


PAPER

Logical characterizations of algebraic circuit classes over integral domains

Timon Barlag¹, Florian Chudigiewitsch² and Sabrina A. Gaube¹

¹Leibniz Universität Hannover, Hannover, Germany and ²Universität zu Lübeck, Lübeck, Germany

Corresponding author: Timon Barlag; Email: barlag@thi.uni-hannover.de

(Received 14 April 2023; revised 28 February 2024; accepted 8 April 2024; first published online 13 May 2024)

Abstract

We present an adapted construction of algebraic circuits over the reals introduced by Cucker and Meer to arbitrary infinite integral domains and generalize the $AC_{\mathbb{R}}$ and $NC_{\mathbb{R}}$ classes for this setting. We give a theorem in the style of Immerman's theorem which shows that for these adapted formalisms, sets decided by circuits of constant depth and polynomial size are the same as sets definable by a suitable adaptation of first-order logic. Additionally, we discuss a generalization of the guarded predicative logic by Durand, Haak and Vollmer, and we show characterizations for the $AC_{\mathbb{R}}$ and $NC_{\mathbb{R}}$ hierarchy. Those generalizations apply to the Boolean AC and NC hierarchies as well. Furthermore, we introduce a formalism to be able to compare some of the aforementioned complexity classes with different underlying integral domains.

Keywords: algebraic circuits; descriptive complexity

1. Introduction

Boolean circuits as a computational model are a fundamental concept in the study of theoretical computer science. Mainly in computational complexity, Boolean circuits play a central part in the analysis of parallel algorithms and the corresponding complexity classes, enabling a finer view and providing new proof methods, especially for subpolynomial complexity classes. An obvious generalization of Boolean circuits is that instead of dealing with truth values as inputs – or the field \mathbb{Z}_2 for the algebraically minded – we consider elements from some other algebraic structure. This approach has its roots in the works of Blum, Shub and Smale, whose model of the *BSS-machine* is able to describe computations over real numbers. Following this, Blum, Shub, Smale and Cucker (Blum et al., 1998) also give a generalization to algebraic circuits over the reals.

1.1 Our contribution

In this article, we provide logical characterizations of circuit complexity classes over integral domains. In particular, we define natural extensions of the classical circuit complexity hierarchies AC^i and NC^i over arbitrary integral domains. The resulting classes are denoted as AC_R^i and NC_R^i , respectively. We adapt the framework of metafinite model theory to define various extensions of first-order logic, which capture these new complexity classes.

We establish an Immerman-style theorem stating that $FO_R = AC_R^0$ and provide a framework to establish complexity-theoretic comparisons of classes with different underlying integral domains and give examples over which integral domain the AC^0 classes are equal.



We adapt the GPR operator, which Durand, Haak and Vollmer use to provide logical characterizations of AC^1 and NC^1 (Durand et al., 2018) to logics over metafinite structures and extend it to be able to characterize the whole AC_R and NC_R hierarchies.

Finally, we define a formalism suitable for comparing complexity classes with different underlying integral domains and we show that a hierarchy of sets of complexity classes emerges, such that each set is able to “simulate” the complexity classes from the sets lower in the hierarchy.

1.2 Related work

Another model of computation that is commonly known under the name “algebraic circuit” are Valiant circuits (Valiant, 1979), which are the foundational model in the emerging subfields of algebraic and geometric complexity theory. They differ from the model we analyse in this work in the way that we use $<$ gates, which are not available in the Valiant setting. This difference is of complexity-theoretical significance, since for our model, we have that $NC_{\mathbb{R}}^i \subsetneq NC_{\mathbb{R}}^{i+1}$ (Cucker, 1992) but we have that $VNC^2 = VNC^3 = \dots = VP$ (Bürgisser, 2013) in the Valiant setting for every field, in particular over the reals.

1.3 Outline of the paper

We start with an overview of some central concepts from algebra and circuit complexity, which are needed for the definition of our model. We then establish our model of algebraic circuits for arbitrary integral domains and the analogous complexity classes induced by them in analogy to the Boolean case. Afterwards, we give a logical characterization of the presented circuit classes inspired by classical descriptive complexity. However, since our models now have an infinite component, we build on the foundations of *metafinite model theory*.

We then go on to define first-order logic over R and show that, like in the classical case, we have that $AC_R^0 = FO_R[Arb_R] + SUM_R + PROD_R$.

In Section 4, we give logical characterizations of AC^i and NC^i . The tool of our choice is an adaptation of the guarded predicative logic of Durand et al. (2018).

In Section 5, we discuss connections between AC_R^0 classes over different integral domains.

2. Preliminaries

First, we discuss the theoretical background of this paper. We give the basic definitions and remarks on the notation used in this paper. We denote the set of natural numbers with 0 as \mathbb{N} and the set of natural numbers without 0 as $\mathbb{N}_{>0}$.

Notation 1. *In this paper, we will generally use overlined letters (e. g. \bar{x}) to denote tuples.*

As the name suggests, algebraic circuit classes make use of concepts originating from abstract algebra. We assume the reader has basic knowledge of algebra, especially rings, integral domains and fields, polynomials and adjoining elements to rings. For an introduction to abstract algebra, the reader may refer to the books by Aluffi (2009) or Lang (2002).

Our considered rings are always assumed to be a commutative ring with unit. Throughout the paper, whenever not otherwise specified, we use R to denote an infinite integral domain.

Remark 2. In particular, we require that for every $r \in R$ the equations

$$\begin{aligned} r \times 0 &= 0 \\ 0 \times r &= 0 \end{aligned}$$

hold.

Example 3. Popular examples of rings include $\mathbb{Z}, \mathbb{Q}, \mathbb{R}$ and \mathbb{C} , as well as sets with adjoined elements like $\mathbb{Z}[j], \mathbb{Z}[\sqrt{-1}], \mathbb{Z}[\sqrt{p}, \sqrt{q}, \dots], \mathbb{R}[j_k], \mathbb{Z}[j_k], \dots$, where $k \in \mathbb{N}, p \neq q$ are primes and j_k denotes the k th root of -1 , that is $i_k^k = -1$. Alternatively, we can adjoin ℓ algebraic independent prime root elements with $2^\ell = k$ and get an analogous construction.

Remark 4. The denotation of $\mathbb{R}[j_k]$ resp. $\mathbb{Z}[j_k]$ is not unique but the constructions of the circuit over the underlying rings are analogous except for the placeholders for the adjoined numbers. For example, \mathbb{Z}^4 can denote $\mathbb{Z}[j_4] = \mathbb{Z}[j_4, j_4^2, j_4^3]$ or $\mathbb{Z}[\sqrt{2}, \sqrt{5}] = \mathbb{Z}[\sqrt{2}, \sqrt{5}, \sqrt{10}]$ or many other rings. Since we only focus on the structure of the tuples, that is the coefficients of adjoin elements, or later the underlying circuits, our short notation for \mathbb{Z}^k for arbitrary $k > 1$ is not unique and may differ, for example \mathbb{Z}^2 may denote $\mathbb{Z}[j]$ or may denote \mathbb{Q} . In the context of the placeholder notation, the arithmetic of the specific ring must be clear, if it is important.

We need some ordering $<$ on our integral domain R . In some cases, like \mathbb{R} and \mathbb{Z} , we have some natural ordering that we want to use. In other cases, like \mathbb{C} , we have to construct some ordering. This ordering does not have to be closed under multiplication, and we only have to distinguish different numbers from each other. So an ordering on tuples $(z_1 = a_1 + b_1i, z_2 = a_2 + b_2i \in \mathbb{C} : (a_1, b_1) <_{\mathbb{C}} (a_2, b_2) \iff a_1 <_{\mathbb{R}} a_2 \text{ or } a_1 = a_2 \text{ and } b_1 <_{\mathbb{R}} b_2)$ is possible.

Definition 5. A strict total order on a set S is a binary relation $<$ on S which is irreflexive, transitive and total.

Example 6. For some field \mathbb{F}_{p^k} for some prime p and a natural number k , we write the finitely many elements in a list and then use the lexicographical on this list. For example, let $R = \mathbb{Z}_3$. Then, we define $<_{\mathbb{Z}_3}$ as $0 < 1 < 2$.

There are multiple possibilities for such a $<$ over the field of complex numbers. Let $z = a + bj \in \mathbb{C}$ and let $<_1$ be the lexicographic order on pairs $(\sqrt{a^2 + b^2}, a)$. Furthermore, let $<_2$ be the lexicographic order on pairs of the form (a, b) . Then, both variants are possible since both distinguish different complex numbers.

Definition 7. A strict total order $<$ over a ring R induces a sign function as follows:

$$\text{sign}_{(R, <)}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise.} \end{cases}$$

In the following, unless explicitly otherwise specified, the symbol R denotes an infinite integral domain with a strict total order $<$ on R . Most of the rings we consider have a natural ordering. In this case, we omit the ordering symbol.

2.1 Algebraic circuits over R

As this work is a generalization of the well established Boolean circuits, some background in circuit complexity is assumed. Standard literature which introduces this topic is the book by Vollmer (1999). The generalization to algebraic circuits over real numbers were first introduced by Cucker

(1992). In analogy to them, Barlag and Vollmer defined an unbounded fan-in version of algebraic circuits (Barlag and Vollmer, 2021).

Definition 8. We define an algebraic circuit C over an integral domain R with a strict total order $<$, or R circuit for short, as a directed acyclic graph. It has gates of the following types:

Input nodes have indegree 0 and contain the respective input values of the circuit.

Constant nodes have indegree 0 and are labelled with elements of R

Arithmetic nodes have an arbitrary indegree ≥ 1 , bounded by the number of nodes in the circuit and are labelled with either $+$ or \times .

Comparison ($<$) nodes have indegree 2.

Output nodes have indegree 1 and contain the output value after the computation of the circuit.

Nodes cannot be predecessors of the same node more than once, and thus, the outdegree of nodes in these algebraic circuits is bounded by the number of gates in the circuit.

During the computation of an algebraic circuit, the arithmetic gates compute their respective functions with the values of their predecessor gates being taken as the function arguments and the comparison gates compute the characteristic function of $<$ in the same way. The values of the output gates at the end of the computation are the result of the computation.

In contrast to the classical setting, where we consider words over an alphabet Σ as inputs, and where languages are thus defined to be subsets of the Kleene closure of the alphabet (in symbols, $L \subseteq \Sigma^*$), we consider vectors of integral domain elements as input. In analogy to Σ^* , we denote for an integral domain R :

$$R^* = \bigcup_{k \in \mathbb{N}_0} R^k$$

With $|x|$, we denote the length of x , i. e., if $x \in R^k$ then $|x| = k$.

Remark 9. We will use the term algebraic circuit to describe circuits in the sense of Blum et al. (1998) rather than arithmetic circuits as for example in Barlag and Vollmer (2021) to distinguish them from arithmetic circuits in the sense of Valiant, see for example (Bürgisser et al., 1997). Valiant circuits are essentially algebraic circuits without sign or comparison gates (Blum et al., 1998, page 350).

Remark 10. In the special case $R = \mathbb{Z}_2$, the definition above yields exactly the Boolean circuits.

Definition 11. We call the number of gates in a circuit the size of the circuit and the longest path from an input gate to an output gate the depth of the circuit.

Remark 12. Unlike the way algebraic circuits with unbounded fan-in gates were introduced previously (Barlag and Vollmer, 2021), algebraic circuits in this context have comparison gates instead of sign gates. This stems from the fact that when dealing with real or complex numbers, we can construct a sign function from $<$ via Definition 7 and the order relation from the sign function via

$$x < y \iff \text{sign}(\text{sign}(y - x) \cdot (2 + \text{sign}(x - y))) = 1.$$

If we consider finite integral domains, however, suddenly it becomes less clear how to construct the $<$ relation from the sign function, while the other way around still works by Definition 7.

Given that we define circuits and logic fragments relative to an ordering, it is natural for both to have access to this ordering. Therefore, we choose to use $<$ gates rather than sign gates. So in the following, all usages of sign are implicit uses of the order relation as by Definition 7.

In the cases which are considered in other literature (\mathbb{R} or \mathbb{F}_2), this has no complexity-theoretic impact, since in the emulation of one formalism using the other, we get a linear overhead in the size and constant overhead in the depth of the circuit.

In order to define complexity classes with respect to algebraic circuits, we have to define the function calculated by such a circuit and define the term of circuit families.

Definition 13. The (n ary) function $f_C: R^n \rightarrow R^m$ computed by an R circuit C (with n input gates and m output gates) is defined by

$$f_C(x_1, \dots, x_n) = (y_1, \dots, y_m),$$

where y_1, \dots, y_m are the values of the output gates of C , when given x_1, \dots, x_n as its inputs.

