On Equivalent Characterizations of the Polynomial Hierarchy in Abstract Models of Computation

Jeremy C. Kirn
Utrecht University, The Netherlands

Lucas Meijer
Utrecht University, The Netherlands

Tillmann Miltzow
Utrecht University, The Netherlands

Hans L. Bodlaender
Utrecht University, The Netherlands

Abstract

We investigate machine models similar to Turing machines that are augmented with the operations of a first-order structure $\mathcal{R}$, and we show that under weak conditions on $\mathcal{R}$, the complexity class $\Sigma_k\mathcal{R}$ may be characterized in four equivalent ways: (1) by polynomial-time algorithms implemented on $\mathcal{R}$-machines together with witness strings, (2) by the $\Sigma_k\mathcal{R}$-complete problem $\Sigma_k\mathrm{SAT}(\mathcal{R})$, (3) by kth existential fragment second-order metafinite logic over $\mathcal{R}$ via descriptive complexity, and (4) via oracles. By characterizing $\Sigma_k\mathcal{R}$ in these four ways, we extend previous work and embed it in one coherent framework. In addition, we derive similar results for $\exists_k\mathcal{R}$, the constant-free Boolean part of $\Sigma_k\mathcal{R}$, by showing that $\exists_k\mathcal{R}$ may be characterized in four analogous ways.

Some conditions on $\mathcal{R}$ must be assumed in order to achieve the above quaternity because there are infinite-vocabulary structures for which $\mathrm{NP}(\mathcal{R}) = \Sigma_1\mathcal{R}$ does not have a complete problem. Surprisingly, even in these cases, we show that $\mathrm{NP}(\mathcal{R})$ *does* have a characterization in terms of existential second-order metafinite logic, suggesting that descriptive complexity theory is well suited to working with infinite-vocabulary structures, such as real vector spaces.

2012 ACM Subject Classification Theory of computation → Abstract machines

Keywords and phrases Machines over a first-order structure, BSS machines, Cook Levin, Fagin, NP, existential theory of the reals, polynomial hierarchy, metafinite model theory, descriptive complexity, oracles

Digital Object Identifier 10.4230/LIPIcs.MFCS.2026.63

Related Version *Full Version*: `https://arxiv.org/abs/2510.05894` [28]

Funding *Jeremy C. Kirn*: generously supported by the Netherlands Organisation for Scientific Research (NWO) under project no. VI.Vidi.213.150.
Lucas Meijer: generously supported by the Netherlands Organisation for Scientific Research (NWO) under project no. VI.Vidi.213.150.
Tillmann Miltzow: generously supported by the Netherlands Organisation for Scientific Research (NWO) under project no. VI.Vidi.213.150.

Acknowledgements The first author gratefully acknowledges conversations with Nima Motamed.

1 Introduction

Algorithms are often presented in terms of primitive operations, and the complexity of these algorithms is often measured by counting how many times the primitive operations are applied. Ignoring the implementation details of these primitive operations is an abstraction that simplifies and enhances algorithm analysis. This abstraction is common in scientific computing, wherein algorithms are presented in terms of operations on the real numbers such

as arithmetic and trigonometric functions, the exponential function, and others [2]. It is also common in algebraic dynamic programming, wherein algorithms are presented in terms of operations that are interpreted in different semirings to solve different problems [18]. Already at this level of abstraction, it is infeasible to solve some algorithmic problems because they are complete for some higher level, Σ_k or Π_k, of a version of the polynomial hierarchy that depends on these operations. We describe an example of this phenomenon in a way that motivates our work, and we then provide an overview of our contribution.

1.1 Motivation

The polynomial hierarchy PH contains some of the most important complexity classes in theoretical computer science. We may characterize the kth existential level Σ_k in at least four different ways, resulting in a quaternity:

1. The class Σ_k consists of decision problems $L \subseteq \{0,1\}^*$ for which there exists a polynomial-time Turing machine M and a polynomial q such that for all $v \in \{0,1\}^n$,

$$v \in L \iff (Q_1 w_1 \in R^{q(n)}) \dots (Q_k w_k \in R^{q(n)}) M(v, w_1, \dots, w_k) = 1,$$

   where the quantifiers $Q_i \in \{\exists, \forall\}$ alternate, starting with $Q_1 = \exists$. This is usually taken as the definition of Σ_k [1]. Although one may define Σ_k equivalently in terms of polynomial-time nondeterministic Turing machines, we work with the above definition of Σ_k because it generalizes in a more straightforward manner to our setting of interest.
2. The class Σ_k consists of decision problems $L \subseteq \{0,1\}^*$ that can be reduced by a polynomial-time Turing machine to $\Sigma_k\text{SAT} \in \Sigma_k$, where $\Sigma_k\text{SAT}$ is the problem of deciding if a quantified Boolean formula, with $k-1$ quantifier alternations starting with $\exists$, is satisfied. The Cook-Levin theorem establishes this characterization of $\Sigma_1 = \text{NP}$, which is expressed by saying that SAT is NP-complete [8][31], and the Σ_k-completeness of $\Sigma_k\text{SAT}$ was established in [43].
3. The class Σ_k consists of decision problems $L \subseteq \{0,1\}^*$ for which there is a sentence Φ of the kth existential fragment of second-order logic $\Sigma_k\text{SO}$ describing L. Fagin's theorem establishes this characterization of $\Sigma_1 = \text{NP}$, which is expressed by saying that $\Sigma_1\text{SO}$ captures NP [14], and the fact that $\Sigma_k\text{SO}$ captures Σ_k was first established in [43].
4. The class Σ_k consists of decision problems $L \subseteq \{0,1\}^*$ for which there is a polynomial-time Turing machine M with access to an oracle $Q \in \Sigma_{k-1}$ and a polynomial q such that for all $v \in \{0,1\}^n$, $v \in L \iff (\exists w \in \{0,1\}^{q(n)})\ M^Q(v,w) = 1$. We express this by saying that $L \in \text{NP}^{\Sigma_{k-1}}$. The fact that $\Sigma_k = \text{NP}^{\Sigma_{k-1}}$ was first established in [43].

In part motivated by the success of the theory of NP-completeness, Blum, Shub, and Smale initiated the study of computational complexity over the real numbers in [4]. They did this by investigating machines over the real numbers that have the ability to write any real number into memory, to evaluate infinite-precision subtraction, addition, and multiplication of real numbers, as well as to test the order relation between two real numbers and branch depending on the answer. Performing each of these operations takes one time step. In our terminology, this is a machine over the structure $\overline{\mathbb{R}} = (\mathbb{R}, (r : r \in \mathbb{R}), -, +, \cdot, \leq)$, which we obtain from the ordered ring of real numbers $\mathbb{R} = (\mathbb{R}, 0, 1, -, +, \cdot, \leq)$ by expanding it with a constant for each $r \in \mathbb{R}$.

Using machines over $\overline{\mathbb{R}}$, one may recreate the above quaternity for $\text{NP}(\overline{\mathbb{R}})$. By definition, $\text{NP}(\overline{\mathbb{R}})$ consists of *real* decision problems $L \subseteq \mathbb{R}^*$ for which there exists a polynomial-time verification algorithm, using *real* witness strings $w \in \mathbb{R}^*$, implemented by an $\overline{\mathbb{R}}$-machine. One may then show that $\text{SAT}(\overline{\mathbb{R}})$ is $\text{NP}(\overline{\mathbb{R}})$-complete under polynomial-time $\overline{\mathbb{R}}$-machine

reductions, where SAT($\overline{\mathbb{R}}$) is the problem of deciding if a quantifier-free formula using the operations of $\overline{\mathbb{R}}$ is satisfiable in $\overline{\mathbb{R}}$. This Cook-Levin theorem over $\overline{\mathbb{R}}$ is implied by results in [3]. One may also show that Σ_1SO($\overline{\mathbb{R}}$) captures NP($\overline{\mathbb{R}}$), where Σ_1SO($\overline{\mathbb{R}}$) is existential second-order metafinite logic over $\overline{\mathbb{R}}$. This Fagin's theorem over $\overline{\mathbb{R}}$ is shown in [20]. Finally, the oracle characterization is trivial at this base level of the hierarchy.

More recently, Schaefer and Štefankovič observed in [39] that many Boolean decision problems $L \subseteq \{0,1\}^*$ in computational geometry and beyond may be reduced to the problem $\exists$SAT($\mathbb{R}$) = ETR of deciding if a quantifier-free formula using the operations of the ordered ring of reals $\mathbb{R} = (\mathbb{R}, 0, 1, -, +, \cdot, \leq)$ is satisfiable in $\mathbb{R}$. Notably, such a formula may only contain the constants 0 and 1, and its satisfiability may be decided by a Turing machine [44]. The above observation prompted the introduction of the complexity class $\exists\mathbb{R}$ consisting of those problems reducible to $\exists$SAT($\mathbb{R}$) with a polynomial-time Turing machine. Already more than 200 problems have been identified as $\exists\mathbb{R}$-complete, attesting to the importance of the complexity class $\exists\mathbb{R}$ [38].

Using machines over $\mathbb{R}$, which are just $\overline{\mathbb{R}}$-machines restricted to write only the constants 0 and 1 into memory, we may recreate the above quaternity for $\exists\mathbb{R}$. The class $\exists\mathbb{R}$ consists of *Boolean* decision problems $L \subseteq \{0,1\}^*$ for which there exists a polynomial-time verification algorithm, using *real* witness strings $w \in \mathbb{R}^*$, implemented by an $\mathbb{R}$-machine. This machine characterization of $\exists\mathbb{R}$ is shown in [5, 13], and it is expressed by saying that $\exists\mathbb{R}$ is the constant-free Boolean part of NP($\overline{\mathbb{R}}$), or $\exists\mathbb{R} = \mathrm{BP}^0(\mathrm{NP}(\overline{\mathbb{R}}))$. By definition, $\exists$SAT($\mathbb{R}$) is $\exists\mathbb{R}$-complete under polynomial-time Turing machine reductions. We will show in this work that $\exists$SO($\mathbb{R}$) captures $\exists\mathbb{R}$, where $\exists$SO($\mathbb{R}$) is the restriction of Σ_1SO($\overline{\mathbb{R}}$) to use only the constants 0 and 1, while a similar result was recently (and independently of our work) established in [33] by restricting Σ_1SO($\overline{\mathbb{R}}$) to use only rational constants. Finally, the oracle characterization is again trivial at this base level of the hierarchy.

