Model-Theoretic Characterization of Boolean and Arithmetic Circuit Classes of Small Depth*

Arnaud Durand
IMJ-PRG, CNRS UMR 7586
University Paris-Diderot
Paris, France
durand@math.univ-paris-diderot.fr

Anselm Haak
Theoretische Informatik
Leibniz Universität Hannover
Hannover, Germany
haak@thi.uni-hannover.de

Heribert Vollmer
Theoretische Informatik
Leibniz Universität Hannover
Hannover, Germany
vollmer@thi.uni-hannover.de

Abstract

In this paper we give a characterization of both Boolean and arithmetic circuit classes of logarithmic depth in the vein of descriptive complexity theory, i.e., the Boolean classes NC^1, SAC^1 and AC^1 as well as their arithmetic counterparts $\#\mathrm{NC}^1$, $\#\mathrm{SAC}^1$ and $\#\mathrm{AC}^1$. We build on Immerman's characterization of constant-depth polynomial-size circuits by formulae of first-order logic, i.e., $\mathrm{AC}^0 = \mathrm{FO}$, and augment the logical language with an operator for defining relations in an inductive way. Considering slight variations of the new operator, we obtain uniform characterizations of the three just mentioned Boolean classes. The arithmetic classes can then be characterized by functions counting winning strategies in semantic games for formulae characterizing languages in the corresponding Boolean class.

Keywords descriptive complexity, counting classes, finite model theory, arithmetic circuits

1 Introduction

The computational power of arithmetic circuits is of current focal interest in computational complexity theory, see the recent surveys [14, 17] or the continuously updated collection of results at [20]. A number of very powerful techniques to prove lower bounds for such circuits have been developed, however only for quite restricted classes.

A long line of research in computational complexity is to characterize complexity classes in a model-theoretic way. Instead of constructing a computational device such as a Turing machine or a family of circuits *deciding* a language L, a formula is built that *defines* the property of those words in L. Best-known is probably Fagin's Theorem stating that languages in NP are exactly those that can be defined in existential second-order logic. More important for this paper is Immerman's theorem, in which the circuit class AC^0 of all languages decidable by Boolean circuits of polynomial size and constant depth is addressed: Immerman showed that AC^0 equals the class of languages definable by first-order formulae: $\mathrm{AC}^0 = \mathrm{FO}$ [12]. The rationale behind this area of *descriptive complexity*, as it is called, is to characterize complexity classes in a model-theoretic way in order to better understand their structure, and to use logical methods in order to get new insights about the considered classes and, most prominently, to obtain lower bounds, see the monographs [13, 16]. The famous lower bound for AC^0, showing that the parity function cannot be computed by such circuit families [10], was obtained independently by Ajtai [1] in a purely logical way.

For arithmetic circuit classes, only one descriptive complexity characterization is known to date. Generalizing in a sense Immerman's Theorem, it was shown very recently that the class $\#\mathrm{AC}^0$ of those functions from binary words to natural numbers computable by polynomial-size constant-depth arithmetic circuits with plus and times gates is equal to the class of those functions computing winning strategies in semantic games for first-order formulae: $\#\mathrm{AC}^0 = \#\mathrm{Win}\text{-}\mathrm{FO}$ [11]. A different way to view this result is to say that $\#\mathrm{AC}^0$ is the class of functions counting Skolem functions for FO-formulae.

Central for this result is a way of looking at arithmetic computation as a counting process: Say that a *proof tree* of a Boolean circuit C for a given input word w is a minimal subtree (of the circuit unfold into a tree) witnessing that the circuit outputs 1 on input w, and let $\#C(w)$ denote the number of such proof trees. It is folklore that $\#\mathrm{AC}^0$ consists of those functions counting proof trees for AC^0-circuits. To prove the mentioned result from [11], a formula has to be constructed whose number of winning strategies (or, number of Skolem functions) equals the number of proof trees of the original circuit.

The class #P is analogously defined by counting accepting paths of nondeterministic polynomial time Turing machines.

*Partially supported by DFG VO 630/8-1 and the ANR-14-CE25-0017-01 project AGGREG

LICS '18, July 9–12, 2018, Oxford, United Kingdom

ACM ISBN 978-1-4503-5583-4/18/07…$15.00
https://doi.org/10.1145/3209108.3209179

Saluja et al. developed a model theoretic characterization of this class by counting assignments to free relational variables of second-order logic [19]. In analogy to this, the mentioned characterization of $\#AC^0$ counts assignments to free *Skolem functions*; this connection is further explored in [7]. A somewhat different look at descriptive complexity of counting classes is given in [2], using formulae involving summation and multiplication. A characterization of $\#AC^0$ in the same vein was given already in [9].

