# A Logical Characterization of Constant-Depth Circuits over the Reals

**Timon Barlag**
Institut für Theoretische Informatik, Leibniz Universität Hannover, Germany
https://www.thi.uni-hannover.de/barlag
barlag@thi.uni-hannover.de

**Heribert Vollmer**
Institut für Theoretische Informatik, Leibniz Universität Hannover, Germany
https://www.thi.uni-hannover.de/vollmer
vollmer@thi.uni-hannover.de

**Abstract**

In this paper we give an Immerman's Theorem for real-valued computation. We define circuits operating over real numbers and show that families of such circuits of polynomial size and constant depth decide exactly those sets of vectors of reals that can be defined in first-order logic on ℝ-structures in the sense of Cucker and Meer.

Our characterization holds both non-uniformly as well as for many natural uniformity conditions.

**2012 ACM Subject Classification** Theory of computation → Circuit complexity; Theory of computation → Complexity classes; Theory of computation → Finite Model Theory; Theory of computation → Complexity theory and logic

**Keywords and phrases** Computation over the reals, descriptive complexity, constant-depth circuit families

## 1 Introduction

Computational complexity theory is a branch of theoretical computer science which focuses on the study and classification of problems with regard to their innate difficulty. This is done by dividing these problems into classes, according to the amount of resources necessary to solve them using particular models of computation. One of the most prominent such models is the Turing machine – a machine operating sequentially on a fixed, finite alphabet.

If one wishes to study problems based on their parallel complexity or in the domain of the real numbers, one requires different models of computation. Theoretical models exist both for real-valued sequential and for real-valued parallel computation, going back to the seminal work by Blum, Shub and Smale [1]. Unlike Turing machines, machines over ℝ obtain not an unstructured sequence of bits as input but a vector of real numbers or an (encoding of an) ℝ-structure. The respective parallel model we are going to have a closer look at is a real analogue to the arithmetic circuit (see, e.g., [12]), which, as its name suggests, resembles electrical circuits in its functioning, however, contrary to these they operate not on electrical signals, i.e., Boolean values, but real numbers.

Descriptive complexity is an area of computational complexity theory, that groups decision problems into classes not by bounds on the resources needed for their solution but by considering the syntactic complexity of a logical formalism able to specify the problems. Most well-known is probably Fagin's characterization of the class NP as those problems that can be described by existential second-order formulas of predicate logic [6]. Since then, many complexity classes have been characterized logically. Most important in our context is a characterization obtained by Neil Immerman, equating problems decidable by (families of) Boolean circuits of polynomial size and constant depth with those describable in first-order logic:

▶ **Theorem 1** ([11]). $\mathrm{AC}^0 = \mathrm{FO}$.

An important issue in circuit complexity is *uniformity*, i.e., the question if a finite description of an infinite family of circuits exists, and if yes, how complicated it is to obtain it. Immerman's Theorem holds both non-uniformly, i.e., under no requirements on the constructability of the circuit family, as well as for many reasonable uniformity conditions. In the non-uniform case, first-order logic is extended by allowing access to arbitrary numerical predicates, in symbols: non-uniform $\mathrm{AC}^0 = \mathrm{FO}[\mathrm{Arb}]$.

The rationale behind the descriptive approach to complexity is the hope to make tools from logic on expressive power of languages available to resource-based complexity and use non-expressibility results to obtain lower bounds on resources such as time, circuit size or depth, etc.

Descriptive complexity seems even more pertinent for real-valued computation than for computation on bit-strings, because formulas just like computation models over $\mathbb{R}$ operate on structured inputs. Usually one considers *metafinite structures*, that is finite first-order structures enriched with a set of functions into another possibly infinite structure, in our case the real numbers $\mathbb{R}$. This study was initiated by Grädel and Meer [8], presenting logical characterizations of $\mathrm{P}_\mathbb{R}$ and $\mathrm{NP}_\mathbb{R}$. Continuing this line of research, Cucker and Meer showed a few logical characterizations for bounded fan-in real arithmetic circuit classes [4], which is what this paper builds on. Cucker and Meer first proved a characterization of $\mathrm{P}_\mathbb{R}$ using fixed-point logic, and building on this characterized the classes of the NC-hierarchy (bounded fan-in circuits of polynomial size and polylogarithmic depth) restricting the number of updating in the definition of fixed points to a polylogarithmic number. They leave out the case of the smallest interesting circuit class $\mathrm{AC}^0$. We now expand on their research by making the framework of logics over metafinite structures amenable for the description of *unbounded fan-in circuits*; we are particularly concerned with a real analogue to the class $\mathrm{AC}^0_\mathbb{R}$ and show that it corresponds to first-order logic over metafinite structures:

▶ **Theorem 2.** $\mathrm{AC}^0_\mathbb{R} = \mathrm{FO}_\mathbb{R}$.

Cucker and Meer only note that 'the expressive power of first-order logic is not too big' since it can only describe properties in $\mathrm{NC}^1_\mathbb{R}$. In a sense we close the missing detail in their picture by considering the base-case.

The logical characterization of Theorem 2 holds for arbitrary uniformity conditions based on time-bounded construction of the circuit family, in particular P-uniformity and LT-uniformity. Extending the framework of Cucker and Meer, we also characterize non-uniform $\mathrm{AC}^0_\mathbb{R}$ by first-order logic enhanced with arbitrary numerical predicates.

This paper is structured as follows: In the next section, we introduce the reader to machines and circuits over $\mathbb{R}$ and the complexity classes they define. We also introduce logics over metafinite structures. Section 3 contains our results, first turning to non-uniform circuits and then generalizing our results to different uniformity conditions. We close by mentioning some questions for further work.

## 2 Preliminaries

In the upcoming section, we give an introduction to the machine models and logic over $\mathbb{R}$ used in this paper – which are mostly taken from Cucker and Meer [4] – and some extensions thereof which we will make use of later on.

## 2.1 Machines over $\mathbb{R}$

Machines over $\mathbb{R}$, which were first introduced by Blum, Shub and Smale [1], operate on an unbounded tape of registers containing real numbers. We will define those machines as they have been defined by Cucker and Meer [4]. In order to describe these machines, we will use $\mathbb{R}^\infty$ to denote arbitrarily long $\mathbb{R}$-vectors and for every such $x \in \mathbb{R}^\infty$ we use $|x|$ to denote its length. To talk about these machines' tape contents we also consider the bi-infinite direct sum $\mathbb{R}_\infty$ whose elementes have the form

$$(..., x_{-2}, x_{-1}, x_0, x_1, x_2, ...)$$

where $x_i \in \mathbb{R}$ for all $i \in \mathbb{Z}$ and $x_k = 0$ for sufficiently large $|k|$. For $\mathbb{R}_\infty$, we define the operations *shift left* $\sigma_\ell \colon \mathbb{R}_\infty \to \mathbb{R}_\infty$ and *shift right* $\sigma_r \colon \mathbb{R}_\infty \to \mathbb{R}_\infty$ which shift the indices of the elements of $\mathbb{R}_\infty$, e.g.

$$\begin{array}{lcccccccclccccccccl}
 & ... & x_{-2} & x_{-1} & x_0 & x_1 & x_2 & ... & & & & ... & x_{-2} & x_{-1} & x_0 & x_1 & x_2 & ... & \\
\sigma_\ell(( & ..., & 0, & \pi, & e, & 5, & 0, & ... & )) & = & ( & ..., & \pi, & e, & 5, & 0, & 0, & ... & ) \\
\sigma_r(( & ..., & 0, & \pi, & e, & 5, & 0, & ... & )) & = & ( & ..., & 0, & 0, & \pi, & e, & 5, & ... & )
\end{array}$$

▶ **Definition 3** (Definition 1 [4])**.** A *machine over* $\mathbb{R}$ consists of an input space $\mathcal{I} = \mathbb{R}^\infty$, an output space $\mathcal{O} = \mathbb{R}^k$ (for some $k \leq \infty$), a state space $\mathcal{S} = \mathbb{R}_\infty$ and a finite connected directed graph with nodes that are labelled $1...N$ for some $N \in \mathbb{N}$ and each of which has one of the following types:

There are only one *input node* and one *output node*. The input node is labelled with 1 and is associated with its next node $\beta(1)$ and the input map $g_I \colon \mathbb{R}^\infty \to \mathbb{R}_\infty$. The output node is labelled with $N$. Once it is reached the computation halts and its result is placed in the output space by the output map $g_O \colon \mathbb{R}_\infty \to \mathbb{R}^k$. It therefore has no next nodes. A *computation node* $m$ is associated with a next node $\beta(m)$ and a map $g_m \colon \mathbb{R}_\infty \to \mathbb{R}_\infty$ where $g_m$ is a polynomial or rational map (i.e., a division of polynomials) on a finite number of coordinates and the identity on all others. A *branch node* $m$ is associated with two nodes: $\beta^+(m)$ and $\beta^-(m)$. The next node of $m$ is $\beta^+(m)$ if $x_0 \geq 0$ and $\beta^-(m)$ otherwise. (Here $x_0$ denotes the zeroth coordinate of the vector $x \in \mathcal{S}$ representing the current state.) A *shift node* $m$ is associated with a next node $\beta(m)$ and a map $\sigma \colon \mathbb{R}_\infty \to \mathbb{R}_\infty$, where $\sigma$ is either a left or a right shift.

The input map $g_I$ places an input $(x_1, ..., x_n) \in \mathbb{R}^\infty$ in $(..., 0, n, x_1, ..., x_n, 0, ...) \in \mathbb{R}_\infty$ where the size of the input $n$ is stored in the zeroth coordinate. When the output space is $\mathbb{R}^\infty$, $g_O$ is the identity map on the first $m$ coordinates of $\mathbb{R}_\infty$, where $m$ is the number of consecutive ones stored in the negative coordinates of $\mathbb{R}_\infty$. If the output space is $\mathbb{R}^k$ for some $k \in \mathbb{N}$, we take $g_O$ as the identity restricted to the first $k$ coordinates of $\mathbb{R}_\infty$.

▶ **Definition 4** (Definition 1 [4])**.** For any given machine $M$, we denote by $f_M$ the function which yields the output of $M$ when given an input $x \in \mathbb{R}^\infty$ and call that function the *input-output function* of $M$. For any function $f \colon \mathbb{R}^\infty \to \mathbb{R}^k$, $k \leq \infty$ we say that $f$ is *computable* if there is a machine $M$ such that $f_M = f$. Additionally, we say that a set $A \subseteq \mathbb{R}^\infty$ is *decidable* if there is a machine $M$ computing its characteristic function.

▶ **Definition 5.** A $\mathbb{R}$-machine is said to work in time $f(n)$ if for every input $x \in \mathbb{R}^\infty$, $M$ reaches its output node after at most $f(n)$ steps. Although generally, $n$ can be anything, in this paper we are mostly concerned with the case where $n$ is part of the input or its length.

We say that $M$ works in *polynomial time* if it works in time bounded by $\mathcal{O}(n^{\mathcal{O}(1)})$. Analogously, we say that $M$ works in *logarithmic time* if it works in time bounded by $\mathcal{O}(\log n)$.

## 2.2 Arithmetic Circuits over $\mathbb{R}$

Arithmetic circuits over $\mathbb{R}$ were first introduced by Cucker [3] and are our main model of computation. We will define them in analogy to how they were defined by Cucker and Meer [4], however in this paper we consider unbounded fan-in. Also, we disallow division and subtraction gates, since it can be shown that losing those gate types does not change computational power within polynomial size.

▶ **Definition 6.** We define the *sign* function and one variation as follows:

$$sign(x) := \begin{cases} 1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0 \end{cases} \qquad sign'(x) := \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$

Since $sign'$ and $sign$ can be obtained from one another other, as $sign'(x) = sign(sign(x)+1)$ and $sign(x) = sign'(x) - sign'(-x)$, we will use both freely whenever we have either one available.

▶ **Definition 7.** An *arithmetic circuit* $C$ over $\mathbb{R}$ is a directed acyclic graph. Its nodes (also called gates) can be of the following types:

| | |
|---|---|
| *Input nodes* | have indegree 0 and contain the respective input values of the circuit. |
| *Constant nodes* | have indegree 0 and are labelled with real numbers. |
| *Arithmetic nodes* | can have an arbitrary indegree only bounded by the number of nodes in the circuit. They can be labelled with either $+$ or $\times$. |
| *Sign nodes* | have indegree 1. |
| *Output nodes* | have indegree 1 and contain the output values of the circuit after the computation. |

Nodes cannot be predecessors of the same node more than once, which leads to the outdegree of nodes in these arithmetic circuits being bounded by the number of gates in the circuit.