Definition 14. A family of R circuits $\mathcal{C} = (C_n)_{n \in \mathbb{N}}$ is a sequence of circuits which contains one circuit for every input length $n \in \mathbb{N}$. The function $f_{\mathcal{C}}: R^* \rightarrow R^*$ computed by a circuit family \mathcal{C} is defined by

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

The size (resp. depth) of a circuit family $(C_n)_{n \in \mathbb{N}}$ is defined as a function mapping $n \in \mathbb{N}$ to the size (resp. depth) of C_n .

Analogously to the classical case, we say that a set $S \subseteq R^*$ can be decided by a circuit family \mathcal{C} , if \mathcal{C} can compute the characteristic function of S .

Definition 15. Let $f_1, f_2: \mathbb{N} \rightarrow \mathbb{N}$ be two functions. We then write $UnbSizeDepth_R(f_1, f_2)$ to denote the class of sets decidable by R circuit families of size $\mathcal{O}(f_1(n))$ and depth $\mathcal{O}(f_2(n))$. We write $SizeDepth_R(f_1, f_2)$ to denote the class of sets decidable by R circuit families of size $\mathcal{O}(f_1(n))$ and depth $\mathcal{O}(f_2(n))$, where each arithmetic gate has indegree bounded by 2 (we call this bounded fan-in).

Definition 16. $AC_R^i := UnbSizeDepth_R(n^{\mathcal{O}(1)}, (\log_2 n)^i)$

Definition 17. $NC_R^i := SizeDepth_R(n^{\mathcal{O}(1)}, (\log_2 n)^i)$

Remark 18. The circuit families we have just introduced do not have any restrictions on the difficulty of constructing any individual circuit given the input length. If it is important to know how hard obtaining a particular circuit is, one can consider so-called uniform circuit families. These families require their circuits to meet restrictions on the difficulties of obtaining them. For more information on uniformity, cf. (Vollmer, 1999).

Uniformity criteria can be defined for algebraic circuit classes in a similar way. See for example (Blum et al., 1998, Section 18.5).

2.2 Structures and first-order logic over integral domains

As we want to characterize circuit complexity classes with logical fragments, this work falls broadly under the umbrella of finite model theory and, in particular, descriptive complexity. Foundational knowledge of these topics is assumed and can be found in the books by Grädel et al. (2007);

Immerman (1999); and Libkin (2004). Traditionally, descriptive complexity is viewed as a sub-discipline of finite model theory. In our setting, however, we want to (carefully) introduce infinite structures to our reasoning. For this, we use *metafinite model theory*, an extension of the finite model formalism first introduced by (Grädel and Gurevich, 1998). A short introduction to metafinite model theory is also featured in the book by Grädel et al. 2007, page 210). The approach was taken by Grädel and Meer (1995) to describe some essential complexity classes over real numbers. These descriptions were later extended by Cucker and Meer (1999), where, among other things, the NC hierarchy over real numbers was defined. For their logical characterizations, these papers introduce a so-called first-order logic with arithmetics, which we will adapt to be used in our setting.

To make proofs easier, we will make use of the well-known trick that any predicate can be emulated by its characteristic function. Thus we only consider predicates, when convenient. We can furthermore emulate constants in the usual way by 0 ary functions.

Definition 19. Let L_s, L_f be finite vocabularies which only contain function symbols. An R structure of signature $\sigma = (L_s, L_f)$ is a pair $\mathcal{D} = (\mathcal{A}, F)$ where

1. \mathcal{A} is a finite structure of vocabulary L_s which we call the skeleton of \mathcal{D} whose universe A we will refer to as the universe of \mathcal{D} and whose cardinality we will refer to by $|A|$
2. and F is a finite set which contains functions of the form $X: A^k \rightarrow R$ for $k \in \mathbb{N}$ which interpret the function symbols in L_f .

We will use $STRUC_R(\sigma)$ to refer to the set of all R structures of signature σ , and we will assume that for any signature $\sigma = (L_s, L_f)$, the symbols in L_s and L_f are ordered.

Remark 20. In this paper, we only consider ranked structures, that is structures, in which the skeleton is ordered.

It often comes in handy to be able to encode an R structure \mathcal{D} over a signature σ as an element of R^* . We do so by simply concatenating the function values of all functions of \mathcal{D} in lexicographical order with respect to the function arguments.

Example 21. Let $A = \{\diamond, \spadesuit\}$, $f_1 = \{\diamond \mapsto 1, \spadesuit \mapsto 0\}$, $f_2 = \{\diamond \mapsto \pi, \spadesuit \mapsto 42\}$ and let $\mathcal{D} = (\mathcal{A}, F)$ be an R structure with universe A , \diamond being ranked below \spadesuit and $F = \{f_1, f_2\}$. Then, \mathcal{D} gets encoded as $(1, 0, \pi, 42)$.

Definition 22 (First-order logic over R). The language of first-order logic over an integral domain R contains for each signature $\sigma = (L_s, L_f)$ a set of formulae and terms. The terms are divided into index terms which take values in universe of the skeleton and number terms which take values in 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 element of R is a number term.
3. For index terms h_1, \dots, h_k and a k ary function symbol $X \in L_f$, $X(h_1, \dots, h_k)$ 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 $\text{sign}(t_1)$.

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

The set FO_R is the smallest set which contains the closure of atomic formulae under the logical connectives $\{\wedge, \vee, \neg, \rightarrow, \leftrightarrow\}$ and quantification $\exists v\psi$ and $\forall v\psi$ where v ranges over \mathcal{A} .

For better readability, we will use inequalities of the form $x \leq y$ and the extensions of equalities $\bar{x} = \bar{y}$ and the inequalities $\bar{x} < \bar{y}$ and $\bar{x} \leq \bar{y}$ to tuples throughout this paper. Note that these extensions are easily definable from $=$ and $<$ in first-order logic.

Remark 23. We call any element of R a number term even though some integral domains can contain elements like ζ , \aleph or \otimes which are not numbers.

Equivalence of FO_R formulae and sets defined by FO_R formulae are done in the usual way, that is a formula φ defines a set S if and only if the elements of S are exactly the encodings of R structures under which φ holds and two such formulae are said to be equivalent if and only if they define the same set.

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 define a further rule, which we call the *maximization rule*. This one is, however, already definable in FO_R , and we thus do not gain any expressive power by using it. We will use it to show that we can represent characteristic functions in FO_R .

Definition 24 (Sum, product and maximization rule). Let t be a number term in which the variables \bar{x} and the variables \bar{w} occur freely and let A denote the universe of the given input structure. Then,

$$\mathbf{sum}_{\bar{x}}(t(\bar{x}, \bar{w}))$$

is also a number term which is interpreted as $\sum_{\bar{x} \in A^{|\bar{x}|}} t(\bar{x}, \bar{w})$. The number terms $\mathbf{prod}_{\bar{x}}(t(\bar{x}, \bar{w}))$ and $\mathbf{max}_{\bar{x}}(t(\bar{x}, \bar{w}))$ are defined analogously.

We call these operators aggregators, and for any formula φ containing aggregators of the above form, the variables in \bar{x} are considered bound in φ .

Example 25. Let $\sigma = (\{\}, \{f_E^2\})$ be the signature of weighted graphs, that is L_s is empty and L_f contains the single function symbol f_E which is interpreted such that $f_E(x, y)$ gives the weight of the edge from x to y or 0 if there is none. Let \mathcal{G} be a (graph) structure over σ . Then, the following $FO_R + \text{SUM}_R$ sentence φ states that there is a node in the skeleton of \mathcal{G} , for which the sum of its outgoing edges is more than double the sum of the outgoing edges of any other node.

$$\varphi := \exists x \forall y (x \neq y \rightarrow \mathbf{sum}_a(f_E(x, a)) > 2 \times \mathbf{sum}_b(f_E(y, b)))$$

Observation 26. $FO_R = FO_R + \text{MAX}_R$.

Proof. An occurrence of $\mathbf{max}_i(F(i))$ essentially assures that there exists an element x , such that for all elements y , $F(x) \geq F(y)$ and takes the value of $F(x)$. Clearly, this can be defined in first order logic by a formula of the form $\forall x \exists y F(x) \geq F(y)$. \square

Furthermore, any characteristic function of a logical formula can be described in FO_R . The proof for this runs analogously to that of Cucker and Meer 1999, Proposition 2), since this proof does not make any use of special properties of the reals.

3. $AC_R^0 = FO_R$

In this section, we present a proof that FO_R captures the class AC_R^0 . The proof idea is similar to the proof of the established result by Immerman which characterizes AC^0 via FO.

When using logics to check, whether a given R tuple would be accepted by a R circuit, one needs to think about how this R tuple would be interpreted as an R structure \mathcal{A} . This can be done by interpreting it as the set of the circuit's input gates along with a single unary function $f_{\text{element}} : A \rightarrow R$ mapping the i th input gate to the i th input of the circuit. We call this kind of structure a *circuit structure*.

In the following, we would like to extend FO_R by additional functions and relations that are not given in the input structure. To that end, we make a small addition to Definition 19 where we defined R structures. Whenever we talk about R structures over a signature (L_s, L_f) , we now also consider structures over signatures of the form (L_s, L_f, L_a) . The additional (also ordered) vocabulary L_a does not have any effect on the R structure, but it contains function symbols, which can be used in a logical formula with this signature. This means that any R structure of signature (L_s, L_f) is also an R structure of signature (L_s, L_f, L_a) for any vocabulary L_a . The symbols in L_a stand for functions that we will use to extend the expressive power of FO_R to capture various complexity classes.

Definition 27. Let F be a set of finitary functions. We will write $\text{FO}_R[F]$ to denote the class of sets that can be defined by FO_R sentences which can make use of the functions in F in addition to what they are given in their structure.

Formally, this means that $\text{FO}_R[F]$ describes exactly those sets $S \subseteq R^*$ for which there exists an FO_R sentence φ over a signature $\sigma = (L_s, L_f, L_a)$ such that for each length n , there is an interpretation I_n interpreting the symbols in L_a as elements of F such that for all R^* tuples s of length n it holds that $s \in S$ if and only if s encodes an R structure over (L_s, L_f, L_a) which models φ when using I_n .

Example 28. Let us take as an example the scenario where we are given a graph as a structure but want to make use of additional functions to interpret that graph as a circuit, by having those functions determine the gate types of the nodes in the graph. We will use the signature $\sigma = (\{E^2\}, \{\}, \{f_{\text{out}}^1, f_{\text{add}}^1, f_{\text{in}}^1\})$, and we will use a set of functions $F = \{F_{\text{out},1}, F_{\text{out},2}, \dots, F_{\text{add},1}, F_{\text{add},2}, \dots, F_{\text{in},1}, F_{\text{in},2}, \dots\}$, where each function is a characteristic function which maps the nodes of the graph to 1, if the gate type matches and 0, otherwise. Hence, if a node v is an addition gate, we have $F_{\text{add},1}(v) = 1$ and $F_{\text{out},1}(v) = 0$. The number in the subscript of the functions of F refers to the size of the encoding of the structure, for which the functions should be used. For example, $F_{\text{add},4}$ is used for structures that are encoded as elements of R^4 . Now suppose we are interested in those circuits where there exists an input gate that has an addition gate as a successor. A fitting $\text{FO}_R[F]$ sentence would be

$$\varphi = \exists x \exists y f_{\text{in}}(x) = 1 \wedge f_{\text{add}}(y) = 1 \wedge E(x, y)$$

The reason why φ works is that for each $n \in \mathbb{N}$, there exists an interpretation I_n which maps the symbols used in φ to functions in F . This interpretation just maps f_{in} to $F_{\text{in},n}$ and f_{add} to $F_{\text{add},n}$.

If we are now given a graph structure of a graph such as the first one in Figure 1, encoded as its adjacency matrix

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

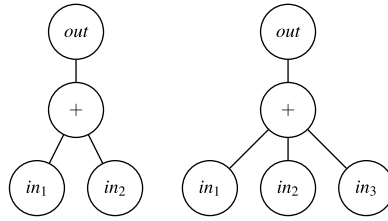


Figure 1. Example graphs that satisfy the imposed conditions of Example 28.

which would amount to the R^{16} tuple $(0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0)$, then our interpretation I_{16} would map f_{in} to $F_{in,16}$ and f_{add} to $F_{add,16}$. This means that the set of encodings of circuits which have gate types according to F that contain at least one input gate followed by an addition gate is definable in $FO_R[F]$. This is because φ is a $FO_R[F]$ sentence over σ such that for each n , the interpretation I_n interprets the symbols f_{in} and f_{add} as elements of F , such that for all $s \in R^*$ it holds that s is an encoding of such a circuit if and only if the graph structure encoded by s satisfies φ .

Definition 29. We write Arb_R to denote the set of all functions $f : R^k \rightarrow R$, where $k \in \mathbb{N}$.

Theorem 30. Let R be an infinite integral domain. Then, $AC_R^0 = FO_R[Arb_R] + SUM_R + PROD_R$.

