Department of Theoretical Computer Science

Leibniz University Hanover

Master Thesis

Parallel Computation with Real Numbers

Timon Barlag
Matriculation Number: 3077970

October 20, 2019

First Examiner: Prof. Dr. Heribert Vollmer
Second Examiner: Dr. Arne Meier

Contents

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst habe und keine anderen Hilfsmittel und Quellen als angegeben verwendet habe.

____

Timon Barlag

1 Introduction

1.1 General 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.

In a day and age where parallel computation is more commonplace than ever, the limitations of this model are all too real. Programs that run on individual chips, in large GPU-clusters or even in clouds require different models, to be studied. For the examination of problems based on their parallel complexity, several different models of computations have been introduced and studied. The model we are going to have a closer look at in this thesis is the arithmetic circuit [Vol99], which, as its name suggests, resembles electrical circuits in its functioning.

Parallel computation is, however, not the only step we are taking away from the Turing machine. Many real-world problems are in fact real-valued, meaning that they cannot be solved by using only discrete computation. Of course, our actual computers cannot perform computations on a non-discrete set either, however, to study these problems it is useful to first assume that exact computations on the reals are available and then approximate them using discrete – and available – techniques. Computation models exist both for real-valued sequential and for real-valued parallel computation [BSS88].

Cucker and Meer showed a few logical characterizations for bounded fan-in real arithmetic circuit classes [CM99], which is what this thesis builds on. We now expand on that research by developing characterizations for unbounded fan-in classes; we are particularly concerned with a real analogue to the class AC^0 – a class which contains problems that can be solved in constant time with a reasonable amount of hardware. That class is also interesting, because it coincides with the class of sets definable by first-order logic in the classical case. We present logical characterizations for $AC^0_{\mathbb{R}}$ and two of its uniform subclasses and we provide a generalization for certain uniform subclasses of $AC^0_{\mathbb{R}}$.

This thesis is structured as follows: In Chapter 2 we start by giving an introduction to computation over the reals by presenting machine models and some basic definitions. We then proceed by defining first-order logic over the real numbers and giving some useful extensions thereof. The main results of this thesis, namely logical characterizations for non-uniform $AC^0_{\mathbb{R}}$ and two of its uniform subclasses are described in Chapter 3. Finally, conclusions are drawn and an outlook on potential future research is provided in Chapter 4.

1.2 Related Work

The theory of computation that we deal with in this thesis was first introduced by Blum, Shub and Smale, to examine computation and complexity over the real numbers or any ordered ring [BSS88]. An interesting property of this theory is that, as shown by Michaux, a real analogue to the classical PSPACE would contain every recursive set over the reals [Mic89], which then makes space complexity much less of an interesting property when classifying problems. Another interesting result about real computation is the existence of inherently sequential problems, which has not yet been shown in the classical case [Cuc92].

Grädel and Meer introduced first-order logic over the reals and used it to provide some results in descriptive complexity [GM95], on which Cucker and Meer expanded by giving characterizations for further real complexity classes, including circuit complexity classes and a real version of NP [CM99]. Meer also investigated counting problems over the reals and gave a characterization for a real version of #P [Mee00].

For more information on real-valued machine models, see [BCSS98] and for more information on arithmetic circuits in the classical case, see [Vol99].

2 Preliminaries

In the upcoming section, we give an introduction to the machine models and logic over $\mathbb{R}$ used in this thesis – which are mostly taken from [CM99] – and some extensions thereof which we will make use of later on.

2.1 Machines over $\mathbb{R}$

Even though this thesis mostly concerns itself with parallel computation, we first introduce a computation model for serial computation over $\mathbb{R}$. We do this mainly to later define uniformity criteria but also to provide a model for real computation akin to the discrete Turing Machine. The model we use was first introduced by Blum et al. [BSS89] and later modified by Cucker and Meer [CM99]. It is reminiscent of discrete register machines such as the Uniform Register Machine (URM) (see, for example [Cut80]).

We start by defining a few notations and operations which we need when talking about the aforementioned machine model.

Definition 1 (§2 [CM99])**.** In the following, arbitrarily long $\mathbb{R}$-vectors will be denoted by $\mathbb{R}^\infty$, i.e.

$$\mathbb{R}^\infty = \bigcup_{n \in \mathbb{N}} \mathbb{R}^n \tag{2.1}$$

For any $x \in \mathbb{R}^\infty$, $|x|$ denotes the length of the vector x, i.e. the only n s.t. $x \in \mathbb{R}^n$.

We denote by $\mathbb{R}_\infty$ the bi-infinite direct sum which contains elements of 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* $\sigma_\ell : \mathbb{R}_\infty \to \mathbb{R}_\infty$ and *shift right* $\sigma_r : \mathbb{R}_\infty \to \mathbb{R}_\infty$ which shift the indices of the elements, e.g.

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

Since we would like to talk about finite computations on $\mathbb{R}_\infty$, we introduce the following, rather specific definition, which we will make use of when defining our sequential machine over $\mathbb{R}$.

Definition 2. When we say that a function $f : \mathbb{R}_\infty \to \mathbb{R}_\infty$ is a *finite* $\mathbb{R}_\infty$*-map* , we mean that there exist indices $\ell, r \in \mathbb{N}$, where $\ell < r$ and there are functions $f_\ell, ..., f_r : \mathbb{R}^{r-\ell} \to \mathbb{R}$, such that

$$\begin{aligned} &f(..., x_{\ell-1}, x_\ell, x_{\ell+1}, ..., x_{r-1}, x_r, x_{r+1}, ...) = \\ &\qquad (...x_{\ell-1}, f_\ell(x_\ell, x_{\ell+1}, ..., x_{r-1}, x_r), ..., f_r(x_\ell, x_{\ell+1}, ..., x_{r-1}, x_r), x_{r+1}, ...). \end{aligned} \tag{2.2}$$

If for all i, f_i is of the form

$$f_i(x_1, ..., x_{r-\ell}) = p(x_1, ..., x_{r-\ell}), \tag{2.3}$$

for a real-valued polynomial p we say that f is a *finite polynomial* $\mathbb{R}_\infty$*-map*, and if the f_i are of the form

$$f_i(x_1, ..., x_{r-\ell}) = \frac{p_{i1}(x_1, ..., x_{r-\ell})}{p_{i2}(x_1, ..., x_{r-\ell})}, \tag{2.4}$$

with real-valued polynomials p_{i1} and p_{i2}, then we say that f is a *finite rational* $\mathbb{R}_\infty$*-map.*

Definition 3 (Definition 1 [CM99])**.** A *machine over* $\mathbb{R}$ consists of an input space $\mathcal{I} = \mathbb{R}^\infty$, an output space $\mathcal{O} = \mathbb{R}^k$ $(k \leq \infty)$, a state space $\mathcal{S} = \mathbb{R}_\infty$ and a finite connected directed graph with nodes that are labeled $1...N$ and each of which has one of the following types:

Input nodes	There is only one input node, this node is labeled with 1. Associated with this node is a next node $\beta(1)$ and the input map $g_I : \mathbb{R}^\infty \to \mathbb{R}_\infty$.
Output nodes	There is only one output node which is labeled with N. It has no next nodes; once it is reached the computation halts and the output map $g_O : \mathbb{R}_\infty \to \mathbb{R}^k$ places the result of the computation in the output space.
Computation nodes	A computation node m is associated with a next node $\beta(m)$ and a map $g_m : \mathbb{R}_\infty \to \mathbb{R}_\infty$, where g_m is a finite polynomial or rational $\mathbb{R}_\infty$-map.
Branch nodes	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.
Shift nodes	A shift node m is associated with a next node $\beta(m)$ and a map $\sigma : \mathbb{R}_\infty \to \mathbb{R}_\infty$, where σ is a left or 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 the input to the output node $in_O \in \mathbb{R}_\infty$. 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 [CM99])**.** 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 $f : \mathbb{R}^\infty \to \mathbb{R}^k$, $k \leq \infty$ 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 thesis 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 an element of $\mathcal{O}(n^{\mathcal{O}(1)})$. Analogously, we say that M works in *logarithmic time* if it works in time bounded by an element of $\mathcal{O}(\log(n))$.

2.2 Arithmetic Circuits over $\mathbb{R}$

In the following we introduce the main model of computation used in this thesis, namely arithmetic circuits over the real numbers as introduced by Cucker [Cuc92]. However, we use a slightly altered version which allows for gates with unbounded fan-in in order to later consider complexity classes for such circuits. As a consequence, we only consider the arithmetic operations $+$ and $\times$, which then allow us to immediately construct subtraction but not division. This is in line with the classical model as given by Vollmer for circuits with unbounded fan-in [Vol99].

We give definitions about the model of computation and about some resulting complexity classes. We start out by defining the *sign* function and one variation thereof which we need during the course of this thesis starting with our introduction of $\mathbb{R}$-circuits.

Definition 6. We define the *sign* function as follows:

$$sign(x) := \begin{cases} 1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0 \end{cases} \tag{2.5}$$

We will also introduce one variation of the sign function here for convenience:

$$sign'(x) := \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{2.6}$$

Remark 7. Since we can easily construct *sign'* from *sign*, as $sign'(x) = sign(sign(x)+1)$, we will use *sign'* freely whenever we have *sign* available.

Definition 8. An *arithmetic circuit* $\mathcal{C}$ over $\mathbb{R}$ is a directed acyclic graph. Its nodes (also called gates) can be of the following types:

Input nodes	Input nodes have indegree 0 and contain the respective input values of the circuit.
Constant nodes	Constant nodes have indegree 0 and are labeled with real numbers.
Arithmetic nodes	Arithmetic nodes can have an arbitrary indegree only bounded by the number of nodes in the circuit. They can be labeled with either $+$ or $\times$.
Sign nodes	Sign nodes have indegree 1.
Output nodes	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	type
input	1
constant	2
$+$	3
$\times$	4
sign	5
output	6

In order to talk about complexity classes of computation models, it is necessary to contemplate which resources one wishes to consider when classifying problems.

For arithmetic circuits, analyzing the longest possible path from an input to an output node can be regarded as an analogue to time complexity of sequential models.

Similarly, the number of gates used by the circuit can be related to space complexity in the sequential setting.

We define those two measures as follows:

Definition 9. For an arithmetic circuit $\mathcal{C}$, the *size* of $\mathcal{C}$ is the number of gates in $\mathcal{C}$ and the *depth* of $\mathcal{C}$ is the length of the longest path from an input gate to an output gate.

For convenience, we now define some auxiliary gate types, which will make some constructions in the later part of this thesis easier. As we will late see, we can construct those gate types with the types we have already available and therefore do not gain any computational power by using them.

Definition 10. In addition to the gate types $1-6$, we also define arithmetic nodes labeled with $-$ or with the relation symbols $=$, $<$, $>$, $\leq$ and $\leq$. All of those nodes have indegree 2.

We denote the types of these gates as follows:

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

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

Definition 11. Let $\mathcal{C}$ be an arithmetic circuit with n input gates and m output gates. We denote $x_1, ..., x_n$ by $\overline{x}$ and inductively associate each gate g with a function $f_g : \mathbb{R}^n \to \mathbb{R}$ as follows:

If g is the ith input gate, then $f_g(\overline{x}) = x_i$.
If g is a constant gate labeled $\alpha \in \mathbb{R}$, then $f_g(\overline{x}) = \alpha$.
If g is an arithmetic $+$ gate with predecessors $g_1, ..., g_k$, then $f_g(\overline{x}) = \sum\limits_{1\leq i\leq k} f_{g_i}(\overline{x})$.
If g is an arithmetic $\times$ gate with predecessors $g_1, ..., g_k$, then $f_g(\overline{x}) = \prod\limits_{1\leq i\leq k} f_{g_i}(\overline{x})$.
If g is a *sign* gate with predecessor g_1, then $f_g(\overline{x}) = sign(f_{g_1}(\overline{x}))$.
If g is an output gate with predecessor g_1, then $f_g(\overline{x}) = f_{g_1}(\overline{x})$.
If g is an arithmetic $-$ gate with predecessors g_1, g_2, then $f_g(\overline{x}) = f_{g_1}(\overline{x}) - f_{g_2}(\overline{x})$.

If g is a relational $=$ gate with predecessors g_1, g_2, then $f_g(\overline{x}) = \begin{cases} 1, & f_{g_1}(\overline{x}) = f_{g_2}(\overline{x}) \\ 0, & f_{g_1}(\overline{x}) \neq f_{g_2}(\overline{x}) \end{cases}$.

If g is a relational $<$ gate with predecessors g_1, g_2, then $f_g(\overline{x}) = \begin{cases} 1, & f_{g_1}(\overline{x}) < f_{g_2}(\overline{x}) \\ 0, & f_{g_1}(\overline{x}) \not< f_{g_2}(\overline{x}) \end{cases}$.

If g is a relational $>$ gate with predecessors g_1, g_2, then $f_g(\overline{x}) = \begin{cases} 1, & f_{g_1}(\overline{x}) > f_{g_2}(\overline{x}) \\ 0, & f_{g_1}(\overline{x}) \not> f_{g_2}(\overline{x}) \end{cases}$.

