

LOGICAL CHARACTERIZATIONS OF ALGEBRAIC CIRCUIT CLASSES OVER INTEGRAL DOMAINS

TIMON BARLAG, FLORIAN CHUDIGIEWITSCH, AND SABRINA A. GAUBE

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_R and NC_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.

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 [Blu+98] 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 a 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 [DHV18] to logics over metafinite structures and extend it to be able to characterize the whole AC_R and NC_R -hierarchies.

Date: February 28, 2023.

Key words and phrases. Algebraic circuits, descriptive complexity.

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 [Val79], which are the foundational model in the emerging subfields of algebraic and geometric complexity theory. They differ from the model we analyze 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 $\text{NC}_{\mathbb{R}}^i \subsetneq \text{NC}_{\mathbb{R}}^{i+1}$ [Cuc92] but we have that $\text{VNC}^2 = \text{VNC}^3 = \dots = \text{VP}$ [Bür13] 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 $\text{AC}_R^0 = \text{FO}_R[\text{Arb}_R] + \text{SUM}_R + \text{PROD}_R$.

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 [DHV18].

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

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

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.

Notation 2.1. In this paper, we will generally use overlined letters (e.g. \overline{x}) to denote tuples.

As the name suggests, algebraic circuit classes make use of concepts originating from abstract algebra. We will first give an overview of some fundamental concepts used in this paper. For further information on the topic, the reader may refer to the books by Aluffi [Alu09] or Lang [Lan02].

Definition 2.2 (Ring). A *ring with unit* $(R, +, \times)$ is a tuple consisting of a set R and two binary operations $+$ and \times such that

- $1 \in R$
- $(R, +)$ is an abelian group, i.e.
 - For every $a, b, c \in R$, we have $(a + b) + c = a + (b + c)$
 - For every $a, b \in R$, we have $a + b = b + a$
 - There is a $0 \in R$, we have $a + 0 = a$
 - For every $a \in R$, there exists an inverse $-a \in R : a + (-a) = 0$
- (R, \times) is a monoid, i.e.
 - $(a \times b) \times c = a \times (b \times c)$
 - There is a $1 \in R$ such that $a \times 1 = 1 \times a = a$

- multiplication is distributive with respect to addition:

$$\begin{aligned} a \times (b + c) &= a \times b + a \times c, \text{ for all } a, b, c \in R \\ (a + b) \times c &= a \times c + b \times c, \text{ for all } a, b, c \in R \end{aligned}$$

A ring is called *commutative*, if $a \times b = b \times a$, for all $a, b \in R$. \triangleleft

In the following a ring is always assumed to have a unit.

Definition 2.3 (Integral domain). A ring R with unit is called integral domain, if it is a commutative ring and if there is no element $a \in R$ which is a zero-divisor, i. e. there is no $a \in R \setminus \{0\}$ for which there is a $b \in R \setminus \{0\}$ with $a \times b = 0$. \triangleleft

Throughout the paper, whenever not otherwise specified, we use R to denote an infinite integral domain.

Remark 2.4. 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.

Definition 2.5 (Polynomial). Let R be a ring. A *polynomial* $f(x)$ in the *indeterminate* x and with *coefficients* in R is a *finite* linear combination of nonnegative exponents of x with coefficients in R :

$$f(x) = \sum_{i \geq 0} a_i x^i = a_0 + a_1 x + a_2 x^2 + \dots,$$

where all a_i are elements of R (the *coefficients*) and we require $a_i = 0$ for infinitely many i . Two polynomials are taken to be equal if all the coefficients are equal:

$$\sum_{i \geq 0} a_i x^i = \sum_{i \geq 0} b_i x^i \iff (\forall i \geq 0): a_i = b_i.$$

The set of polynomials in x over R is denoted by $R[x]$. \triangleleft

Definition 2.6 (Polynomial ring). Let $f(x) = \sum_{i \geq 0} a_i x^i$ and $g(x) = \sum_{i \geq 0} b_i x^i$ be two polynomials. A *polynomial ring* is the set $R[x]$ together with the following two operations:

$$f(x) + g(x) := \sum_{i \geq 0} (a_i + b_i) x^i$$

and

$$f(x) \times g(x) := \sum_{k \geq 0} \sum_{\substack{0 \leq i \leq k, \\ 0 \leq j \leq k, \\ i+j=k}} a_i b_j x^{i+j}$$

The unit of this ring is $1_{R[x]} = 1_R + 0x + 0x^2 + \dots$. \triangleleft

Remark 2.7. A polynomial ring like in Definition 2.6 is also a ring with unit. Analogous to Definition 2.6, we can define polynomial ring S with finitely many variables x_1, \dots, x_n by applying Definition 2.6 inductively:

$$S := R[x_1, \dots, x_n] := (R[x_1, \dots, x_{n-1}])[x_n],$$

where R is a ring. Note that polynomial rings over fields are integral domains.

