can similarly define these problems for the co-classes $\mathbf{\Pi}_i\mathbb{R}_{\mathbb{Z}}$ — we start with a universal quantifier and the whole quantifier structure is flipped.

Definition 4.8 ($\mathbf{\Pi}_i$-k-(IN)FEAS$_{\mathbb{R}}^{\mathbb{Z}}$)**.** *Let $y_1, \ldots, y_i$ be the vectors of real-valued variables: $y_j \in \mathbb{R}^{n_j}$ ($j \in [i], n_j \in \mathbb{N}$).*

For $i, k \in \mathbb{N}$, the $\mathbf{\Pi}_i$-k-(IN)FEAS$_{\mathbb{R}}^{\mathbb{Z}}$ problem for the i^{th} level of the hierarchy is

$$\mathbf{\Pi}_i\text{-}k\text{-FEAS}_{\mathbb{R}}^{\mathbb{Z}} = \{\langle p(y_1, \ldots, y_i)\rangle \mid \forall z_1 \exists z_2 \ldots Q_i z_i : p(z_1, \ldots, z_i) = 0\},$$
$$\mathbf{\Pi}_i\text{-}k\text{-INFEAS}_{\mathbb{R}}^{\mathbb{Z}} = \{\langle p(y_1, \ldots, y_i)\rangle \mid \forall z_1 \exists z_2 \ldots Q_i z_i : p(z_1, \ldots, z_i) \neq 0\}.$$

where p is a polynomial in $n := \sum_{j=1}^{i} n_j$ variables with integer coefficients of degree at most k:

$$p \in \mathbb{Z}[X_1, \ldots, X_n],\ \deg p \leq k,$$

and $Q_i = \begin{cases} \exists, & i \text{ is even,} \\ \forall, & i \text{ is odd.} \end{cases}$

By drawing analogies to the classical polynomial hierarchy and the corresponding SAT and $\overline{\text{SAT}}$ versions, one might intuitively think that the introduced versions of the feasibility and infeasibility problems $\mathbf{\Sigma}_i$-k-FEAS$^{\mathbb{Z}}_{\mathbb{R}}$ and $\mathbf{\Pi}_i$-k-INFEAS$^{\mathbb{Z}}_{\mathbb{R}}$ can be easily shown to be complete for the classes $\mathbf{\Sigma}_i\mathbb{R}_{\mathbb{Z}}$ and $\mathbf{\Pi}_i\mathbb{R}_{\mathbb{Z}}$, respectively. Alas, it is not quite the case.

In fact, $\mathbf{\Sigma}_i$-k-FEAS and $\mathbf{\Sigma}_i$-k-INFEAS are alternately complete for $\mathbf{\Sigma}_i\mathbb{R}_{\mathbb{Z}}$, depending on the parity of i; the same goes for the $\mathbf{\Pi}_i$-k-(IN)FEAS$^{\mathbb{Z}}_{\mathbb{R}}$ problem and $\mathbf{\Pi}_i\mathbb{R}_{\mathbb{Z}}$. The reason is that the last quantifier plays a role as well, therefore a case distinction is needed.

Definition 4.9 ($\mathbf{\Sigma}_i$-POLY$^{\mathbb{Z}}_{\mathbb{R}}$, $\mathbf{\Pi}_i$-POLY$^{\mathbb{Z}}_{\mathbb{R}}$)**.** *For $i \in \mathbb{N}$, we define*

$$\mathbf{\Sigma}_i\text{-}\mathit{POLY}^{\mathbb{Z}}_{\mathbb{R}} := \begin{cases} \mathbf{\Sigma}_i\text{-4-}\mathit{FEAS}^{\mathbb{Z}}_{\mathbb{R}}, & i \text{ is odd} \quad (Q_i = \exists), \\ \mathbf{\Sigma}_i\text{-4-}\mathit{INFEAS}^{\mathbb{Z}}_{\mathbb{R}}, & i \text{ is even} \quad (Q_i = \forall); \end{cases}$$

$$\mathbf{\Pi}_i\text{-}\mathit{POLY}^{\mathbb{Z}}_{\mathbb{R}} := \begin{cases} \mathbf{\Pi}_i\text{-4-}\mathit{INFEAS}^{\mathbb{Z}}_{\mathbb{R}}, & i \text{ is odd} \quad (Q_i = \forall), \\ \mathbf{\Pi}_i\text{-4-}\mathit{FEAS}^{\mathbb{Z}}_{\mathbb{R}}, & i \text{ is even} \quad (Q_i = \exists). \end{cases}$$

Theorem 4.10. *Let $i \in \mathbb{N}$. Then*

(i) $\mathbf{\Sigma}_i$-$\mathit{POLY}^{\mathbb{Z}}_{\mathbb{R}}$ *is* $\mathbf{\Sigma}_i\mathbb{R}_{\mathbb{Z}}$*-complete,*

(ii) $\mathbf{\Pi}_i$-$\mathit{POLY}^{\mathbb{Z}}_{\mathbb{R}}$ *is* $\mathbf{\Pi}_i\mathbb{R}_{\mathbb{Z}}$*-complete.*

Proof sketch. We mainly follow the idea of Matoušek [Mat14], with slight modifications for our case distinction.

(i): We reduce from $\mathbf{\Sigma}_i$-ETR. Let

$$\exists x_1 \forall x_2 \ldots Q_i x_i : \varphi(x) = \varphi(x_1, \ldots, x_i) \overset{!}{\equiv} \text{TRUE}$$

be an $\mathbf{\Sigma}_i$-ETR-sentence. Each x_j is a real-valued vector — the domain is omitted for better readability.

If the last quantifier is existential ($Q_i = \exists$), we can directly use the proof idea of Theorem 3.6: we apply the Tseitin transformation to get a feasibility instance, the existential block with new variables can simply be merged with the last one.

If the last quantifier is universal ($Q_i = \forall$), we doubly negate our sentence and, at first, apply only the inner negation — thus, the last quantifier becomes existential and φ must evaluate to FALSE. We can then apply the Tseitin transformation, but the last added polynomial is $p_\varphi = y_\varphi$ to ensure that $y_\varphi = 0$, i.e., $\varphi \equiv$ FALSE. Finally, we apply the outer negation and get an infeasibility instance. To illustrate the idea, consider the following example.

$$\begin{aligned} & \exists x \forall y : \varphi(x, y) \equiv \text{TRUE} \\ \iff & \neg(\neg[\exists x \forall y : \varphi(x, y) \equiv \text{TRUE}]) \\ \iff & \neg(\forall x \exists y : \varphi(x, y) \equiv \text{FALSE}) \\ \overset{\text{Tseitin}}{\iff} & \neg(\forall x \exists y, z : p(x, y, z) = 0) \\ \iff & \exists x \forall y, z : p(x, y, z) \neq 0. \end{aligned}$$

The higher levels of the hierarchy (i.e., with more quantifiers), are handled analogously since only the last quantifier block matters for the transformation.

With this case distinction, we obtain exactly a $\mathbf{\Sigma}_i$-Poly$^{\mathbb{Z}}_{\mathbb{R}}$-instance — the problem is thus $\mathbf{\Sigma}_i\mathbb{R}_{\mathbb{Z}}$-hard. Furthermore, $\mathbf{\Sigma}_i$-Poly$^{\mathbb{Z}}_{\mathbb{R}}$ is a special case of $\mathbf{\Sigma}_i$-ETR. Hence, $\mathbf{\Sigma}_i$-Poly$^{\mathbb{Z}}_{\mathbb{R}}$ lies in $\mathbf{\Sigma}_i\mathbb{R}_{\mathbb{Z}}$ and is $\mathbf{\Sigma}_i\mathbb{R}_{\mathbb{Z}}$-complete.