Since it is possible to recreate the above quaternity for both NP($\overline{\mathbb{R}}$) and $\exists\mathbb{R}$, these complexity classes faithfully resemble Σ_1 = NP. As we noted above, algorithms are often analyzed over different domains according to different sets of primitive operations that depend on the application area. Given the importance of NP and Σ_k more generally, a fundamental question in the complexity of algorithms is the following: In which domains and with which sets of primitive operations can we recreate the above quaternity for Σ_k?

We formalize this question with first-order structures $\mathcal{R}$, which consist of a set R together with constants $c \in R$, functions $f\colon R^k \to R$, and relations $P \subseteq R^k$. Using register machines over $\mathcal{R}$, we refine our fundamental question as follows: Under what conditions on $\mathcal{R}$ is it possible to recreate the above quaternity for both $\Sigma_k\mathcal{R}$ and $\exists_k\mathcal{R}$ by characterizing them in terms of machine models, complete problems, descriptive complexity, and oracles?

1.2 Overview of Our Contribution

In this work, we define $\Sigma_k\mathcal{R}$ and $\exists_k\mathcal{R}$ in terms of register machines over $\mathcal{R}$, resulting in the polynomial and Boolean hierarchies over a structure, as depicted in Figure 1. We then identify mild conditions on $\mathcal{R}$ which allow us to generalize the above quaternity by characterizing $\Sigma_k\mathcal{R}$ and $\exists_k\mathcal{R}$ in terms of complete problems, descriptive complexity, and oracles.

Each of our results requires us to assume that $\mathcal{R}$ is a *bipointed* structure, by which we mean that $\mathcal{R}$ has at least two distinct constants $0 \neq 1$. Our completeness results require us to assume further that $\mathcal{R}$ is of *finite type*, by which we mean that $\mathcal{R}$ has only finitely many functions and relations. Finally, for completeness with respect to non-Boolean decision problems, we must assume that $\mathcal{R}$ *has all constants*, by which we mean that $\mathcal{R}$ has a unique constant r for each element in its universe R.

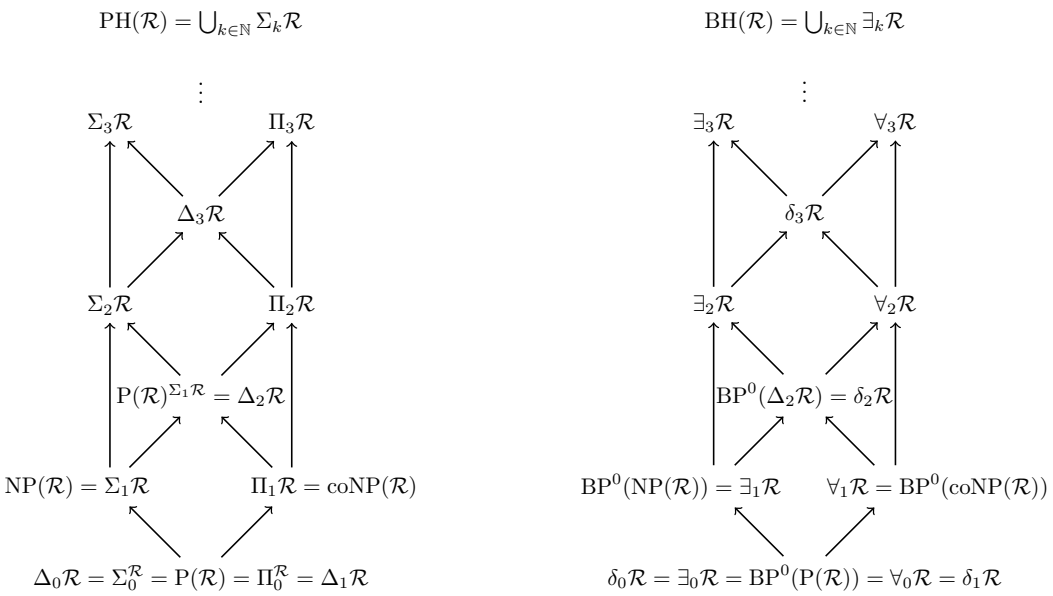

Figure 1 The polynomial hierarchy and the Boolean hierarchy over a structure $\mathcal{R}$.

We begin with complete problems. It is already known from [19] that if $\mathcal{R}$ is a bipointed structure of finite type with all constants, then $\mathrm{SAT}(\mathcal{R})$ is $\mathrm{NP}(\mathcal{R})$-complete under polynomial-time $\mathcal{R}$-machine reductions. Furthermore, it is folklore that, under the same conditions on $\mathcal{R}$, $\Sigma_k\mathrm{SAT}(\mathcal{R})$ is $\Sigma_k\mathcal{R}$-complete under polynomial-time $\mathcal{R}$-machine reductions [7, 11]. This folklore result is sometimes attributed to [37], which is a foundational text that is no longer in wide circulation, preventing us from accessing it. In Theorem 11, we offer a detailed proof that that if $\mathcal{R}$ is a bipointed structure of finite type with all constants, then $\Sigma_k\mathrm{SAT}(\mathcal{R})$ is $\Sigma_k\mathcal{R}$-complete under polynomial-time $\mathcal{R}$-machine reductions. We offer this proof not only for completeness, but also because analyzing this proof allows us to explain why the assumed conditions on $\mathcal{R}$ are probably necessary. Furthermore, our analysis of this proof allows us to derive Corollary 12, in which we show that if $\mathcal{R}$ is a bipointed structure of finite type, then $\exists_k\mathrm{SAT}(\mathcal{R})$ is $\exists_k\mathcal{R}$-complete under polynomial-time Turing machine reductions.

We next turn to descriptive complexity theory. In Theorem 14, we show that if $\mathcal{R}$ is a bipointed structure, then $\Sigma_k\mathrm{SO}(\mathcal{R})$ captures $\Sigma_k\mathcal{R}$ on metafinite structures over $\mathcal{R}$. Note that we are not required to assume that $\mathcal{R}$ is of finite type, nor that $\mathcal{R}$ has all constants, in contrast to our completeness results. Since there is a structure $\mathcal{Z}$ of infinite type for which there is no $\mathrm{NP}(\mathcal{Z})$-complete problem [16], Theorem 14 is strictly more general than any possible generalization of Theorem 11. This suggests that descriptive complexity theory is well suited for dealing with structures of infinite type, such as real vector spaces. Additionally, from Theorem 14 we derive Corollary 15 in which we show that if $\mathcal{R}$ is a bipointed structure, then $\exists_k\mathrm{SO}(\mathcal{R})$ captures $\exists_k\mathcal{R}$ on Boolean metafinite structures over $\mathcal{R}$.

We finally turn to oracle characterizations of these hierarchies. In Theorem 16, we show that if $\mathcal{R}$ is a bipointed structure, then $\Sigma_{k+1}\mathcal{R} = \mathrm{NP}(\mathcal{R})^{\Sigma_k\mathcal{R}}$. Additionally, from Theorem 16 we derive Corollary 17 in which we show that if $\mathcal{R}$ is a bipointed structure, then $\exists_{k+1}\mathcal{R} = \exists_k\mathcal{R}^{\Sigma_k\mathcal{R}^0}$. The class $\Sigma_k\mathcal{R}^0$ is obtained by restricting $\Sigma_k\mathcal{R}$ to those decision problems whose verification algorithms only involve the constants 0 and 1, while still allowing non-Boolean inputs. From this corollary, we can see that it is unlikely that we can replace the non-Boolean oracle class $\Sigma_k\mathcal{R}^0$ with the Boolean oracle class $\exists_k\mathcal{R} \subseteq \Sigma_k\mathcal{R}^0$. In particular, it is unlikely that $\exists\mathbb{R}^{\exists\mathbb{R}} = \exists_2\mathbb{R}$.

We organize our results as follows: In Section 2 we present preliminary definitions, in Section 3 we present our results about complete problems, in Section 4 we present our results about descriptive complexity, and in Section 5 we present our results about oracles. In each section, we provide proof sketches of our results, as well as a discussion of related work. Full proofs appear in the appendix of the full version of our paper, as well as an extended preliminaries section to make our results more accessible [28]. Each section in this appendix may be seen as an extended version of the corresponding section in the main body of our paper.

2 Preliminaries

We present a selection of the preliminaries that are most important for understanding our results. The reader who desires more formal detail or more intuition may find both of these in the extended preliminaries of the appendix of the full version of our paper [28].

2.1 Register Machines Over a Structure

A vocabulary Voc specifies a set of constant symbols c and, for each arity $k \geq 1$, a set function symbols f and a set of relation symbols P. A (first-order) Voc-structure $\mathcal{R}$ specifies an interpretation of each of these symbols in an underlying set R, which we call the universe of $\mathcal{R}$. Conflating a symbol with its interpretation, we may say that $\mathcal{R}$ has constants $c \in R$, functions $f : R^k \to R$, and relations $P \subseteq R^k$.

Recall that we say $\mathcal{R}$ is *bipointed* if it has at least two constants $0 \neq 1$, that $\mathcal{R}$ is of *finite type* if it has only finitely many functions and relations, and that $\mathcal{R}$ *has all constants* if it has a unique constant r for each element of its universe R. Additionally, we treat equality $=$ as a logical relation [45], so we always assume that the equality relation is implicitly present in every structure. This appears to be necessary for most of our results to hold in generality, though equality can be omitted in some particular contexts, as is often done when comparing the complexity of different syntactic fragments of the theory of $\mathbb{R}$, viewed as decision problems in $\exists\mathbb{R}$ and related complexity classes [39, 40].

The symbols of a vocabulary Voc constitute the basic ingredients of a programming language in which we may specify computations over a Voc-structure $\mathcal{R}$. These computations are carried out by a register machine over $\mathcal{R}$, also called an $\mathcal{R}$-machine, as depicted in Figure 2.