Definition 2.8 (Adjoint elements to a ring). When *adjoining* a number $f \notin R$ to a ring R we obtain the set

$$R[f] := \{a + f \cdot b \mid a, b \in R\}.$$

The set $R[f]$ together with the following two operations

$$(a + f \cdot b) + (c + f \cdot d) := (a + c) + f \cdot (b + d)$$

and

$$(a + f \cdot b) \cdot (c + f \cdot d) := ac + f(ad + bc) + f^2(bd),$$

where $+$ and \cdot are the binary operations on the ring, is again a ring, where $1_{R[x]} = 1_R + 0f$ is the unit of this ring. \triangleleft

Example 2.9. Popular examples of rings include $\mathbb{Z}, \mathbb{Q}, \mathbb{R}$ and \mathbb{C} , as well as sets with *adjoined* elements like $\mathbb{Z}[i], \mathbb{Z}[\sqrt{-1}], \mathbb{Z}[\sqrt{p}, \sqrt{q}, \dots], \mathbb{R}[i_k], \mathbb{Z}[i_k], \dots$, where $k \in \mathbb{N}$, $p \neq q$ are primes and i_k denotes the k th root of -1 , i. e. $i_k^k = -1$. Alternatively, we can adjoin ℓ algebraic independent prime root elements with $2^\ell = k$ and get an analogous construction.

Remark 2.10. The denotation of $\mathbb{R}[i_k]$ resp. $\mathbb{Z}[i_k]$ are 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 $\mathbb{Z}[i_4] = \mathbb{Z}[i_4, i_4^2, i_4^3]$ 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, i. e., the coefficients of adjoint elements, or later the underlying circuits, our short notation for \mathbb{Z}^k for arbitrary $k > 1$ is not unique and may differ, e. g. \mathbb{Z}^2 may denote $\mathbb{Z}[i]$ 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.

Definition 2.11 (Field). A field $(F, +, \times)$ is a tuple consisting of a set F and two binary operations $+$ and \times such that

- $1 \in F$
- $0 \in F$
- $(F, +)$ is an abelian group (see 2.2)
- (F, \times) is an abelian group (see 2.2)
- multiplication is distributive with respect to addition:

$$a \times (b + c) = a \times b + a \times c, \text{ for all } a, b, c \in F$$

We do not need two equations for distributivity since multiplication and addition are both commutative. \triangleleft

Remark 2.12. An alternative characterisation for a field is as a commutative ring where $0 \neq 1$ holds and every element $a \neq 0 \in F$ is invertible.

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, 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 2.13. A *strict total order* on a set S is a binary relation $<$ on S which is irreflexive, transitive and total. \triangleleft

Example 2.14. 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 $z = a + bi \in \mathbb{C}$ 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 2.15. A strict total order $<$ over a ring R induces a sign-function as follows:

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

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 [Vol99]. The generalization to *algebraic circuits* over real numbers were first introduced by Cucker [Cuc92]. In analogy to them, Barlag and Vollmer defined an unbounded fan-in version of algebraic circuits [BV21].

Definition 2.16. 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, 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. \triangleleft

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 2.17. We will use the term *algebraic circuit* to describe circuits in the sense of Blum, Cucker, Shub and Smale [Blu+98] rather than *arithmetic circuits* as e.g. in [BV21] to distinguish them from arithmetic circuits in the sense of Valiant, see e.g. [BCS97]. Valiant circuits are essentially algebraic circuits without sign-gates [Blu+98, page 350].

Remark 2.18. In the special case $R = \mathbb{Z}_2$, the definition above yields exactly the Boolean circuits.

Definition 2.19. 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. \triangleleft

Remark 2.20. Unlike the way algebraic circuits with unbounded fan-in gates were introduced previously [BV21], 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 2.15 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 now 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 2.15.

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 2.15.

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 2.21. 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. \triangleleft

Definition 2.22. 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 . \triangleleft

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 2.23. The class AC_R^i is the class of sets decidable by R -circuit families of size $\mathcal{O}(n^{\mathcal{O}(1)})$ and depth $\mathcal{O}((\log_2 n)^i)$. \triangleleft

Definition 2.24. The class NC_R^i is the class of sets decidable by R -circuit families of size $\mathcal{O}(n^{\mathcal{O}(1)})$ and depth $\mathcal{O}((\log_2 n)^i)$, where each arithmetic gate has an indegree bounded by 2 (we call this *bounded fan-in*). \triangleleft

Remark 2.25. 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. [Vol99].