(*ii*): We reduce from $\mathbf{\Pi}_i$-UTR. Let

$$\forall x_1 \exists x_2 \dots Q_i x_i : \varphi(x) = \varphi(x_1, \dots, x_i) \overset{!}{\equiv} \textsc{True}$$

be a $\mathbf{\Pi}_i$-UTR-sentence.

If the last quantifier is existential ($Q_i = \exists$), we can apply the Tseitin transformation directly and get a feasibility instance.

Conversely, if the last quantifier is universal ($Q_i = \forall$), we use the same idea as in (*i*) with double negation and the Tseitin transformation to obtain an infeasibility instance. We thus obtain a $\mathbf{\Pi}_i$-Poly$^{\mathbb{Z}}_{\mathbb{R}}$-instance. Using the same argumentation as in (*i*), the problem is $\mathbf{\Pi}_i\mathbb{R}_{\mathbb{Z}}$-complete. ❑

4.5. The Real Polynomial Hierarchy

Recall that BSS machines define "unrestricted" — that is, the real-valued constants are allowed — complexity classes over $\mathbb{R}$ (Section 3.4.1). In this section, we consider the hierarchy defined via this computational model — the (non-discretized) real polynomial hierarchy.

We have already defined the classes $\mathbf{P}_{\mathbb{R}}$ (Def. 3.9) and $\mathbf{NP}_{\mathbb{R}}$ (Def. 3.10) — the latter one contains all languages with the witness-checking problem in $\mathbf{P}_{\mathbb{R}}$, but we restate the definition here for the convenience of the reader.

A language $L \subseteq \mathbb{R}^\infty$ is in $\mathbf{NP}_{\mathbb{R}}$ if there exists a polynomial time BSS machine M and a polynomial p, such that for all $x \in \mathbb{R}^\infty$:

$$x \in L \iff \exists u \in \mathbb{R}^{p(|x|)} : \ M(x, u) = 1.$$

Similarly to the classical polynomial hierarchy, one can extend this definition to an arbitrary (but finite) number of alternating quantifiers and obtain the real polynomial hierarchy.

Definition 4.11 (Real polynomial hierarchy, **RPH**)**.** *Let $i \in \mathbb{N}, L \subseteq \mathbb{R}^\infty$. $L \in \mathbf{\Sigma}_i\mathbb{R}$ if there exists a polynomial time BSS machine M and a polynomial p such that:*

$$x \in L \iff \exists u_1 \in \mathbb{R}^{p(|x|)} \ \forall u_2 \in \mathbb{R}^{p(|x|)} \ \dots \ Q_i u_i \in \mathbb{R}^{p(|x|)} : M(x, u_1, \dots, u_i) = 1,$$

where $|x| = size_{\mathbb{R}}(x)$ (Definition 2.20 (ii)), i.e., $|x|$ is the dimensionality of the vector x, and $Q_i = \begin{cases} \forall, & i \text{ is even}, \\ \exists, & i \text{ is odd}. \end{cases}$

Similarly, $L \in \mathbf{\Pi}_i\mathbb{R}$ if there exists a polynomial time BSS machine M and a polynomial p such that:

$$x \in L \iff \forall u_1 \in \mathbb{R}^{p(|x|)} \ \exists u_2 \in \mathbb{R}^{p(|x|)} \ \dots \ Q_i u_i \in \mathbb{R}^{p(|x|)} : M(x, u_1, \dots, u_i) = 1,$$

where $Q_i = \begin{cases} \exists, & i \text{ is even,} \\ \forall, & i \text{ is odd.} \end{cases}$

The ***real polynomial hierarchy*** *is the union over the hierarchy levels* ($\mathbf{\Sigma}_0\mathbb{R} = \mathbf{P}_\mathbb{R}$)*:*

$$\mathbf{RPH} := \bigcup_{i\in\mathbb{N}_0} \mathbf{\Sigma}_i\mathbb{R}.$$

It is not hard to see that the properties of the classical **PH** (Observation 2.11) and the discrete real polynomial hierarchy (Observation 4.5) also hold for the non-restricted version of the hierarchy.

As a convention for the notation, we use the same one as in Section 4.3, but we omit the $\mathbb{Z}$ in the subscript since the classes are not restricted anymore and the real-valued constants can be used.

Notation.

$\mathbf{\Sigma}_0\mathbb{R} = \mathbf{\Pi}_0\mathbb{R} = \mathbf{P}_\mathbb{R}$;

$\mathbf{\Sigma}_1\mathbb{R} = \mathbf{NP}_\mathbb{R}$,	$\mathbf{\Pi}_1\mathbb{R} = \mathbf{coNP}_\mathbb{R}$;
...	...
$\mathbf{\Sigma}_i\mathbb{R} = \exists\mathbf{\Pi}_{i-1}\mathbb{R}$,	$\mathbf{\Pi}_i\mathbb{R} = \forall\mathbf{\Sigma}_{i-1}\mathbb{R}$.

4.6. Corresponding Versions of the Real (In-)Feasibility Problem

To define the **DRPH** with BSS machines, one can use the same idea as in Theorem 3.17. We show that there is a problem that is complete for a class $\mathcal{C}_\mathbb{Z}$ in the **DRPH** *and* for the discretized version of the corresponding complexity class in the real polynomial hierarchy $\mathbf{BP}^0(\mathcal{C})$, which implies the equality of the classes $\mathcal{C}_\mathbb{Z} = \mathbf{BP}^0(\mathcal{C})$.

We choose $\mathbf{\Sigma}_i$-Poly$_\mathbb{R}$ and $\mathbf{\Pi}_i$-Poly$_\mathbb{R}$ as suitable problems, which are the unrestricted versions — i.e., the polynomials are allowed to have real coefficients — of $\mathbf{\Sigma}_i$-Poly$^\mathbb{Z}_\mathbb{R}$ and $\mathbf{\Pi}_i$-Poly$^\mathbb{Z}_\mathbb{R}$ (Definition 4.13), respectively. For the convenience of the reader, we define these problems formally in this section as well.

Definition 4.12 (Generalized real k-(in)feasibility problem, $\mathbf{\Sigma}_i$*-k-*(In)Feas$_\mathbb{R}$)**.** *Let* $y_1, \ldots, y_i$ *be the vectors of real-valued variables:* $y_j \in \mathbb{R}^{n_j}$ ($j \in [i], n_j \in \mathbb{N}$).

For $i, k \in \mathbb{N}$*, the* $\mathbf{\Sigma}_i$*-k-(In)*Feas$_\mathbb{R}$ *problem for the* i^{th} *level of the hierarchy is*

$$\mathbf{\Sigma}_i\text{-}k\text{-}\textit{Feas}_\mathbb{R} = \{\langle p(y_1, \ldots, y_i)\rangle \mid \exists z_1 \forall z_2 \ldots Q_i z_i : p(z_1, \ldots, z_i) = 0\},$$
$$\mathbf{\Sigma}_i\text{-}k\text{-}\textit{Infeas}_\mathbb{R} = \{\langle p(y_1, \ldots, y_i)\rangle \mid \exists z_1 \forall z_2 \ldots Q_i z_i : p(z_1, \ldots, z_i) \neq 0\}.$$

where p *is a polynomial in* $n := \sum_{j=1}^{i} n_j$ *variables with* ***real*** *coefficients of degree at most* k*:*

$$p \in \mathbb{R}[X_1, \ldots, X_n],\ \deg p \leq k,$$

and $Q_i = \begin{cases} \forall, & i \text{ is even,} \\ \exists, & i \text{ is odd.} \end{cases}$

The problem $\mathbf{\Pi}_i$-k-(In)Feas$_\mathbb{R}$ is defined analogously, but we start with a universal quantifier and the whole quantifier structure is flipped (cf. Definition 4.8).