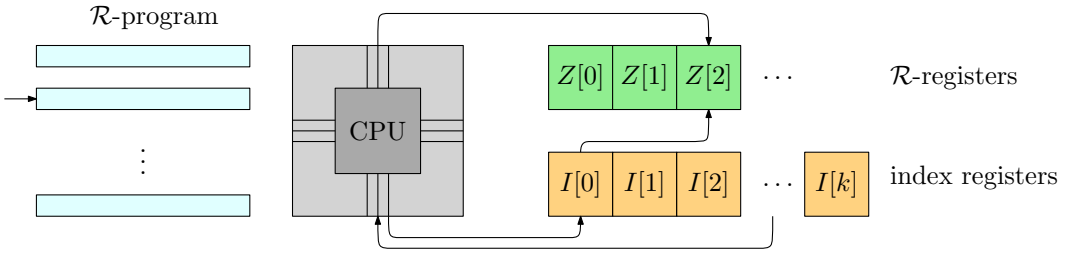

Figure 2 Visualization of an $\mathcal{R}$-machine adapted from [17].

Such machines have an infinite sequence $Z[0], Z[1], Z[2], \ldots$ of $\mathcal{R}$-registers, each of which may store a single element r from the universe R of $\mathcal{R}$, and a finite sequence $I[0], \ldots, I[k]$ of index registers, each of which may store a single natural number $n \in \mathbb{N}$. The primary role of index registers $I[j]$ is to keep track of indexing and counting during computations, as well as to point to $\mathcal{R}$-registers $Z[I[j]]$.

An $\mathcal{R}$-machine manipulates the contents of its registers based on its $\mathcal{R}$-program $\mathcal{P}$, which is a finite enumerated sequence of $\mathcal{R}$-instructions. Importantly, there are $\mathcal{R}$-instructions directing $\mathcal{R}$-machines to evaluate each operation of the structure $\mathcal{R}$ in the following way:

1. the instruction $Z[j_0] := c$ directs a machine to store the constant c in register $Z[j_0]$,
2. the instruction $Z[j_0] := f(Z[j_1], \ldots, Z[j_k])$ directs a machine to evaluate the function $f(Z[j_1], \ldots, Z[j_k])$ and store the result in register $Z[j_0]$,
3. the instruction $\mathsf{if}\ P(Z[j_1], \ldots, Z[j_k])\ \mathsf{then\ goto}\ \ell_1\ \mathsf{else\ goto}\ \ell_2$ directs a machine to test the relation $P(Z[j_1], \ldots, Z[j_k])$ and go to instruction ℓ_1 or ℓ_2 depending on the result.

In addition to the above, $\mathcal{R}$-instructions may also direct an $\mathcal{R}$-machine to branch based on a test of equality between $\mathcal{R}$-registers or index registers, to copy the contents of one $\mathcal{R}$-register to another, and to increment or decrement the contents of an index register. Finally, when we would like to equip an $\mathcal{R}$-machine with an oracle $Q \subseteq R^*$, we allow $\mathcal{R}$-instructions directing $\mathcal{R}$-machines to query this oracle. A full list of $\mathcal{R}$-instructions is presented in the appendix of the full version of our paper [28].

In the formal definition of $\mathcal{R}$-machines below, we include the structure $\mathcal{R}$ as a component of $\mathcal{R}$-machines. We do this so that the $\mathcal{R}$-machine knows how to interpret the symbols in its program, and in this sense we may view the structure $\mathcal{R}$ as the CPU from Figure 2, while the other components of $\mathcal{R}$-machines correspond in the expected way to this picture.

▶ **Definition 1** ($\mathcal{R}$-Machines). *A machine over $\mathcal{R}$, also called an $\mathcal{R}$-machine, is a tuple $M = (\mathcal{P}, \mathcal{R}, \mathbb{N}^{k_{\mathcal{P}}}, R^{\mathbb{N}})$ specifying the following information:*

1. *an $\mathcal{R}$-program $\mathcal{P}$,*
2. *the structure $\mathcal{R}$,*
3. *a space of index states $\mathbb{N}^{k_{\mathcal{P}}}$,*
4. *a space of memory states $R^{\mathbb{N}}$.*

We say that the constants of $\mathcal{R}$, other than 0 and 1, appearing in an instruction of $\mathcal{P}$ are the machine constants of M, and that M is constant free if it has no machine constants. The input and output space of M are $R^* = \bigcup_{n=0}^{\infty} R^n$.

Our definition of $\mathcal{R}$-machines is taken with minimal modification from [25, 17], and it is similar to the original definition of BSS machines given in [4], which closely resemble register machines, a precursor to random access machines introduced in [42]. Because our $\mathcal{R}$-instructions allow only sequential access to memory by incrementing or decrementing an index register, we use the name register machine over $\mathcal{R}$.

A significant difference between BSS machines over a ring $\mathcal{R}$ and $\mathcal{R}$-machines is that BSS machines over $\mathcal{R}$ may write an arbitrary element $r \in R$ into memory, while $\mathcal{R}$-machines may write an element r into memory only if r is a constant of the structure $\mathcal{R}$, or if r can be computed from an input using the operations of $\mathcal{R}$. We choose to limit $\mathcal{R}$-machines in this way so that the power of $\mathcal{R}$-machines more closely reflects the operations present in the structure $\mathcal{R}$, including the constant operations.

An element $r \in R$ usually plays the role of an *input* to an $\mathcal{R}$-machine. When we turn r into a *constant*, we assign it a very different role. As a constant, r may encode an infinite amount of information into $\mathcal{R}$-programs. For example, consider the ordered ring of reals expanded with all constants $\overline{\mathbb{R}}$ and suppose $r_{\text{halt}} \in \mathbb{R}$ encodes the characteristic function of the diagonal halting problem. Upon input $n \in \mathbb{N}$ encoded in binary, an $\overline{\mathbb{R}}$-machine with r_{halt} as a machine constant can extract the nth bit of r_{halt}, as described in [30], to decide if the nth Turing machine halts on input n, thereby deciding the diagonal halting problem. This means that $\overline{\mathbb{R}}$-machines with r_{halt} as a machine constant are strictly more powerful than Turing machines, even when we restrict these $\overline{\mathbb{R}}$-machines to Boolean inputs, whereas constant-free Boolean $\overline{\mathbb{R}}$-machines may be simulated by Turing machines.

Another subtlety concerning constants arises when one regards constants as functions of arity zero. We refrain from taking this view because it would clash with the definition of structures of finite type, which may have infinitely many constants but only finitely many functions. Assuming that $\mathcal{R}$ is of finite type is useful to us because this guarantees that $\mathcal{R}$ has an *effective encoding*, which is an encoding the symbols in the vocabulary of $\mathcal{R}$ as strings in R^* in such a way that an $\mathcal{R}$-machine can interpret that code and then evaluate the corresponding operation. In general, we will use an effective encoding of $\mathcal{R}$ when we show membership of satisfiability problems over $\mathcal{R}$ in complexity classes over $\mathcal{R}$. Note that there is an effective encoding for $\mathcal{R}$ if and only if there is a universal $\mathcal{R}$-machine [24].

Structures with infinitely many constants may still have an effective encoding because a constant may always stand as a code for itself, whereas there are structures with infinitely many unary functions, such as $\mathbb{R}_{\text{lin}}$, for which there is no effective encoding. To get an intuition for this phenomenon, consider why there can be no universal $\mathbb{R}_{\text{lin}}$-machine. Any $\mathbb{R}_{\text{lin}}$-machine can have only finitely many scalar functions μ_r in its program, and it is not possible to express each function in $\{\mu_r : r \in \mathbb{R}\}$ in terms of a finite subset of these functions.

There is a long history capturing computations over a first-order structure with machines and methods similar to ours [15]. We refer the reader to the introduction of [25] for further references into this history. There are multiple incomparable ways to generalize computation to the real numbers [36]. Machines over $\mathbb{R}$ given an algebraic generalization, while computable analysis and type two effectivity give a topological generalization [35]. The algebraic approach provides fertile ground for complexity theory. For an introductory survey of early results in the complexity of computation over $\mathbb{R}$ and other structures, see [32].

2.2 Complexity Classes

We view each subset $L \subseteq R^* = \bigcup_{n=0}^{\infty} R^n$ as a decision problem over a structure $\mathcal{R}$. Under a suitable encoding, the main decision problems we will work with consist of satisfied sentences of a specified grammatical form. A sentence Φ is in prenex normal form when it is a string of quantifiers followed by a quantifier free formula φ as in $Q_1 x_1 \dots Q_n \varphi(x_1, \dots, x_n)$, where $Q_i \in \{\exists, \forall\}$. Supposing Φ has $k-1$ quantifier alternations, we say that Φ is a Σ_k sentence when $Q_1 = \exists$, and we say that Φ is a Π_k sentence when $Q_1 = \forall$.

▶ **Definition 2** (Decision Problems of Satisfied Sentences). *The theory of $\mathcal{R}$ is the set* $\mathsf{Th}(\mathcal{R})$ *of sentences Φ in the vocabulary of $\mathcal{R}$ that are satisfied in $\mathcal{R}$, and the Boolean theory of $\mathcal{R}$ is the subset* $\mathsf{BTh}(\mathcal{R}) \subseteq \mathsf{Th}(\mathcal{R})$ *of sentences Φ such that all constants in Φ are 0 or 1. We identify the following fragments of these theories that we refer to as the kth existential or universal fragment of the theory or Boolean theory of $\mathcal{R}$, respectively:*

1. $\Sigma_k\text{SAT}(\mathcal{R}) = \{\Phi : \Phi \text{ is a } \Sigma_k \text{ sentence and } \mathcal{R} \vDash \Phi\}$,
2. $\Pi_k\text{SAT}(\mathcal{R}) = \{\Phi : \Phi \text{ is a } \Pi_k \text{ sentence and } \mathcal{R} \vDash \Phi\}$,
3. $\exists_k\text{SAT}(\mathcal{R}) = \{\Phi \in \Sigma_k\text{SAT}(\mathcal{R}) : \text{all constants in } \Phi \text{ are 0 or 1}\}$,
4. $\forall_k\text{SAT}(\mathcal{R}) = \{\Phi \in \Pi_k\text{SAT}(\mathcal{R}) : \text{all constants in } \Phi \text{ are 0 or 1}\}$.

A complexity class $\mathcal{C} \subseteq 2^{R^*}$ is any set of decision problems $L \subseteq R^*$. The polynomial hierarchy $\text{PH}(\mathcal{R})$ of complexity classes over a structure $\mathcal{R}$ generalize the classical polynomial hierarchy, and these classes are our main focus.