Uniformity criteria can be defined for algebraic circuit classes in a similar way. See e. g. [Blu+98, 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., Libkin and Immerman [Grä+07; Lib04; Imm99]. Traditionally, descriptive complexity is viewed as a subdiscipline of finite model theory, since allowing infinite structures often makes problems undecidable. In our setting, however, we want to (carefully) reintroduce 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 [GG98]. A short introduction to metafinite model theory is also featured in the book by Grädel et al. [Grä+07, page 210]. The approach was taken by Grädel and Meer [GM95] to describe some essential complexity classes over real numbers. These descriptions were later extended by Cucker and Meer [CM99], where, among other things, the NC-hierarchy over real numbers was defined. For the characterization, 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 use the well known trick that a signature consisting only of functions can emulate any predicates we might want to define via a characteristic function corresponding to the relation the predicate is interpreted as. We can furthermore emulate constants by 0-ary functions.

Definition 2.26. 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 $\text{STRUC}_R(\sigma)$ 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. \triangleleft

Remark 2.27. In this paper, we only consider *ranked structures*, i. e., structures, in which the skeleton is ordered.

Definition 2.28 (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} . \triangleleft

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 2.29. 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. Since we want to do some calculations with these elements, we call them numbers, too.

Equivalence of FO_R -formulae and sets defined by FO_R -formulae are done in the usual way, i. e., 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 2.30 (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 . \triangleleft

Example 2.31. Let $\sigma = (\{\}, f_E^2)$ be the signature of weighted graphs, i. e. $f_E(x, y)$ 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 $\text{FO}_R + \text{SUM}_R$ 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 2.32. $\text{FO}_R = \text{FO}_R + \text{MAX}_R$.

Proof. An occurrence of $\max_i(F(i))$ 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 [CM99, Proposition 2], since this proof does not make any use of special properties of the reals.

3. $\text{AC}_R^0 = \text{FO}_R$

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 2.26 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 3.1. Let F be a set of finite 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. \triangleleft

Formally, this means that $\text{FO}_R[F]$ describes exactly those sets $S \subseteq R^*$ for which there exists a 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 .

Definition 3.2. We write Arb_R to denote the set of all functions $f: R^k \rightarrow R$, where $k \in \mathbb{N}$. \triangleleft

Theorem 3.3. Let R be an infinite integral domain. Then $\text{AC}_R^0 = \text{FO}_R[\text{Arb}_R] + \text{SUM}_R + \text{PROD}_R$.

Proof. The proof for this theorem works similarly to the construction for $\text{FO}_{\mathbb{R}}[\text{Arb}_{\mathbb{R}}] + \text{SUM}_{\mathbb{R}} + \text{PROD}_{\mathbb{R}} = \text{AC}_{\mathbb{R}}^0$ [BV21], 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 $\text{FO}_R[\text{Arb}_R] + \text{SUM}_R + \text{PROD}_R$ -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 mimicing 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 2.15) 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, i. e., 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 $\text{val}_d(g)$ is created when given a circuit family $(C_n)_{n\in\mathbb{N}}$, such that for each n , $\text{val}_d(g)$ evaluates to the value of gate g if g is on the d th level of the respective circuit C_n . 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, Haak and Vollmer [DHV18] 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 [Imm99] in the following way

$$x \leq \frac{y}{2} \iff x + x \leq y.$$

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 4.1. 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.

Proof. $n \cdot \left(2^{-\frac{\log_2 n}{(\log_2 n)^i}}\right)^{(\log_2 n)^i} = n \cdot 2^{-\frac{\log_2 n}{(\log_2 n)^i} \cdot (\log_2 n)^i} = n \cdot 2^{-\log_2 n} = n \cdot \frac{1}{n} = 1 \quad \square$

A more general version of this observation can be found in Lemma A.2 in the appendix. Unfortunately, while it is simple to divide by 2 when the BIT predicate

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

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

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 poly-logarithmic 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 the table in Figure 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.

In order to perform divisions by 2, as stated before, we need to be able to express the BIT predicate in our logic. In the classical case with natural numbers, BIT is defined as follows, where we assume the most significant bit of j 's binary representation to be the bit at index 1:

Definition 4.2. Let the relation $\text{BIT}^2 \subseteq \mathbb{N} \times \mathbb{N}$ be defined as follows:

$$\text{BIT} := \{(i, j) \mid \text{the } i\text{th bit of the binary representation of } j \text{ is } 1, i, j \in \mathbb{N}\}. \quad \triangleleft$$

Since we wish to logically characterize languages decided by circuit families, it is useful to briefly talk about representation of numbers in descriptive complexity. 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 .