Proof. The proof for this theorem works similarly to the construction for $FO_{\mathbb{R}}[Arb_{\mathbb{R}}] + SUM_{\mathbb{R}} + PROD_{\mathbb{R}} = AC_{\mathbb{R}}^0$ (Barlag and Vollmer, 2021), since this construction does not make use of any special properties of the real numbers.

The basic idea for this proof is that we first show that for any $FO_R[Arb_R] + SUM_R + PROD_R$ sentence φ , we can construct a circuit family which accepts its input if and only if the input encodes an R structure that satisfies φ . This is basically done by mimicking the behaviour of the logical operators of φ using the available gate types and evaluating the formula level by level. A universal quantifier as in $\forall x\varphi(x)$, for instance, is implemented by using a sign gate (obtained from $<$ as per Definition 7) on top of a multiplication gate, which has the circuits for $\varphi(a)$ for each a in the skeleton of the encoded structure as its predecessors.

To translate the semantics of FO_R into a circuit, we can mostly use the same translations as in the proof for the real case, as for the most part only properties of integral domains are used. We need ring properties for most of these translations and in particular for universal quantifiers, we also need commutativity of multiplication and no zero dividers, hence we require integral domains.

We do need to change the translation for existential quantifiers and \vee , however. In the real case, the circuit for $\exists x\varphi(x)$ is constructed similarly to the universal case with a sign gate followed by an addition gate with the circuits for $\varphi(a)$ for each a in the skeleton as its predecessors. Since there are infinite integral domains with characteristic greater than 0, that is where adding a positive amount of 1 elements can yield 0, this would not always produce the desired result. However, we can overcome this easily by translating $\exists x\varphi(x)$ to $\neg\forall x\neg\varphi(x)$ and $x \vee y$ to $\neg(\neg x \wedge \neg y)$, as negation, universal quantifiers and \wedge are constructible with integral domain properties.

For the converse inclusion, a number term $val_d(g)$ is created which, when given a circuit family $(C_n)_{n \in \mathbb{N}}$, evaluates to the value of gate g if g is on the d th level of the respective circuit C_n . For this purpose, functions encoding the structure of these circuits are given by the Arb_R extension of the logic. These functions provide information about the gate type of each gate, their constant values if they are constant gates, their index if they are input gates and the edge relation of the circuit.

For this construction, with integral domain properties no changes need to be made to the formula in the real setting. \square

4. Algebraic Circuits and Guarded Functional Recursion

In this section, we generalize the logical characterizations of NC^1 and AC^1 by Durand et al. (2018) to the respective complexity classes over integral domains NC_R^1 and AC_R^1 . We furthermore extend this method to capture the entire NC_R and AC_R hierarchies.

In their paper, Durand, Haak and Vollmer use what they call *guarded predicative recursion* to extend first-order logic in order to capture the logarithmic depth circuit complexity classes NC^1 and AC^1 . This essentially amounts to a recursion operator, which halves a tuple of variables (in their numerical interpretation) which is handed down in each recursion step. This ensures that the total recursion depth is at most logarithmic. The halving of the variable tuple is performed by using the fact that addition is expressible in FO if BIT and $<$ are expressible (Immerman, 1999) in the following way

$$x \leq y/2 \iff x + x \leq y.$$

Note that we do not define the formula of the halving to be equality, since this is not possible for odd numbers. However, this is not an issue since we only want to bound the worst case recursion depth. In order to capture classes of polylogarithmic depth, we would like to find a suitable factor to replace $\frac{1}{2}$ with, so that the recursion process has polylogarithmic depth as well. As it turns out, for any $i \in \mathbb{N}$, we can assure a recursion depth of $\mathcal{O}((\log_2 n)^i)$ by using the factor $2^{-\frac{\log_2 n}{(\log_2 n)^i}}$.

Observation 31. Any number $n \in \mathbb{N}$ can be multiplied by the factor $2^{-\frac{\log_2 n}{(\log_2 n)^i}}$ exactly $(\log_2 n)^i$ times, before reaching 1.

A more general version of this observation can be found in Lemma 58 in the appendix. Unfortunately, while it is simple to divide by 2 when the BIT predicate is available, it is not at all clear if multiplying by a factor such as $2^{-\frac{\log_2 n}{(\log_2 n)^i}}$ can be done in first-order logic.

We can, however, make use of the ability to divide by 2 in order to achieve polylogarithmic recursion depth, by instead of dividing a number, essentially dividing the digits of a base n number individually and carrying over once 0 is reached.

Let us take for example the base 5 number 444. The previously mentioned process is illustrated in Table 1. The table is supposed to be read from top to bottom and then from left to right.

We divide the digits of 444 from the least to most the significant digit until 0 is reached. So, the first step is dividing the rightmost digit of 444 by 2, getting from 444 to 442. After two more divisions of that kind, we reach 0 and in the subsequent step we reset the rightmost digit and divide the second digit once. This works in a similar way to counting down, where instead of taking away 1 in each step, we divide by 2 and carry over in an analogous way. Notably, reaching 000 from 444 takes $63 = (\lfloor \log_2 5 \rfloor + 2)^3 - 1$ steps. This is no coincidence: It can easily be shown that for any base n number of i digits, this sort of process reaches the smallest possible element after less than $(\lfloor \log_2 n - 1 \rfloor + 2)^i - 1$ steps.

Since we wish to logically characterize languages decided by circuit families, it is useful to briefly talk about representation of numbers. In descriptive complexity, tuples of elements from a finite, ordered domain A are often associated with numbers. This is frequently done by interpreting the tuple as a base $|A|$ number with each element of the tuple representing its position in the ordering of A .

Table 1. Illustration of digit-wise division until 0 of the base 5 number 444 which takes $63 = (\lceil \log_2 5 \rceil + 2)^3 - 1$ steps

444	244	144	044
442	242	142	042
441	241	141	041
440	240	140	040
424	224	124	024
422	222	122	022
421	221	121	021
420	220	120	020
414	214	114	014
412	212	112	012
411	211	111	011
410	210	110	010
404	204	104	004
402	202	102	002
401	201	101	001
400	200	100	000

Example 32. For example, let $D = \{a, b, c, d\}$ be ordered such that $a < b < c < d$. Then, the tuple $(b, d, c, a) = (1, 3, 2, 0)$ would be interpreted as the base 4 number 1320, which would correspond to 60 in decimal.

Whenever we talk about the *numerical interpretation* of a tuple, we refer to this sort of interpretation. With this interpretation in mind, we now want to define a BIT predicate, which is needed to express divisions in our logic. In our definition, we assume that the most significant bit of j 's binary representation is the bit at index 1:

Definition 33. For any ranked structure with universe D , let the relation $\text{BIT}^2 \subseteq D^* \times D^*$ be defined as follows:

$$\text{BIT} := \{(i, j) \mid \text{when } i \text{ and } j \text{ are taken as their numerical interpretations, the } i\text{th bit of the binary representation of } j \text{ is } 1, i, j \in D^*\}$$

Note that the case $D^* = \mathbb{N}$ yields the classical BIT predicate. With the BIT predicate and an order relation, we are now able to express division by 2 in first-order logic. We use the fact that whenever BIT and an order are available, we can express multiplication and addition of numerical interpretations of tuples (Immerman, 1999). This result was shown for plain first-order logic, and since our two-sorted first-order logic FO_R is equivalent to first-order logic if the secondary component is ignored, we can apply it here as well.

We express division by 2 as follows:

$$\bar{x} \leq \bar{y}/2 \iff \exists \bar{z} \bar{x} + \bar{x} = \bar{z} \wedge \bar{z} \leq \bar{y}$$

Note again that since the numerical interpretations of tuples are natural numbers, expressing $\bar{x} \leq \bar{y}/2$ is really as good as we can do, since $\bar{x} = \bar{y}/2$ would not work for odd numbers.

Next, we will turn to defining the recursion operator which we have alluded to in the beginning of this section. First, we need a little bit of additional notation, that is *relativized aggregators*.

A relativization of an aggregator is a formula restricting which elements are considered for the aggregator.

Notation 34. For a number term t and an FO_R formula φ , we write

$$\mathbf{max}_{\bar{x}}.(\varphi(\bar{x}))t(\bar{x})$$

as a shorthand for

$$\mathbf{max}_{\bar{x}}(\chi[\varphi(\bar{x})] \times t(\bar{x})).$$

Analogously, we write

$$\mathbf{sum}_{\bar{x}}.(\varphi(\bar{x}))t(\bar{x})$$

for

$$\mathbf{sum}_{\bar{x}}(\chi[\varphi(\bar{x})] \times t(\bar{x}))$$

and

$$\mathbf{prod}_{\bar{x}}.(\varphi(\bar{x}))t(\bar{x})$$

as a shorthand for

$$\mathbf{prod}_{\bar{x}}(\chi[\varphi(\bar{x})] \times t(\bar{x}) + \chi[\neg\varphi(\bar{x})] \times 1).$$

In all these cases, we say that the term $t(\bar{x})$ is in the scope of the respective aggregator. For example in the first case, $t(\bar{x})$ is in the scope of the relativized aggregator $\mathbf{max}_{\bar{x}}.(\varphi(\bar{x}))$.

We now define the GFR_R^i operator and logics extended by GFR_R^i of the form $\mathcal{L} + \text{GFR}_R^i$. The idea is to mimic the behaviour demonstrated in Table 1.

Definition 35 (GFR_R^i). Let F be a set of functions such that BIT is definable in $\text{FO}_R[F]$ and let \mathcal{L} be $\text{FO}_R[F]$ or a logic obtained by extending $\text{FO}_R[F]$ with some construction rules (such as the sum or the product rule as per Definition 24).

For $i \geq 0$, the set of $\mathcal{L} + \text{GFR}_R^i$ formulae over $\sigma = (L_s, L_f, L_a)$ is the set of formulae by the grammar for \mathcal{L} over σ extended by the rule

$$\varphi := [f(\bar{x}, \bar{y}_1, \dots, \bar{y}_i, \bar{y}_{i+1}) \equiv t(\bar{x}, \bar{y}_1, \dots, \bar{y}_i, \bar{y}_{i+1}, f)]\psi(f),$$

where ψ is an \mathcal{L} formula, f is a function symbol, and t is an \mathcal{L} number term with free variables $\bar{x}, \bar{y}_1, \dots, \bar{y}_i, \bar{y}_{i+1}$ such that

1. all \bar{y}_j for $1 \leq j \leq i$ contain the same (positive) number of variables,
2. f only occurs in the form $f(\bar{a}, \bar{z}_1, \dots, \bar{z}_i, \bar{z}_{i+1})$, where $\bar{z}_1, \dots, \bar{z}_i, \bar{z}_{i+1}$ are in the scope of a guarded aggregation

$$A_{\bar{z}_1, \dots, \bar{z}_i, \bar{z}_{i+1}} \cdot \left(\bigvee_{j=1}^i \left(\bar{z}_j \leq \bar{y}_j/2 \wedge \bigwedge_{k=1}^{j-1} \bar{z}_k \leq \bar{y}_k \right) \wedge \xi(\bar{y}_1, \dots, \bar{y}_i, \bar{y}_{i+1}, \bar{z}_1, \dots, \bar{z}_i, \bar{z}_{i+1}) \right)$$

with $A \in \{\mathbf{max}, \mathbf{sum}, \mathbf{prod}\}$, $\xi \in \mathcal{L}$ and ξ not containing any symbols of L_s or L_f and

3. f never occurs in the scope of any aggregation (or quantification) not guarded in this way.

The function symbol f is considered bound in

$$\varphi := [f(\bar{x}, \bar{y}_1, \dots, \bar{y}_i, \bar{y}_{i+1}) \equiv t(\bar{x}, \bar{y}_1, \dots, \bar{y}_i, \bar{y}_{i+1}, f)]\psi(f).$$

The semantics for the GFR_R^i operator are defined as follows: Let φ be an $\mathcal{L} + \text{GFR}_R^i$ formula over σ with the single GFR_R^i occurrence

$$[f(\bar{x}, \bar{y}_1, \dots, \bar{y}_i, \bar{y}_{i+1}) \equiv t(\bar{x}, \bar{y}_1, \dots, \bar{y}_i, \bar{y}_{i+1}, f)]\psi(\bar{z}, f)$$

and let $\mathcal{D} \in \text{STRUC}_R(\sigma)$. Then, the operator which is applied to the formula ψ , namely $[f(\bar{x}, \bar{y}_1, \dots, \bar{y}_i, \bar{y}_{i+1}) \equiv t(\bar{x}, \bar{y}_1, \dots, \bar{y}_i, \bar{y}_{i+1}, f)]$, defines the interpretation of f in ψ in the following way: For all tuples $\bar{a}, \bar{b}_1, \dots, \bar{b}_i, \bar{b}_{i+1}$ of elements of the universe D of \mathcal{D} with the same arities as $\bar{x}, \bar{y}_1, \dots, \bar{y}_i, \bar{y}_{i+1}$, respectively,

$$f(\bar{a}, \bar{b}_1, \dots, \bar{b}_i, \bar{b}_{i+1}) = t(\bar{a}, \bar{b}_1, \dots, \bar{b}_i, \bar{b}_{i+1}, f).$$

This means that the formula $[f(\bar{x}, \bar{y}_1, \dots, \bar{y}_i, \bar{y}_{i+1}) \equiv t(\bar{x}, \bar{y}_1, \dots, \bar{y}_i, \bar{y}_{i+1}, f)]\psi(\bar{c}, f)$ holds for a tuple $\bar{c} \in D^{|\bar{c}|}$ with the same arity as \bar{z} if and only if $\mathcal{D} \models \psi(\bar{c}, f_1)$ where f_1 is the interpretation of f as defined by the GFR_R^i operator. Semantics of formulae with several GFR_R^i operators are defined analogously.