In order to later describe arithmetic circuits, we associate with each gate a number which represents its type. For a gate $g$ these associations are as follows:

| $g$ | input | constant | $+$ | $\times$ | sign | output |
|---|---|---|---|---|---|---|
| type | 1 | 2 | 3 | 4 | 5 | 6 |

For convenience, we define auxiliary gates with types 7–12 which do not grant us additional computational power as we show in Lemma 8. Those are arithmetic gates labelled with $-$ or the relation symbols $=$, $<$, $>$, $\leq$ and $\leq$. All of those nodes have indegree 2.

| $g$ | $-$ | $=$ | $<$ | $>$ | $\leq$ | $\geq$ |
|---|---|---|---|---|---|---|
| type | 7 | 8 | 9 | 10 | 11 | 12 |

We will also refer to nodes of the types $8 - 12$ as *relational* nodes.

Arithmetic nodes compute the respective function they are labelled with, sign gates compute the sign function and relational nodes compute the characteristic function of their respective relation. On any input $x$, a circuit $C$ computes a function $f_C$ by evaluating all gates according to their labels. The values of the output gates at the end of the computation are the result of $C$'s computation, i.e., $f_C(x)$.

In order to talk about complexity classes of arithmetic circuits, one considers the circuit's *depth* and the *size*. The depth of a circuit is the longest path from an input gate to an output gate and the size of a circuit is the number of gates in a circuit.

We say that a directed acyclic graph $C_{sub}$ is a *subcircuit* of a circuit $C$, if and only if $C_{sub}$ is weakly connected, all nodes and edges in $C_{sub}$ are also contained in $C$ and it holds that if there is a path from an input gate to a gate $g$ in $C$, then this path also exists in $C_{sub}$. For any node $g$ in $C$, we denote by the subcircuit *induced* by $g$ that subcircuit $C_{sub,g}$ of $C$, of which $g$ is the top node. We then also say that $g$ is the *root node* of $C_{sub,g}$.

A single circuit can only compute a function with a fixed number of arguments, which is why we call arithmetic circuits a *non-uniform* model of computation. In order to talk about arbitrary functions, we need to consider circuit families, i.e., sequences of circuits which contain one circuit for every input length $n \in \mathbb{N}$. The function computed by a circuit family $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$ is the function computed by the respective circuit, i.e.,

$$f_{\mathcal{C}}(x) = f_{C_{|x|}}(x). \tag{1}$$

A circuit family is said to decide a set if and only if it computes the set's characteristic function. For a function $f: \mathbb{N} \to \mathbb{N}$, we say that a circuit family $\mathcal{C}$ is of size $f$ (depth $f$), if the size (depth) of $C_n$ is bounded by $f(n)$.

▶ **Lemma 8.** *For any arithmetic circuit of polynomial size and constant depth which uses gates of the types* $1-12$*, there exists an arithmetic circuit of polynomial size and constant depth computing the same function, which only uses gates of the types* $1-6$*.*

**Proof.** Let $C$ be an arithmetic circuit with $n$ input gates which uses gates of the types $1-12$, with $size(C) \leq n^q$ and $depth(C) = d$ for $q, d \in \mathbb{N}$. We will construct a circuit $C'$ of polynomial size and constant depth which computes the same function as $C$. We start out by $C' = C$ and proceed as follows: First, since we can represent $t_1 \leq t_2$ by

$$t_1 \leq t_2 \equiv t_1 < t_2 \vee t_1 = t_2 \tag{2}$$

for all $t_1, t_2 \in \mathbb{R}$, we replace every $\leq$ gate in $C'$ by a *sign* gate, followed by an addition gate, which in turn has a $<$ gate and a $=$ gate as its predecessors. Those two gates then each have the nodes $p_1$ and $p_2$ as their predecessors.The *sign* and addition gate at the top represent the $\vee$ in this construction. The overall increase in size is 3 per $\leq$ gate, which leads to the overall increase in size being polynomial in the worst case. The increase in depth is at worst 2 per gate on the longest path from an input gate to the output gate, which means that the overall increase in depth is constant. This means that $C'$ still computes the same function as $C$, its size is still polynomial and its depth is still constant in $n$ and $C'$ now does not contain any $\leq$ gates. For $\geq$ gates, we proceed analogously. We continue similarly for the other cases: Since we can represent $t_1 = t_2$ by

$$t_1 = t_2 \equiv sign'(-(t_1 - t_2)^2) \tag{3}$$

for all $t_1, t_2 \in \mathbb{R}$, we replace every $=$ gate in $C'$ with predecessors $p_1$ and $p_2$ by a *sign* gate at the top, followed by an addition gate which in turn has a constant gate labeled 1 and another *sign* gate as its predecessors. This construction represents *sign*$'$. That second *sign* gate then has a subtraction gate as its predecessor, which has a constant node labeled 0 and a $\times$ gate as its predecessors. The $\times$ gate has two $+$ gates as its predecessors, which in turn each have the same subtraction gate as their predecessor. That subtraction gate then has $p_1$ and $p_2$ as its predecessors. Note here that the $+$ gates here essentially work as identity gates, and we only need them, to have the value of $(p_1 - p_2)$ be multiplied with itself in the $\times$ node. The overall increase in size per $=$ gate in this construction is 9, which means that the total overhead in size is still polynomial in the worst case. In terms of depth, the increase is at worst 6 per gate on the longest path from an input gate to the output gate, meaning that the total increase is still constant. After this step, $C'$ computes the same function as $C$, still has polynomial size and constant depth in $n$ and does not contain any $=$ gates. The construction for $<$ gates with predecessors $p_1$ and $p_2$ works similarly. We make use of $t_1 < t_2$ being representable by

$$t_1 < t_2 \equiv 1 - sign'(t_1 - t_2) \tag{4}$$

for all $t_1, t_2 \in \mathbb{R}$. We therefore replace every $<$ gate by a subtraction gate with 1 and a construction for $sign'$ as above as its predecessors. The $sign'$ construction then has a subtraction gate as its predecessor, which in turn has the nodes $p_1$ and $p_2$ as its predecessors.The increase in size per $<$ gate is 6, leading to a polynomial increase at worst and the increase in depth is at worst 4 per $<$ gate on the longest path from an input gate to the output, meaning that the overall overhead is constant. This means that $C'$ still has polynomial size and constant depth in $n$, still computes the same function as $C$ and now does not contain any $<$ gates. We proceed analogously for $>$ gates. For subtraction gates, we proceed similarly, since we can represent $t_1 - t_2$ by

$$t_1 - t_2 \equiv t_1 + (-1) \times t_2 \tag{5}$$

for all $t_1, t_2 \in \mathbb{R}$. We replace every subtraction gate with predecessors $p_1$ and $p_2$ by an addition gate with $p_1$ and a multiplication gate as its predecessors, where the multiplication gate has a constant node labeled $-1$ and the node $p_2$ as its predecessors.For each gate, this introduces an increase in size of 2 per subtraction gate, leading to the overall overhead still being polynomial in the worst case, and an increase in depth of 1 for each gate on the longest path from an input gate to the output gate, which leads to the overall depth still being constant. Therefore, $C'$ still computes the same function as $C$, has polynomial size and constant depth in $n$ and does not contain any subtraction gates. In total, $C'$ has polynomial size in $n$, constant depth in $n$, only contains gates of the types $1 - 6$ and computes the same function as $C$. ◀

▶ **Definition 9.** The class $\mathrm{AC}^0_{\mathbb{R}}$ is the class of sets decidable by arithmetic circuit families over $\mathbb{R}$ of polynomial size and constant depth.

The circuit families we have just introduced do not have any restrictions on the difficulty of obtaining any individual circuit. This means that even with the strict constraints on the depth and size of the circuits for $\mathrm{AC}^0_{\mathbb{R}}$, it still contains some undecidable problems. For this reason, we also consider so-called *uniform* circuit families, i.e., families of circuits $\mathcal{C} = (C_n)_{n \in \mathbb{N}}$ where the gates are numbered, the predecessors of each gate are ordered and where for any given triple of numbers $(n, v_{nr}, p_{idx})$, a corresponding triple $(t, p_{nr}, c)$ can be computed by an $\mathbb{R}$-machine $M$, where

**i.)** $t$ is the type of the $v_{nr}$th gate $v$ in $C_n$,
**ii.)** $p_{nr}$ is the number of the $p_{idx}$th predecessor of $v$ and
**iii.)** $c$ is the value of $v$ if $v$ is a constant gate, the index $i$ if $v$ is the $i$th input gate and 0 otherwise.

If $v$ has less than $p_{idx}$ predecessors, $M$ returns $(t, 0, 0)$ and if $v_{nr}$ does not encode a gate in $C_n$, $M$ returns $(0, 0, 0)$.

If this computation only takes logarithmic time in $n$, we call $\mathcal{C}$ LT-uniform. If it takes polynomial time in $n$, we call $\mathcal{C}$ P-uniform.

For a circuit complexity class $\mathfrak{C}$, we will by $\mathrm{U}_{\mathrm{LT}}$-$\mathfrak{C}$ denote the subclass of $\mathfrak{C}$, which only contains sets definable by LT-uniform circuit families. We will use $\mathrm{U}_{\mathrm{P}}$-$\mathfrak{C}$ to analogously denote those sets in $\mathfrak{C}$ definable by P-uniform families.

## 2.3 $\mathbb{R}$-structures and First-order Logic over $\mathbb{R}$

The logics we use to characterize real circuit complexity classes are based on first-order logic with arithmetics.

▶ **Definition 10** (Definition 7 [4]). Let $L_s$, $L_f$ be finite vocabularies where $L_s$ can contain function and predicate symbols and $L_f$ only contains function symbols. An $\mathbb{R}$-*structure of signature* $\sigma = (L_s, L_f)$ is a pair $\mathcal{D} = (\mathcal{A}, \mathcal{F})$ where $\mathcal{A}$ is a finite structure of vocabulary $L_s$ which we call the *skeleton* of $\mathcal{D}$ whose universe $A$ we will refer to as the *universe* of $\mathcal{D}$ and whose cardinality we will refer to by $|A|$. $\mathcal{F}$ is a finite set which contains functions of the form $X \colon A^k \to \mathbb{R}$ for $k \in \mathbb{N}$ which interpret the function symbols in $L_f$.

We will use $Struct_{\mathbb{R}}(\sigma)$ to refer to the set of all $\mathbb{R}$-structures of signature $\sigma$ and we will assume that for any fixed signature $\sigma = (L_s, L_f)$, we can fix an ordering on the symbols in $L_s$ and $L_f$.

In order to use $\mathbb{R}$-structures as inputs for machines, we encode them in $\mathbb{R}^\infty$ as follows: We choose an arbitrary ranking $r$ on $A$, i.e., a bijection $r \colon A \to \{0, ..., |A| - 1\}$. We then replace all predicates in $L_s$ by their respective characteristic functions and all functions $f \in L_s$ by $r \circ f$. Those functions are then considered to be elements of $L_f$. We represent each of these functions by concatenating their function values in lexicographical ordering on the respective function arguments according to $r$. To encode $\mathcal{D}$ we only need to concatenate all representations of functions in $L_f$ in the order fixed on the signature. We denote this encoding by $\mathrm{enc}(\mathcal{D})$.

In order to be able to compute $|A|$ from $\mathrm{enc}(\mathcal{D})$, we make an exception for functions and predicates of arity 0. We treat those as if they had arity 1, meaning that e.g. we encode a function $f_1() = 3$ as $|A|$ 3s.
Since

$$|\mathrm{enc}(\mathcal{D})| = \sum_{f \in L_f} |A|^{\max\{ar(f), 1\}}, \tag{6}$$

where $ar(f)$ is the arity of $f$, we can reconstruct $|A|$ from the arities of the functions in $L_f$ and the length of $\mathrm{enc}(\mathcal{D})$. We can do so by using for example binary search, since we know that $|A|$ is between 0 and $|\mathrm{enc}(\mathcal{D})|$. We can therefore compute $|A|$ when given $\varphi$ and $|\mathrm{enc}(\mathcal{D})|$ in time logarithmic in $|\mathrm{enc}(\mathcal{D})|$.