Example 4.3. 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. Given that we are in this setting, we adjust the definition of BIT to suit our purposes as follows, where again, the most significant bit of j 's binary representation is the bit at index 1:

Definition 4.4. For any ranked structure with universe D , let the relation $\text{BIT}^2 \subseteq D^* \times D^*$ be defined as follows:

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

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 [Imm99]. 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, i. e., *relativized aggregators*. A relativization of an aggregator is a formula restricting which elements are considered for the aggregator.

Notation 4.5. For a number term t and a 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

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

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

Definition 4.6 (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 2.30).

For $i \geq 0$, the set of $\mathcal{L} + \text{GFR}_R^i$ -formulae over σ is the set of formulae by the grammar for \mathcal{L} over σ extended by the rule

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

where ψ is a \mathcal{L} formula, f is a function symbol and t is a \mathcal{L} number term with free variables $\overline{x}, \overline{y}_1, \dots, \overline{y}_i, \overline{y}_{i+1}$ such that all \overline{y}_j for $1 \leq j \leq i$ contain the same (positive) number of variables and each (sub-)number term involving the symbol f in t

- (1) is of the form $f(\overline{x}, \overline{z}_1, \dots, \overline{z}_i, \overline{z}_{i+1})$, where $\overline{z}_1, \dots, \overline{z}_i, \overline{z}_{i+1}$ are in the scope of a guarded aggregation

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

with $A \in \{\mathbf{max}, \mathbf{sum}, \mathbf{prod}\}$, $\xi \in \mathcal{L}$ with ξ not containing any relation symbols for relations given in the input structure and

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

The function symbol f is considered bound in

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

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

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

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

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

This means that the formula $[f(\overline{x}, \overline{y}_1, \dots, \overline{y}_i, \overline{y}_{i+1}) \equiv t(\overline{x}, \overline{y}_1, \dots, \overline{y}_i, \overline{y}_{i+1}, f)]\psi(\overline{c}, f)$ holds for a tuple $\overline{c} \in D^{|\overline{c}|}$ with the same arity as \overline{z} if and only if $\mathcal{D} \models \psi(\overline{c}, f_I)$ where f_I 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. \triangleleft

Having defined 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 4.7. *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 $f(\overline{x}, \overline{z}_1, \dots, \overline{z}_i, \overline{z}_{i+1})$ be an occurrence of a GPR^i operator. Then the variables in $\overline{z}_1, \dots, \overline{z}_i, \overline{z}_{i+1}$ are in the scope of a guarded aggregation of the form

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

as per Definition 4.6.

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 conjunction $\bigvee_{j=1}^i \bar{z}_j \leq \bar{y}_j/2 \wedge \bigwedge_{k=1}^{j-1} \bar{z}_k \leq \bar{y}_k$ 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, the first 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 Figure 1. It is also explicitly illustrated in Figure 2 and Figure 3, for a sequence which we will define shortly in Definition 4.12 to make use of exactly this kind of process. \square

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*, i. e., relativized aggregators where we only consider the two elements of maximal size meeting the condition in the relativization.

Definition 4.8. We define the bounded aggregators $\mathbf{sum}_{\bar{x}, \text{bound}}$ and $\mathbf{prod}_{\bar{x}, \text{bound}}$. They are used in the same way as the aggregators defined earlier in Definition 2.30. The semantics are defined as follows:

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

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

With this bounded aggregation, we can now define a slightly weaker version of the guarded functional recursion from Definition 4.6, which we call bounded guarded functional recursion $\text{GFR}_{R, \text{bound}}^i$. This allows us then to define logics of the form $\text{FO}_R[F] + \text{GFR}_R^i$ or $\text{FO}_R[F] + \text{GFR}_{R, \text{bound}}^i$.

Definition 4.9 ($\text{GFR}_{R, \text{bound}}^i$). A formula is in $\text{FO}_R[F] + \text{GFR}_{R, \text{bound}}^i$ if the same conditions as in Definition 4.6 are met, but instead of a guarded aggregation in (1), we require a bounded guarded aggregation. \triangleleft

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 $c^i \cdot \lceil \log_2 n \rceil^i \in \mathcal{O}((\log_2 n)^i)$ for fixed i . We begin by introducing our conventions regarding unary encodings of numbers.

Definition 4.10. 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 4.11. 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 circuit's gates' depth into their gate numbers.

Definition 4.12. For each $n, c, i \in \mathbb{N}$, 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 $d(n, c, i)$.