If g is a relational $\leq$ gate with predecessors g_1, g_2, then $f_g(\overline{x}) = \begin{cases} 1, & f_{g_1}(\overline{x}) \leq f_{g_2}(\overline{x}) \\ 0, & f_{g_1}(\overline{x}) \not\leq f_{g_2}(\overline{x}) \end{cases}$.

If g is a relational $\geq$ gate with predecessors g_1, g_2, then $f_g(\overline{x}) = \begin{cases} 1, & f_{g_1}(\overline{x}) \geq f_{g_2}(\overline{x}) \\ 0, & f_{g_1}(\overline{x}) \not\geq f_{g_2}(\overline{x}) \end{cases}$.

We refer to the function $\varphi_{\mathcal{C}} : \mathbb{R}^n \to \mathbb{R}^m$ associated with $\mathcal{C}$'s output gates as the function *computed by* $\mathcal{C}$.

Lemma 12. *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

$$t_1 \leq t_2 \equiv t_1 < t_2 \vee t_1 = t_2 \tag{2.7}$$

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 as shown in Figure 2.1. 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

$$t_1 = t_2 \equiv sign'(-(t_1 - t_2)^2) \tag{2.8}$$

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 $-$ 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 $-$ gate as their predecessor. That $-$ gate then has p_1 and p_2 as its predecessors. This construction is visualized in Figure 2.2. 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

$$t_1 < t_2 \equiv 1 - sign'(t_1 - t_2) \tag{2.9}$$

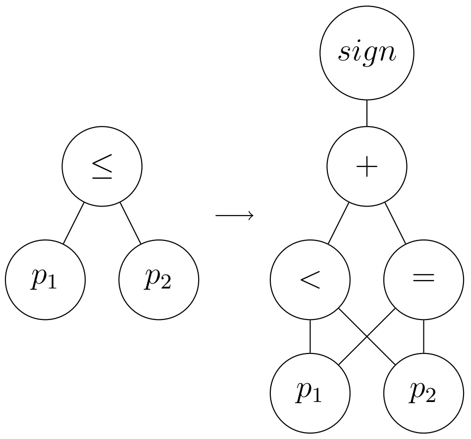

Figure 2.1: Construction for $\leq$ gates in Lemma 12

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. This construction is shown in Figure 2.3. 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

$$t_1 - t_2 \equiv t_1 + (-1) \times t_2 \tag{2.10}$$

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 as shown in Figure 2.4. For each gate, this introduces an increase in size of 2 per $-$ 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.

$\square$

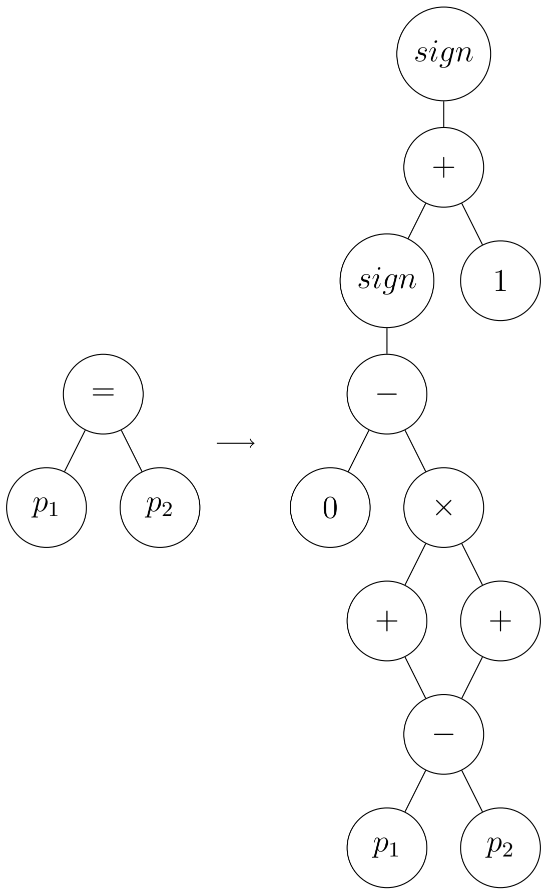

Figure 2.2: Construction for = gates in Lemma 12

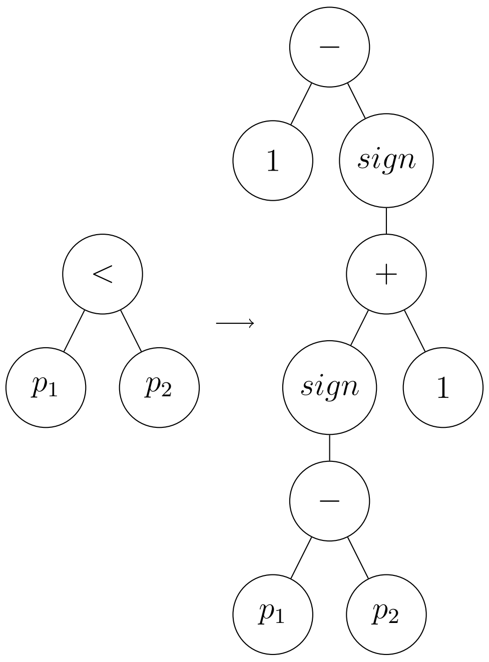

Figure 2.3: Construction for $<$ gates in Lemma 12

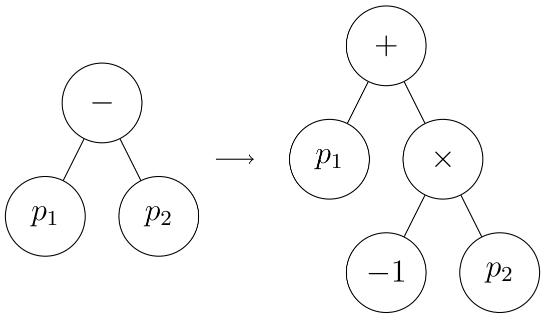

Figure 2.4: Construction for $-$ gates in Lemma 12

Remark 13. Since these auxiliary gate types do not give us any additional computational power, we will assume that whenever we are given a circuit, it only consists of gates of the types $1-6$. We will, however, make use of the gate types $7-12$, when we construct circuits ourselves.

An individual circuit can only compute a function with a fixed number of arguments. In some cases, however, we are interested in functions, which can handle arbitrarily long inputs. The following definition allows us to consider functions of that sort.

Definition 14. A *circuit family* over $\mathbb{R}$ is a sequence $\mathcal{C} = (\mathcal{C}_0, \mathcal{C}_1, \mathcal{C}_2, ...)$ where $\mathcal{C}_n$ is a circuit over $\mathbb{R}$ with n inputs for every $n \in \mathbb{N}$. Let f^n be the function computed by $\mathcal{C}_n$. We then say that $\mathcal{C}$ computes the function $f_{\mathcal{C}} : \mathbb{R}^\infty \to \mathbb{R}^\infty$ where $f_{\mathcal{C}}(w) = f^{|w|}(w)$ for $w \in \mathbb{R}^\infty$.

To denote $\mathcal{C}$ and $f_{\mathcal{C}}$, we also write $(\mathcal{C}_n)_{n\in\mathbb{N}}$ and $(f_n)_{n\in\mathbb{N}}$.

Definition 15. We say that a family of circuits $\mathcal{C}$ *decides* a set $S \subseteq \mathbb{R}^\infty$, if and only if $\mathcal{C}$ computes the characteristic function of S.

Definition 16. For a circuit C, we say that C_{sub} is a *subcircuit* of C, if all nodes and edges in C_{sub} are also contained in C and if for all nodes g in C_{sub} it holds that if there is a path from an input gate to 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 state that g is the *root node* of $C_{sub,g}$. It follows that if we are talking about the root node of a circuit with a single output node, then this output node is what we refer to.

Since arithmetic circuits can only take inputs of fixed length, we call them a non-uniform model of computation. Given that this is quite a restriction, we would also like to talk about a uniform version of this model. To that end, we consider the complexity of computing the correct circuit for any given input. That means that if we have a circuit family $\mathcal{C}$ and are given an input $\overline{x} = (x_1, ..., x_n)$, we would like to classify problems by how hard it is to construct $\mathcal{C}_n$.

The following definition allows us to talk about complexity classes for uniform circuits.

Definition 17. We say that a family of circuits $(\mathcal{C}_n)_{n\in\mathbb{N}}$ is uniform, if its gates are numbered, i.e. for each $n \in \mathbb{N}$, there is an injective function mapping each gate of $\mathcal{C}_n$ to a natural number, and there is an $\mathbb{R}$-machine M, which on input (n, v_{nr}, p_{idx}) returns the tuple

$$(t, p_{nr}, c)$$

where

1. t is the type of the v_{nr}th gate v in $\mathcal{C}_n$,
2. p_{nr} is the number of the p_{idx}th predecessor of v
3. and c is the value of v if v is a constant gate, the index i, if v is the ith input gate and 0 otherwise.

If v_{nr} does not encode a gate in $\mathcal{C}_n$, M returns $(0, 0, 0)$ and if v has less than p_{idx} predecessors M returns $(t, 0, 0)$. If M works in logarithmic time, we say that $\mathcal{C}$ is L-uniform and if M works in polynomial time, we say that $\mathcal{C}$ is P-uniform.

For a circuit complexity class C, we will by U_L-C denote the subclass of C, which only contains sets definable by L-uniform circuit families. We will use U_P-C to analogously denote those sets in C definable by P-uniform families.

Remark 18. Note that ordinarily L-uniformity denotes logspace uniformity rather than logtime uniformity. In the context of $\mathbb{R}$-machines, however, considering space complexity does not make as much sense as in the context of classical Turing machines [Mic89] [Cuc92], which is why we this notation is used for logtime uniformity in the real setting.

Definition 19. The complexity classes $AC^i_{\mathbb{R}}$ are defined as those sets $S \subseteq \mathbb{R}^\infty$ that can be decided by an arithmetic circuit family $(\mathcal{C}_n)_{n\in\mathbb{N}}$ over $\mathbb{R}$, whose size is polynomial in n, whose depth is element of $\mathcal{O}(\log^i n)$ and whose gates have unbounded fan-in.

Definition 20. The complexity classes $NC^i_{\mathbb{R}}$ are defined similarly to $AC^i_{\mathbb{R}}$, but with the additional restriction that their circuit families have gates of at most fan-in 2.

Remark 21. Note that the notation $NC^i_{\mathbb{R}}$ is also used by Cucker and Meer [CM99], however there it is used to denote something slightly different. Their definition is closer to how we would define U_L-$NC^i_{\mathbb{R}}$, albeit the circuits of our classes do not have immediate access to subtraction or division gates.

2.3 $\mathbb{R}$-structures and Logics over $\mathbb{R}$

In order to classify some of the aforementioned complexity classes, we introduce first-order logic over the reals, which was first introduced by Grädel and Meer [GM95]. However, we define it more similarly to how it was defined by Cucker and Meer [CM99] and Blum et al. [BCSS98] and give a few extensions.

We start by defining structures over $\mathbb{R}$. In order to do this, we use two vocabularies; one for talking about the finite, discrete part of the problem which we call the *skeleton* and one for talking about the *arithmetic* part of the problem.

If we were for example to talk about our arithmetic circuits, the structure of the gates and connections of the circuits could be described by the skeleton and the real-valued computations could be described by the arithmetic part.

Definition 22 (Definition 7 [CM99])**.** 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 $\mathcal{D}$. $\mathcal{F}$ is a finite set of functions $X : A^k \to \mathbb{R}$ which interpret the function symbols in L_f.

We will use $\mathit{Struct}_{\mathbb{R}}(\sigma)$ to refer to the set of all $\mathbb{R}$-structures of signature σ 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.

Remark 23. Whenever it is clear from the context, we will use L_s and L_f to either refer to the sets of function and predicate symbols as they are defined or – in the context of a structure – the respective functions and predicates interpreting those symbols.

Definition 24 (Definition 8 [CM99])**.** We will use $|A|$ to denote the cardinality of the universe A of $\mathcal{A}$.

A *ranking* of a $\mathbb{R}$-structure $\mathcal{D} = (\mathcal{A}, \mathcal{F})$ over signature $\sigma = (L_s, L_f)$ is a function r which bijects A with $\{0, 1, ..., |A| - 1\}$.

In order to now be able to use our $\mathbb{R}$-structures as inputs for machines, we need to somehow encode them in $\mathbb{R}^\infty$. The following definition allows us to do just that.

Definition 25. Every $\mathbb{R}$-structure $\mathcal{D} = (\mathcal{A}, \mathcal{F})$ can be identified with a point in $\mathbb{R}^\infty$. To this end, choose an arbitrary ranking r on A. Then replace all predicates in L_s by their respective characteristic functions and all functions $f \in L_s$ by functions f' which for any input $(a_1, ..., a_k)$ map to $r(f(a_1, ..., a_k))$. Those functions are then considered to be elements of L_f. Without loss of generality, we assume that all functions in L_f are total. Then for each $f \in L_f$ of arity k, we need to store $|A|^k$ values. Using the ranking, we represent f by concatenating the function values for all possible tuples $(a_1, ..., a_k)$ in lexicographical ordering of those tuples according to r. To encode $\mathcal{D}$, we then only need to concatenate all representations of functions in L_f in the order fixed on the signature. We will denote this encoding by $\mathrm{enc}(\mathcal{D})$.