Note that the $(i + 1)$ th tuple does not get restricted by the guarded aggregation.

An example is in order. In the following scenario, we would like to illustrate the use of the GFR_R^i operator. A more thorough examination of its recursion depth will follow in Lemma 37.

Example 36. Let us consider the scenario where we are given a graph G , two nodes s and t of G , and we wish to express that there is a path from s to t in G . Let the graphs we consider be encoded by a structure \mathcal{D} , such that each node can be uniquely identified with a quadruple of values of the universe of \mathcal{D} . The signature we will be using is $\sigma = (\{\}, \{f_E^8, f_{start}^4, f_{target}^4\}, \{\})$, where $f_{start}(x_1, x_2, x_3, x_4) = 1$ if (x_1, x_2, x_3, x_4) encodes the starting node, $f_{target}(x_1, x_2, x_3, x_4) = 1$ if (x_1, x_2, x_3, x_4) encodes the target node and $f_E(x_1, \dots, x_8) = 1$ if there is an edge in G from the node encoded by (x_1, x_2, x_3, x_4) to the node encoded by (x_5, x_6, x_7, x_8) . This means that, if $\mathcal{D} = (\mathcal{A}, F)$, then the graph it encodes has size $|\mathcal{A}|^4$.

If we are interested in finding out whether there exists a path of length bounded by $(\log n)^4$ from the start node to the target node, we can use the following $\text{FOR} + \text{SUM}_R + \text{PROD}_R + \text{GFR}_R^4$ sentence. (A more exact treatment of the recursion depth of the GFR_R^i operator will follow this example in Lemma 37.)

$$\begin{aligned} \varphi = & [f(y_1, \dots, y_8) \equiv t(y_1, \dots, y_8, f)] \\ & \exists x_1, x_2, x_3, x_4 f_{start}(x_1, x_2, x_3, x_4) = 1 \wedge \\ & \exists m f(m, m, m, m, x_1, x_2, x_3, x_4) \geq 1 \end{aligned}$$

where

$$\begin{aligned} t(y_1, \dots, y_8, f) = & f_{target}(y_5, y_6, y_7, y_8) + \\ & \text{sum}_{z_1, \dots, z_8} \cdot \left(\bigvee_{j=1}^4 \left(z_j \leq y_j/2 \wedge \bigwedge_{k=1}^{j-1} z_k \leq y_k \right) \right) \\ & (f_E(y_5, y_6, y_7, y_8, z_5, z_6, z_7, z_8) \times f(z_1, \dots, z_8)) \end{aligned}$$

In φ , we first identify the starting node and then dive into the recursion. Note first that the values y_5, y_6, y_7, y_8 essentially take the place of the \bar{y}_{i+1} in Definition 35. Note also that the values y_1, y_2, y_3, y_4 take the place of the $\bar{y}_1 \dots \bar{y}_i$ in the aforementioned definition and are in this example essentially only used as a counter mimicking the behaviour shown in Table 1 to ensure the correct recursion depth. The point of m in φ is thus merely to essentially start this counter which will then

become decreased in the recursion. We could also impose that m be the maximum value of the universe of \mathcal{D} , but we do not need to, as if any such m exists, our requirement that a path with length bounded by $\mathcal{O}((\log n)^4)$ is satisfied.

In t , we first check whether we have already reached the target node. Afterwards, we use a guarded sum-aggregator to decrease the counter stored in z_1, z_2, z_3, z_4 , iterate over all nodes z_5, z_6, z_7, z_8 and jump into the recursion, zeroing out all non-neighbouring nodes of the current node.

In total, φ essentially describes an algorithm which traverses the given graph in a depth-first manner and which terminates after polylogarithmically many steps because of the guarded aggregations. The exact recursion depth is not obvious and will be investigated further in the following.

Having introduced the GFR_R^i operator, it remains to be shown that it indeed ensures polylogarithmic recursion depth in the way that we want it to.

Lemma 37. *Let $\mathcal{D} \in \text{STRUC}_R(\sigma)$ and let φ be a formula containing a GFR_R^i operator. Then, the recursion depth of the GFR_R^i operator is bounded by $\mathcal{O}((\log_2 n)^i)$, where n is the size of the universe of \mathcal{D} .*

Proof. Let n be the size of the universe of the structure under which the GFR_R^i operator is interpreted. The bound for the recursion depth of the GFR_R^i operator stems from the relativization which guards the aggregated variables. Let $f(\bar{x}, \bar{z}_1, \dots, \bar{z}_i, \bar{z}_{i+1})$ be an occurrence of a GPR^i operator. Then, the variables in $\bar{z}_1, \dots, \bar{z}_i, \bar{z}_{i+1}$ are in the scope of a guarded aggregation of the form

$$A_{\bar{z}_1, \dots, \bar{z}_i, \bar{z}_{i+1}} \cdot \left(\bigvee_{j=1}^i \left(\bar{z}_j \leq \bar{y}_j / 2 \wedge \bigwedge_{k=1}^{j-1} \bar{z}_k \leq \bar{y}_k \right) \wedge \xi(\bar{y}_1, \dots, \bar{y}_i, \bar{y}_{i+1}, \bar{z}_1, \dots, \bar{z}_i, \bar{z}_{i+1}) \right)$$

as per Definition 35.

Let z_j be the numerical interpretation of \bar{z}_j for all $1 \leq j \leq i$. We interpret the tuple $\bar{z}_1, \dots, \bar{z}_i$ as a natural number z of base n where the j th digit of z is z_j .

First, observe that in this interpretation, the relativization of the variables used in the recursive call ensures that z strictly decreases in each step. The big disjunction $\bigvee_{j=1}^i \left(\bar{z}_j \leq \bar{y}_j / 2 \wedge \bigwedge_{k=1}^{j-1} \bar{z}_k \leq \bar{y}_k \right)$ makes sure that there is an index j , such that $\bar{z}_j \leq \bar{y}_j / 2$, which means that the numerical interpretation of \bar{z}_j is at most half of the numerical interpretation of \bar{y}_j . It also ensures that all tuples with smaller indices \bar{z}_k (i.e., the more significant tuples in the interpretation of $\bar{z}_1, \dots, \bar{z}_i$ as a base n number) do not increase.

Since each of the tuples \bar{z}_j (in their numerical interpretation) can only get halved at most $\lfloor \log_2 n \rfloor + 1$ times before reaching 0, it takes at most $\lfloor \log_2 n \rfloor + 2$ recursion steps until a tuple other than the i th has been halved, in the worst case that is the $(i - 1)$ th tuple. This process can then be repeated at most $\lfloor \log_2 n \rfloor + 1$ times, before the tuple at the next lower index gets halved. In total, in the worst case, it takes $(\lfloor \log_2 n \rfloor + 2)^j$ recursion steps until the $i - j$ th tuple gets halved in this process.

This means that after $(\lfloor \log_2 n \rfloor + 2)^i - 1$ recursion steps, each tuple has reached 0. Therefore, the total maximum recursion depth is $(\lfloor \log_2 n \rfloor + 2)^i - 1 \in \mathcal{O}((\log_2 n)^i)$.

This process can be thought of as counting down a base $\log_2 n$ number. The idea for it has already been visualized in Table 1. It is also explicitly illustrated in Tables 3 and 4 in the appendix, for a sequence which we will define shortly in Definition 42 to make use of exactly this kind of process. □

Since our final goal is to characterize both AC_R^i and NC_R^i , we need to also define aggregators which model the properties of NC_R circuits. For this purpose, we introduce *bounded aggregators*, that is relativized aggregators where we only consider the two elements of maximal size meeting the condition in the relativization.

Definition 38. We define the bounded aggregators $\text{sum}_{\bar{x},\text{bound}}$ and $\text{prod}_{\bar{x},\text{bound}}$. They are used in the same way as the aggregators defined earlier in Definition 24. The semantics are defined as follows:

$$\text{sum}_{\bar{x},\text{bound}}.(\varphi(\bar{x}))t(\bar{x}, \bar{w}) \equiv \text{sum}_{\bar{x}}.(\varphi(\bar{x}) \wedge \forall \bar{y} \forall \bar{z}((\bar{y} \neq \bar{z} \wedge \bar{x} < \bar{y} \wedge \bar{x} < \bar{z}) \rightarrow (\neg\varphi(\bar{y}) \vee \neg\varphi(\bar{z}))))t(\bar{x}, \bar{w})$$

The bounded aggregators $\text{prod}_{\bar{x},\text{bound}}$ and $\text{max}_{\bar{x},\text{bound}}$ are defined analogously.

With this bounded aggregation, we can now define a slightly weaker version of the guarded functional recursion from Definition 35, which we call bounded guarded functional recursion $\text{GFR}_{R,\text{bound}}^i$. This allows us then to define logics of the form $\text{FOR}[F] + \text{GFR}_{R,\text{bound}}^i$ or $\text{FOR}[F] + \text{GFR}_{R,\text{bound}}^i$.

Definition 39 ($\text{GFR}_{R,\text{bound}}^i$). A formula is in $\text{FOR}[F] + \text{GFR}_{R,\text{bound}}^i$ if the same conditions as in Definition 35 are met, but instead of a guarded aggregation in (2), we require a bounded guarded aggregation.

Our goal in the following is to characterize the AC_R and NC_R hierarchies using first-order logic and guarded functional recursion. For that purpose, we now define a sequence which we will later use as part of the numbers of our gates in order to encode the gates' depth into their numbers. The idea behind the construction of this sequence will be that for a circuit family $(C_n)_{n \in \mathbb{N}}$ with depth bounded by $c \cdot (\log_2 n)^i$, each of the sequence's elements is essentially a i digit base $c \cdot \lfloor \log_2 n \rfloor - 1$ number with each digit being encoded in unary and padded by zeroes. For readability purposes, we will refer to this encoding simply as a unary encoding. The sequence can then be seen as counting down to 0 in that interpretation. This will then result in a length of $c^i \cdot \lfloor \log_2 n \rfloor^i \in \mathcal{O}((\log_2 n)^i)$ for fixed i . We begin by introducing our conventions regarding unary encodings of numbers.

Definition 40. Let $n, \ell \in \mathbb{N}$ such that $\ell \geq n$. Then, we will refer to the function $\text{unary}_\ell : \mathbb{N} \rightarrow \{0, 1\}^\ell$ defined as

$$\text{unary}_\ell(n) := 0^{\ell-n} 1^n$$

as the (length ℓ) unary encoding of n .

Definition 41. For any binary string \bar{a} of the form

$$\bar{a} = 0^{k-m} 1^m$$

for some k, m , we define the function $\text{uval} : \{0, 1\}^* \rightarrow \mathbb{N}$ defined as

$$\text{uval}(\bar{a}) := m$$

and call $\text{uval}(\bar{a})$ the value of the unary encoding \bar{a} .

We now proceed to define the aforementioned sequence d which we will later use to essentially encode our circuits' gates' depth into their gate numbers.

Table 2. The sequence $d(8, 2, 2)$

$\ell \setminus i$	1	2
1	11111	11111
2	11111	01111
3	11111	00111
4	11111	00011
5	11111	00001
6	11111	00000
7	01111	11111
8	01111	01111
9	01111	00111
10	01111	00011
11	01111	00001
12	01111	00000
13	00111	11111
\vdots	\vdots	\vdots
35	00000	00001
36	00000	00000

Definition 42. For each $n, c, i \in \mathbb{N}_{>0}$, we define the sequence $d(n, c, i)$ as follows. For readability purposes, we leave out the arguments (n, c, i) in the definition of the sequence and only write d instead of $d(n, c, i)$.