- (1) 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 $[0, \lfloor c \cdot \log_2 n \rfloor - 1]$.
- (2) $d_1 = 1 \dots 1$ (I. e., $d_{1,j} = \text{unary}_{\lfloor c \cdot \log_2 n \rfloor - 1}(c \cdot \lceil \log_2 n \rceil - 1) = 1 \dots 1$ for all j with $1 \leq j \leq i$.)
- (3) $d_{\ell+1,i} = \begin{cases} \text{unary}_{\lfloor c \cdot \log_2 n \rfloor - 1}(\lfloor c \cdot \log_2 n \rfloor - 1) & \text{if } \text{uval}(d_{\ell,i}) = 0, \\ \text{unary}_{\lfloor c \cdot \log_2 n \rfloor - 1}(\text{uval}(d_{\ell,i}) - 1) & \text{otherwise,} \end{cases}$
for all ℓ where $d_\ell \neq 0 \dots 0$.
- (4) $d_{\ell+1,j} = \begin{cases} \text{unary}_{\lfloor c \cdot \log_2 n \rfloor - 1}(\text{uval}(d_{\ell,j} - 1)) & \text{if } \text{uval}(d_{\ell,j+1}) = 0, \\ d_{\ell,j} & \text{otherwise,} \end{cases}$
for $j < i$. ◁

Examples for the sequence d can be seen in Figures 2 and 3.

Remark 4.13. 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

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

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

$$36 = 2^2 \cdot \lceil \log_2 8 \rceil^2 (= c^i \cdot \lceil \log_2 n \rceil^i).$$

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

Lemma 4.14. Let $n, c, i \in \mathbb{N}$. Then, $d(n, c, i)$ has length $c^i \cdot \lceil \log_2 n \rceil^i$.

$\ell \backslash i$	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

FIGURE 2. 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, 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.

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 4.15. Let $n, c \in \mathbb{N}$ 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 $\underbrace{\overbrace{[n-1, \dots, n-1]}^{c \text{ times}}}$ 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 using the sequence d for circuit descriptions, we need one more lemma which provides a useful property of AC_R^i resp. NC_R^i circuits. We would like to be able to talk about the *depth* of gates, i.e., 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.

$\ell \backslash i$	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

FIGURE 3. The sequence $d(8, 2, 2)$. Since $c \cdot \lceil \log_2 8 \rceil - 1 = 5$, each element of $d(8, 2, 2)$ contains $i = 2$ tuples of length 5. Each line is one element of the sequence, 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.

Lemma 4.16. *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 , i. e., 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. \square

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. This, combined with Lemma 4.14, provides us with a way to logically ensure the polylogarithmic depth of a circuit given as a circuit structure.

Lemma 4.17. *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 the gates of each circuit of \mathcal{C}' are numbered as base n numbers, where for each path from an input gate to the output gate, the first $c \cdot i$ digits encode elements of d in the order that they appear in the sequence.*

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

Proof. Let $L \in \text{AC}_R^i$ or $L \in \text{NC}_R^i$ 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 4.16. 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 $c \in \mathbb{N}$ 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 $\text{depth}(g)$ 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 $c^i \cdot \lceil \log_2 n \rceil^i$, 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 4.17 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 4.18.

- (1) $\text{AC}_R^i = \text{FO}_R[\text{Arb}_R] + \text{SUM}_R + \text{PROD}_R + \text{GFR}_R^i$
- (2) $\text{NC}_R^i = \text{FO}_R[\text{Arb}_R] + \text{SUM}_R + \text{PROD}_R + \text{GFR}_{R, \text{bound}}^i$

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

Let $L \in \text{AC}_R^i$ 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 a $\text{FO}_R[\text{Arb}_R] + \text{SUM}_R + \text{PROD}_R + \text{GFR}_R^i$ 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 4.17 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 4.17 encodes the j th

tuple of an element of $d(n, c, i)$:

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

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 , i. e., 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 4.17. 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

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

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

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

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

yields exactly the product of all predecessor gates of g .

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

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

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

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

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 $\varphi_{\text{const_val}}(\bar{g})$ returns exactly the value of g , and that value is taken by $t(\bar{g}, f)$.

(5) g is the output gate: In that case, g only has one predecessor and thus

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

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

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

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

Let $L \in \text{FO}_R[\text{Arb}_R] + \text{SUM}_R + \text{PROD}_R + \text{GFR}_R^i$ 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 3.3. This is possible, since φ is an $\text{FO}_R[\text{Arb}_R] + \text{SUM}_R + \text{PROD}_R$ formula over $(L_s, L_f, L_a \cup \{f\})$ and by Theorem 3.3, 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 $\bar{a} \in A^k$ for some $k \in \mathbb{N}$ and $\bar{b} \in A^i$, where the length of the tuple \bar{b} is exactly the exponent of the polylogarithm bounding the circuit depth. Notice, that t is an $\text{FO}_R[\text{Arb}_R] + \text{SUM}_R + \text{PROD}_R + \text{GFR}_R^i$

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 labeled $f(\bar{a}, \bar{b})$ 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

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

the number of steps from any $f(\bar{a}, \bar{b})$ before reaching $f(\bar{a}, \bar{0})$, terminating the recursion, is bounded by $\mathcal{O}((\log_2 n)^i)$ as per Lemma 4.7.