### 2.3.1 First-order Logic over $\mathbb{R}$

▶ **Definition 11** (First-order logic). The language of first-order logic contains for each signature $\sigma = (L_s, L_f)$ a set of formulas and terms. The terms are divided into *index terms* which take values in the skeleton and *number terms* which take values in $\mathbb{R}$. These terms are inductively defined as follows:

1. The set of index terms is defined as the closure of the set of variables $V$ under applications of the function symbols of $L_s$.
2. Any real number is a number term.

3. For index terms $h_1, ..., h_k$ and a $k$-ary function symbol $X \in L_f$, $X(h_1, ..., h_k)$ is a number term.
4. If $t_1$, $t_2$ are number terms, then so are $t_1 + t_2$, $t_1 \times t_2$ and $sign(t_1)$.

Atomic formulas are equalities of index terms $h_1 = h_2$ and number terms $t_1 = t_2$, inequalities of number terms $t_1 < t_2$ and expressions of the form $P(h_1, ..., h_k)$, where $P \in L_s$ is a k-ary predicate symbol and $h_1, .., h_k$ are index terms.

The set $\mathrm{FO}_{\mathbb{R}}$ is the smallest set which contains the closure of atomic formulas under the Boolean connectives $\{\wedge, \vee, \neg, \implies, \iff\}$ and quantification $\exists v\psi$ and $\forall v\psi$ where $v$ ranges over $\mathcal{A}$.

Equivalence of and sets defined by $\mathrm{FO}_{\mathbb{R}}$-formulas are done in the usual way, i.e., a formula $\varphi$ defines a set $S$ if and only if the elements of $S$ are exactly the encodings of $\mathbb{R}$-structures under which $\varphi$ holds and two such formulas are said to be equivalent if and only if they define the same set.

### 2.3.2 Extensions to $\mathrm{FO}_{\mathbb{R}}$

In the following, we would like to extend $\mathrm{FO}_{\mathbb{R}}$ by additional functions and relations that are not given in the input structure. To that end, we make a small addition to Definition 10 where we defined $\mathbb{R}$-structures. Whenever we talk about $\mathbb{R}$-structures over a signature $(L_s, L_f)$, we now also consider structures over $(L_s, L_f, L_a)$. The additional vocabulary $L_a$ does not have any effect on the $\mathbb{R}$-structure, but it contains function and relation symbols, which can be used in a logical formula with this signature. This means that any $\mathbb{R}$-structure of signature $(L_s, L_f)$ is also a $\mathbb{R}$-structure of signature $(L_s, L_f, L_a)$ for any alphabet $L_a$.

▶ **Definition 12.** Let $R$ be a set of finite relations and functions. We will write $\mathrm{FO}[R]$ to denote the class of sets $S$ for which there exists a $\mathrm{FO}_{\mathbb{R}}$-sentence $\varphi$ over a signature $\sigma = (L_s, L_f, L_a)$ such that for every input structure there is an interpretation $I$ such that $\varphi$ with interpretation $I$ defines $S$, where $I$ interprets each symbol in $L_a$ as a function or relation from $R$.

With the goal in mind to create a logic which can define sets decided by circuits with unbounded fan-in, we introduce new rules for building number terms: the *sum* and the *product rule*. We will also give another rule, which we call the *maximization rule*, but will later show that we can define this rule in $\mathrm{FO}_{\mathbb{R}}$ and thus do not gain expressive power by using it. We will use this rule to show that we can represent characteristic functions in $\mathrm{FO}_{\mathbb{R}}$.

▶ **Definition 13** (sum, product and maximization rule)**.** Let $t$ be a number term in which the variable $i$ occurs freely with other variables $ow = w_1, ..., w_j$ and let $A$ denote the universe of the given input structure. Then

$$sum_i(t(i, \overline{w})) \tag{7}$$

is also a number term which is interpreted as $\sum_{i \in A} t(i, \overline{w})$. Moreover, $prod_i(t(i, \overline{w}))$ and $\max_i(t(i, \overline{w}))$ are defined analogously.

We will also write $sum_i^q(t(i_1, ..., i_q, \overline{w}))$ to denote $sum_{i_1}(...sum_{i_q}(t(i_1, ..., i_q, \overline{w})))$ for convenience and we will use $prod_i^q$ analogously.

For a logic $\mathcal{L}$, we will by $\mathcal{L} + \mathrm{SUM}_{\mathbb{R}}$, $\mathcal{L} + \mathrm{PROD}_{\mathbb{R}}$ and $\mathcal{L} + \mathrm{MAX}_{\mathbb{R}}$ denote $\mathcal{L}$ extended by the sum rule, the product rule or the maximization rule respectively.

We will now evaluate, which logics can already natively use some of the aforementioned rules. As it turns out, the maximization rule can be used in $\mathrm{FO}_{\mathbb{R}}$ without any extensions and the sum and product rule extend neither $\mathrm{FO}_{\mathbb{R}}[\mathrm{Arb}]$ nor a polynomial extension of $\mathrm{FO}_{\mathbb{R}}$ which we will see later.

▶ **Lemma 14.** $\mathrm{FO}_{\mathbb{R}} = \mathrm{FO}_{\mathbb{R}} + \mathrm{MAX}_{\mathbb{R}}$

**Proof.** Let $\varphi$ be a $\mathrm{FO}_{\mathbb{R}}$ formula which contains $\max_i$-constructions, i.e., number terms of the form $\max_i(t(i,\overline{w}))$ for a number term $t$. We will show that for every such formula, we can construct another $\mathrm{FO}_{\mathbb{R}}$-formula $\varphi'$ which is equivalent to $\varphi$ but which does not contain the term $\max_i(t(i,\overline{w}))$. Since $\max_i$-constructions are number terms, whenever they occur, they are part of atomic (sub-)formulas. For this reason, we only need to show, how to turn atomic formulas with $\max_i$-constructions into semantically equivalent formulas (that are not necessarily atomic anymore). For a given atomic formula with $\max_i$-constructions $\varphi$, define $\varphi'$ as follows: Let $\varphi \triangleq t_1 = t_2$ and let $\max_{i_1}, ..., \max_{i_k}$ be the $\max_i$-occurrences of $\varphi$, ordered by level of nesting, where $\max_{i_1}$ has the lowest level of nesting, the nesting of $\max_{i_2}$ is either the same as $\max_{i_1}$ or greater by 1 and so on. We assume without loss of generality that the variables $x_1, ..., x_k$ and $y_1, ..., y_k$ do not occur in $\varphi$. We also assume for now that there is only one occurrence of $\max_i$ at the lowest level of nesting and that $t_1$ consists only of that outermost $\max_i$-construction, i.e., $t_1 \triangleq \max_{i_1}(F_1(i_1, \overline{w}_1))$. To now construct $\varphi'$, we go through the $\max_i$-occurrences in $\varphi$ in reverse order of nesting, i.e., from the deepest level to the shallowest, and for each occurrence $\max_{i_m}(F_m(i_m, \overline{w}_m))$, we create a subformula $\psi_m$, which ensures that $F_m$ is being maximized with respect to $i_m$. We will use new variables $x_1, ..., x_k, y_1, ..., y_k$ in the subformulas, which will be quantified later, when we connect those subformulas to construct $\varphi'$. $\varphi'$ will then have the form

$$\varphi' \triangleq \exists x_1 \forall y_1 ... \exists x_k \forall x_k \psi_k \wedge ... \wedge \psi_1 \wedge \widehat{\varphi}, \tag{8}$$

where $\widehat{\varphi}$ represents the structure of $\varphi$ without any $\max_i$-constructions. In our case, $\widehat{\varphi}$ would just be $F_1(x_1, \overline{w}_1) = t_2$.

We start with the term $\max_{i_k}(F_k(i_k, i_{k_1}, ..., i_{k_j}, \overline{w}_k))$, where $F_k$ is the number term in $\varphi$ getting maximized by $\max_{i_k}$, $i_{k_1}, ..., i_{k_j}$ are the variables used in $F_k$ from $\max_i$-constructions which occur at lower levels of nesting in $\varphi$ and $\overline{w}_k$ are all other variables used in $F_k$.

We now create the subformula

$$\psi_{i_k} \triangleq F_k(x_k, x_{k_1}, ..., x_{k_j}, \overline{w}_k) \geq F_k(y_k, x_{k_1}, ..., x_{k_j}, \overline{w}_k), \tag{9}$$

which makes sure that $F_k$ is maximal with respect to $i_k$.

Afterwards, we proceed in reverse order of nesting with the other $\max_i$-occurrences in $\varphi$ (meaning that $\max_{i_{k-1}}$ is next) and create the subformulas $\psi_{k-1}, ..., \psi_1$ similarly. For $m \in (k-1, ..., 1)$, we proceed as follows:

Let $\max_{i_m}(F_m(i_m, i_{m_1}, ..., i_{m_j}, \overline{w}_m))$ be the occurrence of $\max_{i_m}$ in $\varphi$ with analogous $F_m, i_m, i_{m_1}, ..., i_{m_j}, \overline{w}_m$ as before. Now replace in $F_m$ all $\max_i$-constructions $\max_i(F_i(i, \overline{w}))$ – where $\overline{w}$ are all variables used in $F_i$ except for $i$ – by parentheses around $F_i$, i.e., $\max_i(F_i(i, \overline{w}))$ would just become $(F_i(i, \overline{w}))$. Denote the result by $F'_m$. We then define

$$\psi_m \triangleq F'_m(x_m, x_{m_1}, ..., x_{m_j}, \overline{w}_m) \geq F'_m(y_m, x_{m_1}, ..., x_{m_j}, \overline{w}_m). \tag{10}$$

Finally, we define

$$\varphi' \triangleq \exists x_1 \forall y_1 ... \exists x_k \forall y_k \;\psi_k \wedge ... \wedge \psi_1 \wedge F'_1(x_1, \overline{w}_1) = t_2. \tag{11}$$

This construction now works for our strong assumption that $t_1 \triangleq \max_{i_1}(F_1)$. However, we only require the following modifications to make it generally applicable: If $\varphi$ contains only one $\max_i$-construction at the lowest level, but then operates on that construction, we can just add the context of that $\max_i$-construction to the term $F_1'$ in $\varphi'$. For example if $\varphi \triangleq 7 = \max_i(F(i)) + 1$, then we could just add the '+1' to the $F_1'(x_1, \overline{w}_1)$ in Formula 11. If $\varphi$ contains several $\max_i$-constructions at the lowest level of nesting, then we can construct as we have previously and just add the subformulas to the conjunction in $\varphi'$.

$\varphi'$ now does not contain any $\max_i$-constructions and is therefore a valid $\mathrm{FO}_\mathbb{R}$-formula. Since for every $\max_i$-occurrence in $\varphi$, there is a subformula in the conjunction of $\varphi'$ making sure that the term maximized by $\max_i$ in $\varphi$ is also maximal in $\varphi'$, $\varphi'$ is also semantically equivalent to $\varphi$.

We can construct $\varphi'$ analogously, if both, $t_1$ and $t_2$ contain $\max_i$-constructions or if $\varphi \triangleq t_1 < t_2$. We have therefore shown that for any $\mathrm{FO}_\mathbb{R}$-formula with $\max_i$-constructions, there exists a semantically equivalent formula which does not contain any such constructions. ◀

▶ **Definition 15.** Let Arb denote the set of all finite relations and functions.