- Each element d_ℓ of d consists of i tuples $d_{\ell,j}$ ($1 \leq j \leq i$), each of which is the length $\lfloor c \cdot \log_2 n \rfloor - 1$ unary encoding of a number in $[0, \lfloor c \cdot \log_2 n \rfloor - 1]$.
- $d_1 = 1 \dots 1$ (I. e., $d_{1,j} = \text{unary}_{\lfloor c \cdot \log_2 n \rfloor - 1}(c \cdot \lfloor \log_2 n \rfloor - 1) = 1 \dots 1$ for all j with $1 \leq j \leq i$)
- $d_{\ell+1,i} = \begin{cases} \text{unary}_{\lfloor c \cdot \log_2 n \rfloor - 1}(\lfloor c \cdot \log_2 n \rfloor - 1) & \text{if } \text{uval}(d_{\ell,i}) = 0, \\ \text{unary}_{\lfloor c \cdot \log_2 n \rfloor - 1}(\text{uval}(d_{\ell,i}) - 1) & \text{otherwise,} \end{cases}$ for all ℓ where $d_\ell \neq 0 \dots 0$.
- $d_{\ell+1,j} = \begin{cases} \text{unary}_{\lfloor c \cdot \log_2 n \rfloor - 1}(\text{uval}(d_{\ell,j} - 1)) & \text{if } \text{uval}(d_{\ell,j+1}) = 0, \\ d_{\ell,j} & \text{otherwise,} \end{cases}$ for $j < i$.

Table 2 shows an abbreviated example for the sequence d . A further example as well as the full version of Table 2 can be found in the appendix as Tables 3 and 4.

Remark 43. Note that subtracting the value 1 in this unary encoding can be seen as an integer division by 2 in binary. This will become useful later when putting this into the context of guarded functional recursion with BIT.

Note that the length (i. e. the number of elements) of $d(8, 1, 2)$ is

$$9 = 1^2 \cdot \lfloor \log_2 8 \rfloor^2 (= c^i \cdot \lfloor \log_2 n \rfloor^i).$$

and the length of $d(8, 2, 2)$ is

$$36 = 2^2 \cdot \lfloor \log_2 8 \rfloor^2 (= c^i \cdot \lfloor \log_2 n \rfloor^i).$$

This is no coincidence. Next, we show that this observation holds in general.

Lemma 44. *Let $n, c, i \in \mathbb{N}_{>0}$. Then, $d(n, c, i)$ has length $c^i \cdot \lfloor \log_2 n \rfloor^i$.*

Proof. Since for each element e in $d(n, c, i)$ the successor rule can be interpreted as subtracting 1 from e when e is seen as a base $c \cdot \lfloor \log_2 n \rfloor$ number with i digits, we are starting at the largest possible element in that sense (i. e. $1 \dots 1$, which would correspond to $(c \cdot \lfloor \log_2 n \rfloor)^i - 1$) and we are counting down to the lowest possible element (i. e. $0 \dots 0$, corresponding to 0), there are exactly $(c \cdot \lfloor \log_2 n \rfloor)^i = c^i \cdot \lfloor \log_2 n \rfloor^i$ elements in $d(n, c, i)$. \square

The remaining problem that stands in the way of using the sequence d for the numbering of gates in descriptions for circuits is that the length of the elements in d depends on n (which will be the number of input gates of our circuit). However, we can remedy this, since we essentially have access to base n numbers in the description for a circuit with n input gates (by virtue of interpreting circuit inputs as circuit structures). Combining those with the BIT predicate and now interpreting the unary encoded tuples in elements of d as binary numbers allows us to encode elements of d using a constant number of digits.

Observation 45. *Let $n, c \in \mathbb{N}_{>0}$ and $1 \leq \ell \leq c \cdot \lfloor \log_2 n \rfloor$. The number $2^\ell - 1$ can be encoded by a base n number of length c .*

Proof. The largest possible number of that form is $2^{c \cdot \lfloor \log_2 n \rfloor} - 1 \leq n^c - 1$, which corresponds to " $\underbrace{n - 1 \dots n - 1}_{c \text{ times}}$ " in base n . Therefore, $2^{c \cdot \lfloor \log_2 n \rfloor} - 1$ can be encoded with c base n digits and thus also all smaller natural numbers can be encoded in this way. \square

We can thus encode the binary valuations of tuples in elements of d as base n numbers of length c . Therefore, each element of d can be encoded using i base n numbers of length c (or $i \cdot c$ base n digits).

Before we proceed to use the sequence d for circuit descriptions, we need one more lemma which provides a useful property of $AC_{\mathbb{R}}^i$ resp. $NC_{\mathbb{R}}^i$ circuits. We would like to be able to talk about the *depth* of gates, that is the distance of a gate to the input gates of the circuit. For this reason, we will establish the fact that for the circuit families we investigate, circuits exist where for each gate g , each input g path has the same length.

Lemma 46. *Let L be in $AC_{\mathbb{R}}^i$ or $NC_{\mathbb{R}}^i$ via the circuit family $\mathcal{C} = (C_n)_{n \in \mathbb{N}}$. Then, there exists a circuit family $\mathcal{C}' = (C'_n)_{n \in \mathbb{N}}$ deciding L , such that for all $n \in \mathbb{N}$ and each gate g in C'_n , each path from an input gate to g in C'_n has the same length. We call C'_n a *balanced DAG*.*

The lemma follows from standard circuit manipulation (cf. also (Vollmer, 1999, Exercise 4.35)). For convenience, a proof is provided in the appendix.

As previously mentioned, whenever we are dealing with balanced DAGs, we will refer to the unambiguous length from input gates to a gate g as the *depth* of g .

Now we will turn to a lemma which will then finally enable us to use the previously defined sequence d to encode our gates' depth into their circuit numbers. The idea is for a gate g , to prepend the $depth(g)$ th element of d to the gate number of g . This way each path from an input gate to the output essentially contains elements of d in order in the prefixes of its gate numbers.

This, combined with Lemma 44, provides us with a way to logically ensure the polylogarithmic depth of a circuit given as a circuit structure.

Lemma 47. *Let L be in AC_R^i or NC_R^i via the circuit family $\mathcal{C} = (C_n)_{n \in \mathbb{N}}$. Then, there exists an AC_R^i (resp. NC_R^i) circuit family $\mathcal{C}' = (C'_n)_{n \in \mathbb{N}}$ deciding L with depth bounded by $c \cdot (\log_2 n)^i$, such that each circuit in \mathcal{C}' is a balanced DAG, numbered by base n numbers and each gate g in \mathcal{C}' encodes the depth(g)'s element of d in the first $c \cdot i$ digits of its gate number.*

The numbering after the first $c \cdot i$ digits can be chosen arbitrarily.

Proof. Let $L \in AC_R^i$ or $L \in NC_R^i$ via the circuit family $\mathcal{C} = (C_n)_{n \in \mathbb{N}}$ the depth of which is bounded by $c_1 \cdot (\log_2 n)^i$. Without loss of generality, let the circuits of \mathcal{C} be balanced DAGs as per Lemma 46. That means that for each circuit C_n in \mathcal{C} , for each gate g in C_n , the length of all paths from input gates to g is the same. Let $c \in \mathbb{N}_{>0}$ be such that $c^i \cdot \lfloor \log_2 n \rfloor^i$ is larger than the depth of C_n (which is bounded by $c_1 \cdot (\log_2 n)^i$). We pad each path in C_n to length $c^i \cdot \lfloor \log_2 n \rfloor^i$ by replacing each edge from an input gate to a gate in C_n by a path of dummy gates of length $c^i \cdot \lfloor \log_2 n \rfloor^i - \text{depth}(C_n)$ so that the resulting circuit has exactly depth $c^i \cdot \lfloor \log_2 n \rfloor^i$.

We now use any base n numbering for the gates of C_n and for each gate g prepend the depth(g)th element of $d(n, c, i)$ to the number of g . Since we made sure that each input-output-path in C_n has length exactly $c^i \cdot \lfloor \log_2 n \rfloor^i$, we can encode exactly the sequence $d(n, c, i)$ in the numbers of each input-output-path. So now for each input-output-path, the first $c \cdot i$ digits of gate numbers encode the elements of $d(n, c, i)$ in the order that they appear in the sequence. \square

With the normal form from Lemma 47 and the previous definitions, we can now turn to a theorem characterizing AC_R^i and NC_R^i logically by tying it all together.

Theorem 48.

1. $AC_R^i = FO_R[Arb_R] + SUM_R + PROD_R + GFR_R^i$ for $i \in \mathbb{N}$
2. $NC_R^i = FO_R[Arb_R] + SUM_R + PROD_R + GFR_{R, \text{bound}}^i$ for $i \in \mathbb{N}_{>0}$

Proof. We start by showing the inclusions of the circuit classes in the respective logics and will then proceed with the converse directions.

Step 1: $AC_R^i \subseteq FO_R[Arb_R] + SUM_R + PROD_R + GFR_R^i$:

Let $L \in AC_R^i$ via the nonuniform circuit family $\mathcal{C} = (C_n)_{n \in \mathbb{N}}$ and let the depth of \mathcal{C} be bounded by $c \cdot (\log_2 n)^i$. We construct an $FO_R[Arb_R] + SUM_R + PROD_R + GFR_R^i$ sentence φ defining L . As the circuit input is interpreted as a circuit structure, the signature σ of φ contains only the single unary function symbol f_{element} .

We define the following additional relations and functions which will essentially encode our given circuits. We will have access to them because of the Arb_R extension of our logic and we use relations here instead of functions for ease of reading, since we essentially have access to relations in functional structures if we consider the respective characteristic functions of the relations instead.

- $G_+(\bar{x}) \iff \bar{x}$ is an addition gate.
- $G_\times(\bar{x}) \iff \bar{x}$ is a multiplication gate.
- $G_<(\bar{x}) \iff \bar{x}$ is a $<$ gate, the left predecessor of which is lexicographically lower than the right predecessor.
- $G_{\text{input}}(\bar{x}) \iff \bar{x}$ is an input gate.

- $G_E(\bar{x}, \bar{y}) \iff \bar{y}$ is a successor gate of \bar{x} .
- $G_{\text{output}}(\bar{x}) \iff \bar{x}$ is the output gate.
- $G_{\text{const}}(\bar{x}) \iff \bar{x}$ is a constant gate.
- $f_{\text{const_val}}(\bar{x}) = y \in R \iff y$ is the value of \bar{x} if \bar{x} is a constant gate and $y = 0$ otherwise.

Without loss of generality let all circuits of \mathcal{C} be in the normal form of Lemma 47 and let the numbering of C_n be such that the last digit of the number of the j th input gate is j for $1 \leq j \leq n$. Recall that this means that for each gate number of a gate g represented as a tuple \bar{g} , the first $c \cdot i$ elements of \bar{g} encode the $\text{depth}(g)$ th element of $d(n, c, i)$. The following sentence φ defines L :

$$\varphi := [f(\bar{y}) \equiv t(\bar{y}, f)] \exists \bar{a} G_{\text{output}}(\bar{a}) \wedge f(\bar{a}) = 1$$

where t is defined as follows (with \bar{z}_j denoting the j th c long subtuple in the $c \cdot i$ long prefix of \bar{z} , which, as per the normal form of Lemma 47, encodes the j th tuple of an element of $d(n, c, i)$):

$$\begin{aligned} t(\bar{y}, f) := & \chi[G_+(\bar{y})] \times \text{sum}_{\bar{z}} \cdot \left(\bigvee_{j=1}^i \left(\bar{z}_j \leq \bar{y}_j/2 \wedge \bigwedge_{k=1}^{j-1} \bar{z}_k \leq \bar{y}_k \right) \wedge G_E(\bar{y}, \bar{z}) \right) f(\bar{z}) + \\ & \chi[G_\times(\bar{y})] \times \text{prod}_{\bar{z}} \cdot \left(\bigvee_{j=1}^i \left(\bar{z}_j \leq \bar{y}_j/2 \wedge \bigwedge_{k=1}^{j-1} \bar{z}_k \leq \bar{y}_k \right) \wedge G_E(\bar{y}, \bar{z}) \right) f(\bar{z}) + \\ & \chi[G_<(\bar{y})] \times \text{max}_{\bar{z}} \cdot \left(\bigvee_{j=1}^i \left(\bar{z}_j \leq \bar{y}_j/2 \wedge \bigwedge_{k=1}^{j-1} \bar{z}_k \leq \bar{y}_k \right) \wedge G_E(\bar{y}, \bar{z}) \right) \\ & \quad \left(\text{max}_{\bar{b}} \cdot \left(\bigvee_{j=1}^i \left(\bar{b}_j \leq \bar{y}_j/2 \wedge \bigwedge_{k=1}^{j-1} \bar{b}_k \leq \bar{y}_k \right) \wedge G_E(\bar{y}, \bar{b}) \wedge \bar{b} < \bar{z} \right) \right) \\ & \quad \left(\chi[f(\bar{b}) < f(\bar{z})] \right) + \\ & \chi[G_{\text{input}}(\bar{y})] \times f_{\text{element}}(y_{|\bar{y}|}) + \\ & \chi[G_{\text{const}}(\bar{y})] \times f_{\text{const_val}}(\bar{y}) + \\ & \chi[G_{\text{output}}(\bar{y})] \times \text{sum}_{\bar{z}} \cdot \left(\bigvee_{j=1}^i \left(\bar{z}_j \leq \bar{y}_j/2 \wedge \bigwedge_{k=1}^{j-1} \bar{z}_k \leq \bar{y}_k \right) \wedge G_E(\bar{y}, \bar{z}) \right) f(\bar{z}). \end{aligned}$$

Here, the relations $G_g(\bar{y})$ for $g \in \{+, \times, <, \text{input}, \text{const}, \text{output}\}$ give information about the gate type of the gate encoded by \bar{y} and are provided by the Arb_R extension of FO_R . They are interpreted as mentioned above. The BIT predicate is provided in the same way.