Since each such step – computing $f(\bar{a}, \bar{b})$, when given values of the next recursive call $f(\bar{c}, \bar{d})$ – 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: $\text{FO}_R[\text{Arb}_R] + \text{SUM}_R + \text{PROD}_R + \text{GFR}_{R, \text{bound}}^i \subseteq \text{NC}_R^i$:

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. 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, by definition, there are no occurrences of f in the scope of unbounded aggregators or quantifiers. For the bounded aggregators, we still construct circuits for all possible values of the aggregated variables, but we only connect the output gates of the maximum two circuits satisfying the relativization. We can do this, since, as ξ does not contain function or relation symbols given in the structure, we can predetermine which circuits will match the relativization and therefore hardwire the connections. In general, $\text{FO}_R[\text{Arb}_R] + \text{SUM}_R + \text{PROD}_R$ formulae without unbounded quantifiers or aggregators can be evaluated in NC_R^0 . Therefore, the gates marked $f(\bar{a}, \bar{b})$ only occur at constant depth in the subcircuits for $t(\bar{a}, \bar{b}, f)$. This means, that in total, the construction leads to a depth in $\mathcal{O}((\log_2 n)^i)$. \square

As mentioned previously, the basis of the idea for guarded functional recursion was the guarded predicative recursion used for plain first-order logic [DHV18]. The same extension to polylogarithmic recursion depth that was showcased in this paper for GFR_R can be applied to GPR, yielding the following results. (A formal definition of GPR^i is presented in the appendix as Definition B.2).

Corollary 4.19.

- (1) $\text{FO}[\text{Arb}] + \text{GPR}^i = \text{AC}^i$
- (2) $\text{FO}[\text{Arb}] + \text{GPR}_{\text{bound}}^i = \text{NC}^i$

5. RELATIONSHIP BETWEEN VERSIONS OF AC_R^0 OVER DIFFERENT INTEGRAL DOMAINS

Intuitively, a circuit deciding a problem in $\text{AC}_{\mathbb{R}}^0$ should also be able to simulate circuits deciding problems in, for example, $\text{AC}_{\mathbb{Z}}^0$ and $\text{AC}_{\mathbb{Z}_2}^0$. Furthermore, we should be able to simulate an $\text{AC}_{\mathbb{Z}_3}^0$ -circuit by an $\text{AC}_{\mathbb{Z}_2}^0$ -circuit by simulating the operations

defined in the integral domain \mathbb{Z}_3 by operations of tuples of \mathbb{Z}_2 . To formalise this intuition, we propose the notion of \mathfrak{C} -simulation maps.

Definition 5.1. Let $f_1, f_2: \mathbb{N} \rightarrow \mathbb{N}$ be two functions and let \mathfrak{C} be a complexity class with $\mathfrak{C} \in \{\text{TIMESPACE}(f_1, f_2), \text{SIZEDEPTH}(f_1, f_2), \text{UNBSIZEDEPTH}(f_1, f_2)\}$.

A \mathfrak{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 $A \in \mathfrak{C}_{R_1^k}$ there exists an $\ell \in \mathbb{N}_{>0}$ and a language $B \in \mathfrak{C}_{R_2^\ell}$ 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 \mathfrak{C} -simulation map from an integral domain R_1 to an integral domain R_2 , we also write $\mathfrak{C}_{R_1} \subseteq_{\text{sim}} \mathfrak{C}_{R_2}$. The relations $=_{\text{sim}}$ and \subsetneq_{sim} are defined analogously. \triangleleft

Similar to the formalism of reductions in classical complexity theory, the relation induced by $\subseteq_{\text{sim}}^{\mathfrak{C}}$ 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 in 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 5.2. *The following three equations hold:*

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

Proof. We split the proof in 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 the circuits simulate are still integral domains.