▶ **Lemma 16.** $\mathrm{FO}_\mathbb{R}[\mathrm{Arb}] = \mathrm{FO}_\mathbb{R}[\mathrm{Arb}] + \mathrm{SUM}_\mathbb{R} = \mathrm{FO}_\mathbb{R}[\mathrm{Arb}] + \mathrm{PROD}_\mathbb{R}$

**Proof.** In order to prove this, we will take an arbitrary $\mathrm{FO}_\mathbb{R}[\mathrm{Arb}]$-sentence $\varphi$ in which number terms of the form $sum_i(t(i, \overline{w}))$ occur and then create an $\mathrm{FO}_\mathbb{R}[\mathrm{Arb}]$-term $\varphi'$ which is semantically equivalent to $\varphi$ but which does not contain any such constructions. Note that the set $L_a$ of the signature of $\varphi'$ will be different to that of $\varphi$. Let $\varphi$ be a valid $\mathrm{FO}_\mathbb{R}[\mathrm{Arb}]$ sentence of signature $\{L_s, L_f, L_a\}$ with the exception that it contains number terms of the form $sum_i(t(i, \overline{w}))$, where $t(i, \overline{w})$ is a number term (which may again contain $sum_i$-constructions). Without loss of generality, we assume that for all number terms of the form $sum_i(t(i, \overline{w}))$, there is no symbol $sum_{(t,i)}$ in the signature of $\varphi$. We now construct a number term $\varphi'$ which is equivalent to $\varphi$ but which does not contain any constructions of the aforementioned form. We define $\varphi'$ step by step. We start out by $\varphi' \triangleq \varphi$. We now take any such instance of a number term $sum_i(t(i, \overline{w}))$ in $\varphi'$ where $t$ itself does not contain any instances of $sum_i$-constructions. Let $\mathcal{D} = \{L_s, L_f, L_a\}$ be the signature of $\varphi'$. Without loss of generality, $\mathcal{D}$ does not contain the symbol $sum_{(t,i)}$. Now add to the set $L_a$ of $\mathcal{D}$ the function symbol $sum_{(t,i)}$ of arity $j = |\overline{w}|$ and replace in $\varphi'$ the term $sum_i(t(i, \overline{w}))$ by $sum_{(t,i)}(\overline{w})$. The interpretation of $sum_{(t,i)}(y_1, ..., y_j)$ for any $(y_1, ..., y_j) \in A^j$ is the sum over all different values $v \in A$ for $i$ if $i$ is in $t$ replaced by $v$ and $w_k$ is replaced by $y_k$ for all $1 \leq k \leq j$, i.e., for all $(y_1, ..., y_k) \in A$:

$$sum_{(t,i)}(y_1, ..., y_j) \coloneqq \sum_{i \in A} t(y_1, ..., y_k) \tag{12}$$

The resulting formula $\varphi'$ is now semantically equivalent to $\varphi$, since we just moved the interpretation of the symbol $sum_i$ to the function symbol $sum_{(t,i)}$, but it does not contain the instance of $sum_i(t(i, \overline{w}))$ that we just removed. If we repeat this process for all remaining occurrences of $sum_i$ in $\varphi'$, we arrive at a sentence which is semantically equivalent to $\varphi$ but which does not contain any instances of $sum_i$. $\mathrm{FO}_\mathbb{R}[\mathrm{Arb}] = \mathrm{FO}_\mathbb{R}[\mathrm{Arb}] + \mathrm{PROD}_\mathbb{R}$ can be shown analogously. ◀

▶ Remark 17. For the sake of simplicity we will in the following only consider *functional* $\mathbb{R}$-*structures*, i.e., $\mathbb{R}$-structures whose signatures do not contain any predicate symbols. This does not restrict what we can express, since any relation $P \in A^k$ can be replaced by its characteristic function $\chi_P \colon A^k \to \mathbb{R}$.

As mentioned before, the reason why we need the maximization rule is that we would like to write characteristic functions as number terms. For a first-order formula $\varphi(v_1, ..., v_r)$ we define its characteristic function $\chi[\varphi]$ on a structure $\mathcal{D}$ by

$$\chi[\varphi](a_1, ..., a_r) = \begin{cases} 1 & \text{if } \mathcal{D} \models \varphi(a_1, ..., a_r) \\ 2 & \text{otherwise} \end{cases} \tag{13}$$

The following result is a slight modification of a result presented by Cucker and Meer [4].

▶ **Proposition 18.** *Let $R$ be a set of functions and predicates. For every $FO_{\mathbb{R}}[R]$-formula $\varphi(v_1, ..., v_k)$, there is a $FO_{\mathbb{R}}[R]$ number term describing $\chi[\varphi]$.*

# 3 Characterizing $\mathrm{AC}^0_{\mathbb{R}}$

In this section, we give descriptive complexity results for the non-uniform set $\mathrm{AC}^0_{\mathbb{R}}$ and some of its uniform subsets. In order to achieve this, we use the previously defined first-order logic over the real numbers and the extensions we defined.

## 3.1 A Characterization for non-uniform $\mathrm{AC}^0_{\mathbb{R}}$

First of all we show an equality which is close to a classical result shown by Immermann [10]. We show that extending our first-order logic over the reals with arbitrary functions lets us exactly describe the non-uniform set $\mathrm{AC}^0_{\mathbb{R}}$.

In the proof for the upcoming theorem, we make use of a convenient property of circuits deciding $\mathrm{AC}^0_{\mathbb{R}}$-sets, namely that for each of those circuits, there exist *tree-like* circuits deciding the same set. We call these circuits *full trees*.

▶ **Lemma 19.** *For every $\mathrm{AC}^0_{\mathbb{R}}$-circuit family $(C_n)_{n\in\mathbb{N}}$, there exists an $\mathrm{AC}^0_{\mathbb{R}}$-circuit family $(C'_n)_{n\in\mathbb{N}}$ computing the same function, such that for all $n \in \mathbb{N}$ and every gate $v$ in $C'_n$, every path from an input gate to $v$ has the same length.*

**Proof.** In order to prove this, we construct, for any given $\mathrm{AC}^0_{\mathbb{R}}$-family $\mathcal{C}$, an $\mathrm{AC}^0_{\mathbb{R}}$-circuit family $\mathcal{C}$' which exhibits this property. Note that since we are talking about circuits deciding sets, we know that the given circuits will each only have one output gate. We give a generic construction for turning any circuit of $\mathcal{C}$ into one of $\mathcal{C}$' which computes the same function. To achieve this, we proceed in two steps: for a given circuit we first create an equivalent circuit where only the input gates have an outdegree $> 1$, and which is thus very tree-like. Secondly, we will pad all paths from input gates to the output gate to have the same length. Due to the tree-likeness of our circuit, this property then translates to all nodes in the circuit.
Step one: Let $C_n$ be an $\mathrm{AC}^0_{\mathbb{R}}$-circuit which contains non-input gates with outdegree $> 1$. We would like to get rid of those gates. To accomplish this, consider all subcircuits of $C_n$ induced by non-input gates which have outdegree $> 1$ in which every other non-input gate has outdegree 1. Since $C_n$ is acyclic, at least one such subcircuit must exist. These subcircuits are all distinct from each other, because only their respective root node has multiple successors (barring the input gates). For each of those subcircuits $C_n^{sub,g}$ now proceed as follows: Let $g$ be the root node of $C_n^{sub,g}$. Replace each connection beyond the first from $g$ to a successor by a copy of $C_n^{sub,g}$, i.e., by a subcircuit which has the same input gates as $C_n^{sub,g}$ and where all other gates and connections are copies of the gates and connections in $C_n^{sub,g}$. After this step, the longest distance between the output node and a non-input node $g$ with multiple successors whose induced subcircuit contains no non-input gates with $> 1$ successors is

reduced by at least one. Repeat this process until there are no more non-input gates with multiple successors in $C_n$ and denote the circuit after the $i$th step by $C_n^i$. Let $q \in \mathbb{N}$ be such that $size(C_n) < n^q$. We show that the size of the circuit resulting in this process is still a polynomial in $n$ by induction. To be exact, we show that $size(C_n^i) < n^{q*(1+2i)}$ for all $i \in \mathbb{N}$.
*Base case* $C_n^1$: After the first step of this process we have increased the size of $C_n$ by less than $n^q$ for each of less than $n^q - 1$ root nodes (because the output node cannot be such a root node) and each of less than $n^q$ successors thereof. This means that $size(C_n^1) < (n^q) + (n^q - 1) * (n^q)^2 < (n^q)^3 = n^{q*(1+2)}$.
*Induction step* $C_n^k \to C_n^{k+1}$: In the $k$th step, we replaced all subcircuits induced by non-input nodes with multiple successors in $C_n^{k-1}$ by copies. This means that all root nodes we consider in the $k+1$th step have not been altered yet and that there are therefore less than $n^q$ of those. Additionally, since all those root nodes have multiple successors, no nodes reachable from these roots have been altered either. Therefore, each of those roots has less than $n^q$ successors. The subcircuits these nodes induce, however, have been altered and are therefore of size less than $n^{q*(1+2k)}$. After the $k+1$th step, we have replaced less than $n^q - 1$ subcircuits of size less than $n^{q*(1+k)}$ by less than $n^q$ copies each. Therefore it follows that $size(C_n^{k+1}) < n^{q*(1+2k)} + n^{q*(1+2k)} * n^q * (n^q - 1) < n^{q*(1+2k)} * n^{2q} = n^{q*(1+2(k+1))}$.

Let $C'_n$ denote the circuit after finishing the procedure above. Since we reduce the distance of the output node to the furthest such root node in each step, we only need to execute this process for a constant number of steps. Therefore $C'_n = C_n^k$ for some $k \in \mathbb{N}$, which means that the size of the $C'_n$ is still a polynomial in $n$. The depth of $C'_n$ is still constant as the procedure we performed did not alter the circuits depth. Additionally, since we only added copies of subcircuits in place of subcircuits with several successors, we also did not change the computed function. This means that $C_n$ is still an $\mathrm{AC}^0_{\mathbb{R}}$-circuit which computes the same function as $C_n$ but does not contain any non-input gates with multiple successors.
Step two: We know that $C'_n$ does not have any nodes with outdegree $> 1$ beyond the input gates. Consider now all paths $p_1, ..., p_k$ from an input gate to the singular output gate. Let $d$ be the depth of $C'_n$, i.e., the length of the longest path from any input gate to the output gate. For every path $p_i$ now add $d - length(p_i)$ successive addition gates in between the first node of $p_i$ – the respective input gate – and $p_i$'s second node. This ensures that all paths from input gates to the output gate have the same length. Denote the resulting circuit by $C''_n$. As we will see, this also results in the property that we wanted in the first place: for every node $v$ in $C''_n$, all paths from input gates to $v$ have the same length. We show this by contradiction: Assume that there is a gate $v$ in $C''_n$ to which there are two paths from input gates with different lengths. We know that $v$ and all its successors have outdegree $\leq 1$, therefore we know that there can be only one path from $v$ to the output node. That means that there would also be two paths of different length from input gates to the ouput node, which is a contradiction. As in step one, we have not added any depth to $\mathcal{C}'$, but we increased its size. The increase in size, however, is less than $size(C'_n) * n * depth(C'_n)$, since there is at most one path from input to output for each outgoing edge of the input gates. There are $n$ input gates and there can be at most $size(C'_n)$ outgoing edges from those, so we have at most $n * size(C'_n)$ paths. Each path now gets padded by less than $depth(C'_n)$ nodes. In the end, since $size(C''_n)$ is a polynomial in $n$ and $depth(C''_n)$ is constant with respect to $n$, the resulting circuit exhibits the properties that we desire and is still of constant depth and polynomial size. It also computes the same function as $C'_n$, since addition gates with only a singular predecessor are essentially just identity gates. ◀

▶ **Theorem 20.** $\mathrm{FO}_{\mathbb{R}}[\mathrm{Arb}] = \mathrm{AC}^0_{\mathbb{R}}$.

**Proof.** $\text{FO}_{\mathbb{R}}[\text{Arb}] \subseteq \text{AC}^0_{\mathbb{R}}$: To show that $\text{FO}_{\mathbb{R}}[\text{Arb}]$ is included in $\text{AC}^0_{\mathbb{R}}$, we will show that for any $\text{FO}_{\mathbb{R}}[\text{Arb}]$-sentence $\varphi$, we can create an $\text{AC}^0_{\mathbb{R}}$ circuit family which decides exactly the set defined by $\varphi$. Given a fixed size $n$ of input $\mathbb{R}$-structures $\mathcal{D} = (\mathcal{A}, \mathcal{F})$ ($n = |\text{enc}(\mathcal{D})|$), we can for any $\text{FO}_{\mathbb{R}}$-formula reconstruct $|A|$ from $n$ as described on page 7. We will denote $|A|$ by $u$. For any such formula $\varphi$ with exactly $k$ free variables $x_1, ..., x_k$ and for all $1 \leq m_1, ..., m_k \leq u$ we then create an arithmetic circuit $C_n^{\varphi(m_1,...,m_k)}$ with the following property: For any input structure $\mathcal{D}$ it holds that $\mathcal{D} \models \varphi$ if and only if $\text{enc}(\mathcal{D})$ is accepted by $C_n^{\varphi(m_1,...,m_k)}$, where $x_i$ is substituted by $m_i$ for all $1 \leq i \leq k$. (That means that any such circuit $C_n^{\varphi(m_1,...,m_k)}$ for a formula $\varphi$ outputs either 1 or 0.) At the very top of the circuit is the output node. The rest of the circuit is defined by induction. A formula $\varphi$ with $k$ free variables $x_1, ..., x_k$ and natural numbers $m_1, ..., m_k$, with $1 \leq m_i \leq u$ for all $i$ are given.