The aim of this paper is to generalize the above mentioned theorem $\#AC^0 = \#\text{Win-FO}$ from [11] to larger circuit classes, in particular the classes $\#NC^1$, $\#SAC^1$ and $\#AC^1$, defined by families of arithmetic circuits of polynomial size and logarithmic depth with bounded fan-in addition and multiplication gates (for $\#NC^1$), unbounded fan-in addition and bounded fan-in multiplication gates ($\#SAC^1$), and unbounded fan-in addition and multiplication gates ($\#AC^1$), see [21]. The mentioned equality between the value computed by an arithmetic circuit and the number of proof trees of the corresponding Boolean circuit does not only hold in the case of the class AC^0 but is a general observation. Thus, a reasonable roadmap to obtain our generalization seems to be to study logical characterizations of the corresponding decision classes NC^1, SAC^1 and AC^1. Such characterizations can be found in the literature: NC^1 can be characterized by an extension of first-order logic by so called monoidal quantifiers [4], and similarly SAC^1 by extending FO by groupoidal quantifiers [15]. However, for such logics with generalized quantifier the notion of winning strategy is not clear. Following a completely different approach, Immerman extended first-order logic by allowing repeated quantifier blocks and thus characterized AC^1 [12]. Here it can be said that in Immerman's notation, $\#AC^1 = \#\text{Win-FO}[\log]$, but this result cannot be transfered to the other log-depth classes NC^1 and SAC^1. Hence we have to start by developing new logical characterizations for the Boolean classes NC^1, SAC^1 and AC^1.

Inspiration comes from a result by Compton and Laflamme, characterizing NC^1 by FO logic augmented with the RPR-operator allowing us to define relations by a certain kind of linear recursion [6] (RPR stands for relational primitive recursion). This approach does not generalize to the classes SAC^1 and AC^1, though. Also, the number of winning strategies does not seem to be related to the number of proof trees; so again, their approach is not suitable for our aim. Instead, we consider a new operator, called GPR ("guarded predicative recursion"), allowing us to define relations by a certain kind of parallel recursion. We show that FO(GPR), first-order logic augmented by GPR, characterizes AC^1, and that slight modifications of the GPR-operator lead to characterizations of NC^1 and SAC^1. In a second step, we show that these characterizations are in a sense "close enough" to the circuit world to mirror the process of counting proof trees by counting winning strategies in semantic games.

Our paper is structured as follows. In the next section, we will give the necessary preliminaries from first order logic and circuit complexity including the respective counting mechanism. In Sect. 3 we briefly recall the result by Compton and Laflamme and then introduce our inductive operator GPR. To demonstrate suitability of our logical approach, we give an example of a formula defining an AC^1-complete problem. We then prove our main results: In Sect. 4 we characterize the Boolean classes NC^1, SAC^1 and AC^1 in a model-theoretic way by first-order logic with different forms of the GPR-operator. This is the technically most demanding part of our paper. We would like to stress that our proofs are completely different from the one for the mentioned result from Compton and Laflamme [6]. In Sect. 5 we characterize the arithmetic classes $\#NC^1$, $\#SAC^1$ and $\#AC^1$ by counting winning-strategies in semantic games for the above logics. Finally, we conclude with a summary and some open problems.

2 Preliminaries

In this paper we will use first-order logic FO with usual syntax and semantics, see, e.g., [8]. $\mathrm{SF}(\varphi)$ denotes the set of all subformulae of formula φ. We consider finite σ-structures where σ is a finite vocabulary consisting of relation and constant symbols. For structure $\mathcal{A}$, $\mathrm{dom}(\mathcal{A})$ denotes its universe. We will always use structures with universe $\{0, 1, \ldots, n-1\}$ for some $n \in \mathbb{N} \setminus \{0\}$. Furthermore, we will always assume that our structures contain the *built-in relation* BIT^2, which is implicitly interpreted in the expected way: $\mathrm{BIT}(i, j)$ is true, iff the i'th bit of the binary representation of j is 1, where the 0'th bit is the LSB. When talking about structures with built-in relations, $\models$ includes the interpretation of the built-in relations in the intended way.

We assume the standard encoding of structures as binary strings (see, e.g., [13]): Relations are encoded row by row by listing their truth values as 0's and 1's. Constants are encoded by the binary representation of their value and thus a string of length $\lceil \log_2(n) \rceil$. A whole structure is encoded by the concatenation of the encodings of its relations and constants except for numerical predicates and constants: These are not encoded, because they are determined by the input length.