The finite case. The simulation of finite integral domains is quite trivial. For $\text{NC}_{\mathbb{F}_p}^i \subseteq_{\text{sim}} \text{NC}_{\mathbb{F}_q}^i$ where $p > q$, choose the length of tuples that the $\text{NC}_{\mathbb{F}_q}^i$ -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 $\text{NC}_{\mathbb{Z}}^i =_{\text{sim}} \text{NC}_{\mathbb{Z}[j_k]}^i$ 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 $\text{NC}_{\mathbb{Z}}^i \subseteq_{\text{sim}} \text{NC}_{\mathbb{Z}[j_k]}^i$ is trivial. For 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

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

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:

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

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 at index ℓ is thus

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

Theorem 5.3. *The following three equations hold:*

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

Proof. We use the strategy from the proof of Theorem 5.2, 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. Observe that for unbounded multiplication, simply iterating the binary multiplication would result in linear depth. But similar to the method for binary multiplication, we can compute in advance which values contribute to a given index of the resulting tuple. And since the value of a tuple entry in the result can only depend on the values $x_{i,j}$, to simulate an unbounded multiplication gate, we need only an unbounded multiplication gate with a linear number of predecessors (but possibly exponential increase in the wire complexity). \square

Corollary 5.4. *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 $\mathbb{C} = \mathbb{R}[j_{(2)}]$. \square

Lemma 5.5. *For all $i \in \mathbb{N}$, we have $\text{NC}_{\mathbb{Z}}^i =_{\text{sim}} \text{NC}_{\mathbb{Q}}^i$ and $\text{AC}_{\mathbb{R}}^i =_{\text{sim}} \text{AC}_{\mathbb{C}}^i$.*

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 $\text{NC}_{\mathbb{Q}}^i \subseteq_{\text{sim}} \text{NC}_{\mathbb{Z}}^0$ (resp. $\text{AC}_{\mathbb{Z}}^i \subseteq_{\text{sim}} \text{AC}_{\mathbb{Q}}^i$), take $f(x) := (a, b)$, where $x = \frac{a}{b}$. \square

Lemma 5.6. $\text{NC}_{\mathbb{Q}}^i \subsetneq \text{NC}_{\mathbb{R}}^i$.

Proof. We use the fact that \mathbb{R} is a transcendental field extension of \mathbb{Q} , i. e., 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 can not 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 AC_R^0 and further characterizations for the AC_R and NC_R hierarchies, using a generalization of the GPR operator of Durand, Haak and Vollmer [DHV18]. 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 $AC_{\mathbb{R}}^0 = FO_{\mathbb{R}}[Arb_{\mathbb{R}}] + SUM_{\mathbb{R}} + PROD_{\mathbb{R}}$ holds both non uniformly and for uniformity criteria given by polynomial time computable circuits ($P_{\mathbb{R}}$ -uniform), logarithmic time computable circuits ($LT_{\mathbb{R}}$ -uniform) and first-order definable circuits ($FO_{\mathbb{R}}$ -uniform) [BV21]. We believe that the results we presented hold here in analogous uniform settings as well, though this would need to be further examined.

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) $NC_{\mathbb{R}}^i$ -classes [Cuc92]. The same proof could also hold for the $NC_{\mathbb{C}}^i$ -classes.

Another model deserving of the name “algebraic circuit” are arithmetic circuits in the sense of Valiant [Val79]. This model is similar to the model presented here, with the exception that generally, only addition and multiplication gates are permitted. Maybe the ideas presented in this paper can lead to further insights with regards 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, i. e., 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. We believe that in the algebraic case, it does make a difference which gate type we bound. This model could then possibly be useful to investigate algebraic structures where the respective operations do not adhere to the same axioms.

APPENDIX A. BOUNDING CIRCUIT DEPTH

Definition A.1. 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 . \triangleleft

Lemma A.2. 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 (that is, 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 A.3. 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.

APPENDIX B. GUARDED PREDICATIVE RECURSION

Notation B.1. Similarly to the relativized aggregators in Notation 4.5, 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 a 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 B.2 (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

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

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

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

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

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^{*i*}_{*R*} in Definition 4.6. \triangleleft