1. Let $\varphi \triangleq \exists y \psi(y)$. If $y$ does not occur free in $\psi$, then $C_n^{\varphi(m_1,...,m_k)} = C_n^{\psi(m_1,...,m_k)}$. Otherwise, the free variables in $\psi$ are $x_1, ..., x_k, y$. $C_n^{\varphi(m_1,...,m_k)}$ now consists of a sign gate with an unbounded fan-in addition gate as its predecessor which in turn has the circuits $C_n^{\psi(m_1,...,m_k,i)}$ as its predecessors for $1 \leq i \leq u$.
2. If $\varphi \triangleq \forall y \psi(y)$, then $C_n^{\varphi(m_1,...,m_k)}$ is defined as in the existential case, but with a multiplication gate below the sign gate.
3. Let $\varphi \triangleq \neg\psi$. Then $C_n^{\varphi(m_1,...,m_k)}$ consists of a subtraction gate, which subtracts the sign of $C_n^{\psi(m_1,...,m_k)}$ from 1.
4. Let $\varphi \triangleq \psi \wedge \xi$. Then $C_n^{\varphi(m_1,...,m_k)}$ consists of a sign gate followed by a multiplication gate with $C_n^{\psi(m_1,...,m_k)}$ and $C_n^{\xi(m_1,...,m_k)}$ as its predecessors. (The sign gate is technically not necessary for this case, but we keep it for consistency with e.g. the construction for $\vee$.)
5. If $\varphi \triangleq \psi \vee \xi$, $\varphi \triangleq \psi \implies \xi$ or $\varphi \triangleq \psi \iff \xi$, then $C_n^{\varphi(m_1,...,m_k)}$ follows analogously to $\varphi \triangleq \psi \wedge \xi$.
6. Let $\varphi \triangleq h_1 = h_2$ for index terms $h_1, h_2$. Then $C_n^{\varphi(m_1,...,m_k)}$ consists of an equality gate with the circuits $C_n^{h_1(m_1,...,m_k)}$ and $C_n^{h_2(m_1,...,m_k)}$ as its predecessors.
7. If $\varphi \triangleq t_1 = t_2$ for number terms $t_1, t_2$, then $C_n^{\varphi(m_1,...,m_k)}$ is defined analogously to the case with index terms.
8. Let $\varphi \triangleq t_1 < t_2$ for number terms $t_1, t_2$. Then $C_n^{\varphi(m_1,...,m_k)}$ consists of a $<$ gate with $C_n^{t_1(m_1,...,m_k)}$ and $C_n^{t_2(m_1,...,m_k)}$ as its predecessors.

For the cases 6, 7 and 8, we also need to show how non-formula index and number terms can be evaluated by our circuit. We will define these by induction as well. Let $h$ be an index term:

1. Let $h \triangleq x$ for $x \in V$. Then $x$ must be $x_i$ for an $i \in 1, ..., k$ and have previously been quantified. Then $C_n^{h(m_1,...,m_k)}$ consists of the constant gate with value $m_i$.
2. Let $h \triangleq f(h_1, ..., h_\ell)$ for a $\ell$-ary function symbol $f \in L_a$ and index terms $h_1, ..., h_\ell$. Then $C_n^{h(m_1,...,m_k)}$ consists of an unbounded addition gate at the top with $u^\ell$ unbounded multiplication gates as its predecessors – one for each possible input to $f$, i.e., for each different encoded tuple $(a_1, ..., a_\ell), a_i \in A$. Each of these multiplication gates has then $\ell$ equality gates as their predecessors – one for each element of $(a_1, ..., a_\ell)$ – and one constant gate containing the function value $f(a_1, ..., a_\ell)$. Of the equality gates, each has a constant gate with the respective value $rank(a_i)$ – which is the value associated with $a_i$ by the ranking of the input structure – as their predecessor and as its other predecessor the root node of the circuit $C_n^{h_r(m_1,...,m_k)}$. The idea behind this construction is to have a subcircuit for each possible input $(a_1, ..., a_\ell)$ to $f$, which returns 0 if there is at least one $h_i$ which does not evaluate to $a_i$ and returns $f(a_1, ..., a_\ell)$ if all of them do. The results of all these subcircuits then get added together, since there is only exactly one, which may return a value other than 0, namely the one representing the given input to $f$.

3. If $h \triangleq f(h_1, ..., h_\ell)$ for a $\ell$-ary function symbol $f \in L_s$ and index terms $h_1, ..., h_\ell$, then $C_n^{h(m_1,...,m_k)}$ is defined as above but with the input gates describing the function values of $f$ instead of constant gates. We know where the correct input gate is, since we know the ordering and arities of the function symbols in the input structure.

Let $t$ be a number term:

1. If $t \triangleq c$ for $c \in \mathbb{R}$, then $C_n^{t(m_1,...,m_k)}$ consists of a constant gate with value $c$.
2. If $t \triangleq f(h_1, ..., h_\ell)$ for a $\ell$-ary function symbol $f \in L_a$ and index terms $h_1, ..., h_\ell$, then $C_n^{t(m_1,...,m_k)}$ is defined analogously to the second case of defining index terms.
3. If $t \triangleq f(h_1, ..., h_\ell)$ for a $\ell$-ary function symbol $f \in L_f$ and index terms $h_1, ..., h_\ell$, then $C_n^{t(m_1,...,m_k)}$ is defined as above but with the input gates describing $f$ instead of constant gates.
4. If $t \triangleq t_1 + t_2$ or $t \triangleq t_1 \times t_2$ for number terms $t_1, t_2$, then $C_n^{t(m_1,...,m_k)}$ consists of a $+$ or $\times$ gate at the top with the circuits $C_n^{t_1(m_1,...,m_k)}$ and $C_n^{t_2(m_1,...,m_k)}$ as its predecessors.

If $\varphi$ is a sentence, then this construction leads to a circuit deciding $S = \{\mathcal{D} \in Struct(\sigma) \mid \mathcal{D} \models \varphi\}$. Since this circuit's depth does not depend on $n$ and its size is polynomial in $n$, $S \in \mathrm{AC}^0_{\mathbb{R}}$.

$\mathrm{AC}^0_{\mathbb{R}} \subseteq \mathrm{FO}_{\mathbb{R}}[\mathrm{Arb}]$: To show that $\mathrm{AC}^0_{\mathbb{R}}$ is included in $\mathrm{FO}_{\mathbb{R}}[\mathrm{Arb}]$, we create, for any given $\mathrm{AC}^0_{\mathbb{R}}$ set $S$, an $\mathrm{FO}_{\mathbb{R}}[\mathrm{Arb}]$-sentence which defines $S$. In order to achieve this, we want to create a sentence, which talks about the structure of the circuits of the $\mathrm{AC}^0_{\mathbb{R}}$-circuit family which decides $S$. Since we have access to arbitrary functions, we can essentially just encode the structure of any given circuit into functions and have the interpretation of the function symbols we use be dependent on the length of the input $n$. However, the function symbols themselves, and thus the formula, do not depend on $n$. Since the depth of our circuits is constant and we can assume that they are full trees as shown in Lemma 19, we can construct a sentence which essentially describes the gates on each level of the circuit. Let $S \in \mathrm{AC}^0_{\mathbb{R}}$ via circuit family $\mathcal{C}$, $depth(C_n) = d$ and let $q$ be such that $size(C_n) \leq n^q$ for all $n \in \mathbb{N}$. Without loss of generality, let $C_n$ be a full tree as described in Lemma 19, i.e., for every gate $g$ in $C_n$ it holds that all paths from input gates to $g$ have the same length. We now create a $\mathrm{FO}_{\mathbb{R}}[\mathrm{Arb}]$-sentence $\varphi$ which defines the set decided by $\mathcal{C}$. The set $L_f$ of the signature of $\varphi$ will only contain one function symbol $f$, which then for every input gate $v$ in $C_n$ leads to $f(v)$ being interpreted as the value of $v$. Since $C_n$ is of size at most $n^q$, we can uniquely identify the gates of $C_n$ with elements of $A^q$. Let $v$ be a gate in $C_n$ encoded by $(v_1, ..., v_q)$. $t_n\colon A^q \to \mathbb{R}$, $c_n\colon A^q \to \mathbb{R}$, $in_n\colon A^{q+1} \to \mathbb{R}$ and $pred_n\colon A^{2q} \to \mathbb{R}$ are functions where $t_n(v_1, ..., v_q)$ is the type of $v$ as per Definition 7, $in_n(v_1, ..., v_q, i)$ is 1 if $v$ is the input gate $i$ of $C_n$ and 0 otherwise, $c_n(v_1, ..., v_q)$ is the value of gate $v$ if $v$ is a constant gate or 0, if it is not and $pred_n(v_1, ..., v_q, w_1, ..., w_q)$ is 1 if $v$ is a predecessor of the gate encoded by $(w_1, ..., w_q)$ and 0 otherwise. We will use $t$, $in$, $c$ and $pred$ as the respective symbols for these functions. Note that this means that the interpretation of these symbols depends on the input structure. We can now create a $q$-ary number term $val_x(v_1, ..., v_q)$ for every $x \leq d$, such that it holds that if $(v_1, ..., v_q)$ encodes a gate in $C_n$ on level $x$ (meaning that every path from an input gate to $v$ has length $x$) then for all inputs $(a_1, ..., a_n)$ to the circuit $C_n$, $val_x(v_1, ..., v_q)$ is the value of the gate encoded by $(v_1, ..., v_q)$ in $C_n$'s computation when given an $\mathbb{R}$-structure $\mathcal{D}$ where $\mathrm{enc}(\mathcal{D}) = (a_1, ..., a_n)$. We will define $val_x$ by induction on $x$. If $x = 0$ then $(v_1, ..., v_q)$ must encode an input gate. We therefore have:

$$val_0(v_1, ..., v_q) \triangleq sum_i(in(v_1, ..., v_q, i) \times f(i)) \tag{14}$$

For $1 \leq x \leq d$, define $val_x$ as follows:

$$
\begin{aligned}
val_x(v_1, ..., v_q) \triangleq\, & \chi[t(v_1, ..., v_q) = 2] \times T_{2,x}(v_1, ..., v_q) \\
& + \chi[t(v_1, ..., v_q) = 3] \times T_{3,x}(v_1, ..., v_q) \\
& + \chi[t(v_1, ..., v_q) = 4] \times T_{4,x}(v_1, ..., v_q) \\
& + \chi[t(v_1, ..., v_q) = 5] \times T_{5,x}(v_1, ..., v_q) \\
& + \chi[t(v_1, ..., v_q) = 6] \times T_{6,x}(v_1, ..., v_q) \qquad (15)
\end{aligned}
$$

where

$$T_{2,x}(v_1, ..., v_q) \triangleq c(v_1, ..., v_q) \qquad (16)$$

$$T_{3,x}(v_1, ..., v_q) \triangleq sum_i^q(pred(i_1, ..., i_q, v_1, ..., v_q) \times val_{x-1}(i_1, ..., i_q)) \qquad (17)$$

$$T_{4,x}(v_1, ..., v_q) \triangleq prod_i^q(pred(i_1, ..., i_q, v_1, ..., v_q) \times val_{x-1}(i_1, ..., i_q)) \qquad (18)$$

$$T_{5,x}(v_1, ..., v_q) \triangleq sum_i^q(pred(i_1, ..., i_q, v_1, ..., v_q) \times sign(val_{x-1}(i_1, ..., i_q))) \qquad (19)$$

$$T_{6,x}(v_1, ..., v_q) \triangleq sum_i^q(pred(i_1, ..., i_q, v_1, ..., v_q) \times val_{x-1}(i_1, ..., i_q)) \qquad (20)$$

We can now use $val_x$ to define a formula $\varphi$ over signature $\{\{\}, \{f\}, \{t, in, c, pred\}\}$ which defines the set decided by $C_n$ as follows: (Recall that $d$ denotes the depth of the circuits of the circuit family defining $S$.)

$$\varphi \triangleq \forall i_1 ... \forall i_q(\chi[t(i_1, ..., i_q) = 6 \implies val_d(i_1, ..., i_q) = 1]) \qquad (21)$$

The formula $\varphi$ is independent of the input length $n$, however the interpretations of its function symbols of $L_a$ are not. ◀

## 3.2 A Characterization for $\mathrm{U_P}$-$\mathrm{AC}^0_{\mathbb{R}}$

Having now developed a description for the non-uniform $\mathrm{AC}^0_{\mathbb{R}}$, in the upcoming part of this paper we derive descriptions for two of its uniform variations. We start by giving a description for the polynomial time uniform $\mathrm{U_P}$-$\mathrm{AC}^0_{\mathbb{R}}$. It turns out that we can define our previously introduced rules $\mathrm{SUM}_{\mathbb{R}}$ and $\mathrm{PROD}_{\mathbb{R}}$ by using a polynomial time extension to our first-order logic and therefore can use that logic to define the sets of $\mathrm{U_P}$-$\mathrm{AC}^0_{\mathbb{R}}$.