We will now prove that φ does indeed define L . Let $\bar{a} \in R^n$ be the input to C_n . We will show that for all $\bar{g} \in R^l$, where l is the encoding length of a gate in \mathcal{C} , the value of the gate encoded by \bar{g} in the computation of C_n when C_n is given \bar{a} as the input is $f(\bar{g})$. Let g be the gate encoded by \bar{g} . We will argue by induction on the depth of the gate g , that is by the distance between g and an input gate.

$d = 0$: If $d = 0$, then g is an input gate. Therefore, the only summand in $t(\bar{g}, f)$ that is not trivially 0 is the fourth one, which is equal to $f_{\text{element}}(g_{|\bar{g}|})$ (which is the value of the $g_{|\bar{g}|}$ th input gate).

$d \rightarrow d + 1$: Since $d + 1 > 0$, g is not an input gate. This means that there are the following 5 possibilities for g :

1. g is an addition gate: In that case, the only summand in $t(\bar{g}, f)$ which is not trivially 0 is the first one. All predecessors of g have a number, the first $c \cdot i$ digits of which are the successor of the first $c \cdot i$ digits of g in the sequence $d(n, c, i)$ because of the normal form of Lemma 47. Additionally, the relativization ensures that the gate encoded by \bar{z} in the respective summand

has exactly the successor in the sequence $d(n, c, i)$ of g 's first $c \cdot i$ digits in its first $c \cdot i$ digits. This means that by the induction hypothesis

$$\text{sum}_{\bar{z}}. \left(\bigvee_{j=1}^i \left(\bar{z}_j \leq \bar{y}_j/2 \wedge \bigwedge_{k=1}^{j-1} \bar{z}_k \leq \bar{y}_k \right) \wedge G_E(\bar{y}, \bar{z}) \right) f(\bar{z})$$

yields exactly the sum of all the values of predecessor gates of g .

2. g is a multiplication gate: Analogously to the above case,

$$\text{prod}_{\bar{z}}. \left(\bigvee_{j=1}^i \left(\bar{z}_j \leq \bar{y}_j/2 \wedge \bigwedge_{k=1}^{j-1} \bar{z}_k \leq \bar{y}_k \right) \wedge G_E(\bar{y}, \bar{z}) \right) f(\bar{z})$$

yields exactly the product of all predecessor gates of g .

3. g is a $<$ gate: In that case, the relativization

$$\text{max}_{\bar{z}}. \left(\bigvee_{j=1}^i \left(\bar{z}_j \leq \bar{y}_j/2 \wedge \bigwedge_{k=1}^{j-1} \bar{z}_k \leq \bar{y}_k \right) \wedge G_E(\bar{y}, \bar{z}) \right)$$

makes sure that \bar{z} is the maximum gate number of a predecessor of g and

$$\left(\text{max}_{\bar{b}}. \left(\bigvee_{j=1}^i \left(\bar{b}_j \leq \bar{y}_j/2 \wedge \bigwedge_{k=1}^{j-1} \bar{b}_k \leq \bar{y}_k \right) \wedge G_E(\bar{y}, \bar{b}) \wedge \bar{b} < \bar{z} \right) \right) \left(\chi[f(\bar{b}) < f(\bar{z})] \right)$$

makes sure that \bar{b} is the gate number of the other predecessor of g and that therefore $\chi[f(\bar{b}) < f(\bar{z})]$ is the value of $t(\bar{g}, f)$, which is exactly the value of g in the computation of C_n .

4. g is a constant gate: Since $f_{\text{const_val}}(\bar{g})$ returns exactly the constant that g is labelled with, that value is taken by g .
5. g is the output gate: In that case, g only has one predecessor and thus

$$\text{sum}_{\bar{z}}. \left(\bigvee_{j=1}^i \left(\bar{z}_j \leq \bar{y}_j/2 \wedge \bigwedge_{k=1}^{j-1} \bar{z}_k \leq \bar{y}_k \right) \wedge G_E(\bar{y}, \bar{z}) \right) f(\bar{z})$$

ensures that g takes the value of that predecessor, since there is only one element matching the relativization.

Finally, by

$$\varphi := [f(\bar{y}) \equiv t(\bar{y}, f)] \exists \bar{a} G_{\text{output}}(\bar{a}) \wedge f(\bar{a}) = 1$$

we make sure that there exists an output gate which has the value 1 at the end of the computation.

Step 2: $\text{NC}_R^i \subseteq \text{FOR}[\text{Arb}_R] + \text{SUM}_R + \text{PROD}_R + \text{GFR}_{R,\text{bound}}^i$:

The proof for this inclusion follows in the same way as the proof for the AC_R^i case, by just replacing all guarded aggregations in the formulae by bounded guarded aggregations.

Step 3: $\text{FOR}[\text{Arb}_R] + \text{SUM}_R + \text{PROD}_R + \text{GFR}_R^i \subseteq \text{AC}_R^i$:

Let $L \in \text{FOR}[\text{Arb}_R] + \text{SUM}_R + \text{PROD}_R + \text{GFR}_R^i$ via a formula φ over some signature $\sigma = (L_s, L_f, L_a)$ and let there be only one occurrence of a GFR_R^i operator in φ . This proof easily extends to the general case. This means that φ is of the form

$$\varphi = [f(\bar{x}, \bar{y}_1, \dots, \bar{y}_i, \bar{y}_{i+1}) \equiv t(\bar{x}, \bar{y}_1, \dots, \bar{y}_i, \bar{y}_{i+1}, f)] \psi(f).$$

We now construct an AC_R^i circuit family $\mathcal{C} = (C_n)_{n \in \mathbb{N}}$ deciding L .

We first construct an AC_R^i family evaluating φ without the occurrences of f as in Theorem 30. This is possible, since φ is an $FOR[Arb_R] + SUM_R + PROD_R$ formula over $(L_s, L_f, L_a \cup \{f\})$, and by Theorem 30, there is such a circuit family for φ .

Next, we explain how we build an AC_R^i circuit family for the whole formula φ from this point. For this, we need to construct an AC_R^i circuit family computing the value of $f(\bar{x}, \bar{y})$ for each pair \bar{a}, \bar{b} with $\bar{a} \in A^{|\bar{x}|}$ for some $k \in \mathbb{N}$ and $\bar{b} \in A^{i \cdot |\bar{y}_1| + |\bar{y}_{i+1}|}$. Notice that t is an $FOR[Arb_R] + SUM_R + PROD_R + GFR_R^i$ number term and can therefore be evaluated by an AC_R^0 circuit family, except for the occurrences of f . We now obtain C_n by taking the n th circuit of all those (polynomially many) AC_R^0 circuit families and for all \bar{a}, \bar{b} replacing the gate labelled $f(\bar{a}, \bar{b})$ by the output gate of the circuit computing $t(\bar{a}, \bar{b}, f)$.

Since all occurrences of $f(\bar{x}, \bar{y})$ are in the scope of a guarded aggregation

$$A_{\bar{z}_1, \dots, \bar{z}_i} \cdot \left(\bigvee_{j=1}^i \bar{z}_j \leq \bar{y}_j / 2 \wedge \bigwedge_{k=1}^{j-1} \bar{z}_k \leq \bar{y}_k \right)$$

the number of steps from any $f(\bar{a}, \bar{b})$ before reaching a function call of the form $f(\bar{c}, \bar{0}, \bar{d})$, with $|\bar{c}| = |\bar{x}|$ and $|\bar{d}| = |\bar{y}_{i+1}|$, terminating the recursion, is bounded by $\mathcal{O}((\log_2 n)^i)$ as per Lemma 37.

Since each such step – computing $f(\bar{a}_1, \bar{b}_1)$, when given values of the next recursive call $f(\bar{a}_2, \bar{b}_2)$ – is done by an AC_R^0 circuit and therefore has constant depth, in total, any path from the first recursive call to the termination has length in $\mathcal{O}((\log_2 n)^i)$. Since the starting circuit deciding φ had constant depth, the circuit we constructed now has polylogarithmic depth in total. And given that we only added polynomially many subcircuits with polynomially many gates each, the whole circuit is an AC_R^i circuit deciding φ .

For the general case of several GFR_R^i operators, we construct a circuit for each operator in the same way and connect them to the circuit evaluating φ .

Step 4: $FOR[Arb_R] + SUM_R + PROD_R + GFR_{R, \text{bound}}^i \subseteq NC_R^i$:

This case can be proven analogously to the case for AC_R^i . Instead of AC_R^0 families for evaluating φ and t , we now need to use NC_R^1 families, which is why this result only holds for $i > 0$. With this, we have logarithmic depth for evaluating t , which would generally be a problem, since repeating this $(\log_2 n)^i$ times would yield a NC_R^{i+1} family.

However, this would only be the case, if the f gate (which would later be replaced by an edge to another copy of the circuit evaluating t) was to occur at logarithmic depth in the circuit evaluating t . Fortunately, due to our requirement that f must never occur in the scope of any unbounded quantifier or aggregator, this is not the case, because the only cases where unbounded fan-in gates would be needed are the ones of unbounded quantifiers and aggregators.

It remains to be shown that evaluating a number term without unbounded quantifiers and aggregators can be done in constant depth with bounded fan-in, that is in NC_R^0 . This might seem surprising at first glance, as NC_R^0 circuits can essentially only access a constant number of their input gates. However, if no unbounded quantifiers or aggregators are present, $FOR[Arb_R] + SUM_R + PROD_R$ number terms only contain variables introduced by bounded aggregators or quantifiers, and their behaviour can essentially be hardwired. We will now show that the bounded aggregators do not introduce logarithmic depth, because, as mentioned above, only the unbounded quantifiers and aggregators would require unbounded fan-in gates and thus introduce logarithmic depth when being restricted to bounded fan-in. The saving grace here is that we only need to aggregate over the maximum two elements which satisfy the relativization of the bounded aggregation. The relativization only depends on the ranking of the input structure and not on the actual functions and relations given therein. Since the ranking is implicitly given to the circuit by the length of the input structure’s encoding and thus the number of input gates of

the circuit, we thus essentially hard-code, which the two maximum elements satisfying the relativization are. Once we have done that we can simply use a single addition (multiplication) gate to mimic the behaviour of a bounded sum (product) construction which has as its two predecessors the circuits for the term, which is being aggregated over, with the two aforementioned maximum values satisfying the relativization.

Putting this all together, f gates only ever occur at constant depth in the circuit for t , and thus, replacing this gate by a copy of the circuit for t and doing this iteratively $(\log n)^i$ many times yields a circuit of depth $\mathcal{O}((\log n)^i)$. \square

As mentioned previously, the basis of the idea for guarded functional recursion was the guarded predicative recursion used for plain first-order logic (Durand et al., 2018). The same extension to polylogarithmic recursion depth that was showcased in this paper for GFR_R can be applied to GPR. Similarly to the relativized aggregators in Notation 34, for relativized quantifiers, we write

$$(\exists x_1, \dots, x_k. \varphi) \psi$$

as a shorthand for $\exists x_1 \dots \exists x_k (\varphi \wedge \psi)$ and

$$(\forall x_1, \dots, x_k. \varphi) \psi$$

as a shorthand for $\forall x_1, \dots, x_k (\varphi \rightarrow \psi)$.

For an FO formula φ and a relation variable P , we write $\varphi(P^+)$ if P does not occur in the scope of a negation in φ .