REFERENCES

- [Alu09] Paolo Aluffi. *Algebra: Chapter 0*. Graduate studies in mathematics. American Mathematical Society, 2009. ISBN: 9780821847817 (cit. on p. 2).
- [BCS97] Peter Bürgisser, Michael Clausen, and Mohammad Amin Shokrollahi. *Algebraic complexity theory*. Vol. 315. Grundlehren der mathematischen Wissenschaften. Springer, 1997. ISBN: 3-540-60582-7 (cit. on p. 6).
- [Blu+98] Lenore Blum et al. *Complexity and Real Computation*. Springer New York, 1998. ISBN: 978-1-4612-6873-4. URL: <https://doi.org/10.1007/978-1-4612-0701-6> (cit. on pp. 1, 6, 7).
- [Bür13] Peter Bürgisser. *Completeness and reduction in algebraic complexity theory*. Bd. 7. Springer Science & Business Media, 2013. ISBN: 978-3-540-66752-0 (cit. on p. 2).
- [BV21] Timon Barlag and Heribert Vollmer. “A Logical Characterization of Constant-Depth Circuits over the Reals”. In: *Logic, Language, Information, and Computation - 27th International Workshop, WoLLIC 2021, Virtual Event, October 5-8, 2021, Proceedings*. Ed. by Alexandra Silva, Renata Wassermann, and Ruy J. G. B. de Queiroz. Vol. 13038. Lecture Notes in Computer Science. Springer, 2021, pp. 16–30. DOI: 10.1007/978-3-030-88853-4_2. URL: https://doi.org/10.1007/978-3-030-88853-4_2 (cit. on pp. 5, 6, 9, 25).
- [CM99] Felipe Cucker and Klaus Meer. “Logics Which Capture Complexity Classes Over The Reals”. In: *J. Symb. Log.* 64.1 (1999), pp. 363–390. URL: <https://doi.org/10.2307/2586770> (cit. on pp. 7, 9).
- [Cuc92] Felipe Cucker. “ $P_{\mathbb{R}} \neq NC_{\mathbb{R}}$ ”. In: *J. Complexity* 8.3 (1992), pp. 230–238. URL: [https://doi.org/10.1016/0885-064X\(92\)90024-6](https://doi.org/10.1016/0885-064X(92)90024-6) (cit. on pp. 2, 5, 25).
- [DHV18] Arnaud Durand, Anselm Haak, and Heribert Vollmer. “Model-Theoretic Characterization of Boolean and Arithmetic Circuit Classes of Small Depth”. In: *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science. LICS '18*. Oxford, United Kingdom: Association for Computing Machinery, 2018, pp. 354–363. ISBN: 9781450355834. DOI: 10.1145/3209108.3209179. URL: <https://doi.org/10.1145/3209108.3209179> (cit. on pp. 1, 2, 10, 22, 25).
- [GG98] Erich Grädel and Yuri Gurevich. “Metafinite Model Theory”. In: *Inf. Comput.* 140.1 (1998), pp. 26–81. DOI: 10.1006/inco.1997.2675. URL: <https://doi.org/10.1006/inco.1997.2675> (cit. on p. 7).
- [GM95] Erich Grädel and Klaus Meer. “Descriptive complexity theory over the real numbers”. In: *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, 29 May-1 June 1995, Las Vegas, Nevada, USA*. Ed. by Frank Thomson Leighton and Allan Borodin. ACM, 1995, pp. 315–324. DOI: 10.1145/225058.225151. URL: <https://doi.org/10.1145/225058> (cit. on p. 7).
- [Grä+07] Erich Grädel et al. *Finite Model Theory and Its Applications*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2007. ISBN: 978-3-540-00428-8. DOI: 10.1007/3-540-68804-8. URL: <https://doi.org/10.1007/3-540-68804-8> (cit. on p. 7).
- [Imm99] Neil Immerman. *Descriptive complexity*. Graduate texts in computer science. Springer, 1999. ISBN: 978-1-4612-6809-3. DOI: 10.1007/978-1-4612-0539-5. URL: <https://doi.org/10.1007/978-1-4612-0539-5> (cit. on pp. 7, 10, 12).

- [Lan02] Serge Lang. *Algebra*. Springer New York, 2002. DOI: 10.1007/978-1-4613-0041-0. URL: <https://doi.org/10.1007/978-1-4613-0041-0> (cit. on p. 2).
- [Lib04] Leonid Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004. ISBN: 3-540-21202-7. DOI: 10.1007/978-3-662-07003-1. URL: <https://doi.org/10.1007/978-3-662-07003-1> (cit. on p. 7).
- [Val79] Leslie G. Valiant. “Completeness classes in algebra”. In: *Proceedings of the eleventh annual ACM symposium on Theory of computing* (1979) (cit. on pp. 2, 25).
- [Vol99] Heribert Vollmer. *Introduction to Circuit Complexity - A Uniform Approach*. Springer, 1999. ISBN: 978-3-540-64310-4. URL: <https://doi.org/10.1007/978-3-662-0392-0> (cit. on pp. 5, 7).

INSTITUT FÜR THEORETISCHE INFORMATIK, LEIBNIZ UNIVERSITÄT HANNOVER, 30167 HANNOVER, GERMANY

Email address: gaube@thi.uni-hannover.de

Email address: barlag@thi.uni-hannover.de

INSTITUT FÜR THEORETISCHE INFORMATIK, UNIVERSITÄT ZU LÜBECK, 23562 LÜBECK, GERMANY

Email address: fch@tcs.uni-luebeck.de