Definition 4.13 ($\mathbf{\Sigma}_i$-Poly$_\mathbb{R}$, $\mathbf{\Pi}_i$-Poly$_\mathbb{R}$)**.** *For $i \in \mathbb{N}$, we define*

$$\mathbf{\Sigma}_i\text{-}\mathit{Poly}_\mathbb{R} := \begin{cases} \mathbf{\Sigma}_i\text{-4-}\mathit{Feas}_\mathbb{R}, & i \text{ is odd} \quad (Q_i = \exists), \\ \mathbf{\Sigma}_i\text{-4-}\mathit{Infeas}_\mathbb{R}, & i \text{ is even} \quad (Q_i = \forall); \end{cases}$$

$$\mathbf{\Pi}_i\text{-}\mathit{Poly}_\mathbb{R} := \begin{cases} \mathbf{\Pi}_i\text{-4-}\mathit{Infeas}_\mathbb{R}, & i \text{ is odd} \quad (Q_i = \forall), \\ \mathbf{\Pi}_i\text{-4-}\mathit{Feas}_\mathbb{R}, & i \text{ is even} \quad (Q_i = \exists). \end{cases}$$

We have already shown that the discretized versions of these problems — $\mathbf{\Sigma}_i$-Poly$_\mathbb{R}^\mathbb{Z}$ and $\mathbf{\Pi}_i$-Poly$_\mathbb{R}^\mathbb{Z}$ — are complete for their corresponding classes $\mathbf{\Sigma}_i\mathbb{R}_\mathbb{Z}$ and $\mathbf{\Pi}_i\mathbb{R}_\mathbb{Z}$, respectively (Theorem 4.10). To finish the definition of **DRPH** with BSS machines, it remains to show that these problems are also complete for the discretized versions of the corresponding classes of **RPH**. We show the completeness of $\mathbf{\Sigma}_i$-Poly$_\mathbb{R}$ and $\mathbf{\Pi}_i$-Poly$_\mathbb{R}$ for the unrestricted **RPH** first, as the idea is easily transferable to the discretized case.

Theorem 4.14. *Let $i \in \mathbb{N}$. Then*

(i) $\mathbf{\Sigma}_i$-$\mathit{Poly}_\mathbb{R}$ *is* $\mathbf{\Sigma}_i\mathbb{R}$*-complete,*

(ii) $\mathbf{\Pi}_i$-$\mathit{Poly}_\mathbb{R}$ *is* $\mathbf{\Pi}_i\mathbb{R}$*-complete.*

Proof sketch. We distinguish depending on the last quantifier. If the last quantifier is existential, we simply apply the proof idea of the real Cook–Levin theorem (Theorem 3.15). For an arbitrary problem in $\mathbf{\Sigma}_i\mathbb{R}$ ($\mathbf{\Pi}_i\mathbb{R}$), we take the corresponding BSS machine verifying the witnesses and co-witnesses in polynomial time, express its computational process with a system of polynomial equations and thus obtain a $\mathbf{\Sigma}_i$-Feas$_\mathbb{R}$ ($\mathbf{\Pi}_i$-Feas$_\mathbb{R}$) instance.

Now consider the case where the last quantifier is universal. Recall that $L \subseteq \mathbb{R}^\infty$ is in $\mathbf{\Sigma}_i\mathbb{R}$ ($\mathbf{\Pi}_i\mathbb{R}$) if there exists a polynomial time BSS machine M and a polynomial p such that:

$$x \in L \Leftrightarrow \exists(\forall) u_1 \in \mathbb{R}^{p(|x|)} \ldots \forall u_i \in \mathbb{R}^{p(|x|)} : M(x, u_1, \ldots, u_i) = 1.$$

We use the same idea as in Theorem 4.14. Firstly, doubly negate the statement and apply the inner negation to convert the last quantifier to be existential. Note that in the negated statement M does not accept. That is to say, we can apply the proof idea of the real Cook–Levin theorem to express M's computation with a system of polynomial equations, but the last added equation would be $\mathrm{out}(x^{(T)}) = 0$ to ensure that $M(x, u) \neq 1$. We then apply the outer negation, thereby constructing the desired $\mathbf{\Sigma}_i$-Infeas$_\mathbb{R}$ ($\mathbf{\Pi}_i$-Infeas$_\mathbb{R}$) instance.

$$\begin{aligned} x \in L &\Leftrightarrow \neg(\neg[\exists(\forall) u_1 \in \mathbb{R}^{p(|x|)} \ldots \forall u_i \in \mathbb{R}^{p(|x|)} : M(x, u_1, \ldots, u_i) = 1]) \\ &\Leftrightarrow \neg(\forall(\exists) u_1 \in \mathbb{R}^{p(|x|)} \ldots \exists u_i \in \mathbb{R}^{p(|x|)} : M(x, u_1, \ldots, u_i) \neq 1) \\ &\Leftrightarrow \neg(\forall(\exists) u_1 \in \mathbb{R}^{p(|x|)} \ldots \exists u_i, v \in \mathbb{R}^{p(|x|)} : p(x, u, v) = 0) \\ &\Leftrightarrow \exists(\forall) u_1 \in \mathbb{R}^{p(|x|)} \ldots \forall u_i, v \in \mathbb{R}^{p(|x|)} : p(x, u, v) \neq 0. \end{aligned}$$

Finally, $\mathbf{\Sigma}_i$-Poly$_\mathbb{R}$ and $\mathbf{\Pi}_i$-Poly$_\mathbb{R}$ lie in $\mathbf{\Sigma}_i\mathbb{R}$ and $\mathbf{\Pi}_i\mathbb{R}$, respectively, yielding the completeness. ❑

Theorem 4.15. *For $i \in \mathbb{N}$:*

(i) $\mathbf{\Sigma}_i\mathbb{R}_\mathbb{Z} = \mathbf{BP}^0(\mathbf{\Sigma}_i\mathbb{R})$,

(ii) $\mathbf{\Pi}_i\mathbb{R}_\mathbb{Z} = \mathbf{BP}^0(\mathbf{\Pi}_i\mathbb{R})$.

Proof sketch. The idea is to show that $\mathbf{\Sigma}_i$-$\textsc{Poly}_\mathbb{R}^\mathbb{Z}$ ($\mathbf{\Pi}_i$-$\textsc{Poly}_\mathbb{R}^\mathbb{Z}$) is complete for both $\mathbf{\Sigma}_i\mathbb{R}_\mathbb{Z}$ ($\mathbf{\Pi}_i\mathbb{R}_\mathbb{Z}$) and $\mathbf{BP}^0(\mathbf{\Sigma}_i\mathbb{R})$ ($\mathbf{BP}^0(\mathbf{\Pi}_i\mathbb{R})$). The first statement was already shown in Theorem 4.10. To prove the latter claim, we simply use the reduction from the proof of Theorem 4.14 since it does not use any other constants than 0 and 1 (cf. Thm. 3.17). ❑

4.7. Definition of (D)RPH Using Oracle BSS Machines

Finally, all technical and formal preparations have been finished and we are ready to define the **DRPH** using the BSS oracle model. Fortunately for us, the proof will resemble the one from the oracle definition of the classical polynomial hierarchy (cf. Theorem 2.19) — one needs to merely replace Turing machines with the BSS model, choose the appropriate complete problems as oracle sets (that is, $\mathbf{\Sigma}_i$-$\textsc{Poly}_\mathbb{R}$ and $\mathbf{\Pi}_i$-$\textsc{Poly}_\mathbb{R}$ instead of corresponding versions of Sat); and the variables' domain is $\mathbb{R}$ instead of $\{0,1\}$. Apart from that, the proof structure remains the same.

We first formulate an oracle definition of the unrestricted **RPH** — it can be easily adapted for the discretized hierarchy.