In order to be able to compute $|A|$ from enc($\mathcal{D}$), we make an exception for constant functions and predicates, i.e. 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

$$|enc(\mathcal{D})| = \sum_{f \in L_f} |A|^{ar(f)}, \tag{2.11}$$

where $ar(f)$ is the arity of f (unless that arity is 0, in which case $ar(f)$ is 1), we can reconstruct $|A|$ from the arities of the functions in L_s and the length of $enc(\mathcal{D})$. We can do so by using for example binary search, since we know that $|A|$ is between 0 and $|enc(\mathcal{D})|$. We can therefore compute $|A|$ when given φ and $|enc(\mathcal{D})|$ in time logarithmic in $|enc(\mathcal{D})|$.

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

Fix a countable set $V = \{v_0, v_1, ...\}$ of variables. (These range over the skeleton of the input structure. We do not consider variables ranging over $\mathbb{R}$.)

Definition 26 (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 $h_1 \doteq h_2$ and number terms $t_1 \doteq t_2$, 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.

Remark 27. Since we have $t_1 < t_2$ and Boolean connectives, we can easily construct $t_1 > t_2$, $t_1 \leq t_2$ and $t_1 \geq t_2$ and will use them freely when working with first-order logic.

The set $FO_{\mathbb{R}}$ 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 $\mathcal{A}$. We do not consider formulas where the quantified variable ranges over $\mathbb{R}$.

A $FO_{\mathbb{R}}$-formula which does not contain any free variables is called a $FO_{\mathbb{R}}$-sentence. We explicitly make this distinction here, since in this thesis, we are mostly concerned with sets defined by sentences. We will get to what exactly this means shortly in Definition 29.

Definition 28. The *interpretation* of a $FO_{\mathbb{R}}$-formula φ given a $\mathbb{R}$-structure $\mathcal{D} = (\mathcal{A}, \mathcal{F})$ of signature $\sigma = (L_s, L_f)$ and an assignment $\mu : V \to A$, which maps each free variable in φ to an element of A, is as one would expect.

1. If $\varphi = h_1 \doteq h_2$ for index terms h_1, h_2, then φ evaluates to true if and only if h_1 and h_2 evaluate to the same value.
2. If $\varphi = t_1 \doteq t_2$ for number terms t_1, t_2, then φ evaluates to true if and only if t_1 and t_2 evaluate to the same value.
3. If $\varphi = t_1 < t_2$ for number terms t_1, t_2, then φ evaluates to true if and only if t_1 evaluates to a value lower than the one t_2 evaluates to.
4. If $\varphi = P(h_1, ..., h_k)$ for a k-ary predicate symbol $P \in L_s$, then φ evaluates to true if and only if P evaluates to true given the values $h_1, ..., h_k$ as arguments.
5. If $\varphi = \psi \wedge \xi$ for $FO_{\mathbb{R}}$-formulas ψ and ξ, then φ evaluates to true if and only if ψ and ξ both evaluate to true.
6. The other Boolean connectives follow analogously.
7. If $\varphi = \exists x \psi$ for a $FO_{\mathbb{R}}$-formula ψ, then φ evaluates to true if and only if there exists a value $a \in A$, such that given $\mathcal{D}$ and $\mu \cup \{x \mapsto a\}$, ψ evaluates to true.
8. If $\varphi = \forall x \psi$ for a $FO_{\mathbb{R}}$-formulas ψ, then the interpretation of φ follows analogously to the existential case.

The interpretation of index terms goes as follows: Let h be an index term.

1. If $h = x$ for a variable x, then h evaluates to $\mu(x)$.
2. If $h = f(h_1, ..., h_k)$ for a k-ary function symbol $f \in L_s$ and index terms $h_1, ..., h_k$, then h evaluates to $f(a_1, ..., a_k)$, where a_i is the value h_i evaluates to.

Analogously let t be a number term.

1. If $t = c$ for $c \in \mathbb{R}$, then t evaluates to c.
2. If $t = f(h_1, ..., h_k)$ for a k-ary function symbol $f \in L_f$ and index terms $h_1, ..., h_k$, then t evaluates to $f(a_1, ..., a_k)$, where a_i is the value h_i evaluates to.
3. If $t = t_1 + t_2$ for number terms t_1 and t_2, then t evaluates to the sum of the values t_1 and t_2 evaluate to.
4. The evaluation for $t = t_1 \times t_2$ and $t = sign(t_1)$ follows analogously.

We say that a $FO_{\mathbb{R}}$-formula is *satisfied* by a $\mathbb{R}$-structure and an assignment, if when given those, it evaluates to true.

We say that a $FO_{\mathbb{R}}$-sentence φ is satisfied by a $\mathbb{R}$-structure $\mathcal{D}$ – in symbols $\mathcal{D} \models \varphi$ – if φ when given $\mathcal{D}$ (and an empty assignment) evaluates to true.

As mentioned previously, we would like to talk about sets which we can describe by using $FO_{\mathbb{R}}$-sentences. In order to do this, we define the following:

Definition 29. A $FO_{\mathbb{R}}$-sentence φ *defines* a set of $\mathbb{R}$-structures S, if and only if the structures satisfying φ are exactly the structures in S.

Accordingly, the class $FO_{\mathbb{R}}$ is then the class of all sets definable by $FO_{\mathbb{R}}$-sentences, i.e.

$$FO_{\mathbb{R}} = \bigcup_{\substack{\mathbb{R}\text{-signatures } \sigma, \\ FO_{\mathbb{R}}\text{-sentences } \varphi}} \{\mathcal{D} \mid \mathcal{D} \in Struct(\sigma), \mathcal{D} \models \varphi\} \tag{2.12}$$

Definition 30. Two $FO_{\mathbb{R}}$-sentences are called *semantically equivalent*, if and only if they define the same set, i.e. they are satisfied by the same $\mathbb{R}$-structures.

2.3.2 A few Extensions to $FO_{\mathbb{R}}$

In the following, we would like to consider logics, which also have access to functions and relations that are not given by their input structure. To that end, we make a small addition to Definition 22, 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. The interpretation of the symbols of L_a is then analogous to the interpretation of the symbols in L_s and L_f.

Definition 31. Let R be a set of finite relations and functions. We will write $FO[R]$ to denote the class of sets S for which there exists a $FO_{\mathbb{R}}$-sentence φ over a signature $\sigma = (L_s, L_f, L_a)$ such that for every input structure there is an interpretation I interpreting the symbols in L_a in such a way that φ with interpretation I defines S.

Definition 32. Let Arb denote the set of all finite relations and functions. Then $FO_{\mathbb{R}}[Arb]$ contains the sets definable by $FO_{\mathbb{R}}$ sentences which can contain function and relation symbols, whose interpretations have no restrictions in terms of computability or complexity.

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 $FO_{\mathbb{R}}$ and thus do not gain expressive power by using it. We will use this rule to show that we can represent the characteristic function in $FO_{\mathbb{R}}$.

Those rules should then allow us to represent the unbounded fan-in of circuits deciding $AC^0_{\mathbb{R}}$ sets.

Definition 33 (The sum, product and maximization rule)**.** Let t be a number term in which the variable i occurs freely with other variables $\overline{w} = w_1, ..., w_j$ and let A denote the universe of the given input structure. Then

$$\underset{i}{sum}(t(i, \overline{w}))$$

is also a number term which is interpreted as

$$\sum_{i \in A} t(i, \overline{w}),$$

that is, the sum over all values $v \in A$, if i is in t replaced by v and the other variables are replaced by their respective values.

We will analogously define

$$\underset{i}{prod}(t(i, \overline{w}))$$

to be interpreted as

$$\prod_{i \in A} t(i, \overline{w})$$

and

$$\max_{i}(t(i, \overline{w}))$$

to be interpreted as the maximum value for t if i is replaced by an element of A and the other variables are replaced by their respective values.

Remark 34. Note that $\underset{i}{sum}$, $\underset{i}{prod}$ and $\max_{i}$ work essentially in the same way as quantifiers do. We will for that reason, when dealing with a formula φ in which $\underset{i}{sum}$, $\underset{i}{prod}$ or $\max_{i}$ occur, consider the variable i to be bound in φ.

Definition 35. For a logic L, we will by $L + SUM_{\mathbb{R}}$ denote L extended by the sum rule and by $L + PROD_{\mathbb{R}}$ denote L extended by the product rule.

Definition 36. For a number $q \in \mathbb{N}$ and a number term t in which $\overline{w} = w_1, ..., w_j$ occur and $i_1, ..., i_q$ occur freely, we will for convenience write

$$\underset{i}{sum}^q(t(i_1, ..., i_q, \overline{w}))$$

to denote

$$\underset{i_1}{sum}(...\underset{i_q}{sum}(t(i_1, ..., i_q, \overline{w})))$$

which is then interpreted in accordance to Definition 33 as

$$\sum_{i_1 \in A} ... \sum_{i_q \in A} t(i_1, ..., i_q, \overline{w})$$

and

$$\underset{i}{prod}^q(t(i_1, ..., i_q, \overline{w}))$$

to denote

$$\underset{i_1}{prod}(...\underset{i_q}{prod}(t(i_1, ..., i_q, \overline{w})))$$

which is interpreted as

$$\prod_{i_1 \in A} ... \prod_{i_q \in A} t(i_1, ..., i_q, \overline{w}).$$

We would now like to see which extensions of our first-order logic can already use the aforementioned rules naturally. We demonstrate that, as mentioned before, we can use $\underset{i}{\max}$ in $FO_{\mathbb{R}}$ without extending it and we can use $\underset{i}{sum}$ and $\underset{i}{prod}$ in $FO_{\mathbb{R}}[Arb]$. We will later show that we can also define $\underset{i}{sum}$ and $\underset{i}{prod}$ in a polynomial extension of $FO_{\mathbb{R}}$.

Lemma 37. *For any $FO_{\mathbb{R}}$-formula φ which contains $\underset{i}{\max}$-constructions, we can define a semantically equivalent formula φ which does not contain any $\underset{i}{\max}$-constructions.*

Proof. Let φ be a $FO_{\mathbb{R}}$ formula which contains $\underset{i}{\max}$-constructions, i.e. number terms of the form $\underset{i}{\max}(t(i, \overline{w}))$ for a number term t. We will show that for every such formula, we can construct another $FO_{\mathbb{R}}$-formula φ' which is equivalent to φ but which does not contain the term $\underset{i}{\max}(t(i, \overline{w}))$. Since $\underset{i}{\max}$-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 $\underset{i}{\max}$-constructions into semantically equivalent formulas (that are not necessarily atomic anymore).

For a given atomic formula with $\underset{i}{\max}$-constructions φ, define φ' as follows: Let $\varphi = t_1 \doteq t_2$ and let $\underset{i_1}{\max}, ..., \underset{i_k}{\max}$ be the $\underset{i}{\max}$-occurrences of φ, ordered by level of nesting, where $\underset{i_1}{\max}$ has the lowest level of nesting, the nesting of $\underset{i_2}{\max}$ is either the same as $\underset{i_1}{\max}$ 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 φ. We also assume for now that there is only one occurrence of $\underset{i}{\max}$ at the lowest level of nesting and that t_1 consists only of that outermost $\underset{i}{\max}$-construction, i.e. $t_1 = \underset{i_1}{\max}(F_1(i_1, \overline{w}_1))$. To now construct φ', we go through the $\underset{i}{\max}$-occurrences in φ in reverse order of nesting, i.e. from the deepest level to the shallowest, and for each occurrence $\underset{i_m}{\max}(F_m(i_m, \overline{w}_m))$, we create a subformula ψ_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 φ'. φ' will then have the form

$$\varphi' = \exists x_1 \forall y_1 ... \exists x_k \forall x_k \psi_k \wedge ... \wedge \psi_1 \wedge \widehat{\varphi}, \tag{2.13}$$

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

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

We now create the subformula

$$\psi_{i_k} \coloneqq F_k(x_k, x_{k_1}, ..., x_{k_j}, \overline{w}_k) \geq F_k(y_k, x_{k_1}, ..., x_{k_j}, \overline{w}_k), \tag{2.14}$$

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

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

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

$$\psi_m \coloneqq F'_m(x_m, x_{m_1}, ..., x_{m_j}, \overline{w}_m) \geq F'_m(y_m, x_{m_1}, ..., x_{m_j}, \overline{w}_m). \tag{2.15}$$

Finally, we define

$$\varphi' \coloneqq \exists x_1 \forall y_1 ... \exists x_k \forall y_k \ \psi_k \wedge ... \wedge \psi_1 \wedge F'_1(x_1, \overline{w}_1) \doteq t_2. \tag{2.16}$$

This construction now works for our strong assumption that $t_1 = \max\limits_{i_1}(F_1)$. However, we only require the following modifications to make it generally applicable: If φ contains only one $\max\limits_{i}$-construction at the lowest level, but then operates on that construction, we can just add the context of that $\max\limits_{i}$-construction to the term F'_1 in φ'. For example if $\varphi = 7 \doteq \max\limits_{i}(F(i)) + 1$, then we could just add the '+1' to the $F'_1(x_1, \overline{w}_1)$ in Formula 2.16. If φ contains several $\max\limits_{i}$-constructions at the lowest level of nesting, then we can construct as we have previously and just add the subformulas to the conjunction in φ'.