▶ **Definition 3** (Polynomial Hierarchy Over $\mathcal{R}$). *We define two sequences of complexity classes $\Sigma_k\mathcal{R}$ and $\Pi_k\mathcal{R}$ for $k \in \mathbb{N}$ as follows. A decision problem $L \subseteq R^*$ is in $\Sigma_k\mathcal{R}$ if and only if there is a polynomial q and a polynomial-time $\mathcal{R}$-machine M such that for all $n \in \mathbb{N}$ and all $v \in R^n$*

$$v \in L \iff (Q_1 w_1 \in R^{q(n)}) \dots (Q_k w_k \in R^{q(n)}) M(v, w_1, \dots, w_k) = 1,$$

where the quantifiers $Q_i \in \{\exists, \forall\}$ alternate, starting with $Q_1 = \exists$. The class $\Pi_k\mathcal{R}$ is defined by the same condition except the first quantifier is $Q_1 = \forall$. Define the polynomial hierarchy over $\mathcal{R}$ as the class $\mathrm{PH}(\mathcal{R}) = \bigcup_{k\in\mathbb{N}} \Sigma_k\mathcal{R}$.

We define $\mathrm{P}(\mathcal{R}) = \Sigma_0\mathcal{R}$, and we note that this is the class of decision problems that can be decided by a polynomial-time $\mathcal{R}$-machine. Furthermore, we define $\mathrm{NP}(\mathcal{R}) = \Sigma_1\mathcal{R}$ and $\mathrm{coNP}(\mathcal{R}) = \Pi_1\mathcal{R}$, and we note that these classes generalize classical NP and coNP. We may relate a complexity class $\mathcal{C}$ over a structure to classical complexity classes by taking the constant-free Boolean part of $\mathcal{C}$.

▶ **Definition 4** (Constant-Free Boolean Part). *The Boolean part of a decision problem L is the set* $\mathrm{BP}(L) = L \cap \{0,1\}^*$. *The Boolean part of a complexity class $\mathcal{C}$ is the set* $\mathrm{BP}(\mathcal{C}) = \{\mathrm{BP}(L) : L \in \mathcal{C}\}$. *If $\mathcal{C}$ is defined in terms of $\mathcal{R}$-machines, then $\mathcal{C}^0$ is the class obtained by restricting to $\mathcal{R}$-machines with no machine constants. We write* $\mathrm{BP}^0(\mathcal{C})$ *for* $\mathrm{BP}(\mathcal{C}^0)$, *and we say that* $\mathrm{BP}^0(\mathcal{C})$ *is the constant-free Boolean part of $\mathcal{C}$.*

By taking the constant-free Boolean part of each level of $\mathrm{PH}(\mathcal{R})$, we obtain the Boolean hierarchy over a structure $\mathcal{R}$, which is a hierarchy of Boolean complexity classes.

▶ **Definition 5** (Boolean Hierarchy Over $\mathcal{R}$). *We define the complexity classes $\exists_k\mathcal{R}$ and $\forall_k\mathcal{R}$ as the constant-free Boolean parts of $\Sigma_k\mathcal{R}$ and $\Pi_k\mathcal{R}$, respectively, for $k \in \mathbb{N}$. That is,* $\exists_k\mathcal{R} = \mathrm{BP}^0(\Sigma_k\mathcal{R})$ and $\forall_k\mathcal{R} = \mathrm{BP}^0(\Pi_k\mathcal{R})$. *The Boolean hierarchy over $\mathcal{R}$ is the complexity class* $\mathrm{BH}(\mathcal{R}) = \bigcup_{k\in\mathbb{N}} \exists_k\mathcal{R}$.

As we will see, it is quite fruitful to analyze the polynomial and Boolean hierarchies over $\mathcal{R}$ in terms of $\mathcal{R}$-machines with oracles.

▶ **Definition 6** (Oracle Complexity Classes). *Let $\mathcal{D}$ be a complexity class over $\mathcal{R}$. For each of level of the polynomial and Boolean hierarchies over $\mathcal{R}$, the complexity classes $\Sigma_k\mathcal{R}^{\mathcal{D}}$, $\Pi_k\mathcal{R}^{\mathcal{D}}$, $\exists_k\mathcal{R}^{\mathcal{D}}$, and $\forall_k\mathcal{R}^{\mathcal{D}}$ are obtained by modifying the definition of the corresponding class to allow $\mathcal{R}$-machines with oracles from $\mathcal{D}$.*

2.3 Metafinite Structures

Metafinite structures may be thought of as finite structures $\mathcal{A}$ that are given finite access to a potentially infinite structure $\mathcal{R}$ by a finite set $\mathbf{F}$ of weight functions $W\colon A^k \to R$. They were first used to generalize Fagin's theorem from NP to $\mathrm{NP}(\overline{\mathbb{R}})$ in [20], and their general theory was developed in [21]. Just as $\mathcal{R}$-machines separate the finite and discrete operations of index registers from the potentially infinite and continuous operations of $\mathcal{R}$-registers, metafinite structures separate the finite and discrete aspect of a structure from the potentially infinite and continuous aspects of a structure.

In order to talk about metafinite structures, we need a notion of metafinite vocabularies, which will consist of three separate vocabularies $\mathsf{MVoc} = (\mathsf{P}(\mathsf{MVoc}), \mathsf{S}(\mathsf{MVoc}), \mathsf{W}(\mathsf{MVoc}))$. The *primary vocabulary* $\mathsf{P}(\mathsf{MVoc})$ is a finite vocabulary, the *secondary vocabulary* $\mathsf{S}(\mathsf{MVoc})$ is a potentially infinite vocabulary, and the *weight vocabulary* $\mathsf{W}(\mathsf{MVoc})$ is a finite set of function symbols F, each with a given arity. We say that MVoc is a *metafinite vocabulary over* $\mathcal{R}$ whenever the secondary vocabulary of MVoc is the vocabulary of $\mathcal{R}$.

▶ **Definition 7.** *A metafinite structure for a metafinite vocabulary* MVoc *is a triple $\mathcal{D} = (\mathcal{A}, \mathcal{R}, \mathbf{F})$ where*

1. *the primary part of $\mathcal{D}$ is a finite structure $\mathcal{A}$ for vocabulary* $\mathsf{P}(\mathsf{MVoc})$,
2. *the secondary part of $\mathcal{D}$ is a potentially infinite structure $\mathcal{R}$ for vocabulary* $\mathsf{S}(\mathsf{MVoc})$,
3. *the set of weight functions of $\mathcal{D}$ is a finite set $\mathbf{F}$ of functions $F\colon A^{k_F} \to R$.*

We say that $\mathcal{D}$ is a Boolean metafinite structure if the image of each weight function $F \in \mathbf{F}$ is contained in $\{0,1\}$, so that $F\colon A^{k_F} \to \{0,1\} \subseteq R$.

The *size* of a metafinite structure $\mathcal{D}$, denoted by $|\mathcal{D}|$, is the cardinality of the universe A of its primary part $\mathcal{A}$. In practice, we will conflate the function $F\colon A^{k_F} \to R$ with the symbol for F. Whenever the secondary part of $\mathcal{D}$ is $\mathcal{R}$, then we say that $\mathcal{D}$ is an *$\mathcal{R}$-structure*. For a given metafinite vocabulary MVoc over $\mathcal{R}$, let $\mathrm{Struct}(\mathsf{MVoc})$ be the class of metafinite structures in this vocabulary.

▶ **Example 8.** A simple class of metafinite structures is obtained by letting $\mathcal{A} = (V, E)$ be a finite graph, letting $\mathcal{R} = \mathbb{R}$ be the ordered ring of real numbers, and letting $\mathbf{F}$ contain a single binary function $F\colon V^2 \to \mathbb{R}$ that assigns 0 to all pairs (v_1, v_2) that are not related by E. The resulting $\mathbb{R}$-structure is a finite graph with edges that have real number weights. Conversely, every graph with real weights can be interpreted as an $\mathbb{R}$-structure.

An important restriction of metafinite logic is that first-order variables, those from $\mathsf{Var} = \{x_1, x_2, x_3, \dots\}$, range only over the primary part $\mathcal{A}$ of an MVoc-structure $\mathcal{D} = (\mathcal{A}, \mathcal{R}, \mathbf{F})$. Second-order variables of positive arity k, those from $\mathsf{VAR}_k = \{X_1^k, X_2^k, X_3^k, \dots\}$, range over all possible functions $W\colon A^k \to R$, even those not in $\mathbf{F}$. First-order metafinite logic allows only quantification over first-order variables, while second-order metafinite logic allows quantification over second-order variables as well. Note that second-order quantification subsumes quantification over the secondary structure by accessing a second-order variable $X\colon A \to R$ at a specific point $X(a_0)$.

A sentence Φ of second-order metafinite logic is in prenex normal form when it is a string of second-order quanifiers followed by a first-order sentence φ, as in $QX_1 \dots QX_N\varphi(X_1, \dots, X_N)$, where $Q_i \in \{\exists, \forall\}$. Supposing Φ has $k-1$ quantifier alternations, we say that Φ is a Σ_k sentence when $Q_1 = \exists$, and we say that Φ is a Π_k sentence when $Q_1 = \forall$.

▶ **Definition 9** (Fragments of Second-Order Metafinite Logic). *Let $\mathrm{SO}(\mathcal{R})$ be the set of all second-order sentences in any metafinite vocabulary MVoc over $\mathcal{R}$. We identify the following fragments of metafinite second order logic over $\mathcal{R}$ that we refer to as the kth existential or universal fragment or Boolean fragment of $\mathrm{SO}(\mathcal{R})$, respectively:*

1. $\Sigma_k\mathrm{SO}(\mathcal{R}) = \{\Phi \in \mathrm{SO}(\mathcal{R}) : \Phi \text{ is a } \Sigma_k \text{ sentence}\}$,
2. $\Pi_k\mathrm{SO}(\mathcal{R}) = \{\Phi \in \mathrm{SO}(\mathcal{R}) : \Phi \text{ is a } \Pi_k \text{ sentence}\}$,
3. $\exists_k\mathrm{SO}(\mathcal{R}) = \{\Phi \in \Sigma_k\mathrm{SO}(\mathcal{R}) : \text{all constants from } \mathcal{R} \text{ in } \Phi \text{ are 0 or 1}\}$,
4. $\forall_k\mathrm{SO}(\mathcal{R}) = \{\Phi \in \Pi_k\mathrm{SO}(\mathcal{R}) : \text{all constants from } \mathcal{R} \text{ in } \Phi \text{ are 0 or 1}\}$.