Definition 4.16 (Real polynomial hierarchy, **RPH**. Alternative definition using oracles)**.** *Having* $\mathbf{coNP}_\mathbb{R} = \mathbf{\Pi}_1\mathbb{R}$*, we define for* $i \in \mathbb{N}$*:*

$$\mathbf{\Sigma}_{i+1}\mathbb{R} := \mathbf{NP}_\mathbb{R}^{\mathbf{\Sigma}_i\mathbb{R}},$$
$$\mathbf{\Pi}_{i+1}\mathbb{R} := \mathbf{coNP}_\mathbb{R}^{\mathbf{\Sigma}_i\mathbb{R}}.$$

The following theorem shows the equivalence of this definition to Definition 4.11.

Theorem 4.17. *For $i \geq 2$, $\mathbf{\Sigma}_i\mathbb{R} = \mathbf{NP}_\mathbb{R}^{\mathbf{\Sigma}_{i-1}\text{-}\textsc{Poly}_\mathbb{R}}$. That is, $\mathbf{\Sigma}_i\mathbb{R}$ is the set of all languages decidable by a polynomial time nondeterministic BSS machine with access to a $\mathbf{\Sigma}_{i-1}$-$\textsc{Poly}_\mathbb{R}$-oracle.*

Proof. We proceed by induction on i.

Base case ($i = 2$). We need to show that $\mathbf{\Sigma}_2\mathbb{R} = \mathbf{NP}_\mathbb{R}^{\mathbf{\Sigma}_1\text{-}\textsc{Poly}_\mathbb{R}} (= \mathbf{NP}_\mathbb{R}^{4\text{-}\textsc{Feas}_\mathbb{R}})$.

"$\subseteq$": Suppose $L \in \mathbf{\Sigma}_2\mathbb{R}$, i.e., there exists a polynomial time BSS machine M and a polynomial p such that:

$$x \in L \iff \exists u_1 \in \mathbb{R}^{p(|x|)}\ \forall u_2 \in \mathbb{R}^{p(|x|)} :\ M(x, u_1, u_2) = 1.$$

We observe that for fixed x and u_1, the remaining statement is a $\mathbf{coNP}_\mathbb{R}$-statement and can thus be determined by a 4-$\textsc{Feas}_\mathbb{R}$-oracle. This means that we can construct a nondeterministic BSS machine N with a 4-$\textsc{Feas}_\mathbb{R}$-oracle deciding L: on the input x, we guess u_1 nondeterministically and then use the oracle to check if

$$\forall u_2 \in \mathbb{R}^{p(|x|)} :\ M(x, u_1, u_2) = 1.$$

This works since $x \in L$ if and only if there exists such u_1 that makes N accept.

"$\supseteq$": Suppose $L \in \mathbf{NP}_{\mathbb{R}}^{4\text{-}\textsc{Feas}_{\mathbb{R}}}$, meaning that L is decidable by a nondeterministic polynomial time 4-$\textsc{Feas}_{\mathbb{R}}$-oracle BSS machine. Since the nondeterminism is used for guessing the solutions, this is equivalent to the existence of a witness y verifiable by a deterministic BSS machine N with a 4-$\textsc{Feas}_{\mathbb{R}}$-oracle:

$$x \in L \iff \exists y \in \mathbb{R}^{p(|x|)} : N^{4\text{-}\textsc{Feas}_{\mathbb{R}}}(x, y) = 1$$

for some polynomial p.

The idea is to simulate the oracle using the additional $\forall$ quantifier. On each query q_i, if $q_i \in 4\text{-}\textsc{Feas}_{\mathbb{R}}$, then there exists a witness y' containing a satisfying assignment for q_i, that is, an assignment that sets q_i to zero. Conversely, if $q_i \notin 4\text{-}\textsc{Feas}_{\mathbb{R}}$, then $q_i \neq 0$ for *every* assignment.

We construct a BSS machine M without an oracle to simulate N. The new witness y' contains the original witness y and, additionally, the answers $a_i \in \{0, 1\}$ to oracle queries q_i. If the answer is positive ($a_i = 1$), then y' also contains a satisfying assignment $s_i \in \mathbb{R}^\infty$ for q_i, i.e., $q_i(s_i) = 0$. If the answer is negative ($a_i = 0$), then the "co-witness" z' should contain an unsatisfiable assignment u_i for the query. Thus, the "co-witness" z' represents (all) possible assignments for the queries with a negative answer. We can now construct the BSS machine M formally.

Algorithm 4.1: Simulating an oracle BSS machine with a co-witness.

Input: $x \in \mathbb{R}^\infty$, witness y', co-witness z', encoding of N.
Output: 1 if M accepts, 0 otherwise.

1 simulate N (copy every step till an oracle query)
2 **if** N makes oracle query q_i **then**
3 | **if** $a_i = 1$, verify that s_i is contained in y' and $q_i(s_i) = 0$
4 | **if** $a_i = 0$, REJECT if u_i in z' is a satisfying assignment for q_i, i.e., $q_i(u_i) = 0$
5 continue the simulation of N (step 1)
6 **if** all verifications are successful **and** N accepts, ACCEPT
7 **else** REJECT

We have:

$$x \in L \iff \exists y' \in \mathbb{R}^{p(|x|)}\ \forall z' \in \mathbb{R}^{p(|x|)} : M(x, y', z') = 1$$

for some polynomial p.

Note that $\text{TIME}(N) = \text{poly}(|x|)$, meaning that there are at most polynomially many oracle queries q_i. Furthermore, an assignment contains one value for each variable of the formula encoded in $|x|$, which means its length does not exceed the input length: $|s_i| = |u_i| \leq |x|$. We thus get $|y'|, |z'| \leq \text{poly}(|x|)$, i.e., the witness y' and the co-witness z' are polynomial in the input size.

Putting everything together yields $L \in \mathbf{\Sigma}_2\mathbb{R}$.

Induction assumption. $\mathbf{\Sigma}_{i-1}\mathbb{R} = \mathbf{NP}_{\mathbb{R}}^{\mathbf{\Sigma}_{i-2}\text{-}\textsc{Poly}_{\mathbb{R}}}$ for an arbitrary but fixed $i > 2$, i.e., any language in $\mathbf{NP}^{\mathbf{\Sigma}_{i-2}\text{-}\textsc{Poly}_{\mathbb{R}}}$ can be written with $(i-1)$ alternating quantifiers, starting with an existential one.

Induction step. We need to show that $\mathbf{\Sigma}_i\mathbb{R} = \mathbf{NP}_{\mathbb{R}}^{\mathbf{\Sigma}_{i-1}\text{-}\textsc{Poly}_{\mathbb{R}}}$.

"$\subseteq$": The left inclusion is again easy. Let L be in $\mathbf{\Sigma}_i\mathbb{R}$ and recall that

$$x \in L \iff \exists u_1 \in \mathbb{R}^{p(|x|)}\ \forall u_2 \in \mathbb{R}^{p(|x|)}\ \ldots\ Q_i u_i \in \mathbb{R}^{p(|x|)} : M(x, u_1, \ldots, u_i) = 1.$$

Just like in the base case, for fixed x and u_1, the remaining statement is a $\mathbf{co\Sigma}_{i-1}\mathbb{R}$-statement and can thus be verified with a $\mathbf{\Sigma}_{i-1}$-$\textsc{Poly}_{\mathbb{R}}$-oracle.

"$\supseteq$": The right inclusion requires a little bit more work, but we follow the same idea as in the base case. Suppose $L \in \mathbf{NP}_{\mathbb{R}}^{\mathbf{\Sigma}_{i-1}\text{-}\textsc{Poly}_{\mathbb{R}}}$, that is, L is decidable by a nondeterministic polynomial time $\mathbf{\Sigma}_{i-1}$-$\textsc{Poly}_{\mathbb{R}}$-oracle BSS machine, which is equivalent to the existence of a witness y verifiable by a deterministic BSS machine N with a $\mathbf{\Sigma}_{i-1}$-$\textsc{Poly}_{\mathbb{R}}$-oracle:

$$x \in L \iff \exists y \in \mathbb{R}^{p(|x|)} : N^{\mathbf{\Sigma}_{i-1}\text{-}\textsc{Poly}_{\mathbb{R}}}(x, y) = 1$$

for some polynomial p.