Definition 49 (GPR^i). Let \mathcal{R} be a set of relations such that BIT is definable in $\text{FO}[\mathcal{R}]$. The set of $\text{FO}[\mathcal{R}] + \text{GPR}^i$ formulae over σ is the set of formulae by the grammar for $\text{FO}[\mathcal{R}]$ formulae over σ extended by the rule

$$\varphi := [P(\bar{x}, \bar{y}_1, \dots, \bar{y}_i, \bar{y}_{i+1}) \equiv \theta(\bar{x}, \bar{y}_1, \dots, \bar{y}_i, \bar{y}_{i+1}, P^+)] \psi(P^+),$$

where ψ and θ are $\text{FO}[\mathcal{R}]$ formulae over σ , $\bar{x}, \bar{y}_1, \dots, \bar{y}_i, \bar{y}_{i+1}$ are tuples of variables, P is a relation variable and each atomic sub-formula involving P in θ

1. is of the form $P(\bar{x}, \bar{y}_1, \dots, \bar{y}_i, \bar{y}_{i+1})$, the $\bar{y}_1, \dots, \bar{y}_i, \bar{y}_{i+1}$ are in the scope of a guarded quantification

$$Q \bar{y}_1, \dots, \bar{y}_i, \bar{y}_{i+1}. \left(\bigvee_{j=1}^i \left(\bar{z}_j \leq \bar{y}_j / 2 \wedge \bigwedge_{k=1}^{j-1} \bar{z}_k \leq \bar{y}_k \right) \wedge \xi(\bar{y}_1, \dots, \bar{y}_i, \bar{y}_{i+1}, \bar{z}_1, \dots, \bar{z}_i, \bar{z}_{i+1}) \right)$$

with $Q \in \{\forall, \exists\}$, $\xi \in \text{FO}[\mathcal{R}]$ with ξ not containing any relation symbols for relations given in the input structure and

2. never occurs in the scope of any quantification not guarded this way.

The semantics for GPR^i are defined analogously to the semantics for GFR_R^i in Definition 35.

As stated, the proof of Theorem 48 can easily be adapted to the GPR^i operator, so we obtain the following result.

Corollary 50.

1. $\text{FO}[\text{Arb}] + \text{GPR}^i = \text{AC}^i$ for $i \in \mathbb{N}$
2. $\text{FO}[\text{Arb}] + \text{GPR}_{\text{bound}}^i = \text{NC}^i$ for $i \in \mathbb{N}_{>0}$

5. Relationship Between Versions of AC_R^0 Over Different Integral Domains

Intuitively, a circuit deciding a problem in $AC_{\mathbb{R}}^0$ should also be able to simulate circuits deciding problems in, for example, $AC_{\mathbb{Z}}^0$ and $AC_{\mathbb{Z}_2}^0$. Furthermore, we should be able to simulate an $AC_{\mathbb{Z}_3}^0$ circuit by an $AC_{\mathbb{Z}_2}^0$ circuit by simulating the operations defined in the integral domain \mathbb{Z}_3 by operations of tuples of \mathbb{Z}_2 . To formalize this intuition, we propose the notion of \mathcal{C} simulation maps. Since the focus of this paper is on circuit complexity, we introduce simulation maps only in the context of circuit size and depth, but this formalism does not depend on specific properties of circuits and can easily be adapted to time and space complexity classes, which can, for example be defined via a suitably adapted definition of R machines as presented in (Blum et al., 1998).

Definition 51. Let $f_1, f_2: \mathbb{N} \rightarrow \mathbb{N}$ be two functions and let \mathcal{C} be a complexity class with $\mathcal{C} \in \{\text{SizeDepth}(f_1, f_2), \text{UnbSizeDepth}(f_1, f_2)\}$.

A \mathcal{C} simulation map from an integral domain R_1 to an integral domain R_2 is an injective function $f: R_1^* \rightarrow R_2^*$ such that the following holds:

For all $k \in \mathbb{N}_{>0}$ and $A \in \mathcal{C}_{R_1^k}$, there exists an $\ell \in \mathbb{N}_{>0}$ and a language $B \in \mathcal{C}_{R_2^\ell}$ such that for all $\bar{x} = (x_1, x_2, \dots, x_{|\bar{x}|})$ with $x_i \in R_1^k$ for all i :

$$\bar{x} \in A \iff f(\bar{x}) = (f(x_1), f(x_2), \dots, f(x_{|\bar{x}|})) \in B.$$

If there exists a \mathcal{C} simulation map from an integral domain R_1 to an integral domain R_2 , we also write $\mathcal{C}_{R_1} \subseteq_{\text{sim}} \mathcal{C}_{R_2}$. The relations $=_{\text{sim}}$ and \subsetneq_{sim} are defined analogously.

Similar to the formalism of reductions in classical complexity theory, the relation induced by $\subseteq_{\text{sim}}^{\mathcal{C}}$ is reflexive and transitive. Since in this work the main focus is on the AC_R and NC_R hierarchies, we will proceed to restrict ourselves to these classes. But note that the simulation methods we show trivially extend to larger complexity classes.

The formalism essentially divides our integral domains into a three-tier hierarchy. The first tier in this hierarchy consists of the complexity classes over finite integral domains, the second tier consists of the classes over integral domains which are simulatable by \mathbb{Z} , and the third tier consists of classes over integral domains which are simulatable by \mathbb{R} . In this setting, a complexity class over a certain integral domain is able to simulate the complexity classes over integral domains in the same tier or below. To make this explicit, we show the following three equalities:

Theorem 52. The following three equations hold:

1. $NC_{\mathbb{F}_p}^i =_{\text{sim}} NC_{\mathbb{F}_q}^i$ for all prime powers p, q and $i \in \mathbb{N}$.
2. $NC_{\mathbb{Z}}^i =_{\text{sim}} NC_{\mathbb{Z}[j_k]}^i$ for all $i \in \mathbb{N}$.
3. $NC_{\mathbb{R}}^i =_{\text{sim}} NC_{\mathbb{R}[j_k]}^i$ for all $i \in \mathbb{N}$.

Proof. We split the proof into two main parts: the finite case, in which simulations of integral domains are trivial, and the infinite case, where we have to be more careful about the new operations our simulating circuits execute to uphold that the structure of the circuits simulate are still integral domains.

The finite case. The simulation of finite integral domains is quite trivial. For $NC_{\mathbb{F}_p}^i \subseteq_{\text{sim}} NC_{\mathbb{F}_q}^i$ where $p \leq q$, the simulation is straightforward. For $NC_{\mathbb{F}_p}^i \subseteq_{\text{sim}} NC_{\mathbb{F}_q}^i$ where $p > q$, choose the length of tuples that the $NC_{\mathbb{F}_q}^i$ circuit uses to be the smallest $k \in \mathbb{N}$ such that $p \leq q^k$ holds. Map the p elements to the first p elements in \mathbb{F}_q^k and define addition and multiplication tables of constant size which simulate the addition and multiplication in \mathbb{F}_p .

The infinite case. We will prove that $NC_{\mathbb{Z}}^i =_{\text{sim}} NC_{\mathbb{Z}[j_k]}^i$ for any fixed $k \in \mathbb{N}$. Note that the following proof does not depend on the countability of the set, so the proof for the uncountable case runs analogously. The direction $NC_{\mathbb{Z}}^i \subseteq_{\text{sim}} NC_{\mathbb{Z}[j_k]}^i$ is trivial. In the other direction, note that when simulating infinite integral domains, we can no longer hard-code the addition and multiplication tables in a trivial way. Furthermore, integral domains are not closed under cartesian products, since for two integral domains R_1, R_2 , we have for $R_1 \times R_2$ that $(1, 0) \times (0, 1) = (0, 0)$ using componentwise multiplication. We fix this by still using k tuples to emulate the adjoint elements and using componentwise addition, but adapting multiplication so that it simulates multiplication of two elements with adjoint elements, where the m th index of the tuple stands for the m th power of the adjoint element, which we call j here. Explicitly, we use the integral domain $(\mathbb{Z}^k, +_{\mathbb{Z}^k}, \times_{\mathbb{Z}^k})$, where $+_{\mathbb{Z}^k}$ is componentwise addition, and $\times_{\mathbb{Z}^k}$ is defined as follows:

If we want to multiply two tuples of length k

$$\bar{z} = (x_1, x_2, \dots, x_k) \times_{\mathbb{Z}^k} (y_1, y_2, \dots, y_k),$$

we simulate the multiplication in the original, adjoint integral domain

$$(x_1 + x_2j + \dots + x_kj^{k-1}) \times_{\mathbb{Z}[j_k]} (y_1 + y_2j + \dots + y_kj^{k-1}),$$

by constructing the following matrix of constant size which corresponds to expanding the multiplication:

$$A = (a_{uv}) = \begin{pmatrix} x_1y_1 & x_1y_2j & \dots & x_1y_kj^{k-1} \\ x_2y_1j & x_2y_2j^2 & \dots & x_2y_kj^k (= -x_2y_k) \\ \vdots & \ddots & & \\ x_ky_1j^{k-1} & x_ky_2j^k (= -x_ky_2) & \dots & x_ky_kj^{2k-2} \end{pmatrix}.$$

Observe that, due to the fact that $j^k = -1$, every entry a_{uv} of the matrix contributes to the term at index $(u + v - 2) \bmod k$ in the tuple. The entry of the resulting tuple \bar{z} at index ℓ is thus

$$z_\ell = \sum_{\substack{1 \leq u, v \leq k \\ (u+v-2) \bmod k = \ell}} a_{uv}. \quad \square$$

Theorem 53. *The following three equations hold:*

1. $AC_{\mathbb{F}_p}^i =_{\text{sim}} AC_{\mathbb{F}_q}^i$ for all prime powers p, q and $i \in \mathbb{N}$.
2. $AC_{\mathbb{Z}}^i =_{\text{sim}} AC_{\mathbb{Z}[j_k]}^k$ for all $i \in \mathbb{N}$.
3. $AC_{\mathbb{R}}^i =_{\text{sim}} AC_{\mathbb{R}[j_k]}^i$ for all $i \in \mathbb{N}$.

Proof. We use the strategy from the proof of Theorem 52 and show that unbounded addition and multiplication can be realized without a significant increase in the complexity.

For n given tuples of length k

$$(x_{1,1}, x_{1,2}, \dots, x_{1,k}), (x_{2,1}, x_{2,2}, \dots, x_{2,k}), \dots, (x_{n,1}, x_{n,2}, \dots, x_{n,k}),$$

the simulation of unbounded addition by unbounded componentwise addition of tuples is straightforward. For unbounded multiplication, we extend the strategy for the bounded case by

computing the matrix A iteratively. That is, for a sequence of tuples $\bar{x}_1, \dots, \bar{x}_n$, first compute the matrix A_1 from the tuples \bar{x}_1 and \bar{x}_2 , resulting in a tuple \bar{z}_1 . The matrix A_2 is then computed from the tuples \bar{z}_1 and \bar{x}_3 , and so on. In the resulting tuple \bar{z} , the value of each tuple entry can only depend on the values $x_{i,j}$. So to simulate an unbounded multiplication gate, we need only an unbounded multiplication gate with a linear number of predecessors. \square

Corollary 54. *For all $i \in \mathbb{N}$, we have $\text{NC}_{\mathbb{R}}^i =_{\text{sim}} \text{NC}_{\mathbb{C}}^i$ and $\text{AC}_{\mathbb{R}}^i =_{\text{sim}} \text{AC}_{\mathbb{C}}^i$.*

Proof. Observe that $\mathbb{C} \cong \mathbb{R}[j_{(2)}]$. \square

Lemma 55. *For all $i \in \mathbb{N}$, we have $\text{NC}_{\mathbb{Z}}^i =_{\text{sim}} \text{NC}_{\mathbb{Q}}^i$ and $\text{AC}_{\mathbb{Z}}^i =_{\text{sim}} \text{AC}_{\mathbb{Q}}^i$.*

Proof. For $\text{NC}_{\mathbb{Z}}^i \subseteq_{\text{sim}} \text{NC}_{\mathbb{Q}}^i$ (resp. $\text{AC}_{\mathbb{Z}}^i \subseteq_{\text{sim}} \text{AC}_{\mathbb{Q}}^i$), take $f(x) := x$ and for $\text{NC}_{\mathbb{Q}}^i \subseteq_{\text{sim}} \text{NC}_{\mathbb{Z}}^i$ (resp. $\text{AC}_{\mathbb{Q}}^i \subseteq_{\text{sim}} \text{AC}_{\mathbb{Z}}^i$), take $f(x) := (a, b)$, where $x = \frac{a}{b}$. \square

Lemma 56. *For all $i \in \mathbb{N}$, $\text{NC}_{\mathbb{Q}}^i \subsetneq_{\text{sim}} \text{NC}_{\mathbb{R}}^i$ and $\text{AC}_{\mathbb{Q}}^i \subsetneq_{\text{sim}} \text{AC}_{\mathbb{R}}^i$.*

Proof. We use the fact that \mathbb{R} is a transcendental field extension of \mathbb{Q} , that is there is no finite set of numbers M which we can adjoin to \mathbb{Q} in order to get $\mathbb{Q}[M] = \mathbb{R}$. In our setting, this means that we cannot simulate all numbers $r \in \mathbb{R}$ by a set of finite tuples (a_0, \dots, a_n) of numbers where $a_0, \dots, a_n \in \mathbb{Q}$. \square

6. Conclusion

In this paper, we introduced algebraic complexity classes with respect to algebraic circuits over integral domains. We showed a logical characterization for $\text{AC}_{\mathbb{R}}^0$ and further characterizations for the $\text{AC}_{\mathbb{R}}$ and $\text{NC}_{\mathbb{R}}$ hierarchies, using a generalization of the GPR operator of Durand *et al.* (2018). We constructed a formalism to be able to compare the expressiveness of complexity classes with different underlying integral domains. We then showed that using this formalism, we obtain a hierarchy of sets of complexity classes, each set being able to “simulate” the complexity classes from the sets below.