3 Cook-Levin Theorem Over a Structure

The idea behind our hardness results is a generalization of the idea behind the classical Cook-Levin theorem: It is possible to encode the computation of a machine in a logical formula. In the classical setting, the machine is a polynomial-time Turing machine and the formula is a Boolean formula. In our setting, the machine is a polynomial-time $\mathcal{R}$-machine, and the formula is a first-order formula in the vocabulary of $\mathcal{R}$.

▶ **Lemma 10** (Encoding Computations in First-Order Formulas). *Let $\mathcal{R}$ be a bipointed structure with all constants, M be a polynomial-time $\mathcal{R}$-machine, q be a polynomial, and $v \in R^*$ be a string. Then for all $k \in \mathbb{N}$, there is a first-order formula $\exists\overline{y}\varphi_{v,k}(\overline{x}_1, \dots, \overline{x}_k, \overline{y})$ in the vocabulary of $\mathcal{R}$, with $\varphi_{v,k}$ quantifier free, such that for all strings $w_1, \dots, w_k \in R^{q(|v|)}$,*

$$M(v, w_1, \dots, w_k) = 1 \text{ if and only if } \mathcal{R} \vDash \exists\overline{y}\ \varphi_{v,k}(w_1, \dots, w_k, \overline{y}).$$

Table 1 Computation table of M running in time n^m on input $v \in R^n$.

	Registers						
Time	$L[0]$	$I[0]$	$\dots$	$I[k_{\mathcal{P}}-1]$	$Z[0]$	$\dots$	$Z[n^m]$
0	ℓ_0	$\nu_{0,0}$	$\dots$	$\nu_{k_{\mathcal{P}}-1,0}$	$\rho_{0,0}$	$\dots$	$\rho_{n^m,0}$
$\vdots$							
t	ℓ_t	$\nu_{0,t}$	$\dots$	$\nu_{k_{\mathcal{P}}-1,t}$	$\rho_{0,t}$	$\dots$	$\rho_{n^m,t}$
$\vdots$							
n^m	ℓ_{n^m}	ν_{0,n^m}	$\dots$	$\nu_{k_{\mathcal{P}}-1,n^m}$	ρ_{0,n^m}	$\dots$	ρ_{n^m,n^m}

Proof Sketch. The time bound of M allows us to represent the computation of M on input v in a finite computation table, as depicted in Table 1. Since the program of M is built from the operations of $\mathcal{R}$, we may use a formula $\varphi_{v,k} \equiv \texttt{start}_v \wedge \texttt{update} \wedge \texttt{accept}$ in the vocabulary of $\mathcal{R}$ to describe this table. We need to assume that $\mathcal{R}$ has all constants because the subformula $\texttt{start}_v$ uses constants to for each v_i in $v = (v_1, \dots, v_n) \in R^n$ to express the zeroth row of the table. The subformula `update` expresses that row $t+1$ of the table results from modifying row t of the table according to the program of M, while the subformula `accept` expresses that M returns 1 in the last row of the table. The computation table of M on input $(v, w_1, \dots, w_k)$ is then captured as a sequence $\overline{z} \in R^*$ such that $M(v, w_1, \dots, w_k) = 1$ if and only if $\mathcal{R} \vDash \varphi_{v,k}(w_1, \dots, w_k, \overline{z})$. ◀

▶ **Theorem 11** (Higher Cook-Levin Analogue). *Let $\mathcal{R}$ be a bipointed structure of finite type with all constants. Then for all $k \in \mathbb{N}$ with $k \geq 1$, $\Sigma_k\mathrm{SAT}(\mathcal{R})$ is $\Sigma_k\mathcal{R}$-complete and $\Pi_k\mathrm{SAT}(\mathcal{R})$ is $\Pi_k\mathcal{R}$-complete under polynomial-time $\mathcal{R}$-machine reductions.*

Proof Sketch. To show that $\Sigma_k\mathrm{SAT}(\mathcal{R})$ is in $\Sigma_k(\mathcal{R})$, it suffices to show that a polynomial-time $\mathcal{R}$-machine can decide if $\mathcal{R} \vDash \varphi(\overline{r})$ for any quantifier-free $\mathsf{Voc}(\mathcal{R})$-formula $\varphi(x_1, \dots, x_m)$ and any $\overline{r} = (r_1, \dots, r_m) \in R^m$. We show this by arguing that a polynomial-time $\mathcal{R}$-machine can recursively decide if $\mathcal{R} \vDash \varphi(\overline{r})$ based on the syntactic structure of φ. In order to implement the base case of this algorithm, we need $\mathcal{R}$ to have an effective encoding so that an $\mathcal{R}$-machine can take the encoding of a symbol in the vocabulary of $\mathcal{R}$ and use it to evaluate the corresponding operation. Our assumption that $\mathcal{R}$ is of finite type guarantees the existence of an effective encoding that may, moreover, be decoded in constant time. Hence, this recursive algorithm may be implemented on a polynomial-time $\mathcal{R}$-machine.

To show hardness, suppose $L \in \Sigma_k\mathcal{R}$. For any $v \in R^n$, let $Q\overline{y}\varphi_{v,k}$ be the formula depending on the machine associated with L and v defined above, or its negation if k is even, and let $\Phi_{v,k}$ be the Σ_k sentence $\Phi_{v,k} \equiv \exists\overline{x}_1 \forall\overline{x}_2 \dots Q\overline{x}_k Q\overline{y}\ \varphi_{v,k}(\overline{x}_1, \overline{x}_2, \dots, \overline{x}_k, \overline{y})$. Then by Lemma 10 together with the logical replacement theorem of [29], which allows us to prepend the appropriate string of quantifiers, $v \in L$ if and only if $\mathcal{R} \vDash \Phi_{v,k}$. Since the number of atomic subformulas in $\varphi_{v,k}$ is polynomial in $|v|$, an $\mathcal{R}$-machine can construct $\Phi_{v,k}$ in polynomial time from v and L. Thus $v \mapsto \Phi_{v,k}$ is a polynomial-time reduction from L to $\Sigma_k\mathrm{SAT}(\mathcal{R})$. ◀

As a corollary, we see that $\mathrm{SAT}(\mathcal{R})$ is $\mathrm{NP}(\mathcal{R})$-complete, as was shown in [19] with special cases shown already in [4, 3]. Theorem 11 was claimed already in [11] and attributed to [37], which is a foundational text that is no longer in wide circulation, preventing us from accessing it. We offer the present proof not only for completeness, but also because we slightly modify

the proof to obtain Corollary 12 and because we analyze the proof in order to demonstrate that the assumptions of the theorem are probably necessary, or may be weakened only slightly.

We conjecture that the assumption that $\mathcal{R}$ is of finite type is not strictly necessary because there are structures $\mathcal{R}$ of infinite type for which $\mathrm{SAT}(\mathcal{R})$ is $\mathrm{NP}(\mathcal{R})$ complete, as shown in [34, 24]. However, we have a strong intuition that the existence of an effective encoding for $\mathcal{R}$ is a necessary condition for $\mathrm{SAT}(\mathcal{R})$ to be in $\mathrm{NP}(\mathcal{R})$ because if there is an operation of $\mathcal{R}$ that $\mathcal{R}$-machines cannot decode, then $\mathcal{R}$-machines should not be able to test the truth of formulas with this operation. Indeed, the known infinite-type structures for which $\mathrm{SAT}(\mathcal{R})$ is $\mathrm{NP}(\mathcal{R})$-complete are Megiddo extensions of finite-type structures, and these extensions always have an effective encoding [24]. Given the fact that $\mathcal{R}$ has an effective encoding if and only if there is a universal $\mathcal{R}$-machine, we conjecture that $\mathrm{SAT}(\mathcal{R})$ is in $\mathrm{NP}(\mathcal{R})$ if and only if there is a universal $\mathcal{R}$-machine that runs with polynomial-overhead.

Previously, most research into the polynomial hierarchy over $\mathcal{R}$ focused on the cases when $\mathcal{R}$ is either $\mathbb{R}_{\mathrm{add}} = (\mathbb{R}, 0, 1, -, +)$ or $\mathbb{R}_{\mathrm{lin}}$ or a number of related structures [9]. Many of the results for the additive reals remain true when $\mathcal{R}$ is a ring, and in this case we already knew already that a k-fold quantified version of polynomial feasibility is complete for each level of the polynomial hierarchy over $\mathcal{R}$ [3, 6].

Exotic quantifiers are used in [6] to extend the polynomial hierarchy over $\mathbb{R}$ in order to achieve completeness results for problems from semialgebraic geometry within these newly defined classes. When restricting to the constant-free Boolean part of these newly defined classes, the exotic quantifiers can be eliminated, thereby yielding completeness results in the Boolean hierarchy over $\mathbb{R}$, which is treated in our next corollary.

▶ **Corollary 12** (Higher Boolean Cook-Levin Analogue)**.** *Let $\mathcal{R}$ be a bipointed structure of finite type. Then for all $k \in \mathbb{N}$ with $k \geq 1$, $\exists_k\mathrm{SAT}(\mathcal{R})$ is $\exists_k\mathcal{R}$-complete and $\forall_k\mathrm{SAT}(\mathcal{R})$ is $\forall_k\mathcal{R}$-complete under polynomial-time Turing machine reductions.*

Proof Sketch. All constants in a sentence Φ of $\exists_k\mathrm{SAT}(\mathcal{R})$ are either 0 or 1, implying that the encoding of Φ is a Boolean string, so $\exists_k\mathrm{SAT}(\mathcal{R})$ is in $\exists\mathcal{R}$ by definition. The reduction in Theorem 11 from $L \in \Sigma_k\mathcal{R}$ to $\Sigma_k\mathrm{SAT}(\mathcal{R})$ restricts to a reduction from $L \in \exists_k\mathcal{R}$ to $\exists_k\mathrm{SAT}(\mathcal{R})$. Finally, the polynomial-time $\mathcal{R}$-machine implementing this reduction takes only Boolean inputs and need not execute operations of $\mathcal{R}$ other than writing 0 and 1 and testing for equality, so this reduction may be implemented on a polynomial-time Turing machine. ◀

As a corollary we get that $\exists\mathrm{SAT}(\mathcal{R})$ is $\exists\mathcal{R}$-complete. Indeed, this same sort of argument is used in [5] to show that $\exists\mathrm{SAT}(\mathcal{R})$ is $\exists\mathcal{R}$-complete when $\mathcal{R}$ is a ring. When comparing Corollary 12 with Theorem 11, we see that the assumption that $\mathcal{R}$ has all constants has been dropped. This is possible because, even though $\exists\mathcal{R}$ is defined in terms of $\mathcal{R}$-machines, it is a Boolean complexity class, by which we mean that $L \subseteq \{0,1\}^*$ for every $L \in \exists\mathcal{R}$.