We want to construct a BSS machine M without an oracle for simulating N. The new witness y' contains the original witness y and, additionally, the answer $a_j \in \{0, 1\}$ for each query q_j. If $a_j = 1$, — that is, $q_j \in \mathbf{\Sigma}_{i-1}$-$\textsc{Poly}_{\mathbb{R}}$ — then

$$\exists y'_{j,1} \in \mathbb{R}^{p(|x|)}\ \forall y'_{j,2} \in \mathbb{R}^{p(|x|)} : T_1^{\mathbf{\Sigma}_{i-2}\text{-}\textsc{Poly}_{\mathbb{R}}}(q_j, y'_{j,1}, y'_{j,2}) = 1$$

for some BSS machine T_1 with access to a $\mathbf{\Sigma}_{i-2}$-$\textsc{Poly}_{\mathbb{R}}$-oracle and some polynomial p.

By induction assumption, we can simulate T_1 with alternating quantifiers and a BSS machine without an oracle, meaning that we can "unfold" the above statement — we omit the variables' domain $\mathbb{R}^{p(|x|)}$ for better readability:

$$\exists y'_{j,1}\ \forall y'_{j,2}\ \ldots\ Q_{i-1} y'_{j,i-1} : y'_{j,1}, \ldots, y'_{j,i-1} \text{ is a satisfying assignment for } q_j.$$

Conversely, if $a_j = 0$ ($q_j \notin \mathbf{\Sigma}_{i-1}$-$\textsc{Poly}_{\mathbb{R}}$), then

$$\forall z'_{j,2} \in \mathbb{R}^{p(|x|)}\ \exists z'_{j,3} \in \mathbb{R}^{p(|x|)} : T_2^{\mathbf{\Sigma}_{i-2}\text{-}\textsc{Poly}_{\mathbb{R}}}(q_j, z'_{j,2}, z'_{j,3}) = 1$$

for some BSS machine T_2 with a $\mathbf{\Sigma}_{i-2}$-$\textsc{Poly}_{\mathbb{R}}$-oracle and some polynomial p.

Again, by induction assumption, we can simulate T_2 with an oracle-free BSS machine and thereby unfold the above statement (the domain is omitted):

$$\forall z'_{j,2}\ \exists z'_{j,3}\ \ldots\ Q_i z'_{j,i} : z'_{j,2}, \ldots, z'_{j,i} \text{ is } \textit{not} \text{ a satisfying assignment for } q_j.$$

Note that in the latter case we numerate the quantifiers from 2 to i — this is done for convenience. One can merge the two statements by reformulating:

$$\begin{aligned}
&\exists y'_{j,1}\ \forall (y'_{j,2}, z'_{j,2})\ \ldots\ Q_{i-1}(y'_{j,i-1}, z'_{j,i-1})\, Q_i z'_{j,i} : \\
&\qquad (a_j = 1 \text{ AND } y'_{j,1}, \ldots, y'_{j,i-1} \text{ is a satisfying assignment for the query } q_j) \\
&\qquad \text{OR} \\
&\qquad (a_j = 0 \text{ AND } z'_{j,2}, \ldots, z'_{j,i} \text{ is not a satisfying assignment for the query } q_j).
\end{aligned}$$

This means that we have the desired quantifiers structure. We can now construct the BSS machine M for simulating the oracle formally.

Algorithm 4.2: Simulating a $\mathbf{\Sigma}_{i-1}$-$\textsc{Poly}_{\mathbb{R}}$-oracle BSS machine with an oracle-free BSS machine.

Input: $x \in \mathbb{R}^\infty$, (co-)witnesses y' and z', encoding of N.
Output: 1 if M accepts, 0 otherwise.

1 simulate N (copy every step till an oracle query)
2 **if** N makes oracle query q_j **then**
3 | **if** $a_j = 1$, verify that $y'_{j,1}, \ldots, y'_{j,i-1}$ in y' is a satisfying assignment for q_j
4 | **if** $a_j = 0$, REJECT if $z'_{j,2}, \ldots, z'_{j,i}$ in z' is a satisfying assignment for q_j
5 continue the simulation of N (step 1)
6 **if** all verifications are successful **and** N accepts, ACCEPT
7 **else** REJECT

This gives us

$$x \in L \iff \exists y'_1 \; \forall (y'_2, z'_2) \; \ldots \; Q_{i-1}(y'_{i-1}, z'_{i-1}) \; Q_i z'_i : M(x, y'_1, \ldots, z'_i) = 1.$$

The argumentation for the polynomial length of the (co-)witnesses is the same as in the base case. We thus get $L \in \mathbf{\Sigma}_i\mathbb{R}$, which finishes the proof. ∎

Note that we do not use real-valued constants (in fact, we do not use any other constants than 0 and 1) in the above proof — it is thus can be used to define **DRPH**.

Corollary 4.18 (Alternative definition of **DRPH** using oracles)**.** *For $i \in \mathbb{N}$, we define:*

$$\mathbf{\Sigma}_{i+1}\mathbb{R}_{\mathbb{Z}} = \exists \mathbb{R}^{\mathbf{\Sigma}_i \mathbb{R}_{\mathbb{Z}}},$$
$$\mathbf{\Pi}_{i+1}\mathbb{R}_{\mathbb{Z}} = \forall \mathbb{R}^{\mathbf{\Sigma}_i \mathbb{R}_{\mathbb{Z}}}.$$

To prove the corollary, one simply uses the idea of the above proof, but chooses the discretized versions of the problems as oracle sets — that is, $\mathbf{\Sigma}_{i-1}$-$\textsc{Poly}_{\mathbb{R}}^{\mathbb{Z}}$ instead of the unrestricted version $\mathbf{\Sigma}_{i-1}$-$\textsc{Poly}_{\mathbb{R}}$.

Alternatively, one can discretize both sides of the equations from the oracle definition of **RPH** (Definition 4.16) — when discretizing a complexity class defined via an oracle model, both the base and the oracle classes are discretized:

$$\begin{aligned}
\mathbf{\Sigma}_{i+1}\mathbb{R}_{\mathbb{Z}} &= \mathbf{BP}^0(\mathbf{\Sigma}_{i+1}\mathbb{R}) \\
&= \mathbf{BP}^0(\mathbf{NP}_{\mathbb{R}}^{\mathbf{\Sigma}_i \mathbb{R}}) \\
&= \mathbf{BP}^0(\mathbf{NP}_{\mathbb{R}})^{\mathbf{\Sigma}_i \mathbb{R}_{\mathbb{Z}}} \\
&= \exists \mathbb{R}^{\mathbf{\Sigma}_i \mathbb{R}_{\mathbb{Z}}}.
\end{aligned}$$

4.8. Definition of DRPH with the (Oracle) Real RAM

Since the algorithmic membership in $\exists\mathbb{R}$ is usually proven using the real RAM, it makes sense to consider this model for the higher levels of **DRPH** as well — it might facilitate the construction of verification algorithms. In this section, we take a look on how **DRPH** can be defined with the real RAM and its oracle extension.