For future work, it would be interesting to investigate the logical characterizations made in this paper in the uniform setting. We know that for the real numbers, the characterization $\text{AC}_{\mathbb{R}}^0 = \text{FO}_{\mathbb{R}}[\text{Arb}_{\mathbb{R}}] + \text{SUM}_{\mathbb{R}} + \text{PROD}_{\mathbb{R}}$ holds both non-uniformly and for uniformity criteria given by polynomial time computable circuits ($\text{P}_{\mathbb{R}}$ uniform), logarithmic time computable circuits ($\text{LT}_{\mathbb{R}}$ uniform) and first-order definable circuits ($\text{FO}_{\mathbb{R}}$ uniform) (Barlag and Vollmer, 2021). We believe that the results we presented here hold in analogous uniform settings as well, though this would need to be further examined. The case of $\text{P}_{\mathbb{R}}$ uniformity seems like it should follow relatively directly with similar ideas as in the $\text{AC}_{\mathbb{R}}^0$ setting. For $\text{LT}_{\mathbb{R}}$ and $\text{FO}_{\mathbb{R}}$ uniformity, though, some more work would be required, since polylogarithmic depth can neither be trivially simulated in logarithmic time nor can it be simply encoded in fixed length gate numbers, which is how the proofs for the real cases go. However, the periodic nature of the circuits simulating $\text{GFR}_{\mathbb{R}}^i$ extensions makes it plausible, and they are describable in a more compact way.

Another open direction is to find interesting problems (potentially even complete) for these new complexity classes. This could even provide new insights for the classical case.

A promising approach to the separation of algebraic circuit complexity classes could be an adaption of the approach taken by Cucker, who showed that the problem FER, which essentially asks whether a point lies on a Fermat curve, separates the (logarithmic time uniform) $\text{NC}_{\mathbb{R}}^i$ classes (Cucker, 1992). The same proof could also hold for the $\text{NC}_{\mathbb{C}}^i$ classes.

Another model deserving of the name “algebraic circuit” is arithmetic circuits in the sense of Valiant (1979). This model is similar to the model presented here, with the exception that generally, only addition and multiplication gates are permitted. Hence, there is no *sign* or comparison function, which gives easy access to control flow constructions such as conditional statements and also allows for the study of functions which are not differentiable. Maybe the ideas presented in this paper can lead to further insights with regard to this model of computation as well.

In the Boolean case, in addition to the AC and NC hierarchies, one commonly investigated hierarchy is the so-called SAC hierarchy. This hierarchy is defined by bounding the fan-in of only one gate type, that is either the conjunction or the disjunction gates. It is known that it does not make a difference in that setting which gate type is bounded. A possible next step is to define a sensible analogue of the SAC hierarchy in the algebraic setting or as a previous step in the real setting. We believe that in the algebraic case, it does make a difference which gate type we bound. The next step is to decide which version of bounded gates yields the more interesting class and afterwards one can relate it to any GPR^i like operator in the real or the algebraic setting. This model could then possibly be useful to investigate algebraic structures where the respective operations do not adhere to the same axioms.

Acknowledgements. We thank Sebastian Berndt and Anselm Haak for fruitful discussions.

References

- Aluffi, P. (2009). *Algebra: Chapter 0. Graduate Studies in Mathematics*. American Mathematical Society.
- Barlag, T. and Vollmer, H. (2021). A logical characterization of constant-depth circuits over the reals. In: Silva, A., Wassermann, R., and de Queiroz, R. J. G. B. (eds.), *Logic, Language, Information, and Computation - 27th International Workshop, WoLLIC 2021, Virtual Event, October 5-8, 2021, Proceedings*, vol. 13038. *Lecture Notes in Computer Science*. Springer, 16-30.
- Blum, L., Cucker, F., Shub, M. and Smale, S. (1998). *Complexity and Real Computation*. Springer New York.
- Bürgisser, P. (2013) *Completeness and Reduction in Algebraic Complexity Theory, Bd., 7*, Springer Science & Business Media.
- Bürgisser, P., Clausen, M. and Shokrollahi, M. A. (1997). *Algebraic Complexity Theory*, Vol. 315. Grundlehren der mathematischen Wissenschaften. Springer.
- Cucker, F. (1992). $\text{P}_{\mathbb{R}} \neq \text{NC}_{\mathbb{R}}$. *Journal of Complexity* 8 (3) 230–238
- Cucker, F. and Meer, K. (1999). Logics which capture complexity classes over the reals. *The Journal of Symbolic Logic* 64 (1) 363–390.
- Durand, A., Haak, A. and Vollmer, H. (2018). Model-theoretic characterization of boolean and arithmetic circuit classes of small depth. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS, New York, NY, USA: Association for Computing Machinery*, vol. 18, 354–363
- Grädel, E. and Gurevich, Y. (1998). Metafinite model theory. *Information & Computation*. 140 (1) 26–81
- Grädel, E., Kolaitis, P. G., Libkin, L., Marx, M., Spencer, J., Vardi, M. Y., Venema, Y. and Weinstein, S. (2007) *Finite model theory and its applications*, Texts in Theoretical Computer Science. An EATCS Series, Springer.
- Grädel, E. and Meer, K. (1995). Descriptive complexity theory over the real numbers. In Leighton, F. T. and Borodin, A. (eds.) *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*, May-1 June 1995, ACM, 315–324.
- Immerman, N. (1999). *Descriptive Complexity*. Graduate Texts in Computer Science, Springer.
- Lang, S. (2002). *Algebra*, Springer New York.
- Libkin, L. (2004) *Elements of finite model theory*, Texts in Theoretical Computer Science. An EATCS Series. Springer.
- Valiant, L. G. (1979). Completeness classes in algebra. In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*.
- Vollmer, H. (1999). *Introduction to Circuit Complexity - A Uniform Approach*, Springer.

Appendix

A. Bounding Circuit Depth

In this section, we show in general how to bind the depth of a circuit with a recursively defined constraint on the number of gates in the layers of the circuit.

Definition 57. Let C be a circuit and let L_k be the set of all gates $g \in C$ with $\text{depth}(g) = k$. Then, we call L_k the k th layer of C .

Lemma 58. Let $\mathcal{C} = (C_n)_{n \in \mathbb{N}}$ be a circuit family, f a sublinear function and $g_C(k)$ the number of gates at layer k for a circuit $C \in \mathcal{C}$. Furthermore, let $\alpha := 2^{-\frac{\log_2 n}{f(n)}}$.

A circuit C has depth $\mathcal{O}(f(n))$ if we request that for every layer k , the inequality $g_C(k) \leq \lfloor \alpha \cdot g_C(k-1) \rfloor$ holds.

Proof. We want to bound the depth of the circuit (i.e., the number of layers k) by the sublinear function f . For any input size n , we therefore set $k = f(n) < n$. Furthermore, if we require that the inequality $g_C(k) \leq \lfloor \alpha \cdot g_C(k-1) \rfloor$ holds, the circuit reaches its maximum depth when $n\alpha^k \leq 1$ holds, since the factor of α is applied k times, once for each layer. Substitution yields

$$\begin{aligned} n \cdot \alpha^k \leq 1 &\iff n \cdot 2^{-\frac{\log_2 n}{f(n)} k} \leq 1 \\ &\iff n \cdot 2^{-\log_2 n} \leq 1. \end{aligned} \quad \square$$

Corollary 59. A circuit C has depth $\mathcal{O}((\log_2 n)^i)$ if we request that for every layer k , the inequality $g_C(k) \leq \left\lfloor 2^{-\frac{\log_2 n}{(\log_2 n)^i}} \cdot g_C(k-1) \right\rfloor$ holds.

B. Examples for the Sequence $d(n, c, i)$

Here, we develop two examples of the sequence $d(n, c, i)$ from Definition 42. Table 3 shows the sequence $d(8, 1, 2)$. Since $c \cdot \lfloor \log_2 8 \rfloor - 1 = 2$, each element of $d(8, 1, 2)$ contains $i = 2$ tuples of length 2. Each line is one element of the sequence, and the columns determine the tuples in the elements. This means that the first element here is the element (11, 11), the second one is (11, 01) and so on.

Table 4 shows the sequence $d(8, 2, 2)$. Since $c \cdot \lfloor \log_2 8 \rfloor - 1 = 5$, each element of $d(8, 2, 2)$ contains $i = 2$ tuples of length 5. Each line is one element of the sequence, and the columns determine the tuples in the elements. This means that the first element here is the element (11111, 11111), the second one is (11111, 01111) and so on.

Table 3. The sequence $d(8, 1, 2)$

$\ell \setminus i$	1	2
1	11	11
2	11	01
3	11	00
4	01	11
5	01	01
6	01	00
7	00	11
8	00	01
9	00	00

Table 4. The sequence $d(8, 2, 2)$

$\ell \backslash i$	1	2
1	11111	11111
2	11111	01111
3	11111	00111
4	11111	00011
5	11111	00001
6	11111	00000
7	01111	11111
8	01111	01111
9	01111	00111
10	01111	00011
11	01111	00001
12	01111	00000
13	00111	11111
14	00111	01111
15	00111	00111
16	00111	00011
17	00111	00001
18	00111	00000
19	00011	11111
20	00011	01111
21	00011	00111
22	00011	00011
23	00011	00001
24	00011	00000
25	00001	11111
26	00001	01111
27	00001	00111
28	00001	00011
29	00001	00001
30	00001	00000
31	00000	11111
32	00000	01111
33	00000	00111
34	00000	00011
35	00000	00001
36	00000	00000

C. Proof of Lemma 46

Lemma 60. *Let L be in AC_R^i or NC_R^i via the circuit family $\mathcal{C} = (C_n)_{n \in \mathbb{N}}$. Then, there exists a circuit family $\mathcal{C}' = (C'_n)_{n \in \mathbb{N}}$ deciding L , such that for all $n \in \mathbb{N}$ and each gate g in C'_n , each path from an input gate to g in C'_n has the same length. We call C'_n a balanced DAG.*

Proof. Let L be in AC_R^i or NC_R^i via $\mathcal{C} = (C_n)_{n \in \mathbb{N}}$. For each circuit $C_n \in \mathcal{C}$, we construct a circuit C'_n , such that $f_{C_n} = f_{C'_n}$ and for each gate g in C'_n , all input g paths in C'_n have the same length. We transform C_n into C'_n by creating paths of dummy gates to replace edges that go over more than one level of depth.

Let the depth of C_n be bounded by $c_1 \cdot (\log_2 n)^i$, and let its size be bounded by $c_2 \cdot n^{c_3}$. We proceed by structural induction over the depth d of gates g in C_n , that is, the maximum length of paths from an input gate to g .

$d = 1$: Each gate g at depth d is a direct successor of an input gate. Therefore, no changes need to be made, since all gates at depth d only have input g paths of length 1 and therefore have the desired property.

$d \rightarrow d + 1$: For each gate at depth $d + 1$, all predecessors are gates of depth $< d + 1$ for which it holds that all paths from input gates to them are of the same length. Keep all those predecessors at depth d as they are and replace the edge from predecessors of smaller depths $d' < d$ to g by a path of dummy (unary addition) gates of length $d - d'$. Now all paths from input gates to g have exactly length $d + 1$, and we only added at most $c_1 \cdot (\log_2 n)^i$ gates per predecessor of g . In total, the number of dummy gates added for g is bounded by $c_2 \cdot n^{c_3} \cdot c_1 \cdot (\log_2 n)^i$.

The resulting circuit is C'_n . Since addition gates with only a singular predecessor are essentially identity gates, the value of each gate in any computation of C'_n remains the same. Thus, $f_{C_n} = f_{C'_n}$.

Additionally, for each gate in C_n , we add at most $c_1 \cdot (\log_2 n)^i \cdot c_2 \cdot n^{c_3}$ gates to arrive at C'_n . Therefore, the size of C'_n is bounded by $c_1 \cdot (\log_2 n)^i \cdot c_2 \cdot n^{c_3} \cdot c_2 \cdot n^{c_3} = c_1 \cdot (\log_2 n)^i \cdot (c_2 \cdot n^{c_3})^2 \in \mathcal{O}(n^{\mathcal{O}(1)})$. The depth of C'_n does not change, since we only ever add gates, when longer paths within C_n exist so that they end up at the same length.

In total, C'_n computes the same function as C_n – and thus decides L – and has the property that for each of its gates g , all input g paths have the same length. □