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

We can construct φ' analogously, if both, t_1 and t_2 contain $\max\limits_{i}$-constructions or if $\varphi = t_1 < t_2$. We have therefore shown that for any $FO_{\mathbb{R}}$-formula with $\max\limits_{i}$-constructions, there exists a semantically equivalent formula which does not contain any such constructions.

□

Example 38. Let $\varphi = 7 \doteq \max_i(f(i) + 2)$. Then

$$\varphi' = \exists x \forall y \; f(x) + 2 \geq f(y) + 2 \wedge 7 \doteq f(x) + 2 \tag{2.17}$$

Example 39. Let $\varphi = 5 \doteq \max_i(\max_j(f(j) + g(i)) + f(i))$. Then

$$\begin{aligned}\varphi' = &\exists x_1 \forall y_1 \exists x_2 \forall y_2 \\ &\quad f(x_2) + g(x_1) \geq f(y_2) + g(x_1) \\ &\wedge (f(x_2) + g(x_1)) + f(x_1) \geq (f(x_2) + g(y_1)) + f(y_1) \\ &\wedge 5 \doteq f(x_2) + g(x_1) + f(x_1)\end{aligned} \tag{2.18}$$

Lemma 40. *For any $FO_{\mathbb{R}}[Arb]$-formula φ, which contains $\underset{i}{sum}$-constructions, we can define a semantically equivalent $FO_{\mathbb{R}}[Arb]$ formula φ' which does not contain $\underset{i}{sum}$-constructions.*

Proof. In order to prove this, we will take an arbitrary $FO_{\mathbb{R}}[Arb]$-sentence φ in which number terms of the form $\underset{i}{sum}(t(i, \overline{w}))$ occur and then create an $FO_{\mathbb{R}}[Arb]$-term φ' which is semantically equivalent to φ but which does not contain any such constructions. Note that the set L_a of the signature of φ' will be different to that of φ.

Let φ be a valid $FO_{\mathbb{R}}[Arb]$ sentence of signature $\{L_s, L_f, L_a\}$ with the exception that it contains number terms of the form $\underset{i}{sum}(t(i, \overline{w}))$, where $t(i, \overline{w})$ is a number term (which may again contain $\underset{i}{sum}$-constructions). Without loss of generality, we assume that for all number terms of the form $\underset{i}{sum}(t(i, \overline{w}))$, there is no symbol $sum_{t,i}$ in the signature of φ. We now construct a number term φ' which is equivalent to φ but which does not contain any constructions of the aforementioned form.

We define φ' step by step. We start out by $\varphi' := \varphi$.

We now take any such instance of a number term $\underset{i}{sum}(t(i, \overline{w}))$ in φ' where t itself does not contain any instances of $\underset{i}{sum}$-constructions. Let $\mathcal{D} = \{L_s, L_f, L_a\}$ be the signature of φ'. 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 φ' the term $\underset{i}{sum}(t(i, \overline{w}))$ 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 $(y_1, ..., y_k) \in A$:

$$sum_{t,i}(y_1, ..., y_j) := \sum_{i \in A} t(y_1, ..., y_k) \tag{2.19}$$

The resulting formula φ' is now semantically equivalent to φ, since we just moved the interpretation of the symbol $\underset{i}{sum}$ to the function symbol $sum_{t,i}$, but it does not contain the instance of $\underset{i}{sum}(t(i, \overline{w}))$ that we just removed. If we repeat this process for all remaining occurrences of $\underset{i}{sum}$ in φ', we arrive at a sentence which is semantically equivalent to φ but which does not contain any instances of $\underset{i}{sum}$.

□

Example 41. Let $\varphi = \exists x\ 1 \doteq \underset{i}{sum}(f(i) + f(x))$. Then

$$\varphi' = \exists x\ 1 \doteq sum_{f(i)+f(x),i}(x) \tag{2.20}$$

where $sum_{f(i)+f(x),i}(x)$ is a function symbol which for any input structure $\mathcal{D} = (\mathcal{A}, \mathcal{F})$ is interpreted as

$$\sum_{i \in A} f(i) + f(x). \tag{2.21}$$

Example 42. Let $\varphi = \exists x \forall y\ 7 \doteq \underset{i}{sum}(\underset{j}{sum}(f(i) + g(j)) + f(y)) + g(x)$ a formula of signature $\{L_s, L_f, L_a\}$. Then after the first iteration

$$\varphi' = \exists x \forall y\ 7 \doteq \underset{i}{sum}(sum_{f(i)+g(j),j}(i) + f(y)) + g(x) \tag{2.22}$$

and after the second iteration

$$\varphi' = \exists x \forall y\ 7 \doteq sum_{sum_{f(i)+g(j),j}(i)+f(y),i}(y) + g(x) \tag{2.23}$$

where $sum_{f(i)+g(j),j}(i)$ is a 1-ary function symbol interpreted as

$$sum_{f(i)+g(j),j}(i) \coloneqq \sum_{j \in A} f(i) + g(j) \tag{2.24}$$

and $sum_{sum_{f(i)+g(j),j}(i)+f(y),i}(y)$ is another 1-ary function symbol interpreted as

$$sum_{sum_{f(i)+g(j),j}(i)+f(y),i}(y) \coloneqq \sum_{i \in A} sum_{f(i)+g(j),j}(i) + f(y) = \sum_{i \in A} \left(\sum_{j \in A} f(i) + g(j) \right) + f(y) \tag{2.25}$$

for an input structure of signature $\{L_s, L_f, L_a \cup \{sum_{f(i)+g(j),j}, sum_{sum_{f(i)+g(j),j}(i)+f(y),i}\}\}$.

Corollary 43. *For any $q \in \mathbb{N}$, we can also define $\underset{i}{prod}$, $\underset{i}{sum}^q$ and $\underset{i}{prod}^q$ in $FO_{\mathbb{R}}[Arb]$.*

Remark 44. For the following, we 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 $\chi_P : A^k \to \mathbb{R}$.

We assume for the same reason that for any logic $FO_{\mathbb{R}}[R]$, R only contains functions.

As mentioned before, the reason why we need the maximization rule is that we would like to write characteristic functions as number terms. The following result is a slight modification of a result presented by Cucker and Meer [CM99].

Proposition 45. *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 $\chi[\varphi]$.*

Proof. We will prove this proposition by induction on the construction of φ. If φ is atomic, then it is of the form $t_1 \doteq t_2$, $t_1 < t_2$ for number terms t_1, t_2, since we only consider functional $\mathbb{R}$-structures. For atomic formulas, we have

$$\chi[t_1 \doteq t_2] = sign'[-(t_1 - t_2)^2] \tag{2.26}$$

and

$$\chi[t_1 < t_2] = 1 - [sign'(t_1 - t_2)]. \tag{2.27}$$

If φ is of the form $\varphi = \exists x \psi(x)$, then

$$\chi[\varphi] = \max_x \chi[\psi(x)]. \tag{2.28}$$

If φ has the form $\varphi = \neg\psi$, then

$$\chi[\varphi] = 1 - \chi[\psi] \tag{2.29}$$

and if $\varphi = \psi \wedge \xi$, then

$$\chi[\varphi] = \chi[\psi] \times \chi[\xi]. \tag{2.30}$$

Since $\varphi = \forall x \psi(x)$ and the remaining Boolean connectives can be constructed from the above, we have now shown that we can describe $\chi[\varphi]$ in $FO_{\mathbb{R}}[R]$ for any $\varphi \in FO_{\mathbb{R}}[R]$. □

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

In this section, we show descriptive complexity results for the non-uniform set $AC^0_{\mathbb{R}}$ 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 $AC^0_{\mathbb{R}}$

First of all we show an equality which is close to a classical result presented by Vollmer [Vol99]. We show that extending our first-order logic over the reals with arbitrary functions lets us exactly describe the non-uniform set $AC^0_{\mathbb{R}}$.

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

Lemma 46. *For every $AC^0_{\mathbb{R}}$-circuit family $(\mathcal{C}_n)_{n\in\mathbb{N}}$, i.e. circuit families with unbounded fan-in gates, constant depth and polynomial size, there exists an $AC^0_{\mathbb{R}}$-circuit family $(\mathcal{C}'_n)_{n\in\mathbb{N}}$ computing the same function, such that for every gate v in $\mathcal{C}'_n$, every path from an input gate to v has the same length.*

Proof. In order to prove this, we construct, for any given $AC^0_{\mathbb{R}}$-family $\mathcal{C}$, an $AC^0_{\mathbb{R}}$-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 $\mathcal{C}_n$ be an $AC^0_{\mathbb{R}}$-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 $\mathcal{C}_n$ induced by non-input gates which have outdegree > 1 in which every other non-input gate has outdegree 1. Since $\mathcal{C}_n$ 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 $\mathcal{C}_n^{sub,g}$ now proceed as follows: Let g be the root node of $\mathcal{C}_n^{sub,g}$. Replace each connection beyond the first from g to a successor by a copy of $\mathcal{C}_n^{sub,g}$, i.e. by a subcircuit which has the same input-gates as $\mathcal{C}_n^{sub,g}$ and where all other gates and connections are copies of the gates and connections in $\mathcal{C}_n^{sub,g}$. 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 $\mathcal{C}_n$ and denote the circuit after the ith step by $\mathcal{C}_n^i$.

Let $q \in \mathbb{N}$ be such that $size(\mathcal{C}_n) < n^q$. 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 $size(\mathcal{C}_n^i) < n^{q*(1+2i)}$ for all $i \in \mathbb{N}$.

Base case $\mathcal{C}_n^1$: After the first step of this process we have increased the size of $\mathcal{C}_n$ 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 $size(\mathcal{C}_n^1) < (n^q) + (n^q - 1) * (n^q)^2 < (n^q)^3 = n^{q*(1+2)}$.
Induction step $\mathcal{C}_n^k \to \mathcal{C}_n^{k+1}$: In the kth step, we replaced all subcircuits induced by non-input nodes with multiple successors in $\mathcal{C}_n^{k-1}$ by copies. This means that all root nodes we consider in the k+1th 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 $size(\mathcal{C}_n^{k+1}) < n^{q*(1+2k)} + n^{q*(1+2k)} * n^q * (n^q - 1) < n^{q*(1+2k)} * n^{2q} = n^{q*(1+2(k+1))}$.

Let $\mathcal{C}'_n$ 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 $\mathcal{C}'_n = \mathcal{C}_n^k$ for some $k \in \mathbb{N}$, which means that the size of the $\mathcal{C}'_n$ is still a polynomial in n. The depth of $\mathcal{C}'_n$ 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 $\mathcal{C}_n$ is still an $AC^0_{\mathbb{R}}$-circuit which computes the same function as $\mathcal{C}_n$ but does not contain any non-input gates with multiple successors.

Step two:
We know that $\mathcal{C}'_n$ 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 $\mathcal{C}'_n$, 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 $\mathcal{C}''_n$. As we will see, this also results in the property that we wanted in the first place: for every node v in $\mathcal{C}''_n$, all paths from input gates to v have the same length. We show this by contradiction:

Assume that there is a gate v in $\mathcal{C}''_n$ to which there are two paths from input gates with different lengths. We know that v and all its successors have outdegree ≤ 1, therefore

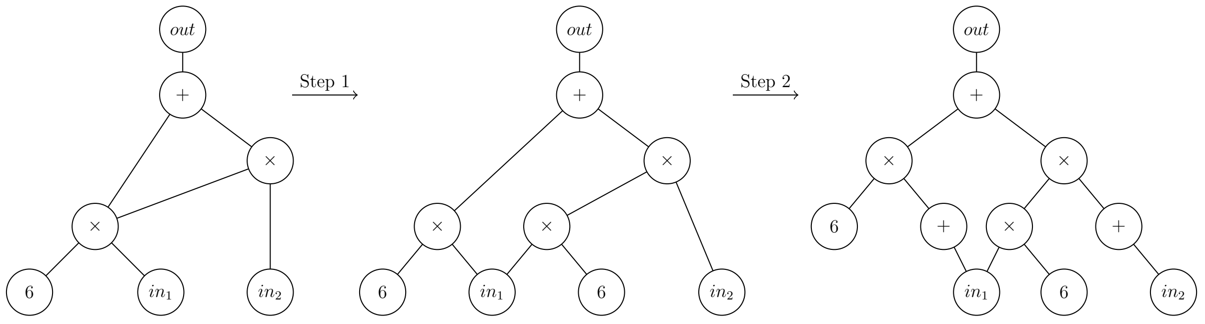

Figure 3.1: Example for the two steps in the construction for Lemma 46

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 $size(\mathcal{C}'_n) * n * depth(\mathcal{C}'_n)$, 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 $size(\mathcal{C}'_n)$ outgoing edges from those, so we have at most $n * size(\mathcal{C}'_n)$ paths. Each path now gets padded by less than $depth(\mathcal{C}'_n)$ nodes. In the end, since $size(\mathcal{C}''_n)$ is a polynomial in n and $depth(\mathcal{C}''_n)$ 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 $\mathcal{C}'_n$, since addition gates with only a singular predecessor are essentially just identity gates.