For this reason, we introduce another notation here:

▶ **Definition 21.** By $\mathrm{FTIME}_{\mathbb{R}}(f(n))$ we will denote all functions that for a finite set $S$ and $k \in \mathbb{N}$ map from $S^k$ to $\mathbb{R}$ or to $S$ and that are computable by a $\mathbb{R}$-machine in time bounded by $\mathcal{O}(f(|S|))$.

▶ **Theorem 22.** $\mathrm{FO}_{\mathbb{R}}[\mathrm{FTIME}_{\mathbb{R}}(n^{\mathcal{O}(1)})] = \mathrm{U_P}$-$\mathrm{AC}^0_{\mathbb{R}}$

**Proof.** $\mathrm{FO}_{\mathbb{R}}[\mathrm{FTIME}_{\mathbb{R}}(n^{\mathcal{O}(1)})] \subseteq \mathrm{U_P}$-$\mathrm{AC}^0_{\mathbb{R}}$: The construction of the circuit is analogous to the one in Theorem 20. We now need to demonstrate that the constructed circuit is P-uniform. This follows from the fact that the circuit's size is polynomial in the length of its input $n$ and that the construction of each gate takes at most polynomial time. In fact, the time it takes to construct the next gate when constructing the circuit in, for example, a depth-first manner is constant in all cases except for those, in which a function or a predicate of $L_a$ needs to be evaluated. In those cases, the required time is polynomial. That means that the entire circuit can be constructed in polynomial time. We will choose as the numbering of the circuit just the order, in which the gates are first constructed. Since we can compute $|A|$ from $n = |\mathrm{enc}(\mathcal{D})|$ in logarithmic time as described on page 7, it follows that there exists a

machine which on input $(n, v_{nr}, p_{idx})$ can compute $(t, p_{nr}, c)$ as described on page 6 in time bounded by a polynomial in $n$.
$\mathrm{U_P}$-$\mathrm{AC}^0_{\mathbb{R}} \subseteq \mathrm{FO}_{\mathbb{R}}[\mathrm{FTIME}_{\mathbb{R}}(n^{\mathcal{O}(1)})]$: For a given $\mathrm{U_P}$-$\mathrm{AC}^0_{\mathbb{R}}$ set $S$, we can also create a formula in the same way as in Theorem 20. We only need to show that we can define the number terms $t(v_1, ..., v_q)$, $c(v_1, ..., v_q)$, $in(v_1, ..., v_q, i)$, $pred(v_1, ..., v_q, w_1, ..., w_q)$, $sum_i(F(i_1, ..., i_q, \overline{w}))$ and $prod_i^q(F(i_1, ..., i_q, \overline{w}))$ in $\mathrm{FO}_{\mathbb{R}}[\mathrm{FTIME}_{\mathbb{R}}(n^{\mathcal{O}(1)})]$, since we can then just use the construction from Theorem 20. Let $A$ be the universe of the input structure.

1. Since the family defining $S$ is P-uniform, clearly $t(v_1, ..., v_q)$, $c(v_1, ..., v_q)$ and $in(v_1, ..., v_q, i)$ can be defined in $\mathrm{FO}_{\mathbb{R}}[\mathrm{FTIME}_{\mathbb{R}}(n^{\mathcal{O}(1)})]$.
2. $sum_i(F(i_1, ..., i_q, \overline{w}))$ can be defined in $\mathrm{FO}_{\mathbb{R}}[\mathrm{FTIME}_{\mathbb{R}}(n^{\mathcal{O}(1)})]$ as follows:
   a. If $F(i_1, ..., i_q, \overline{w}) \triangleq c$ for $c \in \mathbb{R}$, then $sum_i(F(i_1, ..., i_q)) = |A| * c$, which can be computed in a single step.
   b. If $F(i_1, ..., i_q, \overline{w}) \triangleq X(h_1(i_1, ..., i_q, \overline{w}), ..., h_k(i_1, ..., i_q, \overline{w}))$ for $X \in L_f$ and $h_1, ..., h_k$ index terms, then the evaluation of $X$ takes constant time since its interpretation is given in the input. The evaluation of each index term also takes at most polynomial time and $X$ needs to be evaluated no more than $|A|^q$ times. Afterwards, all the results need to be summed up, which can be done in polynomial time. Therefore the whole evaluation takes polynomial time.
   c. If $F(i_1, ..., i_q, \overline{w}) \triangleq X(h_1(i_1, ..., i_q, \overline{w}), ..., h_k(i_1, ..., i_q, \overline{w}))$ for $X \in L_a$ and $h_1, ..., h_k$ index terms, then the evaluation of $X$ takes polynomial time. But since this only introduces a polynomial addition to the calculation above, this evaluation can be done in polynomial time as well.
   d. If $F(i_1, ..., i_q, \overline{w}) \triangleq t_1(i_1, ..., i_q, \overline{w}) + t_2(i_1, ..., i_q, \overline{w})$ or $t_1(i_1, ..., i_q, \overline{w}) * t_2(i_1, ..., i_q, \overline{w})$ or $sign(t_1(i_1, ..., i_q, \overline{w}))$ for number terms $t_1, t_2$, then clearly the evaluation takes polynomial time.
3. $prod_i(F(i_1, ..., i_q, \overline{w}))$ can be defined analogously.
4. $pred(v_1, ..., v_q, w_1, ..., w_q)$ can be defined in $\mathrm{FO}_{\mathbb{R}}[\mathrm{FTIME}_{\mathbb{R}}(n^{\mathcal{O}(1)})]$ in the following way: We define the predicate

$$pred_k := \left\{ (v_1, ..., v_q, w_1, ..., w_q, k_1, ..., k_q) \;\middle|\; \begin{array}{l} v \text{ is the } k\text{th predecessor of } w \\ \text{where } v \text{ is the gate encoded by} \\ (v_1, ..., v_q),\ w \text{ is the gate encoded} \\ \text{by } (w_1, ..., w_q) \text{ and } k \text{ is the num-} \\ \text{ber encoded by } (k_1, ..., k_q). \end{array} \right\} \tag{22}$$

which we can evaluate in polynomial time, since $S$ is P-uniform. We can now define $pred(v_1, ..., v_q, w_1, ..., w_q)$ in $\mathrm{FO}_{\mathbb{R}}[\mathrm{FTIME}_{\mathbb{R}}(n^{\mathcal{O}(1)})]$ as follows:

$$pred(v_1, ..., v_q, w_1, ..., w_q) \triangleq \chi[\exists k_1, ..., \exists k_q pred_k(v_1, ..., v_q, w_1, ..., w_q, k_1, ..., k_q)] \tag{23}$$

Therefore we can define $S$ using a $\mathrm{FO}_{\mathbb{R}}[\mathrm{FTIME}_{\mathbb{R}}(n^{\mathcal{O}(1)})]$ sentence. ◀

## 3.3 A Characterization for $\mathrm{U_{LT}}$-$\mathrm{AC}^0_{\mathbb{R}}$

We have demonstrated that the same construction as in the proof of Theorem 20 can be applied in the P-uniform case if we restrict our logic to a polynomial extension rather than a universal one. For the second uniformity result, we will produce a description for $\mathrm{U_{LT}}$-$\mathrm{AC}^0_{\mathbb{R}}$ sets. The construction is again very similar to the one for the non-uniform case. This time, however, we need to explicitly extend our logic by the sum and product rule, since we could not define them in $\mathrm{FO}_{\mathbb{R}}[\mathrm{FTIME}_{\mathbb{R}}(\log n)]$ (and we believe that they cannot be defined therein).

Additionally, we take advantage of the tree-like nature of the circuits we constructed with our method so far and number their gates in a post-order fashion. This will be helpful for showing LT-uniformity, since it gives us the path from the output gate to any other gate and hence allows us to construct it without needing to construct the entire circuit.

▶ **Theorem 23.** $\mathrm{FO}_{\mathbb{R}}[\mathrm{FTIME}_{\mathbb{R}}(\log n)] + \mathrm{SUM}_{\mathbb{R}} + \mathrm{PROD}_{\mathbb{R}} = \mathrm{U}_{\mathrm{LT}}\text{-}\mathrm{AC}^0_{\mathbb{R}}$

**Proof.** $\mathrm{FO}_{\mathbb{R}}[\mathrm{FTIME}_{\mathbb{R}}(\log n)] + \mathrm{SUM}_{\mathbb{R}} + \mathrm{PROD}_{\mathbb{R}} \subseteq \mathrm{U}_{\mathrm{LT}}\text{-}\mathrm{AC}^0_{\mathbb{R}}$: Just as in the polynomial case, we will use the same construction as in Theorem 20 for the logarithmic case, adding only the construction for $sum_i$ and $prod_i$, which are quite natural operations for unbounded fan-in circuits. Showing the LT-uniformity of the resulting circuit, however, is not as simple as it was in Theorem 22, since we cannot just construct the entire circuit to retrieve the information for a singular gate. We can, however, construct only part of the circuit to arrive at the gate which we would like to retrieve in order to remain within logarithmic time. We will essentially construct the path from the output node to the node we are looking for, which has constant length. Let $S$ be the set of $\mathbb{R}$-structues defined by a given $\mathrm{FO}_{\mathbb{R}}[\mathrm{FTIME}_{\mathbb{R}}(\log n)] + \mathrm{SUM}_{\mathbb{R}} + \mathrm{PROD}_{\mathbb{R}}$-sentence $\varphi$. To create a circuit family deciding $S$, define the structure of our circuit depending on $\varphi$ similarly to the proof of Theorem 20. Here however, we will make sure that for each gate $v$, we know the *fullsize* of all of its direct subcircuits, i.e., the subcircuits induced by $v$'s predecessor gates, in order to make sure that we continue our construction at the right predecessor of $v$. In doing so, we can always compute the *fullsize*s of the predecessor subcircuits of any given node in time constant in the length of the input. We will additionally number our nodes in post-order, to ensure that we know where to continue constructing our circuit. The circuit is then constructed/structured as follows:

Since the input gates do not behave tree-like, we explicitly give the numbering they get, whenever it is needed: the $i$th input gate is numbered $\mathit{fullsize}(C_n) + i$. At the very top of the circuit, there is the output node numbered $\mathit{fullsize}(C_n)$. Its predecessor is then numbered $\mathit{fullsize}(C_n) - 1$ and structured as follows: Let the root gate of the subcircuit representing $\varphi$ be numbered $q$.

1. Let $\varphi \triangleq \exists y \psi(y)$. Then the construction is as in Theorem 20. The *sign* gate is numbered $q$, the addition gate is numbered $q-1$ and the root of the $i$th predecessor circuit $C_n^{\psi(m_1,\ldots,m_k,i)}$ is numbered $q - 2 - (u - i) * \mathit{fullsize}(C_n^{\psi(m_1,\ldots,m_k,1)})$. (We can use the *fullsize* of $C_n^{\psi(m_1,\ldots,m_k,1)}$ here for each $i$, because $C_n^{\psi(m_1,\ldots,m_k,i)}$ has the same *fullsize* for each $i$.)
2. If $\varphi \triangleq \forall y \psi(y)$, then the construction is as in Theorem 20 and the numbering is analogous to the existential case.
3. Let $\varphi \triangleq \neg\psi$. Then the construction is as in Theorem 20, the subtraction gate is numbered $q$, the constant gate with value 1 is numbered $q-1$ and the root of $C_n^{\psi(m_1,\ldots,m_k)}$ is numbered $q-2$.
4. Let $\varphi \triangleq \psi \wedge \xi$. Then the construction is as in Theorem 20, the sign node is numbered $q$, the $\times$ gate is numbered $q-1$, the root of $C_n^{\psi(m_1,\ldots,m_k)}$ is numbered $q-2-\mathit{fullsize}(C_n^{\xi(m_1,\ldots,m_k)})$ and the root of $C_n^{\xi(m_1,\ldots,m_k)}$ is numbered $q-2$.
5. If $\varphi \triangleq \psi \vee \xi$, $\varphi \triangleq \psi \implies \xi$ or $\varphi \triangleq \psi \iff \xi$, then $C_n^{\varphi(m_1,\ldots,m_k)}$ and its numbering follows analogously to $\varphi \triangleq \psi \wedge \xi$.
6. Let $\varphi \triangleq h_1 = h_2$ for index terms $h_1, h_2$. Then the construction is as in Theorem 20, the equality gate is numbered $q$, the root of $C_n^{h_1(m_1,\ldots,m_k)}$ is numbered $q - 1 - \mathit{fullsize}(C_n^{h_2(m_1,\ldots,m_k)})$ and the root of $C_n^{h_1(m_1,\ldots,m_k)}$ is numbered $q-1$.
7. If $\varphi \triangleq t_1 = t_2$ for number terms $t_1, t_2$, then $C_n^{\varphi(m_1,\ldots,m_k)}$ is defined and numbered analogously to the case with index terms.