Theorem 4.19 (Alternative definition of **DRPH** with real RAM)**.** *Let $L \subseteq \{0,1\}^*$, $i \in \mathbb{N}$. $L \in \mathbf{\Sigma}_i\mathbb{R}_\mathbb{Z}$ if there exists a polynomial time real RAM R and a polynomial p such that for every $x \in \{0,1\}^*$:*

$$x \in L \iff \exists u_1 \in \mathbb{R}^{p(|x|)}\ \forall u_2 \in \mathbb{R}^{p(|x|)} \dots\ Q_i u_i \in \mathbb{R}^{p(|x|)} : R(x, u_1, \dots, u_i) = 1,$$

where $Q_i = \begin{cases} \forall, & i \text{ is even,} \\ \exists, & i \text{ is odd.} \end{cases}$

Similarly, $L \in \mathbf{\Pi}_i\mathbb{R}_\mathbb{Z}$ if there exists a polynomial time real RAM R and a polynomial p such that for every $x \in \{0,1\}^*$:

$$x \in L \iff \forall u_1 \in \mathbb{R}^{p(|x|)}\ \exists u_2 \in \mathbb{R}^{p(|x|)} \dots\ Q_i u_i \in \mathbb{R}^{p(|x|)} : R(x, u_1, \dots, u_i) = 1,$$

where $Q_i = \begin{cases} \exists, & i \text{ is even,} \\ \forall, & i \text{ is odd.} \end{cases}$

Proof sketch. To prove the theorem, one can simply use the analog of the Cook–Levin theorem for the real RAM (Theorem 3.20). Note that its proof does not use any assumptions on how the (co-)witnesses in the input are quantified. Dobbins et al. [DKMR23] use this observation to show the validity of this definition for the class $\forall\exists\mathbb{R}$, but it can easily be extended to the higher levels of **DRPH**. This means that we can simply transfer the proof to an arbitrary quantifier structure, thereby showing the completeness of $\mathbf{\Sigma}_i$-ETR and $\mathbf{\Pi}_i$-UTR for the corresponding complexity classes defined using the real RAM model and (co-)witnesses. Recall that $\mathbf{\Sigma}_i$-ETR and $\mathbf{\Pi}_i$-UTR are per definition complete for $\mathbf{\Sigma}_i\mathbb{R}_\mathbb{Z}$ and $\mathbf{\Pi}_i\mathbb{R}_\mathbb{Z}$, respectively — thus, the equivalence of the definitions follows. ❑

Having the oracle real RAM, one can prove Corollary 4.18 using it. The proof is entirely analogous to the one of Theorem 4.17, but one can choose $\mathbf{\Sigma}_i$-ETR directly as the oracle set — without supplementary case distinctions or reductions. Note that no further restrictions or discretization steps are needed since one obtains the result directly for the desired classes, which is another advantage of using the oracle real RAM model:

$$\mathbf{\Sigma}_{i+1}\mathbb{R}_\mathbb{Z} = \exists\mathbb{R}^{\mathbf{\Sigma}_i\text{-ETR}} = \exists\mathbb{R}^{\mathbf{\Sigma}_i\mathbb{R}_\mathbb{Z}},$$
$$\mathbf{\Pi}_{i+1}\mathbb{R}_\mathbb{Z} = \forall\mathbb{R}^{\mathbf{\Sigma}_i\text{-ETR}} = \forall\mathbb{R}^{\mathbf{\Sigma}_i\mathbb{R}_\mathbb{Z}}.$$

4.9. The Zeroth Level of the Hierarchy

So far, we have considered the higher levels of **DRPH**. We have shown that $\mathbf{\Sigma}_i\mathbb{R}_\mathbb{Z} = \mathbf{BP}^0(\mathbf{\Sigma}_i\mathbb{R})$ and $\mathbf{\Pi}_i\mathbb{R}_\mathbb{Z} = \mathbf{BP}^0(\mathbf{\Pi}_i\mathbb{R})$ for $i \in \mathbb{N}$ (Theorem 4.15). One might intuitively think that this result also holds for the zeroth level, that is, $\varnothing\mathbb{R} \stackrel{?}{=} \mathbf{BP}(\mathbf{P}^0_\mathbb{R})$. Alas, it is presumably not the case.

In fact, there is no unique candidate for the zeroth level of **DRPH** — both $\varnothing\mathbb{R}$ and $\mathbf{BP}(\mathbf{P}^0_\mathbb{R})$ are valid contenders for this "position". In this section, we discuss the advantages and disadvantages of choosing each class.

The complexity class $\varnothing\mathbb{R}$ is defined by making $\mathbf{\Sigma}_0$-$\textsc{Poly}_{\mathbb{R}}^{\mathbb{Z}}$ a complete problem. Since there are no free variables and no quantified ones, there are no real-valued variables at all — the problem simply consists of evaluating a polynomial expression with no variables and checking whether it is (un)equal to zero. An example of such an instance would be

$$1 \cdot 1 + 0 \cdot 1 - 1 = 0.$$

The problem is equivalent to the Circuit Value Problem (CVP), which deals with evaluating the output of a given Boolean circuit on a given input — or an arithmetic circuit since both computational models are equivalent over $\mathbb{Q}$ and finite fields [vS91].

A problem is said to be **P**-complete if it lies in **P** and any other problem in **P** can be reduced to it. However, to define **P**-completeness, one needs more restrictive reductions — polynomial time reductions do not suffice because any non-trivial problem in **P** would also be **P**-complete.

There are two common choices for such reductions. The first one are *log-space reductions* — that is, the reductions computable by a deterministic Turing machine in polynomial time with logarithmic extra space usage [AB06]. It is important to note that the TM is still allowed to read the input and write the output and is not "charged space" for it. Formally, one can define a dedicated model for space-bounded computation, which is suited for performing log-space reductions. It is called a *log space transducer* (LST) and is a Turing machine with a read-only input tape, (logarithmically) space-bounded read/write work tapes and a write-once output tape [Sze94]. Note that it is not necessary to require the runtime to be polynomial since such a machine always runs in polynomial time [Sze94].

Another option are so-called **NC** *reductions* (or *Nick's Class*, named after Nicholas Pippenger [Pip79]) that are computable in polylogarithmic time $(\mathcal{O}(\log^c n))$ on polynomially many processors $(\mathcal{O}(n^k))$ for some constants $c, k \in \mathbb{R}$; n denotes the input size.

The CVP was shown to be **P**-complete under log-space reductions, that is, it lies in **P** and every other problem in **P** is reducible to CVP in polynomial time using logarithmic space [Lad75]. Since the same problem is per definition $\varnothing\mathbb{R}$-complete, it follows that $\varnothing\mathbb{R} = \mathbf{P}$.

When it comes to the unrestricted class $\mathbf{P}_{\mathbb{R}}$, the problem $\mathbf{\Sigma}_0$-$\textsc{Poly}_{\mathbb{R}}$ becomes the Real Circuit Value Problem ($\text{CVP}_{\mathbb{R}}$), where the circuit is allowed to have real-valued input [BC09], which was shown to be $\mathbf{P}_{\mathbb{R}}$-complete under **NC** reductions by Cucker and Torrecillas [CT91].

Let us now consider the problem PosSLP (*positive straight-line program*), where we are given an arithmetic circuit with constants $\{0, 1\}$ and operations $\{+, -, \times\}$ computing an integer N and we need to decide whether $N > 0$. Note that the are no variables in the input.

Theorem 4.20 (Allender et al. [AKBM06])**.** $\mathbf{BP}(\mathbf{P}_{\mathbb{R}}^0) = \mathbf{P}^{PosSLP}$.

Proof sketch. We follow the proof of [AKBM06].

"$\supseteq$": We simulate a classical Turing machine with a BSS machine. On a query to the PosSLP oracle, we can evaluate the circuit using the BSS model in polynomial time and use a branching instruction to determine whether the result is positive.

"$\subseteq$": We simulate the computational process of the BSS machine, but do not compute the results directly. The registers' content (state space) is stored as arithmetic circuits. For simulating an operation on registers, we combine two corresponding circuits to a new one. For division, we have two separate circuits for the nominator and denominator. To simulate a branching instruction, we query the PosSLP oracle to determine whether the content of a given register (represented by an arithmetic circuit) is positive. ❑

The intuitive reason why restricted BSS machines are more powerful than classical Turing machines — even if we take real numbers out — is the branching instruction allowing to check whether the content of a given register is positive. The real RAM possesses such branching instruction as well, meaning that one can apply the proof idea to show the same result for the zeroth level defined with the real RAM model, the result would still be $\mathbf{BP}(\mathbf{P}^0_{\mathbb{R}})$ (or $\mathbf{P}^{\text{PosSLP}}$), analogously to the BSS model.