An example of the construction in this proof is given in Figure 3.1. □

Before we go ahead with the proof of our description of $FO_{\mathbb{R}}[Arb]$, we need to make one final proposition. When given a circuit family C, we need a way to identify the gates of C_n which only depends on n. In order to do so, we will make use of the fact that the size of our circuits is a polynomial in its input length. We therefore state the following:

Proposition 47. *For any finite, ordered set S and any constant q, we can uniquely identify each vector $v = (v_1, ..., v_q)$ with a number $n \in \mathbb{N}$, where $v_i \in S$ for all i. This computation can be done in constant time.*

Proof. Let r be the function associating each element in S with its position in the ordering of S. Given $\overline{v}$, we compute $f(\overline{v}) = r(v_1) + r(v_2) * |S| + r(v_3) * |S|^2 + ... + r(v_q) * |S|^{q-1}$, which uniquely identifies $\overline{v}$ with a number and which can be done in constant time with our $\mathbb{R}$-machines. □

Theorem 48. $FO_{\mathbb{R}}[Arb] = AC^0_{\mathbb{R}}$.

Proof. $FO_{\mathbb{R}}[Arb] \subseteq AC^0_{\mathbb{R}}$:
To show that $FO_{\mathbb{R}}[Arb]$ is included in $AC^0_{\mathbb{R}}$, we will show that for any $FO_{\mathbb{R}}[Arb]$-sentence φ, we can create an $AC^0_{\mathbb{R}}$ circuit family which decides exactly the set defined by φ.

Given a fixed size n of input $\mathbb{R}$-structures $\mathcal{D} = (\mathcal{A}, \mathcal{F})$ ($n = |enc(\mathcal{D})|$), we can for any $FO_{\mathbb{R}}$-formula reconstruct $|A|$ from n as per Definition 25. We will denote $|A|$ by u. For

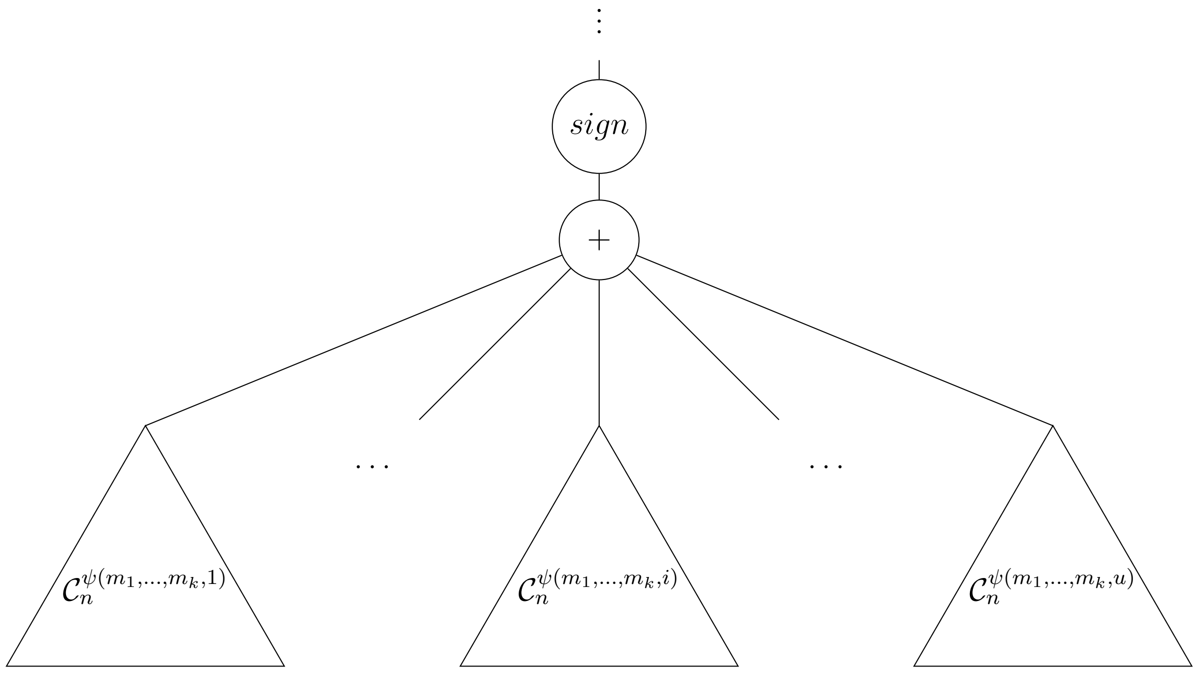
sign
+
$\mathcal{C}_n^{\psi(m_1,...,m_k,1)}$
$\mathcal{C}_n^{\psi(m_1,...,m_k,i)}$
$\mathcal{C}_n^{\psi(m_1,...,m_k,u)}$

Figure 3.2: Construction of the circuit for Theorem 48 if $\varphi = \exists y \psi(y)$.

any such formula φ 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 $\mathcal{C}_n^{\varphi(m_1,...,m_k)}$ with the following property: For any input structure $\mathcal{D}$ it holds that $\mathcal{D} \models \varphi$ if and only if $enc(\mathcal{D})$ is accepted by $\mathcal{C}_n^{\varphi(m_1,...,m_k)}$, where x_i is substituted by m_i for all $1 \leq i \leq k$. (That means that any such circuit $\mathcal{C}_n^{\varphi(m_1,...,m_k)}$ for a formula φ 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 φ 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 $\varphi = \exists y \psi(y)$. If y does not occur free in ψ, then $\mathcal{C}_n^{\varphi(m_1,...,m_k)} = \mathcal{C}_n^{\psi(m_1,...,m_k)}$. Otherwise, the free variables in ψ are $x_1, ..., x_k, y$. $\mathcal{C}_n^{\varphi(m_1,...,m_k)}$ now consists of a sign gate with an unbounded fan-in addition gate as its predecessor which in turn has the circuits $\mathcal{C}_n^{\psi(m_1,...,m_k,i)}$ as its predecessors for $1 \leq i \leq u$. This construction is visualized in Figure 3.2.

2. If $\varphi = \forall y \psi(y)$, then $\mathcal{C}_n^{\varphi(m_1,...,m_k)}$ is defined as in the existential case, but with a multiplication gate below the sign gate.

3. Let $\varphi = \neg\psi$. Then $\mathcal{C}_n^{\varphi(m_1,...,m_k)}$ consists of a subtraction gate, which subtracts the sign of $\mathcal{C}_n^{\psi(m_1,...,m_k)}$ from 1.

4. Let $\varphi = \psi \wedge \xi$. Then $\mathcal{C}_n^{\varphi(m_1,...,m_k)}$ consists of a sign gate followed by a multiplication gate with $\mathcal{C}_n^{\psi(m_1,...,m_k)}$ and $\mathcal{C}_n^{\xi(m_1,...,m_k)}$ 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 $\varphi = \psi \vee \xi$, $\varphi = \psi \implies \xi$ or $\varphi = \psi \iff \xi$, then $\mathcal{C}_n^{\varphi(m_1,...,m_k)}$ follows analogously to $\varphi = \psi \wedge \xi$.

6. Let $\varphi = h_1 \doteq h_2$ for index terms h_1, h_2. Then $\mathcal{C}_n^{\varphi(m_1,...,m_k)}$ consists of an equality gate with the circuits $\mathcal{C}_n^{h_1(m_1,...,m_k)}$ and $\mathcal{C}_n^{h_2(m_1,...,m_k)}$ as its predecessors.

7. If $\varphi = t_1 \doteq t_2$ for number terms t_1, t_2, then $\mathcal{C}_n^{\varphi(m_1,...,m_k)}$ is defined analogously to the case with index terms.

8. Let $\varphi = t_1 < t_2$ for number terms t_1, t_2. Then $\mathcal{C}_n^{\varphi(m_1,...,m_k)}$ consists of a $<$-gate with $\mathcal{C}_n^{t_1(m_1,...,m_k)}$ and $\mathcal{C}_n^{t_2(m_1,...,m_k)}$ 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 $h = x$ for $x \in V$. Then x must be x_i for an $i \in 1, ..., k$ and have previously been quantified. Then $\mathcal{C}_n^{h(m_1,...,m_k)}$ consists of the constant gate with value m_i.

2. Let $h = f(h_1, ..., h_\ell)$ for a ℓ-ary function symbol $f \in L_a$ and index terms $h_1, ..., h_\ell$. Then $\mathcal{C}_n^{h(m_1,...,m_k)}$ consists of an unbounded addition gate at the top with u^ℓ 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 ℓ 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 $\mathcal{C}_n^{h_r(m_1,...,m_k)}$. 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. The described construction is visualized in Figure 3.3.

3. If $h = f(h_1, ..., h_\ell)$ for a ℓ-ary function symbol $f \in L_s$ and index terms $h_1, ..., h_\ell$, then $\mathcal{C}_n^{h(m_1,...,m_k)}$ 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 $t = c$ for $c \in \mathbb{R}$, then $\mathcal{C}_n^{t(m_1,...,m_k)}$ consists of a constant gate with value c.

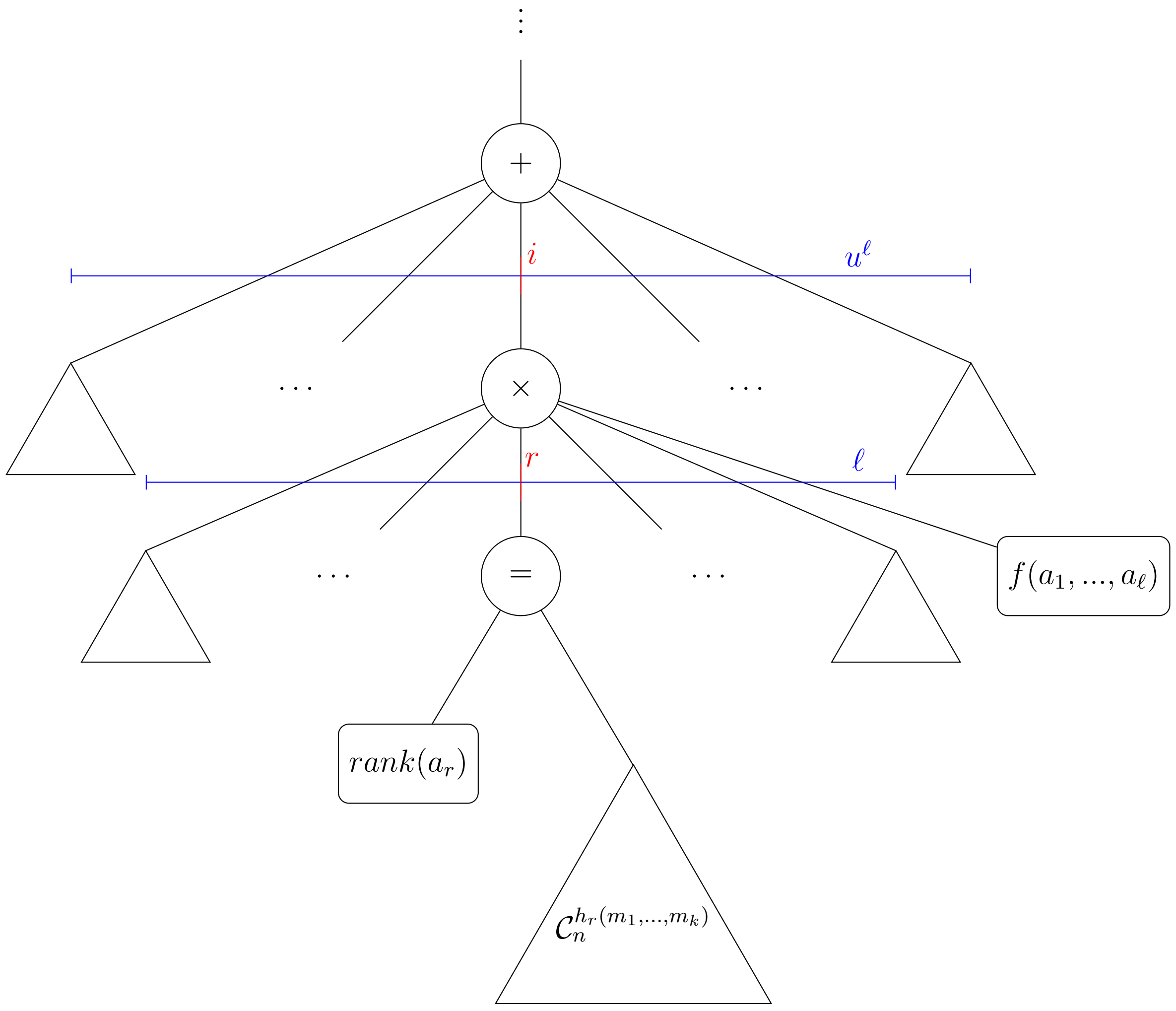

Figure 3.3: Construction of the circuit for Theorem 48 if $h = f(h_1, ..., h_\ell)$ for $f \in L_a$.

2. If $t = f(h_1, ..., h_\ell)$ for a ℓ-ary function symbol $f \in L_a$ and index terms $h_1, ..., h_\ell$, then $\mathcal{C}_n^{t(m_1,...,m_k)}$ is defined analogously to the second case of defining index terms.

3. If $t = f(h_1, ..., h_\ell)$ for a ℓ-ary function symbol $f \in L_f$ and index terms $h_1, ..., h_\ell$, then $\mathcal{C}_n^{t(m_1,...,m_k)}$ is defined as above but with the input gates describing f instead of constant gates.