8. Let $\varphi \triangleq t_1 < t_2$ for number terms $t_1, t_2$. Then $C_n^{\varphi(m_1,\dots,m_k)}$ is defined as in Theorem 20 and numbered analogously to the case of equality.

For the cases 6, 7 and 8, we also need to show how construction and numbering can be done for non-formula index and number terms. We will define these by induction as well. Let $h$ be an index term:

1. Let $h \triangleq x$ for $x \in V$. Then the construction is as in Theorem 20 (The constant gate is numbered $q$).
2. Let $h \triangleq f(h_1, \dots, h_\ell)$ for a $\ell$-ary function symbol $f \in L_a$ and index terms $h_1, \dots, h_\ell$. Then the construction is as in Theorem 20. The addition gate is numbered $q$, the subcircuits induced by the multiplication gates each have the same *fullsize*, namely $s_{h,\times} := \sum\limits_{1 \le i \le \ell} (\mathit{fullsize}(C_n^{(h_i(m_1,\dots,m_k))}) + 2) + 2$. Therefore the $i$th $\times$ gate is numbered $nr_{h,\times,i} := q - 1 - (n^\ell - i) * s_{h,\times}$. For the $i$th $\times$ gate the constant gate containing the function value (in the case of characteristic functions 1 or 0) is numbered $nr_{h,\times,i} - 1$. For the subcircuit induced by the respective $r$th equality gate, the numbering is as follows: the equality gate itself is numbered $nr_{h,=,i,r} := nr_{h,\times,i} - 2 - (\sum\limits_{r+1 \le j \le k} \mathit{fullsize}(C_n^{h_j(m_1,\dots,m_k)}) + 2)$. The constant gate containing the value $rank(a_r)$ is numbered $nr_{h,=,i,r} - 1 - \mathit{fullsize}(C_n^{h_r(m_1,\dots,m_k)})$ and the root of $C_n^{h_r(m_1,\dots,m_k)}$ is numbered $nr_{h,=,i,r} - 1$.
3. If $h \triangleq f(h_1, \dots, h_\ell)$ for a $\ell$-ary function symbol $f \in L_s$ and index terms $h_1, \dots, h_\ell$, then the construction is as in Theorem 20 and the numbering is done as above, but with the input gates numbered according to the rule at the top.

Let $t$ be a number term:

1. If $t \triangleq c$ for $c \in \mathbb{R}$, then the construction is as in Theorem 20.
2. If $t \triangleq f(h_1, \dots, h_\ell)$ for a $\ell$-ary function symbol $f \in L_a$ and index terms $h_1, \dots, h_\ell$, then the construction is as in Theorem 20 and the numbering is done as described in the case of index terms.
3. If $t \triangleq f(h_1, \dots, h_\ell)$ for a $\ell$-ary function symbol $f \in L_f$ and index terms $h_1, \dots, h_\ell$, then the construction is as in Theorem 20 and the numbering is done as described in the case of index terms.
4. If $t \triangleq t_1 + t_2$ or $t \triangleq t_1 \times t_2$ for number terms $t_1, t_2$, then the construction is as in Theorem 20. The addition gate is numbered $q$, the root of $C_n^{t_2 m_1,\dots,m_k}$ is numbered $q - 1$ and the root of $C_n^{t_1(m_1,\dots,m_k)}$ is numbered $q - 1 - \mathit{fullsize}(C_n^{t_2(m_1,\dots,m_k)})$.
5. If $t \triangleq \mathit{sum}_i(t_1(i))$ for a number term $t_1$ in which $i$ occurs freely, then $C_n^{t(m_1,\dots,m_k)}$ consists of an addition gate at the top, numbered $q$, with the root nodes of the circuits $C_n^{t_1(m_1,\dots,m_k,i)}, 1 \le i \le u$ as its $u$ predecessors, numbered $q - 1 - (u - i) * \mathit{fullsize}(C_n^{t_1(m_1,\dots,m_k,i)})$, similar to the case of existential quantification. If $i$ does not occur freely in $t_1$, then the predecessors of node $q$ are $u$ gates, of which each induces a copy of the circuit $C_n^{t_1(m_1,\dots,m_k)}$ and which are numbered the same way as for the case where $i$ is free in $t_1$.
6. If $t \triangleq \mathit{prod}_i(t_1(i))$ for a number term $t_1$, then the construction and numbering is done as above, just using a multiplication gate instead of an addition gate.

Note that this numbering gives each gate a distinct number and makes sure that for all non-input gates $v$ it holds that $v$'s number is higher than those of $v$'s predecessors. Additionally, it holds that for any two predecessors $v_1$ and $v_2$ of $v$, if $v_1$ is numbered lower than $v_2$, then all nodes in $v_1$'s induced subcircuit are also numbered lower than $v_2$ and vice versa. Since we can compute the *fullsize* of any subcircuit in constant time, we can also compute the number of the node where we need to continue in constant time. Note also that since the input gates do not behave tree-like there are holes in the numbering. Now we define an $\mathbb{R}$-machine

$M$ which on input $(n, v_{nr}, p_{idx})$ returns $(t, p_{nr}, c)$ as described on page 6. As described on page 7, we know that we can compute $|A|$ from $n = |\text{enc}(\mathcal{D})|$ in time logarithmic in $n$. To now produce the desired output, we take advantage of our node numbering. We know that our last node – the output node – has number $\mathit{fullsize}(C_n)$ and its singular predecessor node has number $\mathit{fullsize}(C_n) - 1$. Let $\mathit{fs}_\varphi$ denote $\mathit{fullsize}(C_n^{\varphi(m_1,\dots,m_k)})$ – which is the same as $\mathit{fullsize}(C_n^{\varphi(m_1,\dots,m_k,i)})$ etc., since the variable assignments do not have an effect on the size of the circuit – and $u$, as in the proof of Theorem 20, the size of the universe of the input structure $|A|$. Now the machine works as follows: If $v_{nr} > \mathit{fullsize}C_n + n$, then return $(0, 0, 0)$. If $v_{nr} = \mathit{fullsize}(C_n)$ then return $(6, \mathit{fullsize}(C_n) - 1, 0)$ if $p_{idx} = 1$ and $(6, 0, 0)$ otherwise. If $v_{nr} = \mathit{fullsize}(C_n) + i$, for $i \in \{1, \dots, n\}$ then return $(1, 0, i)$. Otherwise proceed as follows: Let $q$ be the number of the root of the current subcircuit. (We will use $q$ to describe both the value $q$ and the register in which we store that value.)

1. Let $\varphi \triangleq \exists y \psi(y)$. If $v_{nr} = q$, then return $(5, q - 1, 0)$ if $p_{idx} = 1$ and $(5, 0, 0)$ otherwise. If $v_{nr} = q - 1$, then return $(3, q - 2 - (u - p_{idx}) * \mathit{fs}_\psi, 0)$ if $p_{idx} \leq u$ and $(5, 0, 0)$ otherwise. Otherwise gate $v_{nr}$ is contained in the subcircuit induced by the gate numbered $y = q - 2 - (\left\lceil \frac{q-1-v_{nr}}{\mathit{fs}_\psi} \right\rceil - 1) * \mathit{fs}_\psi$ where $y$ is the smallest natural number such that $y \geq v_{nr}$ and $y = q - 2 - (u - i) * \mathit{fs}_\psi$ for some $i \in \{1, \dots, u\}$. We can compute $y$ in time logarithmic in $u$ by using binary search on $i$. We therefore store $y$ in $q$ and continue with the construction of the subcircuit induced by node $y$.
2. If $\varphi \triangleq \forall y \psi(y)$, then the construction is analogous to the existential case.
3. Let $\varphi \triangleq \neg\psi$. If $v_{nr} = q$, then return $(7, q - 1, 0)$ if $p_{idx} = 1$, $(7, q - 2, 0)$ if $p_{idx} = 2$ and $(7, 0, 0)$ otherwise. If $v_{nr} = q - 1$, then return $(2, 0, 1)$. Otherwise, store $q - 2$ in $q$ and continue with the construction of $C_n^{\psi(m_1,\dots,m_k)}$.
4. Let $\varphi \triangleq \psi \wedge \xi$. If $v_{nr} = q$ then return $(5, q - 1, 0)$ if $p_{idx} = 1$ and $(5, 0, 0)$ otherwise. If $v_{nr} = q - 1$ then return $(4, q - 2 - \mathit{fullsize}(C_n^{\xi(m_1,\dots,m_k)}), 0)$ if $p_{idx} = 1$, $(4, q - 2, 0)$ if $p_{idx} = 2$ and $(4, 0, 0)$ otherwise. Otherwise, if $v_{nr} \leq q - 2 - \mathit{fullsize}(C_n^{\xi(m_1,\dots,m_k)})$, store $q - 2 - \mathit{fullsize}(C_n^{\xi(m_1,\dots,m_k)})$ in $q$ and construct $C_n^{\psi(m_1,\dots,m_k)}$ and otherwise store $q - 2$ in $q$ and construct $C_n^{\psi(m_1,\dots,m_k)}$.
5. If $\varphi \triangleq \psi \vee \xi$, $\varphi \triangleq \psi \implies \xi$ or $\varphi \triangleq \psi \iff \xi$, then proceed analogously to $\varphi \triangleq \psi \wedge \xi$.
6. If $\varphi \triangleq h_1 = h_2$ for index terms $h_1, h_2$, then proceed analogously to the Boolean connectives.
7. If $\varphi \triangleq t_1 = t_2$ for number terms $t_1, t_2$, then proceed analogously to the Boolean connectives.
8. If $\varphi \triangleq t_1 < t_2$ for number terms $t_1, t_2$, then proceed analogously to the Boolean connectives.

For the cases 6, 7 and 8, we also need to explain how to construct the subcircuits for non-formula index and number terms. We will define these by induction as well. Let $h$ be an index term:

1. Let $h \triangleq x$ for $x \in V$. Then if the constant gate is numbered $v_{nr}$, $x$ must be $x_i$ for some $x_i \in V$, thus return $(2, 0, m_i)$. Otherwise return $(0, 0, 0)$.
2. Let $h \triangleq f(h_1, \dots, h_\ell)$ for a $\ell$-ary function symbol $f \in L_a$ and index terms $h_1, \dots, h_\ell$. Note: the numbers $n_{h,\times,i}$ and $nr_{h,=,i,r}$ are as they were defined in the structure of the circuit. If $v_{nr} = q$ then return $(5, q - 1, 0)$ if $p_{idx} = 1$ and $(5, 0, 0)$ otherwise. If $v_{nr} = q - 1$ then return $(3, nr_{h,\times,p_{idx}}, 0)$ if $p_{idx} \leq u^\ell$ and $(3, 0, 0)$ otherwise. Otherwise let $nr_\times \coloneqq \min\{nr_{h,\times,i} \mid nr_{h,\times,i} \geq v_{nr},\ 1 \leq i \leq u^\ell\}$. (The smallest number $nr_{h,\times,i}$ greater than or equal to $v_{nr}$) If $v_{nr} = nr_\times$ then return $(4, n_{h,=,i,p_{idx}}, 0)$ if $p_{idx} \leq \ell$, return $(4, nr_\times - 1, 0)$ if $p_{idx} = l + 1$ and return $(4, 0, 0)$, otherwise. ($i$ here is the corresponding $i$ from the definition of $nr_\times$.) If $v_{nr} = nr_\times - 1$ then return $(2, 0, c)$

where $c$ is the value of $f(a_1, ..., a_\ell)$ if $(a_1, ..., a_\ell)$ is the lexicographically $i$th input to $f$. Otherwise let $nr_= := \min\{nr_{h,=,i,r} \mid nr_{h,=,i,r} \geq v_{nr},\ 1 \leq i \leq u^\ell,\ 1 \leq r \leq \ell\}$. (The smallest number $nr_{h,=,i,r}$ greater than or equal to $v_{nr}$) If $v_{nr} = nr_=$ then return $(8, nr_= - 1 - \mathit{fullsize}(C_n^{h_i(m_1,...,m_k)}), 0)$ if $p_{idx} = 1$, $(8, nr_= - 1, 0)$ if $p_{idx} = 2$ and $(8, 0, 0)$ otherwise. If $v_{nr} = nr_= - 1 - \mathit{fullsize}(C_n^{h_i(m_1,...,m_k)})$ then return $(2, 0, a)$ where $a$ is the value of the $r$th element of the lexicographically $i$th input to $f$. Otherwise store $nr_= - 1$ in $q$ and continue with the construction of $C_n^{h_i(m_1,...,m_k)}$, where $i$ is the respective $i$ from the definition of $nr_=$.