Allender et al. [AKBM06] showed the placement of PosSLP within the counting hierarchy **CH** [AW97]:

$$\text{PosSLP} \in \mathbf{P}^{\mathbf{PP}^{\mathbf{PP}^{\mathbf{PP}}}} \subseteq \mathbf{CH}.$$

PP stands for *probabilistic polynomial time* and denotes the complexity class of all problems solvable by a probabilistic Turing machine with an error probability of less than $\frac{1}{2}$ [Gil77]. A probabilistic Turing machine is a TM that can choose the next configuration based on some probability distribution [Gil77].

Furthermore, Toda showed that $\mathbf{PH} \subseteq \mathbf{P}^{\mathbf{PP}}$. This means that **PP** is "as hard as the polynomial hierarchy": **PP** does not lie in **PH** unless **PH** collapses to a finite level [Tod91]. This implies that $\mathbf{P}^{\mathbf{PP}}$ is probably unequal to **P**, hence, PosSLP $\notin \mathbf{P}$ and $\mathbf{P}^{\text{PosSLP}} \neq \mathbf{P}$.

This gives us reason to believe that $\mathbf{BP}(\mathbf{P}^0_{\mathbb{R}}) \neq \varnothing\mathbb{R}$ (unless the polynomial hierarchy collapses):

$$\mathbf{BP}(\mathbf{P}^0_{\mathbb{R}}) = \mathbf{P}^{\text{PosSLP}} \overset{?}{\neq} \mathbf{P} = \varnothing\mathbb{R}.$$

The question remains how one should define the zeroth level of **DRPH**. On the one hand, if we define the hierarchy via complete problems, the consistent way to define the predecessor of $\exists\mathbb{R}$ and $\forall\mathbb{R}$ would be to define it as $\varnothing\mathbb{R}$. However, it does not add any additional interest to studying this class since $\varnothing\mathbb{R} = \mathbf{P}$.

On the other hand, if we define **DRPH** using BSS machines or the real RAM, it would be sensible to define the zeroth level as $\mathbf{BP}(\mathbf{P}^0_{\mathbb{R}}) = \mathbf{P}^{\text{PosSLP}}$ — note that $\mathbf{BP}(\mathbf{P}^0_{\mathbb{R}})$ is contained both in $\mathbf{BP}(\mathbf{NP}^0_{\mathbb{R}})$ and $\mathbf{BP}(\mathbf{coNP}^0_{\mathbb{R}})$; moreover, the class is presumably more "powerful" than the classical **P**.

5. Conclusion

The thesis aimed at defining the discrete real polynomial hierarchy **DRPH** with an oracle model — we have accomplished this task using both BSS machines and the real RAM model.

We discussed how the aforementioned computational models can be extended with an oracle: we summarized the existing results for oracle BSS machines and introduced the oracle real RAM — this is an important contribution of this work. We then showed how the complexity class $\exists\mathbb{R}$ can be defined with BSS machines and the real RAM. This is attained with the help of real analogs of the Cook-Levin theorem, which we have explored deeply for the BSS model. Eventually, we defined **DRPH** with oracle BSS and real random access machines, which is a central result of the thesis.

While the definition with oracle BSS machines seems more natural because of the proximity to oracle Turing machines, the definition via the oracle real RAM is more practical since the model is considered the standard in computational geometry.

5.1. Future Work

Having these definitions, it can be easier to explore higher levels of **DRPH** — until now, most research was focused on the first two levels; little research has been done for the third and higher levels.

Furthermore, the oracle definitions allow the usage of the "oracle" superscript notation, e.g., $\forall\mathbb{R}^{\exists\mathbb{R}}$. It facilitates the exploration of so-called hybrid classes from **DRPH**, where some quantifiers are Boolean, because it allows an easier construction of verification algorithms for membership proofs. For instance, deciding the existence of an evolutionary stable strategy in multi-player symmetric games lies in $\exists\forall\mathbb{R}$, but real-valued variables in the existential block are not needed [BH22], meaning that we could write $\mathbf{NP}^{\forall\mathbb{R}}$. It might be interesting to investigate which problems from higher levels of **DRPH** may have Boolean "intermediate stages".

It is known that $\mathbf{PH} \subseteq \mathbf{DRPH} \subsetneq \mathbf{RPH}$, where the statement holds for each level of the corresponding hierarchies, e.g., $\mathbf{NP} \subseteq \exists\mathbb{R} \subsetneq \mathbf{NP}_{\mathbb{R}}$. The interplay between different hierarchies — i.e., how similar or different are the classes from the corresponding levels — is worth considering for future research. We have seen that $\mathbf{DRPH} = \mathbf{BP}^0(\mathbf{RPH})$ — again, the discretization steps are applied to each level. Another thought-provoking idea would be to examine what happens if only one discretization step is applied to **RPH**, that is, how powerful are $\mathbf{BP}(\mathbf{RPH})$ and $\mathbf{RPH}^0$.

One might also contemplate experimenting with different types of oracles, e.g., by extending BSS machines or the real RAM with a random oracle and inspecting how powerful such computational models would be.

Bibliography

[AAM17] Mikkel Abrahamsen, Anna Adamaszek, and Tillmann Miltzow. "The Art Gallery Problem is $\exists\mathbb{R}$-complete". In: *CoRR* Volume abs/1704.06969 (2017). arXiv: *1704.06969*.

[AB06] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2006. ISBN: 978-0-521-42426-4.

[AKBM06] E. Allender, J. Kjeldgaard-Pedersen, P. Burgisser, and P. Miltersen. "On the complexity of numerical analysis". In: *21st Annual IEEE Conference on Computational Complexity (CCC'06)*. 2006, 9 pp.–339. DOI: *10.1109/CCC.2006.30*.

[AKM21] Mikkel Abrahamsen, Linda Kleist, and Tillmann Miltzow. "Training Neural Networks is ER-complete". In: *CoRR* Volume abs/2102.09798 (2021). arXiv: *2102.09798*.

[AM19] Mikkel Abrahamsen and Tillmann Miltzow. "Dynamic Toolbox for ETRINV". In: *CoRR* Volume abs/1912.08674 (2019). arXiv: *1912.08674*.

[AMS22] Mikkel Abrahamsen, Tillmann Miltzow, and Nadja Seiferth. *Framework for $\exists\mathbb{R}$-Completeness of Two-Dimensional Packing Problems*. 2022. arXiv: *2004.07558*.

[AW97] Eric Allender and Klaus Wagner. "Counting Hierarchies: Polynomial Time And Constant Depth Circuits". In: Volume 40 (Nov. 1997).

[BC06] Peter Bürgisser and Felipe Cucker. "Counting complexity classes for numeric computations II: Algebraic and semialgebraic sets". In: *Journal of Complexity* Volume 22 (2006), pp. 147–191. ISSN: 0885-064X. DOI: *https://doi.org/10.1016/j.jco.2005.11.001*.

[BC09] Peter Bürgisser and Felipe Cucker. "Exotic Quantifiers, Complexity Classes, and Complete Problems". English. In: *Foundations of Computational Mathematics* Volume 9 (Apr. 2009), pp. 135–170. ISSN: 1615-3375. DOI: *10.1007/s10208-007-9006-9*.

[BCSS98] Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale. *Complexity and real computation*. Berlin, Heidelberg: Springer-Verlag, 1998. ISBN: 0387982817.

[Ber+22] Daniel Bertschinger, Christoph Hertrich, Paul Jungeblut, Tillmann Miltzow, and Simon Weber. *Training Fully Connected Neural Networks is $\exists\mathbb{R}$-Complete*. 2022. arXiv: *2204.01368*.