To the best of our knowledge, the first results about higher levels of the Boolean hierarchy over a structure $\mathcal{R}$ appear in [9], wherein $\mathcal{R}$ is taken to be $\mathbb{R}_{\mathrm{add}}$ or $\mathbb{R}_{\mathrm{lin}}$, among other structures. In these cases, the authors show that $\exists\mathbb{R}_{\mathrm{add}} = \mathrm{NP} = \exists\mathbb{R}_{\mathrm{lin}}$, and furthermore that the Boolean hierarchy over these structures coincides with the classical polynomial hierarchy.

The authors of [6] introduce the Boolean hierarchy over $\mathbb{R}$ in terms of machine models and show completeness for every level of this hierarchy of a problem that is essentially the circuit version of $\exists_k\mathrm{SAT}(\mathbb{R})$ and $\forall_k\mathrm{SAT}(\mathbb{R})$. This hierarchy also is introduced in [40] as the real hierarchy by *defining* $\exists_k\mathbb{R}$ as the closure of a problem essentially equivalent to $\exists_k\mathrm{SAT}(\mathbb{R})$ under polynomial-time Turing machine reductions. Corollary 12 offers another proof that their definition in terms of complete problems coincides the machine-based definition found here and in [6, 22].

A number of problems have already been identified as complete for classes in the second level of this hierarchy, including area-universality [12], Hausdorff-distance [26], escape-games [10], surjectivity [6, 41], and image density [6, 27].

4 Fagin's Theorem Over a Structure

The idea behind our results in this section is a generalization of the idea behind Fagin's theorem: It is possible to encode the computation of a machine in second-order logical formula. In the setting of Fagin's theorem, the machine is a polynomial-time Turing machine, and the formula is a second-order formula that is interpreted in finite structures. In our setting, the machine is a polynomial-time $\mathcal{R}$-machine, and the formula is a second-order metafinite formula that is interpreted in metafinite structures over $\mathcal{R}$, which we will abbreviate to $\mathcal{R}$-structures.

▶ **Lemma 13** (Encoding Computations in Second-Order Formulas). *Let $\mathcal{R}$ be a bipointed structure, M be a polynomial-time $\mathcal{R}$-machine, q be a positive integer, and* MVoc *be a metafinite vocabulary over $\mathcal{R}$. Then for all $k \in \mathbb{N}$, there is a second-order formula $\exists \overline{Y} \varphi_{M,k}(X_1, \ldots, X_k, \overline{Y})$ in the vocabulary* MVoc*, with $\varphi_{M,k}$ first-order, such that for all* MVoc*-structures $\mathcal{D}$ and all weight functions $W_1, \ldots, W_k \colon A^q \to R$,*

$$M(\mathcal{D}, W_1, \ldots, W_k) = 1 \text{ if and only if } \mathcal{D} \vDash \exists \overline{Y} \varphi_{M,k}(W_1, \ldots, W_k, \overline{Y}).$$

Proof Sketch. The time bound of M allows us to represent the computation of M on input $\mathcal{D}$ in a finite computation table, as depicted in Table 1, which we may describe with the metafinite formula $\varphi_{M,k} \equiv \texttt{start}_M \wedge \texttt{update} \wedge \texttt{accept}$ whose subformulas have the same intended interpretation as those of $\varphi_{v,k}$ in Lemma 10.

However, while $\varphi_{v,k}$ describes an accepting computation of M on a *fixed* input v, the formula $\varphi_{M,k}$ describes an accepting computation of M on input $\mathcal{D}$ for *any* MVoc-structure $\mathcal{D}$. In order to achieve this, we construct a subformula $\texttt{input}_M$ of $\texttt{start}_M$ such that $\texttt{input}_M$ expresses the encoding of $\mathcal{D}$ as a string in R^* when it is interpreted in $\mathcal{D}$. The computation table of M on input $(\mathcal{D}, W_1, \ldots, W_k)$ is then captured as a sequence $\overline{Z}$ of weight functions on $\mathcal{D}$ such that $M(\mathcal{D}, W_1, \ldots, W_k) = 1$ if and only if $\mathcal{D} \vDash \varphi_{M,k}(W_1, \ldots, W_k, \overline{Z})$. ◀

▶ **Theorem 14** (Higher Fagin's Analogue). *Let $\mathcal{R}$ be a bipointed structure. Then for all $k \in \mathbb{N}$ with $k \geq 1$, $\Sigma_k\mathrm{SO}(\mathcal{R})$ captures $\Sigma_k\mathcal{R}$ and $\Pi_k\mathrm{SO}(\mathcal{R})$ captures $\Pi_k\mathcal{R}$ on $\mathcal{R}$-structures*

Proof Sketch. We first show that the data complexity of $\Sigma_1\mathrm{SO}(\mathcal{R})$ is $\mathrm{NP}(\mathcal{R})$. Suppose Φ is a $\Sigma_1\mathrm{SO}(\mathcal{R})$ sentence, which means that Φ is of the form $\Phi \equiv \exists X_1 \ldots \exists X_N \varphi(X_1, \ldots, X_N)$, where $\varphi(X_1, \ldots, X_N)$ is a first-order formula in some metafinite vocabulary MVoc over $\mathcal{R}$. To show that $\{\mathcal{D} \in \mathrm{Struct}(\mathsf{MVoc}) : \mathcal{D} \vDash \Phi\} \in \Sigma_k\mathcal{R}$, it suffices to show that given an MVoc-structure $\mathcal{D} = (\mathcal{A}, \mathcal{R}, \mathbf{F})$ and weight functions $W_1, \ldots, W_N$ with $W_i \colon A^{k_i} \to R$, an $\mathcal{R}$-machine can decide in polynomial time if $\mathcal{D} \vDash \varphi(W_1, \ldots, W_N)$. This is essentially a matter of showing that an $\mathcal{R}$-machine requires only polynomial-time overhead when simulating the operations of the primary part $\mathcal{A}$ of an $\mathcal{R}$-structure $\mathcal{D} = (\mathcal{A}, \mathcal{R}, \mathbf{F})$ that it receives as input, which may be done by analyzing the encoding of the structure $\mathcal{D}$ as a string in R^* according to our knowledge of the fixed metafinite vocabulary MVoc of $\mathcal{D}$.

We next show that if $L \in \Sigma_k\mathcal{R}$ is a decision problem of MVoc-structures, then there is a sentence Φ of $\Sigma_k\mathrm{SO}(\mathsf{MVoc})$ describing L. Each such $L \in \Sigma_k\mathcal{R}$ is of the form

$$L = \{\mathcal{D} \in \mathrm{Struct}(\mathsf{MVoc}) : (\exists W_1 : A^q \to R) \ldots (QW_N : A^q \to R)\ M(\mathcal{D}, W_1, \ldots, W_N) = 1\}$$

for some positive integer q and some polynomial-time $\mathcal{R}$-machine M. Let $Q\overline{Y}\varphi_{M,k}$ be the formula associated with the machine M defined above, or its negation if k is even, and let $\Phi_{M,k}$ be the $\Sigma_k\mathrm{SO}(\mathsf{MVoc})$ sentence $\Phi_{M,k} \equiv \exists Z_1 \ldots QZ_N Q\overline{Y}\varphi_{M,k}(Z_1, \ldots, Z_N, \overline{Y})$. Then by Lemma 13 together with the logical replacement theorem [29], which allows us to prepend the appropriate string of quantifiers, we see that $L = \{\mathcal{D} \in \mathrm{Struct}(\mathsf{MVoc}) : \mathcal{D} \vDash \Phi_{M,k}\}$. We conclude that $\Sigma_k\mathrm{SO}(\mathcal{R})$ captures $\Sigma_k\mathcal{R}$ on $\mathcal{R}$-structures. ◀

While Theorem 11 assumes that $\mathcal{R}$ is of finite type with all constants, Theorem 14 makes neither of these assumptions. This is due mainly to the way in which descriptive complexity theory approaches complexity classes. We describe in more detail why we can do without each of these assumptions below.

The assumption that $\mathcal{R}$ is of finite type is used Theorem 11 to guarantee that $\mathcal{R}$ has an effective encoding, which we require when showing that $\Sigma_k\mathrm{SAT}(\mathcal{R})$ is in $\Sigma_k\mathcal{R}$. The corresponding part of Theorem 14 shows that the data complexity of $\Sigma_k\mathrm{SO}(\mathcal{R})$ is $\Sigma_k\mathcal{R}$, which we do by constructing a machine that takes input $(\mathcal{D}, \overline{W})$ and decides, for a fixed formula φ, if $\mathcal{D} \vDash \varphi(\overline{W})$. Since φ is fixed and not part of the input, we do not need to encode φ in a way that a machine can understand. Hence, we do not need the assumption that $\mathcal{R}$ has an effective encoding, let alone the assumption that $\mathcal{R}$ is of finite type. Notably, this means that Theorem 14 applies to the structure $\mathbb{R}_{\mathrm{lin}}$, as well as structures $\mathcal{Z}$ of infinite type for which we know that $\mathrm{NP}(\mathcal{Z})$ has no complete problems [16].

The assumption that $\mathcal{R}$ has all constants is used in Theorem 11 when showing that $\mathrm{SAT}(\mathcal{R})$ is $\mathrm{NP}(\mathcal{R})$-hard in order to build the formula $\varphi_{v,k}$, which hard-codes the fixed input v. The corresponding part of Theorem 14 builds the formula $\varphi_{M,k}$, which does describe a fixed input, so no constants other than 0 or 1 or the machine constants of M are necessary, and these are a subset of the constants of the structure $\mathcal{R}$.