3. If $h \triangleq f(h_1, ..., h_\ell)$ for a $\ell$-ary function symbol $f \in L_s$ and index terms $h_1, ..., h_\ell$, construct analogously to the case above, except for the gates for function values. Those would be input gates in this case and if $v_{nr}$ was the number of one of those, the machine would have returned already in the very beginning.

Let $t$ be a number term:

1. If $t \triangleq c$ for $c \in \mathbb{R}$. Then if the constant gate is numbered $v_{nr}$, return $(2, 0, c)$. Otherwise return $(0, 0, 0)$.
2. If $t \triangleq f(h_1, ..., h_\ell)$ for a $\ell$-ary function symbol $f \in L_a$ and index terms $h_1, ..., h_\ell$, then construct analogously to the case of index terms.
3. If $t \triangleq f(h_1, ..., h_\ell)$ for $\ell$-ary function symbol $f \in L_f$ and index terms $h_1, ..., h_\ell$, then construct analogously to the case of index terms.
4. If $t \triangleq t_1 + t_2$ or $t \triangleq t_1 \times t_2$ for number terms $t_1, t_2$, continue constructing as in the case of Boolean connectives.
5. Let $t \triangleq \mathit{sum}_i(t_1(i))$ for a number term $t_1$. If $v_{nr} = q$ then return $(3, q - 1 - (u - p_{idx}) * \mathit{fullsize}(C_n t_1(m_1, ..., m_k, p_{idx})), 0)$ if $p_{idx} \leq u$ and $(3, 0, 0)$ otherwise. Otherwise store $q - 1 - (u - p_{idx}) * \mathit{fullsize}(C_n t_1(m_1, ..., m_k, p_{idx}))$ in $q$ and continue with the construction of $C_n^{t_1(m_1,...,m_k,p_{idx})}$.
6. If $t \triangleq \mathit{prod}_i(t_1(i))$ for a number term $t_1$, then continue constructing as in the case of $\mathit{sum}_i$.

The way $M$ works, after decoding the input structure, it only ever needs to perform a constant number of operations on each level of the circuit, with the exception of the predicates and functions which are not given in the input structure. For those, $M$ needs logarithmic time. This means in total that since the circuit only has constant depth and hence a constant number of levels, $M$ works in logarithmic time. Therefore, $S$ is an element of $\mathrm{U_{LT}}\text{-}\mathrm{AC}^0_\mathbb{R}$.

$\mathrm{U_{LT}}\text{-}\mathrm{AC}^0_\mathbb{R} \subseteq \mathrm{FO}_\mathbb{R}[\mathrm{FTIME}_\mathbb{R}(\log n)] + \mathrm{SUM}_\mathbb{R} + \mathrm{PROD}_\mathbb{R}$: Showing that a set $S \in \mathrm{U_{LT}}\text{-}\mathrm{AC}^0_\mathbb{R}$ can be defined using $\mathrm{FO}_\mathbb{R}[\mathrm{FTIME}_\mathbb{R}(\log n)] + \mathrm{SUM}_\mathbb{R} + \mathrm{PROD}_\mathbb{R}$ is done in the same way as it was done in the polynomial case (Theorem 22). We construct the formula analogously and we can compute the functions we need for that construction in logarithmic time as follows:

1. We can compute $t(v_1, ..., v_q)$, $c(v_1, ..., v_q)$, $\mathit{in}(v_1, ..., v_q, i)$ and $\mathit{pred}(v_1, ..., v_q)$ in logarithmic time analogous to Theorem 22, since our circuit family is LT-uniform.
2. $\mathit{sum}_i$ and $\mathit{prod}_i$ are given in the specification of $\mathrm{FO}_\mathbb{R}[\mathrm{FTIME}_\mathbb{R}(\log n)] + \mathrm{SUM}_\mathbb{R} + \mathrm{PROD}_\mathbb{R}$.

◀

With the construction shown in the previous theorem we can now generalize that, whenever we have a variant of $\mathrm{AC}^0_\mathbb{R}$ given by a time complexity uniformity criterion that is at least logarithmic, we can describe it using first-order logic extended with functions of that class' time complexity and the sum and product rule. This result is formalized as follows:

► **Corollary 24.** *For any function $f : \mathbb{N} \to \mathbb{N}$ with $f(n) \geq \log n$ for all $n$, it holds that*

$$\mathrm{U}_f\text{-}\mathrm{AC}^0_\mathbb{R} = \mathrm{FO}_\mathbb{R}[\mathrm{FTIME}_\mathbb{R}(f(n))] + \mathrm{SUM}_\mathbb{R} + \mathrm{PROD}_\mathbb{R}, \quad (24)$$

*where $\mathrm{U}_f\text{-}\mathrm{AC}^0_\mathbb{R}$ is the class of sets decidable by circuit families, which can be constructed as described on page 6 in time bounded by $\mathcal{O}(f(n))$.*

▶ Remark 25. The logarithmic bound for $f$ in Corollary 24 stems from the time it takes to decode an encoded $\mathbb{R}$-structure as mentioned on page 7.

▶ Remark 26. Even though we have only considered functional $\mathbb{R}$-structures in this paper, our findings can be generalized for $\mathbb{R}$-structures which use relations as well, since any relation can be expressed via its characteristic function.

## 4 Conclusion

We showed that the computational power of circuits of polynomial size and constant depth over the reals can be characterized in a logical way by first-order logic on metafinite structures. This result is in analogy to corresponding characterizations for Boolean circuits [11] and arithmetic circuits [9]. In the Boolean and arithmetic context, it is shown that the numerical predicates of addition and multiplication play a special role: If we enhance first-order logic by these, we obtain a logic as powerful as LT-uniform $\mathrm{AC}^0$-circuits, i.e., $\mathrm{U_{LT}}\text{-}\mathrm{AC}^0 = \mathrm{FO}[+, \times]$. This does not seem to hold in our case of computation over the real numbers: $\mathrm{U_{LT}}\text{-}\mathrm{AC}^0_{\mathbb{R}}$ looks more powerful than $\mathrm{FO}_{\mathbb{R}}[+, \times]$, since real numbers can be manipulated more general by $\mathbb{R}$-machines operating in logarithmic time than in first-order formulas. Maybe an equivalence can be obtained with a more powerful logic for real numbers, but this is a question for further research.

We consider it worthwhile to study logical characterizations of analogues of further circuit classes of unbounded or semi-unbounded fan-in, most prominently $\mathrm{SAC}^1_{\mathbb{R}}$ and $\mathrm{AC}^1_{\mathbb{R}}$. In the theory of arithmetic complexity, i.e., computation over arbitrary semi-rings, first an analogue of Immerman's Theorem was shown in [9], and this was later used to obtain logical characterizations of the larger arithmetic classes $\mathrm{NC}^1$, $\mathrm{SAC}^1$ and $\mathrm{AC}^1$ [5]. Remarkably these characterizations did not build on logics with repeated quantifier blocks (like in [11]) or restricted fixed-point logic (like in [4]). Instead, new logical characterizations of the Boolean classes $\mathrm{NC}^1$, $\mathrm{SAC}^1$ and $\mathrm{AC}^1$ were given, somewhat similar to earlier ideas from Compton and Laflamme [2], and these were then shifted to the arithmetic setting. Maybe this can also be useful in our context to develop characterizations for $\mathrm{SAC}^1_{\mathbb{R}}$ and $\mathrm{AC}^1_{\mathbb{R}}$ (and maybe obtain a new characterization of $\mathrm{AC}^1_{\mathbb{R}}$).

In the theory of computation over the reals, separations among classes are known which are widely open in the discrete world; we only mention the separation of $\mathrm{NC}_{\mathbb{R}}$ and $\mathrm{P}_{\mathbb{R}}$ [3]. In the circuit world, the most prominent open question is if $\mathrm{TC}^0 = \mathrm{NC}^1$ (see the discussion in [12]). In our context, it is intriguing to study the landscape between $\mathrm{AC}^0_{\mathbb{R}}$ and $\mathrm{NC}^1_{\mathbb{R}}$. Is there any meaningful way to add computational power to $\mathrm{AC}^0_{\mathbb{R}}$ without already arriving at the full power of $\mathrm{NC}^1_{\mathbb{R}}$? Observe that up to date, no reasonable real analogue of the class $\mathrm{TC}^0$ is known. In Boolean complexity, $\mathrm{TC}^0$ is obtained by enriching $\mathrm{AC}^0$-circuits with majority gates. Here, the class $\mathrm{AC}^0_{\mathbb{R}}$ is closed under all reasonable forms of majority and threshold operations. A first step forward will be to separate $\mathrm{AC}^0_{\mathbb{R}}$ and $\mathrm{NC}^1_{\mathbb{R}}$, a real world analogue of a classical circuit separation from the eighties [7].

## References


1 Lenore Blum, Mike Shub, and Steve Smale. On a theory of computation over the real numbers; NP completeness, recursive functions and universal machines (extended abstract). In *29th Annual Symposium on Foundations of Computer Science*, pages 387–397. IEEE Computer Society, 1988. doi:10.1109/SFCS.1988.21955.

2 Kevin J. Compton and Claude Laflamme. An algebra and a logic for $\mathrm{NC}^1$. *Inf. Comput.*, 87(1/2):240–262, 1990. doi:10.1016/0890-5401(90)90063-N.

3 Felipe Cucker. $P_{\mathbb{R}}$ != $NC_{\mathbb{R}}$. *J. Complexity*, 8(3):230–238, 1992. doi:10.1016/0885-064X(92)90024-6.
4 Felipe Cucker and Klaus Meer. Logics which capture complexity classes over the reals. *J. Symb. Log.*, 64(1):363–390, 1999. doi:10.2307/2586770.
5 Arnaud Durand, Anselm Haak, and Heribert Vollmer. Model-theoretic characterization of Boolean and arithmetic circuit classes of small depth. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 354–363. ACM, 2018. doi:10.1145/3209108.3209179.
6 Ronald Fagin. Generalized first-order spectra and polynomial time recognizable sets. In R. Karp, editor, *Complexity of Computations*, volume 7 of *SIAM–AMS Proceedings*, pages 43–73. American Mathematical Society, Providence, RI, 1974.
7 Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984. doi:10.1007/BF01744431.
8 Erich Grädel and Klaus Meer. Descriptive complexity theory over the real numbers. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*, pages 315–324, 1995. doi:10.1145/225058.225151.
9 Anselm Haak and Heribert Vollmer. A model-theoretic characterization of constant-depth arithmetic circuits. *Ann. Pure Appl. Log.*, 170(9):1008–1029, 2019. doi:10.1016/j.apal.2019.04.006.
10 Neil Immerman. Languages that capture complexity classes. *SIAM Journal of Computing*, 16:760–778, 1987. doi:10.1137/0216051.
11 Neil Immerman. Expressibility and parallel complexity. *SIAM J. Comput.*, 18(3):625–638, 1989. doi:10.1137/0218043.
12 Heribert Vollmer. *Introduction to Circuit Complexity - A Uniform Approach*. Springer, 1999. doi:10.1007/978-3-662-03927-4.