[BGS75] Theodore Baker, John Gill, and Robert Solovay. "Relativizations of the $\mathcal{P} =? \mathcal{NP}$ Question". In: *SIAM Journal on Computing* Volume 4 (1975), pp. 431–442. eprint: *https://doi.org/10.1137/0204037*.

[BH22] Manon Blanc and Kristoffer Arnsfelt Hansen. *Computational Complexity of Multi-Player Evolutionarily Stable Strategies*. 2022. arXiv: *2203.07407*.

[BKR86] Michael Ben-Or, Dexter Kozen, and John H. Reif. "The Complexity of Elementary Algebra and Geometry". In: *J. Comput. Syst. Sci.* Volume 32 (1986), pp. 251–264. DOI: *10.1016/0022-0000(86)90029-2*.

[BSS89] Lenore Blum, Mike Shub, and Steve Smale. "On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines". In: *Bulletin (New Series) of the American Mathematical Society* Volume 21 (1989), pp. 1–46.

[Can88] John Canny. "Some Algebraic and Geometric Computations in PSPACE". In: *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*. Chicago, Illinois, USA: Association for Computing Machinery, 1988, pp. 460–467. ISBN: 0897912640. DOI: *10.1145/62212.62257*.

[CKM10] Wesley Calvert, Ken Kramer, and Russell G. Miller. "The Cardinality of an Oracle in Blum-Shub-Smale Computation". In: *Proceedings Seventh International Conference on Computability and Complexity in Analysis, CCA 2010, Zhenjiang, China, 21-25th June 2010*. Edited by Xizhong Zheng and Ning Zhong. Vol. 24. 2010, pp. 56–66. DOI: *10.4204/EPTCS.24.10*.

[Cou23] Michael Coulombe. *You're Too Slow! The Case for Constant-time Algorithms*. 2023.

[CT91] Felipe Cucker and A. Torrecillas. "Two P-Complete Problems in the Theory of the Reals". In: *Proceedings of the 18th International Colloquium on Automata, Languages and Programming*. Berlin, Heidelberg: Springer-Verlag, 1991, pp. 556–565. ISBN: 3540542337.

[DKMR23] Michael Gene Dobbins, Linda Kleist, Tillmann Miltzow, and Paweł Rzążewski. "Completeness for the Complexity Class $\forall\exists\mathbb{R}$ and Area- Universality". In: *Discrete & Computational Geometry* Volume 70 (July 2023), pp. 154–188. ISSN: 1432-0444. DOI: *10.1007/s00454-022-00381-0*.

[EvM19] Jeff Erickson, Ivor van der Hoog, and Tillmann Miltzow. "A Framework for Robust Realistic Geometric Computations". In: *CoRR* Volume abs/1912.02278 (2019). arXiv: *1912.02278*.

[Gil77] John Gill. "Computational Complexity of Probabilistic Turing Machines". In: *SIAM J. Comput.* Volume 6 (Dec. 1977), pp. 675–695. ISSN: 0097-5397. DOI: *10.1137/0206049*.

[Hen49] Leon Henkin. "Alfred Tarski. A decision method for elementary algebra and geometry. U. S. Air Force Project Rand, R-109. Prepared for publication by J. C. C. McKinsey. Litho-printed. The Rand Corporation, Santa Monica, California, 1948, iii 60 pp." In: *Journal of Symbolic Logic* Volume 14 (1949), pp. 188–188. DOI: *10.2307/2267068*.

[Jun23a] Paul Jungeblut. *On the Complexity of Lombardi Graph Drawing*. 2023. arXiv: *2306.02649*.

[Jun23b] Tim Junginger. *Robustness of the Discrete Real Polynomial Hierarchy*. 2023.

[Lad75] Richard E. Ladner. "The circuit value problem is log space complete for P". In: *SIGACT News* Volume 7 (Jan. 1975), pp. 18–20. ISSN: 0163-5700. DOI: *10.1145/990518.990519*.

[Mat14] Jiří Matoušek. *Intersection graphs of segments and* $\exists\mathbb{R}$. 2014. arXiv: *1406.2636*.

[Mou02] Lucia Moura. "Introduction to the theory of NP-completeness". In: *University of Ottawa Lectures* (2002).

[MW19] Dylan M. McKay and Richard Ryan Williams. "Quadratic Time-Space Lower Bounds for Computing Natural Functions with a Random Oracle". In: *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*. Edited by Avrim Blum. Vol. 124. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019, 56:1–56:20. ISBN: 978-3-95977-095-8. DOI: *10.4230/LIPIcs.ITCS.2019.56*.

[MZ05] Klaus Meer and Martin Ziegler. "An Explicit Solution to Post's Problem over the Reals". In: *Fundamentals of Computation Theory*. Edited by Maciej Liśkiewicz and Rüdiger Reischuk. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 467–478. ISBN: 978-3-540-31873-6.

[ODo17] Ryan O'Donnell. *Undergraduate Complexity Theory*. Carnegie Mellon University, 2017.

[Pip79] Nicholas Pippenger. "On simultaneous resource bounds". In: *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*. 1979, pp. 307–311. DOI: *10.1109/SFCS.1979.29*.

[Sch79] Arnold Schönhage. "On the power of random access machines". In: *Automata, Languages and Programming*. Edited by Hermann A. Maurer. Berlin, Heidelberg: Springer Berlin Heidelberg, 1979, pp. 520–529. ISBN: 978-3-540-35168-9.

[Shi17] Yaroslav Shitov. "The Complexity of Positive Semidefinite Matrix Factorization". In: *SIAM Journal on Optimization* Volume 27 (2017), pp. 1898–1909. eprint: *https://doi.org/10.1137/16M1080616*.

[Sho90] Peter W. Shor. "Stretchability of Pseudolines is NP-Hard." In: *Applied Geometry And Discrete Mathematics, Proceedings of a DIMACS Workshop, Providence, Rhode Island, USA, September 18, 1990*. Edited by Peter Gritzmann and Bernd Sturmfels. Vol. 4. DIMACS/AMS, 1990, pp. 531–554. DOI: *10.1090/DIMACS/004/41*.

[SŠ17] Marcus Schaefer and Daniel Štefankovič. "Fixed Points, Nash Equilibria, and the Existential Theory of the Reals". In: *Theory of Computing Systems* Volume 60 (Feb. 2017). DOI: *10.1007/s00224-015-9662-0*.

[SŠ23] Marcus Schaefer and Daniel Štefankovič. *Beyond the Existential Theory of the Reals*. 2023. arXiv: *2210.00571*.

[Sto76] Larry J. Stockmeyer. "The polynomial-time hierarchy". In: *Theoretical Computer Science* Volume 3 (1976), pp. 1–22. ISSN: 0304-3975.

[Sze94] Andrzej Szepietowski. “Turing Machines with Sublogarithmic Space”. In: *Lecture Notes in Computer Science*. 1994.

[Tod91] Seinosuke Toda. “PP is as Hard as the Polynomial-Time Hierarchy”. In: *SIAM Journal on Computing* Volume 20 (1991), pp. 865–877. eprint: *https://doi.org/10.1137/0220053*.

[Tri90] Eberhard Triesch. “A note on a theorem of Blum, Shub, and Smale”. In: *Journal of Complexity* Volume 6 (1990), pp. 166–169. ISSN: 0885-064X. DOI: *https://doi.org/10.1016/0885-064X(90)90004-W*.

[Tse68] G. S. Tseitin. “On the complexity of derivation in the propositional calculus”. In: *Zap. Nauchn. Semin. LOMI, Leningrad: Nauka* Volume 8 (1968), pp. 234–259 (in Russian).

[vS91] Joachim von zur Gathen and Gadiel Seroussi. “Boolean circuits versus arithmetic circuits”. In: *Information and Computation* Volume 91 (1991), pp. 142–154. ISSN: 0890-5401. DOI: *https://doi.org/10.1016/0890-5401(91)90078-G*.