Our proof strategy generalizes that found in [20], which shows that $\Sigma_1\mathrm{SO}(\overline{\mathbb{R}})$ captures $\mathrm{NP}(\overline{\mathbb{R}})$. A generalization of Fagin's theorem to a class of structures on the natural numbers, called arithmetical structures, is already found in [21], though the authors use a different measure of the size of an input, so their results are incomparable with ours. Already we know that $\Sigma_1\mathrm{SO}(\mathcal{R})$ captures $\mathrm{NP}(\mathcal{R})$ on structures of finite type, as shown in [11] with a proof that relies on a prior descriptive characterization of $\mathrm{P}(\mathcal{R})$.

The authors of [33] show that by restricting $\Sigma_1\mathrm{SO}(\overline{\mathbb{R}})$ to those formulas with only rational constants, one can capture $\exists\mathbb{R}$ over discrete $\overline{\mathbb{R}}$-structures, whose definition makes use of the fact that $\mathbb{N} \subseteq \mathbb{R}$. Since we work with structures $\mathcal{R}$ whose universe may contain neither $\mathbb{Q}$ nor $\mathbb{N}$, we restrict $\Sigma_k\mathrm{SO}(\mathcal{R})$ to those formulas with only 0 and 1 as constants, and we capture $\exists_k\mathcal{R}$ over Boolean $\mathcal{R}$-structures.

In the same way that we modified the proof that $\Sigma_1\mathrm{SO}(\mathcal{R})$ captures $\mathrm{NP}(\mathcal{R})$ on $\mathcal{R}$-structures to show that $\exists\mathrm{SO}(\mathcal{R})$ captures $\exists\mathcal{R}$ on Boolean $\mathcal{R}$-structures, we may modify the proof that $\Sigma_k\mathrm{SO}(\mathcal{R})$ captures $\Sigma_k\mathcal{R}$ on $\mathcal{R}$-structures to show that $\exists_k\mathrm{SO}(\mathcal{R})$ captures $\exists_k\mathcal{R}$ on Boolean $\mathcal{R}$-structures.

▶ **Corollary 15** (Higher Boolean Fagin's Analogue). *Let $\mathcal{R}$ be a bipointed structure. Then for all $k \in \mathbb{N}$ with $k \geq 1$, $\exists_k\mathrm{SO}(\mathcal{R})$ captures $\exists_k\mathcal{R}$ and $\forall_k\mathrm{SO}(\mathcal{R})$ captures $\forall_k\mathcal{R}$ on Boolean metafinite structures over $\mathcal{R}$.*

Proof Sketch. Suppose $\Phi \equiv \exists\overline{X}_1 \ldots Q\overline{X}_N\varphi(X_1, \ldots, X_N)$ is a sentence of $\exists_k\mathrm{SO}(\mathsf{MVoc})$, and let $L = \{\mathcal{D} \in \mathrm{BStruct}(\mathsf{MVoc}) : \mathcal{D} \vDash \Phi\}$. Since all constants from $\mathcal{R}$ occurring in φ are either 0 or 1, the $\mathcal{R}$-machine from Theorem 14 that decides if $\mathcal{D} \vDash \varphi(\overline{W})$ is constant free. Moreover, since L is a decision problem of Boolean $\mathcal{R}$-structures, we see that $L \in \mathrm{BP}(\mathrm{NP}^0(\mathcal{R})) = \exists\mathcal{R}$. From this we conclude that the data complexity of $\exists\mathrm{SO}(\mathcal{R})$ is $\exists\mathcal{R}$.

Each $L \in \exists_k\mathcal{R}$ is of the form specified in Theorem 14 where M is a polynomial-time constant-free $\mathcal{R}$-machine. Since M is constant free, all constants from $\mathcal{R}$ occurring in $\varphi_{M,k}$ are 0 or 1. Thus, $\exists X \exists \overline{Y} \varphi_{M,1}(X, \overline{Y})$ is an $\exists\mathrm{SO}(\mathcal{R})$ sentence, and from this we conclude that $\exists\mathrm{SO}(\mathcal{R})$ captures $\exists\mathcal{R}$ on Boolean $\mathcal{R}$-structures. ◀

An advantage of using the class of Boolean $\mathcal{R}$-structures is that they may be identified with the class of finite structures that is used in classical descriptive complexity theory. Indeed, this is the perspective taken by the authors of [23] when they show that $\exists\mathrm{SO}(\mathbb{R}_{\mathrm{add}})$ captures NP on finite structures, using the fact that $\exists\mathbb{R}_{\mathrm{add}} = \mathrm{NP}$ established in [9]. Given the similar result that $\exists\mathbb{R}_{\mathrm{lin}} = \mathrm{NP}$, one implication of Corollary 15 is that $\exists\mathrm{SO}(\mathbb{R}_{\mathrm{lin}})$ captures NP on Boolean $\mathbb{R}_{\mathrm{lin}}$-structures. In this way, we can see that Boolean $\mathcal{R}$-structures have a natural place in descriptive complexity theory.

5 Oracles and Hierarchies

The classical polynomial hierarchy was originally defined in terms of oracle classes [43]. Since the polynomial hierarchy is now usually defined in terms of machine models and witness strings, characterizing each level of this hierarchy in terms of oracles requires proof. The proof strategy we use below is a generalization of that found in [1], for the classical polynomial hierarchy, and [9], for the polynomial hierarchy over $\mathbb{R}_{\mathrm{add}}$ and a number of related structures. The latter proof was already generalized to rings in [3].

▶ **Theorem 16** (Oracle Polynomial Hierarchy Characterization). *Let $\mathcal{R}$ be a bipointed structure. Then for all $k \in \mathbb{N}$, $\Sigma_{k+1}\mathcal{R} = \mathrm{NP}(\mathcal{R})^{\Sigma_k\mathcal{R}}$ and $\Pi_{k+1}\mathcal{R} = \mathrm{coNP}(\mathcal{R})^{\Sigma_k\mathcal{R}}$.*

Proof Sketch. We proceed by induction on $k \in \mathbb{N}$. The base case holds immediately, and in the inductive step establishing that $\Sigma_{k+1}\mathcal{R} = \mathrm{NP}(\mathcal{R})^{\Sigma_k\mathcal{R}}$, the difficult inclusion to show is $\mathrm{NP}(\mathcal{R})^{\Sigma_k\mathcal{R}} \subseteq \Sigma_{k+1}\mathcal{R}$. Our inductive hypothesis implies that it is sufficient to show that a polynomial-time $\mathcal{R}$-machine M with a $Q \in \Sigma_k\mathcal{R}$ oracle may be simulated by a polynomial-time $\mathcal{R}$-machine N with an $S \in \Pi_{k-1}\mathcal{R}$ oracle, together with the help of witness strings that are provided in addition to the input. This is possible because for each $Q \in \Sigma_k\mathcal{R}$ there is some $S \in \Pi_{k-1}\mathcal{R}$ such that $u \in Q$ if and only if there is some w such that $(u, w) \in S$, and $u \notin Q$ if and only if $(u, w') \notin S$ for all w'. Given the answers $b_1, \ldots, b_p \in \{0, 1\}$ to the queries $u_1, \ldots, u_p$ that M makes on input v, as well as the witness strings $w_1, \ldots, w_p$ and $w'_1, \ldots, w'_p$, the machine N can simulate M by branching according to the answers $b_1, \ldots, b_p$. Furthermore, N can check that these answers are correct by querying $(u_i, w_i) \in S$ if $b_i = 1$ and querying $(u_i, w'_i) \in S$ if $b_i = 0$. Thus, $\Sigma_{k+1}\mathcal{R} = \mathrm{NP}(\mathcal{R})^{\Sigma_k\mathcal{R}}$, and this implies $\Pi_{k+1}\mathcal{R} = \mathrm{coNP}(\mathcal{R})^{\Sigma_k\mathcal{R}}$, thereby completing our induction. ◀

One usually states the corresponding theorem for the classical polynomial hierarchy as $\Sigma_{k+1} = \mathrm{NP}^{\Sigma_k\mathrm{SAT}}$, where $\Sigma_k\mathrm{SAT}$ is the problem of deciding if a quantified Boolean formula of the appropriate form is true. This is equivalent to saying that $\Sigma_{k+1} = \mathrm{NP}^{\Sigma_k}$ because $\Sigma_k\mathrm{SAT}$ is Σ_k-complete. We use the oracle class $\Sigma_k\mathcal{R}$, rather than the single oracle $\Sigma_k\mathrm{SAT}(\mathcal{R})$, so that Theorem 16 applies to structures for which $\Sigma_k\mathrm{SAT}(\mathcal{R})$ is not $\Sigma_k\mathcal{R}$-complete. Modifying the proof of Theorem 16, we obtain the following corollary.

▶ **Corollary 17** (Oracle Boolean Hierarchy Characterization). *Let $\mathcal{R}$ be a bipointed structure. Then for all $k \in \mathbb{N}$, $\exists_{k+1}\mathcal{R} = \exists\mathcal{R}^{\Sigma_k\mathcal{R}^0}$ and $\forall_{k+1}\mathcal{R} = \forall\mathcal{R}^{\Sigma_k\mathcal{R}^0}$.*

This corollary tells us that it is unlikely that we can replace the non-Boolean oracle class $\Sigma_k\mathcal{R}^0$ with the Boolean oracle class $\exists_k\mathcal{R}$, as one might expect from a straightforward analogy with the classical polynomial hierarchy. For example, while $\mathrm{NP}^{\mathrm{NP}} = \Sigma_2$ and

$\mathrm{NP}(\mathcal{R})^{\mathrm{NP}(\mathcal{R})} = \Sigma_2\mathcal{R}$, Corollary 17 tells us that $\exists\mathcal{R}^{\mathrm{NP}(\mathcal{R})^0} = \exists_2\mathcal{R}$, so it is unlikely that $\exists\mathcal{R}^{\exists\mathcal{R}} = \exists_2\mathcal{R}$. It is instructive to reflect on why our proof requires the constant-free, non-Boolean oracle class $\mathrm{NP}(\mathcal{R})^0$ when showing that $\exists\mathcal{R}^{\mathrm{NP}(\mathcal{R})^0} = \exists_2\mathcal{R}$.