4. If $t = t_1 + t_2$ or $t = t_1 \times t_2$ for number terms t_1, t_2, then $\mathcal{C}_n^{t(m_1,...,m_k)}$ consists of a $+$ or $\times$ gate at the top with the circuits $\mathcal{C}_n^{t_1(m_1,...,m_k)}$ and $\mathcal{C}_n^{t_2(m_1,...,m_k)}$ as its predecessors.

If φ 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, $S \in AC^0_{\mathbb{R}}$.

$AC^0_{\mathbb{R}} \subseteq FO_{\mathbb{R}}[Arb]$:
To show that $AC^0_{\mathbb{R}}$ is included in $FO_{\mathbb{R}}[Arb]$, we create, for any given $AC^0_{\mathbb{R}}$ set S, an $FO_{\mathbb{R}}[Arb]$-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 $AC^0_{\mathbb{R}}$-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 46, we can construct a sentence which essentially describes the gates on each level of the circuit.

Let $S \in AC^0_{\mathbb{R}}$ via circuit family $\mathcal{C}$, $depth(\mathcal{C}_n) = d$ and let q be such that $size(\mathcal{C}_n) \leq n^q$ for all $n \in \mathbb{N}$. Without loss of generality, let $\mathcal{C}_n$ be a full tree as described in Lemma 46, i.e. for every gate g in $\mathcal{C}_n$ it holds that all paths from input gates to g have the same length. We now create a $FO_{\mathbb{R}}[Arb]$-sentence φ which defines the set decided by $\mathcal{C}$. The set L_f of the signature of φ will only contain one function symbol f, which then for every input gate v in $\mathcal{C}_n$ leads to $f(v)$ being interpreted as the value of v.

Since $\mathcal{C}_n$ is of size at most n^q, we can uniquely identify the gates of $\mathcal{C}_n$ with elements of A^q. Let v be a gate in $\mathcal{C}_n$ encoded by $(v_1, ..., v_q)$. $t_n : A^q \to \mathbb{R}$, $c_n : A^q \to \mathbb{R}$, $in_n : A^{q+1} \to \mathbb{R}$ and $pred_n : A^{2q} \to \mathbb{R}$ are functions where $t_n(v_1, ..., v_q)$ is the type of v as per Definition 8, $in_n(v_1, ..., v_q, i)$ is 1 if v is the input gate i of $\mathcal{C}_n$ 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 $\mathcal{C}_n$ 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 $\mathcal{C}_n$, $val_x(v_1, ..., v_q)$ is the value of the gate encoded by $(v_1, ..., v_q)$ in $\mathcal{C}_n$'s computation when given an $\mathbb{R}$-structure $\mathcal{D}$ where $enc(\mathcal{D}) = (a_1, ..., a_n)$.

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:

$$val_0(v_1, ..., v_q) = \underset{i}{sum}(in(v_1, ..., v_q, i) \times f(i)) \tag{3.1}$$

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

$$\begin{aligned} val_x(v_1, ..., v_q) = & \chi[t(v_1, ..., v_q) \doteq 2] \times T_{2,x}(v_1, ..., v_q) \\ & + \chi[t(v_1, ..., v_q) \doteq 3] \times T_{3,x}(v_1, ..., v_q) \\ & + \chi[t(v_1, ..., v_q) \doteq 4] \times T_{4,x}(v_1, ..., v_q) \\ & + \chi[t(v_1, ..., v_q) \doteq 5] \times T_{5,x}(v_1, ..., v_q) \\ & + \chi[t(v_1, ..., v_q) \doteq 6] \times T_{6,x}(v_1, ..., v_q) \end{aligned} \tag{3.2}$$

where

$$T_{2,x}(v_1, ..., v_q) = c(v_1, ..., v_q) \tag{3.3}$$

$$T_{3,x}(v_1, ..., v_q) = \underset{i}{sum}^q(pred(i_1, ..., i_q, v_1, ..., v_q) \times val_{x-1}(i_1, ..., i_q)) \tag{3.4}$$

$$T_{4,x}(v_1, ..., v_q) = \underset{i}{prod}^q(pred(i_1, ..., i_q, v_1, ..., v_q) \times val_{x-1}(i_1, ..., i_q)) \tag{3.5}$$

$$T_{5,x}(v_1, ..., v_q) = \underset{i}{sum}^q(pred(i_1, ..., i_q, v_1, ..., v_q) \times sign(val_{x-1}(i_1, ..., i_q))) \tag{3.6}$$

$$T_{6,x}(v_1, ..., v_q) = \underset{i}{sum}^q(pred(i_1, ..., i_q, v_1, ..., v_q) \times val_{x-1}(i_1, ..., i_q)) \tag{3.7}$$

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

$$\varphi = \forall i_1 ... \forall i_q(\chi[t(i_1, ..., i_q) \doteq 6 \implies val_d(i_1, ..., i_q) \doteq 1]) \tag{3.8}$$

The formula φ is independent of the input length n, however the interpretations of its function symbols of L_a are not.

$\square$

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

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

For this reason, we introduce another notation here:

Definition 49. By $FTIME_{\mathbb{R}}(f(n))$ 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 50. $FO_{\mathbb{R}}[FTIME_{\mathbb{R}}(n^{\mathcal{O}(1)})] = U_P$-$AC^0_{\mathbb{R}}$

Proof. $FO_{\mathbb{R}}[FTIME_{\mathbb{R}}(n^{\mathcal{O}(1)})] \subseteq U_P$-$AC^0_{\mathbb{R}}$:

The construction of the circuit is analogous to the one in Theorem 48. 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 $n = |enc(\mathcal{D})|$ in logarithmic time as per Definition 25, it follows that there exists a machine which on input (n, v_{nr}, p_{idx}) can compute (t, p_{nr}, c) as in Definition 17 in time bounded by a polynomial in n.

U_P-$AC^0_{\mathbb{R}} \subseteq FO_{\mathbb{R}}[FTIME_{\mathbb{R}}(n^{\mathcal{O}(1)})]$:

For a given U_P-$AC^0_{\mathbb{R}}$ set S, we can also create a formula in the same way as in Theorem 48. We only need to show that we can define the number terms $t(v_1, ..., v_q)$, $c(v_1, ..., v_q)$, $in(v_1, ..., v_q, i)$, $pred(v_1, ..., v_q, w_1, ..., w_q)$, $\underset{i}{sum}(F(i_1, ..., i_q, \overline{w}))$ and $\underset{i}{prod}^q(F(i_1, ..., i_q, \overline{w}))$ in $FO_{\mathbb{R}}[FTIME_{\mathbb{R}}(n^{\mathcal{O}(1)})]$, since we can then just use the construction from Theorem 48.

Let A be the universe of the input structure.

1. Clearly, $t(v_1, ..., v_q)$, $c(v_1, ..., v_q)$ and $in(v_1, ..., v_q, i)$ can be defined in $FO_{\mathbb{R}}[FTIME_{\mathbb{R}}(n^{\mathcal{O}(1)})]$, since the family defining S is *P-uniform*.

2. $\underset{i}{sum}(F(i_1, ..., i_q, \overline{w}))$ can be defined in $FO_{\mathbb{R}}[FTIME_{\mathbb{R}}(n^{\mathcal{O}(1)})]$ as follows:

   a) If $F(i_1, ..., i_q, \overline{w}) = c$ for $c \in \mathbb{R}$, then $\underset{i}{sum}(F(i_1, ..., i_q)) = |A| * c$, which can be computed in a single step.

   b) If $F(i_1, ..., i_q, \overline{w}) = X(h_1(i_1, ..., i_q, \overline{w}), ..., h_k(i_1, ..., i_q, \overline{w}))$ 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 $F(i_1, ..., i_q, \overline{w}) = X(h_1(i_1, ..., i_q, \overline{w}), ..., h_k(i_1, ..., i_q, \overline{w}))$ 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 $F(i_1, ..., i_q, \overline{w}) = t_1(i_1, ..., i_q, \overline{w}) + t_2(i_1, ..., i_q, \overline{w})$ 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. $\underset{i}{prod}(F(i_1, ..., i_q, \overline{w}))$ can be defined analogously.

4. $pred(v_1, ..., v_q, w_1, ..., w_q)$ can be defined in $FO_{\mathbb{R}}[FTIME_{\mathbb{R}}(n^{\mathcal{O}(1)})]$ in the following way:

   We define the predicate

$$pred_k := \left\{ (v_1, ..., v_q, w_1, ..., w_q, k_1, ..., k_q) \;\middle|\; \begin{array}{l} v \text{ is the } k\text{th predecessor of } w \\ \text{where } v \text{ is the gate encoded} \\ \text{by } (v_1, ..., v_q),\ w \text{ is the gate} \\ \text{encoded by } (w_1, ..., w_q) \text{ and} \\ k \text{ is the number encoded by} \\ (k_1, ..., k_q). \end{array} \right\} \quad (3.9)$$

   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 $FO_{\mathbb{R}}[FTIME_{\mathbb{R}}(n^{\mathcal{O}(1)})]$ as follows:

$$pred(v_1, ..., v_q, w_1, ..., w_q) = \chi[\exists k_1, ..., \exists k_q pred_k(v_1, ..., v_q, w_1, ..., w_q, k_1, ..., k_q)] \quad (3.10)$$

Therefore we can define S using a $FO_{\mathbb{R}}[FTIME_{\mathbb{R}}(n^{\mathcal{O}(1)})]$ sentence. □

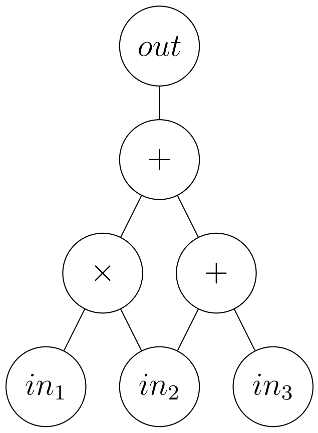

Figure 3.4: The size of this circuit is 7, however its $fullsize$ is 8, since the second input gate has two successors.

3.3 A Characterization for U_L-$AC^0_{\mathbb{R}}$

We have demonstrated that the same construction as in the proof of Theorem 48 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 U_L-$AC^0_{\mathbb{R}}$ 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 $FO_{\mathbb{R}}[FTIME_{\mathbb{R}}(\log(n))]$ and we assume that they cannot be defined therein.

Additionally, since the circuits we have constructed with our method so far have been very tree-like, we would like to take advantage of that and number our gates in a post-order fashion. To do this while constructing the circuit, we would like to have a measure of the size of the subcircuit induced by a node. However, since we would like to use a tree numbering scheme, we cannot just use the actual size of the subcircuit, which we defined earlier. We know that our circuits are structured exactly as trees, with the exception of the leaf nodes. The leaf nodes, which are our circuits' input nodes, can have several successors. So what we would like is a measure for the size of the circuit if it were a tree, i.e. in our case if each input node was counted once for each connection to our circuit.

Definition 51. By the $fullsize$ of a circuit we denote the number of gates of the circuit where each input gate is counted once for each connection it has to the circuit. An example for this is given in Figure 3.4.

Remark 52. Note that *fullsize* does not refer to the size of the full tree of a circuit. Even though the names sound similar, they refer to different properties!

Since we would like to have access to the $fullsize$ of our circuits during our computations, we need to see, how efficiently we can compute the $fullsize$ of circuits in our construction.

As it turns out, the number of computation steps we need does not depend on the size of our given input structure and is therefore constant for our purposes.

Lemma 53. *For a circuit constructed for a given $FO_{\mathbb{R}}$-sentence and $\mathbb{R}$-structure, as in Theorem 48, we can compute the fullsize of the circuit for φ or any circuit for a subformula or number or index term of φ in constant time with respect to the given input structure.*

Proof. Note that since the variable assignments of the notation for Theorem 48 do not make a difference for the size of the circuit, we will omit them in this proof.

Let φ be the given formula and $u = |A|$ be the size of the input structure. We give the *fullsize* of the circuit for every subformula and term of φ by induction in the same way, as the circuit is constructed in the proof of Theorem 48.

1. Let $\varphi = \exists y \psi(y)$. Then $fullsize(\mathcal{C}_n^{\varphi}) = u * fullsize(\mathcal{C}_n^{\psi})$.
2. If $\varphi = \forall y \psi(y)$, then the *fullsize* is computed as in the existential case.
3. Let $\varphi = \neg\psi$. Then $fullsize(\mathcal{C}_n^{\varphi}) = 2 + fullsize(\mathcal{C}_n^{\psi})$.
4. Let $\varphi = \psi \wedge \xi$. Then $fullsize(\mathcal{C}_n^{\varphi}) = 1 + fullsize(\mathcal{C}_n^{\psi}) + fullsize(\mathcal{C}_n^{\xi})$.
5. If $\varphi = \psi \vee \xi$, $\varphi = \psi \implies \xi$ or $\varphi = \psi \iff \xi$, then the *fullsize* can be computed analogously to $\varphi = \psi \wedge \xi$.
6. Let $\varphi = h_1 \doteq h_2$ for index terms h_1, h_2. Then $fullsize(\mathcal{C}_n^{\varphi}) = 1 + fullsize(\mathcal{C}_n^{h_1}) + fullsize(\mathcal{C}_n^{h_2})$.
7. If $\varphi = t_1 \doteq t_2$ for number terms t_1, t_2, then the *fullsize* can be computed as for index terms.
8. If $\varphi = t_1 < t_2$ for number terms t_1, t_2, then the *fullsize* can be computed as for equality.