In order to show the inclusion $\exists\mathcal{R}^{\mathrm{NP}(\mathcal{R})^0} \subseteq \exists_2\mathcal{R}$, we use a constant-free machine N and witness strings to simulate a constant-free machine M with an oracle $Q \in \mathrm{NP}(\mathcal{R})^0$. If Q were not in the constant-free part of $\mathrm{NP}(\mathcal{R})$, we could hide a constant in the oracle Q, making a constant-free simulation impossible. In order to show the inclusion $\exists_2\mathcal{R} \subseteq \exists\mathcal{R}^{\mathrm{NP}(\mathcal{R})^0}$, we replace witness strings $w \in R^*$ with an oracle $Q \in \mathrm{NP}(\mathcal{R})^0$. Since witness strings can be non-Boolean, it should generally be impossible to replace them with an oracle in the Boolean part of $\mathrm{NP}(\mathcal{R})^0$. Thus, we take $\mathrm{NP}(\mathcal{R})^0$ as our oracle class because we need to take the constant-free part of $\mathrm{NP}(\mathcal{R})$, but we cannot take the Boolean part of $\mathrm{NP}(\mathcal{R})^0$.

6 Conclusion

We have characterized the polynomial and Boolean hierarchies over a first-order structure $\mathcal{R}$ in terms of machine models, complete problems, descriptive complexity, and oracles. Additionally, we have shown that descriptive complexity is well suited to working with algorithms over infinite vocabulary structures, such as real vector spaces. In this way, we have answered fundamental questions about the complexity of algorithms presented in terms of the primitive operations present in a first-order structure.

References

1 Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009. doi:10.1017/CBO9780511804090.

2 Lenore Blum. Computing over the reals: Where Turing meets Newton. *Notices of the AMS*, 51(9):1024–1034, 2004. URL: https://www.ams.org/journals/notices/200409/fea-blum.pdf?adat=October%202004&trk=200409fea-blum&cat=none&type=.pdf.

3 Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale. *Complexity and real computation*. Springer-Verlag, New York, 1998. With a foreword by Richard M. Karp. doi:10.1007/978-1-4612-0701-6.

4 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. *Bull. Amer. Math. Soc. (N.S.)*, 21(1):1–46, 1989. doi:10.1090/S0273-0979-1989-15750-9.

5 Peter Bürgisser and Felipe Cucker. Counting complexity classes for numeric computations. II. Algebraic and semialgebraic sets. *Journal of Complexity*, 22(2):147–191, 2006. doi:10.1016/j.jco.2005.11.001.

6 Peter Bürgisser and Felipe Cucker. Exotic quantifiers, complexity classes, and complete problems. *Foundations of Computational Mathematics*, 9(2):135–170, 2009. doi:10.1007/s10208-007-9006-9.

7 Olivier Chapuis and Pascal Koiran. Saturation and stability in the theory of computation over the reals. *Ann. Pure Appl. Logic*, 99(1-3):1–49, 1999. doi:10.1016/S0168-0072(98)00060-8.

8 Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, pages 151–158. ACM, 1971. doi:10.1145/800157.805047.

9 Felipe Cucker and Pascal Koiran. Computing over the reals with addition and order: higher complexity classes. *Journal of Complexity*, 11(3):358–376, 1995. doi:10.1006/jcom.1995.1018.

10 Julian D'Costa, Engel Lefaucheux, Eike Neumann, Joël Ouaknine, and James Worrell. On the complexity of the escape problem for linear dynamical systems over compact semialgebraic sets. In *Proceedings of the 46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021*, volume 202 of *LIPIcs*, pages 33:1–33:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.MFCS.2021.33.

11 Paulin Jacobé de Naurois. *Completeness results and syntactic characterizations of complexity classes over arbitrary structures.* PhD thesis, éditeur inconnu, 2004.

12 Michael Gene Dobbins, Linda Kleist, Tillmann Miltzow, and PawełRzażewski. $\forall\exists\mathbb{R}$-completeness and area-universality. In *Proceedings of the 44th International Workshop on Graph-theoretic Concepts in Computer Science, WG 2018*, volume 11159 of *Lecture Notes in Computer Science*, pages 164–175. Springer, 2018. doi:10.1007/978-3-030-00256-5_14.

13 Jeff Erickson, Ivor van der Hoog, and Tillmann Miltzow. Smoothing the gap between NP and ER. In *Proceedigns of the 61st Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 1022–1033, 2020. doi:10.1137/20M1385287.

14 Ronald Fagin. Generalized first-order spectra and polynomial-time recognizable sets. In R.Karp, editor, *Proceedings Complexity of Computation*, volume 7, pages 43–73, 1974.

15 Solomon Feferman. *About and around computing over the reals*, chapter 3, pages 55–76. MIT Press, Cambridge, MA, 2013.

16 Christine Gaßner. On NP-completeness for linear machines. *Journal of Complexity*, 13(2):259–271, 1997. doi:10.1006/JCOM.1997.0444.

17 Christine Gaßner. An introduction to a model of abstract computation. In *Contemporary Logic and Computing*, volume II of *Landscapes in Logic*, pages 574–603. College Publications, 2020.

18 Robert Giegerich, Carsten Meyer, and Peter Steffen. A discipline of dynamic programming over sequence data. *Science of Computer Programming*, 51(3):215–263, 2004. doi:10.1016/j.scico.2003.12.005.

19 John B. Goode. Accessible telephone directories. *The Journal of Symbolic Logic*, 59(1):92–105, 1994. doi:10.2307/2275252.

20 Erich Grädel and Klaus Meer. Descriptive complexity theory over the real numbers. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing, STOC 1995*, pages 315–324, 1995. doi:10.1145/225058.225151.

21 Erich Graedel and Yuri Gurevich. Metafinite model theory. *Information and Computation*, 140(1):26–81, 1998. doi:10.1006/INCO.1997.2675.

22 Thekla Hamm, Lucas Meijer, Tillmann Miltzow, and Subhasree Patro. Oracle separations for RPH, 2025. doi:10.48550/arXiv.2502.09279.

23 Miika Hannula and Jonni Virtema. Tractability frontiers in probabilistic team semantics and existential second-order logic over the reals. *Annals of Pure and Applied Logic*, 173(10):103108, 2022. doi:10.1016/j.apal.2022.103108.

24 Armin Hemmerling. Computability over structures of infinite signature. *Mathematical Logic Quarterly*, 44:394–416, 1998. doi:10.1002/MALQ.19980440311.

25 Armin Hemmerling. Computability of string functions over algebraic structures. *Mathematical Logic Quarterly*, 44:1–44, 2006. doi:10.1002/malq.19980440102.

26 Paul Jungeblut, Linda Kleist, and Tillmann Miltzow. The complexity of the Hausdorff distance. In *Proceedings of the 38th International Symposium on Computational Geometry, SoCG 2022*, volume 224 of *LIPIcs*, pages Art. No. 48, 17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/lipics.socg.2022.48.

27 Tim Janik Junginger. Robustness of the discrete real polynomial hierarchy. Bachelor's thesis, Karlsruher Institut für Technologie, Karlsruhe, 2023. (last accessed 10/19/2023). URL: https://i11www.iti.kit.edu/_media/teaching/theses/ba-junginger-23.pdf.

28 Jeremy C Kirn, Lucas Meijer, Tillmann Miltzow, and Hans L Bodlaender. On equivalent characterizations of the polynomial hierarchy in abstract models of computation. *arXiv preprint*, 2025. arXiv:2510.05894.

29 Stephen Cole Kleene. *Mathematical logic.* Dover Publications, 2002.

30 Pascal Koiran. Computing over the reals with addition and order. *Theoretical Computer Science*, 133(1):35–47, 1994. doi:10.1016/0304-3975(93)00063-B.

31 Leonid A. Levin. Universal search problems. *Problemy Peredachi Informatsii*, 9(3):265–266, 1973.

32 Klaus Meer and Christian Michaux. A survey on real structural complexity theory. *Bull. Belg. Math. Soc. Simon Stevin*, 4(1):113–148, 1997. URL: http://projecteuclid.org/euclid.bbms/1105730626.

33 Klaus Meer and Adrian Wurm. Some structural complexity results for $\exists\mathbb{R}$, 2025. doi:10.48550/arXiv.2502.00680.

34 Nimrod Megiddo. A general NP-completeness theorem. In *From Topology to Computation: Proceedings of the Smalefest*, pages 432–442. Springer, 1990. doi:10.1007/978-1-4612-2740-3_39.

35 Eike Neumann and Arno Pauly. A topological view on algebraic computation models. *Journal of Complexity*, 44:1–22, 2018. doi:10.1016/j.jco.2017.08.003.

36 Philippos Papayannopoulos. Unrealistic models for realistic computations: how idealisations help represent mathematical structures and found scientific computing. *Synthese*, 199(1):249–283, 2021. doi:10.1007/s11229-020-02654-8.

37 Bruno Poizat. *Les Petits Cailloux : Une Approche Modèle-Théorique de L'Algorithmie*. Lyon : Aléas, 1995.

38 Marcus Schaefer, Jean Cardinal, and Tillmann Miltzow. The existential theory of the reals as a complexity class: A compendium, 2024. doi:10.48550/arxiv.2407.18006.

39 Marcus Schaefer and Daniel Štefankovič. Fixed points, Nash equilibria, and the existential theory of the reals. *Theory of Computing Systems*, 60(2):172–193, 2017. doi:10.1007/s00224-015-9662-0.

40 Marcus Schaefer and Daniel Štefankovič. Beyond the existential theory of the reals. *Theory of Computing Systems*, 2023. doi:10.1007/s00224-023-10151-x.

41 Marcus Schaefer and Daniel Štefankovič. The complexity of tensor rank, 2024. doi:10.48550/arXiv.1612.04338.

42 John C. Shepherdson and Howard E. Sturgis. Computability of recursive functions. *Journal of the ACM*, 10(2):217–255, 1963. doi:10.1145/321160.321170.

43 Larry J Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976. doi:10.1016/0304-3975(76)90061-X.

44 Alfred Tarski. *A Decision Method for Elementary Algebra and Geometry*. The Rand Corporation, Santa Monica, CA, 1948.

45 Dirk van Dalen. *Logic and Structure*. Springer, 1994. doi:10.1007/978-1-4471-4558-5.