The *fullsize* of index terms is computed as follows. Let h be an index term.

1. Let $h = x$ for $x \in V$. Then $fullsize(\mathcal{C}_n^{h}) = 1$.
2. Let $h = f(h_1, ..., h_\ell)$ for a ℓ-ary function symbol $f \in L_a$ and index terms $h_1, ..., h_\ell$. Then $fullsize(\mathcal{C}_n^{h}) = 1 + u^\ell * (2 + \sum\limits_{1 \le i \le \ell} (fullsize(\mathcal{C}_n^{h_i}) + 2))$.
3. If $h = f(h_1, ..., h_\ell)$ for a ℓ-ary function symbol $f \in L_s$, then the *fullsize* can be computed as above.

The *fullsize* of number terms is computed as follows. Let t be a number term.

1. Let $t = c$ for $c \in \mathbb{R}$. Then $fullsize(\mathcal{C}_n^{t}) = 1$.

2. If $t = f(h_1, ..., h_\ell)$ for a ℓ-ary function symbol $f \in L_a$ and index terms $h_1, ..., h_\ell$, then the fullsize can be computed as with function symbols for index terms.

3. If $t = f(h_1, ..., h_\ell)$ for a ℓ-ary function symbol $f \in L_f$ and index terms $h_1, ..., h_\ell$, then the fullsize can be computed as with function symbols for index terms.

4. Let $t = t_1 + t_2$ or $t = t_1 \times t_2$ for number terms t_1, t_2, then $fullsize(\mathcal{C}_n^t) = 1 + fullsize(\mathcal{C}_n^{t_1}) + fullsize(\mathcal{C}_n^{t_2})$.

To get the $fullsize$ of the entire circuit for φ, we need to add 1 to the final $fullsize$, since the output gate is not considered for subformulas.

We have now shown how to compute the $fullsize$ of the circuit for φ and any of its subformulas and terms. Each individual computation can be done in constant time, and since the formula is constant, only a constant amount of those operations is required. □

Theorem 54. $FO_{\mathbb{R}}[FTIME_{\mathbb{R}}(\log(n))] + SUM_{\mathbb{R}} + PROD_{\mathbb{R}} = U_L\text{-}AC^0_{\mathbb{R}}$

Proof. $FO_{\mathbb{R}}[FTIME_{\mathbb{R}}(\log(n))] + SUM_{\mathbb{R}} + PROD_{\mathbb{R}} \subseteq U_L\text{-}AC^0_{\mathbb{R}}$:
Just as in the polynomial case, we will use the same construction as in Theorem 48 for the logarithmic case, adding only the construction for $\underset{i}{sum}$ and $\underset{i}{prod}$, which are quite natural operations for unbounded fan-in circuits. Showing the L-uniformity of the resulting circuit, however, is not as simple as it was in Theorem 50, 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 $FO_{\mathbb{R}}[FTIME_{\mathbb{R}}(\log(n))] + SUM_{\mathbb{R}} + PROD_{\mathbb{R}}$-sentence φ. To create a circuit family deciding S, define the structure of our circuity depending on φ similarly to the proof of Theorem 48. 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 ith input gate is numbered $fullsize(\mathcal{C}_n) + i$.

At the very top of the circuit, there is the output node numbered $fullsize(C_n)$. Its predecessor is then numbered $fullsize(C_n) - 1$ and structured as follows: Let the root gate of the subcircuit representing φ be numbered q.

1. Let $\varphi = \exists y \psi(y)$. Then the construction is as in Theorem 48. The *sign* gate is numbered q, the addition gate is numbered $q-1$ and the root of the ith predecessor circuit $\mathcal{C}_n^{\psi(m_1,\dots,m_k,i)}$ is numbered $q-2-(u-i)*\mathit{fullsize}(\mathcal{C}_n^{\psi(m_1,\dots,m_k,1)})$. (We can use the $\mathit{fullsize}$ of $\mathcal{C}_n^{\psi(m_1,\dots,m_k,1)}$ here for each i, because $\mathcal{C}_n^{\psi(m_1,\dots,m_k,i)}$ has the same $\mathit{fullsize}$ for each i.)

2. If $\varphi = \forall y \psi(y)$, then the construction is as in Theorem 48 and the numbering is analogous to the existential case.

3. Let $\varphi = \neg\psi$. Then the construction is as in Theorem 48, the subtraction gate is numbered q, the constant gate with value 1 is numbered $q-1$ and the root of $\mathcal{C}_n^{\psi(m_1,\dots,m_k)}$ is numbered $q-2$.

4. Let $\varphi = \psi \wedge \xi$. Then the construction is as in Theorem 48, the sign node is numbered q, the $\times$-gate is numbered $q-1$, the root of $\mathcal{C}_n^{\psi(m_1,\dots,m_k)}$ is numbered $q-2-\mathit{fullsize}(\mathcal{C}_n^{\xi(m_1,\dots,m_k)})$ and the root of $\mathcal{C}_n^{\xi(m_1,\dots,m_k)}$ is numbered $q-2$.

5. If $\varphi = \psi \vee \xi$, $\varphi = \psi \implies \xi$ or $\varphi = \psi \iff \xi$, then $\mathcal{C}_n^{\varphi(m_1,\dots,m_k)}$ and its numbering follows analogously to $\varphi = \psi \wedge \xi$.

6. Let $\varphi = h_1 \doteq h_2$ for index terms h_1, h_2. Then the construction is as in Theorem 48, the equality gate is numbered q, the root of $\mathcal{C}_n^{h_1(m_1,\dots,m_k)}$ is numbered $q-1-\mathit{fullsize}(\mathcal{C}_n^{h_2(m_1,\dots,m_k)})$ and the root of $\mathcal{C}_n^{h_1(m_1,\dots,m_k)}$ is numbered $q-1$.

7. If $\varphi = t_1 \doteq t_2$ for number terms t_1, t_2, then $\mathcal{C}_n^{\varphi(m_1,\dots,m_k)}$ is defined and numbered analogously to the case with index terms.

8. Let $\varphi = t_1 < t_2$ for number terms t_1, t_2. Then $\mathcal{C}_n^{\varphi(m_1,\dots,m_k)}$ is defined as in Theorem 48 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 $h = x$ for $x \in V$. Then the construction is as in Theorem 48 (The constant gate is numbered q).

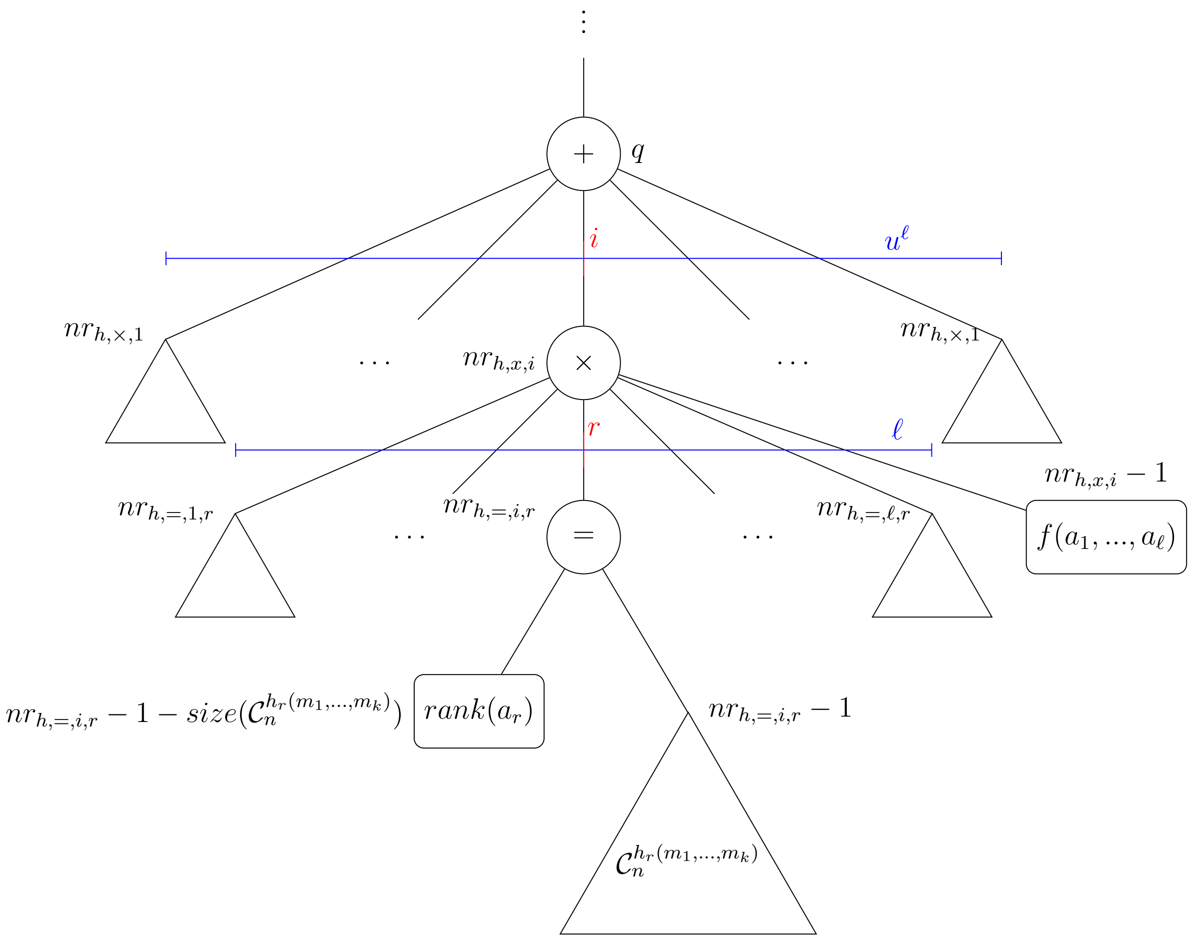

Figure 3.5: Numbering of the circuit for Theorem 54 if $h = f(h_1, ..., h_\ell)$ for $f \in L_a$.

2. Let $h = f(h_1, ..., h_\ell)$ for a ℓ-ary function symbol $f \in L_a$ and index terms $h_1, ..., h_\ell$. Then the construction is as in Theorem 48. The addition gate is numbered q, the subcircuits induced by the $\times$-gates each have the same $fullsize$, namely $s_{h,\times} := \sum_{1\le i\le \ell} (fullsize(\mathcal{C}_n^{(h_i(m_1,...,m_k))}) + 2) + 2$. Therefore the ith $\times$-gate is numbered $nr_{h,\times,i} := q - 1 - (n^\ell - i) * s_{h,\times}$. For the ith $\times$-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 rth equality gate, the numbering is as follows: the equality gate itself is numbered $nr_{h,=,i,r} := nr_{h,\times,i} - 2 - (\sum_{r+1\le j\le k} fullsize(\mathcal{C}_n^{h_j(m_1,...,m_k)}) + 2)$. The constant gate containing the value $rank(a_r)$ is numbered $nr_{h,=,i,r} - 1 - fullsize(\mathcal{C}_n^{h_r(m_1,...,m_k)})$ and the root of $\mathcal{C}_n^{h_r(m_1,...,m_k)}$ is numbered $nr_{h,=,i,r} - 1$. This numbering is visualized in Figure 3.5.

3. If $h = f(h_1, ..., h_\ell)$ for a ℓ-ary function symbol $f \in L_s$ and index terms $h_1, ..., h_\ell$, then the construction is as in Theorem 48 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 $t = c$ for $c \in \mathbb{R}$, then the construction is as in Theorem 48.
2. If $t = f(h_1, ..., h_\ell)$ for a ℓ-ary function symbol $f \in L_a$ and index terms $h_1, ..., h_\ell$, then the construction is as in Theorem 48 and the numbering is done as described in the case of index terms.
3. If $t = f(h_1, ..., h_\ell)$ for a ℓ-ary function symbol $f \in L_f$ and index terms $h_1, ..., h_\ell$, then the construction is as in Theorem 48 and the numbering is done as described in the case of index terms.
4. If $t = t_1 + t_2$ or $t = t_1 \times t_2$ for number terms t_1, t_2, then the construction is as in Theorem 48. The addition gate is numbered q, the root of $\mathcal{C}_n^{t_2 m_1,...,m_k}$ is numbered $q - 1$ and the root of $\mathcal{C}_n^{t_1(m_1,...,m_k)}$ is numbered $q - 1 - \mathit{fullsize}(\mathcal{C}_n^{t_2(m_1,...,m_k)})$.
5. If $t = \underset{i}{sum}(t_1(i))$ for a number term t_1 in which i occurs freely, then $\mathcal{C}_n^{t(m_1,...,m_k)}$ consists of an addition gate at the top, numbered q, with the root nodes of the circuits $\mathcal{C}_n^{t_1(m_1,...,m_k,i)}, 1 \le i \le u$ as its u predecessors, numbered $q - 1 - (u - i) * \mathit{fullsize}(\mathcal{C}_n^{t_1(m_1,...,m_k,i)})$, 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 $\mathcal{C}^{t_1(m_1,...,m_k)}$ and which are numbered the same way as for the case where i is free in t_1.
6. If $t = \underset{i}{prod}(t_1(i))$ 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 $\mathit{fullsize}$ 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 in Definition 17. From Definition 25, we know that we can compute $|A|$ from $n = |enc(\mathcal{D})|$ 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 $\mathit{fullsize}(\mathcal{C}_n)$ and its singular predecessor node has number $\mathit{fullsize}(\mathcal{C}_n) - 1$. Let fs_φ denote $\mathit{fullsize}(\mathcal{C}_n^{\varphi(m_1,...,m_k)})$ – which is the same as $\mathit{fullsize}(\mathcal{C}_n^{\varphi(m_1,...,m_k,i)})$ etc., since the variable assignments do not have an effect on the size of the circuit – and u, as in the proof of Theorem 48, the size of the universe of the input structure $|A|$. Now the machine works as follows:

If $v_{nr} > fullsize\mathcal{C}_n + n$, then return $(0,0,0)$.

If $v_{nr} = fullsize(\mathcal{C}_n)$ then return $(6, fullsize(\mathcal{C}_n) - 1, 0)$ if $p_{idx} = 1$ and $(6,0,0)$ otherwise.

If $v_{nr} = fullsize(\mathcal{C}_n) + i$, for $i \in \{1, ..., n\}$ 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 $\varphi = \exists y \psi(y)$.

   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 $(3, q - 2 - (u - p_{idx}) * fs_\psi, 0)$ if $p_{idx} \leq u$ and $(5,0,0)$ otherwise.

   Otherwise gate v_{nr} is contained in the subcircuit induced by the gate numbered $y = q - 2 - (\left\lceil \frac{q-1-v_{nr}}{fs_\psi} \right\rceil - 1) * fs_\psi$ where y is the smallest natural number such that $y \geq v_{nr}$ and $y = q - 2 - (u - i) * fs_\psi$ for some $i \in \{1, ..., u\}$. 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 $\varphi = \forall y \psi(y)$, then the construction is analogous to the existential case.

3. Let $\varphi = \neg\psi$.

   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 $\mathcal{C}_n^{\psi(m_1,...,m_k)}$.

4. Let $\varphi = \psi \wedge \xi$.

   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 $(4, q-2-fullsize(\mathcal{C}_n^{\xi(m_1,...,m_k)}), 0)$ if $p_{idx} = 1$, $(4, q-2, 0)$ if $p_{idx} = 2$ and $(4,0,0)$ otherwise.

   Otherwise, if $v_{nr} \leq q-2-fullsize(\mathcal{C}_n^{\xi(m_1,...,m_k)})$, store $q-2-fullsize(\mathcal{C}_n^{\xi(m_1,...,m_k)})$ in q and construct $\mathcal{C}_n^{\psi(m_1,...,m_k)}$ and otherwise store $q-2$ in q and construct $\mathcal{C}_n^{\psi(m_1,...,m_k)}$.

5. If $\varphi = \psi \vee \xi$, $\varphi = \psi \implies \xi$ or $\varphi = \psi \iff \xi$, then proceed analogously to $\varphi = \psi \wedge \xi$.

6. If $\varphi = h_1 \doteq h_2$ for index terms h_1, h_2, then proceed analogously to the Boolean connectives.

7. If $\varphi = t_1 \doteq t_2$ for number terms t_1, t_2, then proceed analogously to the Boolean connectives.

8. If $\varphi = t_1 < t_2$ 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 $h = x$ 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 $h = f(h_1, ..., h_\ell)$ for a ℓ-ary function symbol $f \in L_a$ and index terms $h_1, ..., h_\ell$.

   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 $nr_\times := \min\{nr_{h,\times,i} \mid nr_{h,\times,i} \geq v_{nr},\ 1 \leq i \leq u^\ell\}$. (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 ith 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 $(8, nr_= - 1 - \mathit{fullsize}(\mathcal{C}_n^{h_i(m_1,...,m_k)}), 0)$ if $p_{idx} = 1$, $(8, nr_= - 1, 0)$ if $p_{idx} = 2$ and $(8, 0, 0)$ otherwise.

   If $v_{nr} = nr_= - 1 - \mathit{fullsize}(\mathcal{C}_n^{h_i(m_1,...,m_k)})$ then return $(2, 0, a)$ where a is the value of the rth element of the lexicographically ith input to f.

   Otherwise store $nr_= - 1$ in q and continue with the construction of $\mathcal{C}_n^{h_i(m_1,...,m_k)}$, where i is the respective i from the definition of $nr_=$.

3. If $h = f(h_1, ..., h_\ell)$ for a ℓ-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 $t = c$ 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 $t = f(h_1, ..., h_\ell)$ for a ℓ-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 $t = f(h_1, ..., h_\ell)$ for ℓ-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 $t = t_1 + t_2$ or $t = t_1 \times t_2$ for number terms t_1, t_2, continue constructing as in the case of Boolean connectives.

5. Let $t = \underset{i}{sum}(t_1(i))$ for a number term t_1.

   If $v_{nr} = q$ then return $(3, q - 1 - (u - p_{idx}) * fullsize(\mathcal{C}_n t_1(m_1, ..., m_k, p_{idx})), 0)$ if $p_{idx} \leq u$ and $(3, 0, 0)$ otherwise.

   Otherwise store $q - 1 - (u - p_{idx}) * fullsize(\mathcal{C}_n t_1(m_1, ..., m_k, p_{idx}))$ in q and continue with the construction of $\mathcal{C}_n^{t_1(m_1,...,m_k,p_{idx})}$.

6. If $t = \underset{i}{prod}(t_1(i))$ for a number term t_1, then continue constructing as in the case of $\underset{i}{sum}$.

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 $U_L\text{-}AC^0_{\mathbb{R}}$.

$U_L\text{-}AC^0_{\mathbb{R}} \subseteq FO_{\mathbb{R}}[FTIME_{\mathbb{R}}(\log(n))] + SUM_{\mathbb{R}} + PROD_{\mathbb{R}}$:
Showing that a set $S \in U_L\text{-}AC^0_{\mathbb{R}}$ can be defined using $FO_{\mathbb{R}}[FTIME_{\mathbb{R}}(\log(n))] + SUM_{\mathbb{R}} + PROD_{\mathbb{R}}$ is done in the same way as it was done in the polynomial case (Theorem 50). 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 $t(v_1, ..., v_q)$, $c(v_1, ..., v_q)$, $in(v_1, ..., v_q, i)$ and $pred(v_1, ..., v_q)$ in logarithmic time analogous to Theorem 50, since our circuit family is L-uniform.

2. $\underset{i}{sum}$ and $\underset{i}{prod}$ are given in the specification of $FO_{\mathbb{R}}[FTIME_{\mathbb{R}}(\log(n))] + SUM_{\mathbb{R}} + PROD_{\mathbb{R}}$.

□

With the construction shown in the previous theorem we can now generalize that whenever we have a variant of $AC^0_{\mathbb{R}}$ 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 55. *For any function $f : \mathbb{N} \to \mathbb{N}$ with $f(n) \geq \log(n)$ for all n, it holds that*

$$U_f\text{-}AC^0_{\mathbb{R}} = FO_{\mathbb{R}}[FTIME_{\mathbb{R}}(f(n))] + SUM_{\mathbb{R}} + PROD_{\mathbb{R}}, \tag{3.11}$$

where U_f-$AC^0_{\mathbb{R}}$ is the class of sets decidable by circuit families, which can be constructed analogous to Definition 17 in time bounded by $\mathcal{O}(f(n))$.

Remark 56. The restriction in Corollary 55 to functions that are at least logarithmic stems solely from the time it takes to decode the size of the universe of an input structure from its encoding (see Definition 25). If an algorithm were found, which could do so in time independent from or only sublogarithmically dependent on the length of the encoding, this restriction could be loosened. This question boils down to this:

For fixed numbers $a_1, ..., a_k, k \in \mathbb{N}$, find the natural root of a polynomial $p(x) = -c + a_1 * x + a_2 * x^2 + ... + a_k * x^k$ when given $c \in \mathbb{N}$ as the input in time independent from or only sublogarithmically dependent on c.

Remark 57. For the results that we have shown in this thesis, whenever we extended our first-order logic by sets of functions or predicates as per Definition 31, we only ever needed functions $A^\ell \to \mathbb{R}$ for $\ell \in \mathbb{N}$. Therefore, at least in our case, an extension containing real valued functions can be seen as equivalent to extensions also containing predicates and functions mapping into the universe A.

4 Conclusions

4.1 Summary and Main Results

In this thesis, we started out by introducing two models of computation over the real numbers – arithmetic circuits and $\mathbb{R}$-machines – and first-order logic over the reals, similar to how they were used by Cucker and Meer [CM99]. We had to make some adaptations to account for unbounded fan-in circuits and we made some extensions to our logic in order to be able to describe those circuits. Afterwards, we showed characterizations for $AC^0_{\mathbb{R}}$, U_P-$AC^0_{\mathbb{R}}$ and U_L-$AC^0_{\mathbb{R}}$ using the logic and extensions we had introduced.

We found that $AC^0_{\mathbb{R}}$ coincides with the class of $\mathbb{R}$-structures definable by first-order sentences without restriction on the functions and relations used therein, which is in line with the classical result [Vol99]. We further discovered that we can describe variants of $AC^0_{\mathbb{R}}$ given by a time complexity uniformity criterion by extending our first-order logic with functions of that time complexity and the sum and product rule.

4.2 Outlook

The results of this thesis suggest several directions for further research.

A question which could immediately improve a result in this thesis is whether retrieving the size of the universe of an encoded $\mathbb{R}$-structure can be done time-independently of the length of that encoding, thus depending only on its signature. As already briefly discussed in Remark 56, if such an algorithm were found, Corollary 55 could immediately be improved.

Moreover, to be able to better put the results of this thesis in the context of the results presented by Cucker and Meer [CM99], it is worth looking into the relation between the bounded fan-in circuits which have immediate access to subtraction and division gates and the unbounded fan-in circuits used here. Similar analysis is of interest concerning the logics and uniformity definition.

Furthermore, a continued exploration of real-valued parallel complexity and its relation to classical complexity theory is certainly of interest. The classes $NC_{\mathbb{R}}$, $AC_{\mathbb{R}}$ and a real analogue to the classical TC pose interesting questions, such as whether their relations behave differently over the real numbers than they do in the classical case. The $AC_{\mathbb{R}}$ hierarchy could be further explored, e.g. to find a general characterization for $AC^i_{\mathbb{R}}$ and its uniform subclasses. In that context, other classical circuit problems, such as Minimum Circuit Size, also merit investigation in the real setting. These circuit theoretic questions also warrant examination in their connection to other preexisting logics, such as team logics.

Glossary

fan-in The fan-in of a gate in an arithmetic circuit is the indegree of that gate, i.e. the amount of predecessor gates it has. 5, 13, 17, 24, 27, 36, 44

functional $\mathbb{R}$-structure A functional $\mathbb{R}$-structure is a $\mathbb{R}$-structure, which does not contain predicate symbols. 22, 23

indegree The indegree of a gate of an arithmetic circuit is the number of predecessors that gate has. 6, 7

level The level of a gate in an arithmetic circuit which is a full tree is the distance of that gate to any input gate. 30

outdegree The outdegree of a gate of an arithmetic circuit is the number of successors that gate has. 6, 24, 25

Bibliography

[BCSS98] Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale. Complexity and Real Computation. Springer, 1998.

[BSS88] 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, White Plains, New York, USA, 24-26 October 1988, pages 387–397. IEEE Computer Society, 1988.

[BSS89] Lenore Blum, Mike Shub, and Steve Smale. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. Bull. Amer. Math. Soc. (N.S.), 21(1):1–46, 07 1989.

[CM99] Felipe Cucker and Klaus Meer. Logics which capture complexity classes over the reals. J. Symb. Log., 64(1):363–390, 1999.

[Cuc92] Felipe Cucker. $P_{\mathbb{R}}$!= $NC_{\mathbb{R}}$. J. Complexity, 8(3):230–238, 1992.

[Cut80] Nigel Cutland. Computability: An Introduction to Recursive Function Theory. Cambridge University Press, 1980.

[GM95] Erich Grädel and Klaus Meer. Descriptive complexity theory over the real numbers. In Frank Thomson Leighton and Allan Borodin, editors, Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, 29 May-1 June 1995, Las Vegas, Nevada, USA, pages 315–324. ACM, 1995.

[Mee00] Klaus Meer. Counting problems over the reals. Theor. Comput. Sci., 242(1-2):41–58, 2000.

[Mic89] Christian Michaux. Une remarque à propos des machines sur $\mathbb{R}$ introduites par blum, shub et smale. (a remark about the machines over $\mathbb{R}$ introduced by blum, shub and smale). Comptes Rendus de l'Académie des Sciences. Série I, 01 1989.

[Vol99] Heribert Vollmer. Introduction to Circuit Complexity - A Uniform Approach. Texts in Theoretical Computer Science. An EATCS Series. Springer, 1999.