Since we want to talk about languages accepted by Boolean circuits, we will need the vocabulary

$$\tau_{\text{string}} = (\leq^2, S^1)$$

of binary strings. A binary string is represented as a structure over this vocabulary as follows: Let $w \in \{0, 1\}^*$ with $|w| = n$. Then the structure representing this string is the structure with universe $\{0, \ldots, n-1\}$, $\leq^2$ interpreted as the $\leq$-relation on $\mathbb{N}$ restricted to the universe and $x \in S$, iff the x'th bit of w is 1. The structure corresponding to string w is denoted by $\mathcal{A}_w$. Also, by the above, w is the encoding of structure $\mathcal{A}_w$. We will often use the natural extensions of $=$ and $\leq$ to tuples, denoting these by the same symbols. We denote by FO not only the set of first-order formulae, but also the complexity class of all languages definable in first-order logic with built-in BIT:

Definition 2.1. A language $L \subseteq \{0,1\}^*$ is in FO if there is an FO-formula φ over vocabulary $\tau_{\text{string}} \cup (\text{BIT}^2)$ such that for all $w \in \{0,1\}^*$:

$$w \in L \Leftrightarrow \mathcal{A}_w \models \varphi.$$

We will also use relativized quantifiers. A relativization of a quantifier is a formula restricting the domain of elements considered for that quantifier. We also use these for quantification of tuples. More precisely, 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, respectively,

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

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

Furthermore, we consider bounded variants of relativized quantifiers, that is, quantifiers where we only consider the maximal two elements meeting the condition expressed by the relativization. Notation: $\exists_\text{b}$, $\forall_\text{b}$. Formally, the semantics for $\exists_\text{b}$ can be given in FO as follows:

$$\begin{aligned}(\exists_\text{b}(\overline{x}).\varphi(\overline{x})\ \psi(\overline{x}) \equiv \Big(\exists\overline{x}.\big(\varphi(\overline{x})\wedge \\ &\forall\overline{y}\forall\overline{z}\ (\overline{y}\neq\overline{z}\wedge\overline{x}<\overline{y}\wedge\overline{x}<\overline{z}) \\ &\quad\rightarrow(\neg\varphi(\overline{y})\vee\neg\varphi(\overline{z}))\big)\Big)\ \psi(\overline{x})\end{aligned}$$

The semantics for $\forall_\text{b}$ is defined analogously.

For the definition of uniform circuit families we will need FO-interpretations, which are mappings between structures over different vocabularies.

Definition 2.2. Let σ, τ be vocabularies, $\tau = (R_1^{a_1},\dots,R_r^{a_r})$. A first-order interpretation (or FO-interpretation)

$$I : \text{STRUC}[\sigma] \rightarrow \text{STRUC}[\tau]$$

is given by a tuple of FO-formulae $\varphi_0, \varphi_1, \dots, \varphi_r$ over vocabulary σ. For some k, φ_0 has k free variables and φ_i has $k \cdot a_i$ free variables for all $i \geq 1$. For each structure $\mathcal{A} \in \text{STRUC}[\sigma]$, these formulae define the structure

$$I(\mathcal{A}) = \langle |I(\mathcal{A})|, R_1^{I(\mathcal{A})}, \dots, R_r^{I(\mathcal{A})}\rangle \in \text{STRUC}[\tau],$$

where the universe is defined by φ_0 and the relations by $\varphi_1,\dots,\varphi_r$ in the following way:

$$|I(\mathcal{A})| = \{\langle b^1,\dots,b^k\rangle \mid \mathcal{A} \models \varphi_0(b^1,\dots,b^k)\}$$

$$R_i^{I(\mathcal{A})} = \{(\overline{b}_1,\dots\overline{b}_{a_i}) \in |I(\mathcal{A})|^{a_i} \mid \mathcal{A} \models \varphi_i(\overline{b}_1,\dots\overline{b}_{a_i})\},$$

where $\overline{b}_i = \langle b_i^1,\dots,b_i^k\rangle$ for all i.

For better readability, we will write $\varphi_{\text{universe}}$ instead of φ_0 and φ_{R_i} instead of φ_i for all i.

We will next recall the definition of Boolean circuits and complexity classes defined using them. A circuit is a directed acyclic graph (dag), whose nodes (also called gates) are marked with either a Boolean function (in our case $\wedge$ or $\vee$), a constant (0 or 1), or a (possibly negated) bit of the input. Also, one gate is marked as the output gate. On any input, a circuit computes a Boolean function by evaluating all gates according to what they are marked with. The value of the output gate then is the function value for that input. If C is a circuit, we denote the function it computes by $C(x)$. When we want circuits to work on different input lengths, we have to consider families of circuits: A family contains a circuit for any input length $n \in \mathbb{N}$. Families of circuits allow us to talk about languages being accepted by circuits: A circuit family $C = (C_n)_{n\in\mathbb{N}}$ is said to accept (or decide) the language L, if it computes its characteristic function c_L:

$$C_{|x|}(x) = c_L(x) \text{ for all } x.$$

The complexity classes in circuit complexity are classes of languages that can be decided by circuit families with certain restrictions on their resources. The resources relevant here are depth, size and fan-in (number of children) of gates. The depth here is the length of a longest path from any input gate to the output gate of a circuit and the size is the number of non-input gates in a circuit. Depth and size of a circuit family are defined as functions accordingly.
Above, we have not restricted the computability of the circuit $C_{|x|}$ from x in any way. This is called non-uniformity, which allows such circuit families to even compute non-recursive functions. Since we want to capture a kind of efficient computability, we need some notion of uniformity. For this, we first define the vocabulary for Boolean circuits as FO-structures:

$$\tau_{\text{circ}} = (E^2, G_\wedge^1, G_\vee^1, \text{Input}^2, \text{negatedInput}^2, \text{output}^1),$$

where the relations are interpreted as follows:

- $E(x,y)$: gate y is a child of gate x
- $G_\wedge(x)$: gate x is an and-gate
- $G_\vee(x)$: gate x is an or-gate
- Input(x,i): gate x is an input gate associated with the i'th input bit
- negatedInput(x,i): gate x is a negated input gate associated with the i'th input bit
- output(x): gate x is the output gate

We will now define FO-uniformity of Boolean circuits and the complexity classes relevant in this paper.

Definition 2.3. A circuit family $C = (C_n)_{n\in\mathbb{N}}$ is first-order uniform (FO-uniform) if there is an FO-interpretation

$$I : \text{STRUC}[\tau_{\text{string}} \cup (\text{BIT}^2)] \rightarrow \text{STRUC}[\tau_{\text{circ}}]$$

mapping any structure $\mathcal{A}_w$ over τ_{string} with built-in BIT to the circuit $C_{|w|}$ given as a structure over the vocabulary τ_{circ}.

Note that implicitly, the mapping I only depends on the length of the input and not the input bits. Furthermore, this uniformity by [3] coincides with the maybe better known DLOGTIME-uniformity for many familiar circuit classes (and in particular for all classes studied in this paper). All circuit classes we consider in this paper are FO-uniform.

Definition 2.4. A language $L \subseteq \{0,1\}^*$ is in AC^i if there is an FO-uniform circuit family with depth $O((\log n)^i)$ accepting L. NC^i is defined analogously with bounded fan-in gates. SAC^i is defined analogously with bounded fan-in $\wedge$-gates and unbounded fan-in $\vee$-gates.

We will also call circuit families with the above restrictions on their resources AC^i, NC^i and SAC^i circuit families, respectively. Observe that these kind of circuit families always have polynomial size, since their gates are tuples of elements of the first-order input structure.

For this paper, the classes AC^0, AC^1, NC^1 and SAC^1 are of particular interest. It is known that the class AC^0 coincides with the class FO [4, 13]: $\mathrm{AC}^0 = \mathrm{FO}$.

We will next define counting variants of the above classes. The idea for counting classes in general is to use a model of computation and identify a kind of witness for acceptance in that model. For a nondeterministic Turing machine, we usually consider the accepting paths on a given input as witnesses. Considering polynomial time computations, this concept gives rise to the class #P. A witness that a Boolean circuit accepts its input is a so-called *proof tree*: a minimal subtree of the circuit showing that it evaluates to true for a given input. For this, we first unfold the given circuit into tree shape, and we further require that it is in *negation normal form* (meaning that negations only occur directly in front of literals)—note that this is always the case for τ_{circ}-structures. A proof tree then is a subtree that contains the output gate, for every included $\vee$-gate exactly one child and for every included $\wedge$-gate all children, such that every input gate which we reach in this way is a true literal. This allows us to define the following counting complexity classes:

Definition 2.5. A function $f\colon \{0,1\}^* \to \mathbb{N}$ is in $\#\mathrm{AC}^i$ (resp., $\#\mathrm{NC}^i$, $\#\mathrm{SAC}^i$) if there is an AC^i (NC^i, SAC^i) circuit family $C = (C_n)_{n\in\mathbb{N}}$ such that for all $w \in \{0,1\}^*$,

$$f(w) = \text{number of proof trees of } C_{|w|}(w).$$

Remark 1. *Alternative definitions of these classes are in terms of arithmetic circuits: An arithmetic circuit is a circuit with gates for addition and multiplication. When using these arithmetic circuits with binary inputs and apply the respective resource bounds, we get the same classes as by counting proof trees in Boolean circuits. For details, see, e.g., [21]. In this paper, we will use both definitions depending on the context.*

Note that, already at the first level, the classes $\#\mathrm{AC}^1$, $\#\mathrm{NC}^1$, $\#\mathrm{SAC}^1$, though based on relatively close circuit classes, have a rather different computational power. It can be seen, through the connections between SAC^1 circuits and multiplicatively disjoint circuits (see [18]) that $\#\mathrm{NC}^1$ and $\#\mathrm{SAC}^1$ are subclasses of #P. On the contrary, the class $\#\mathrm{AC}^1$ can output numbers bigger than $2^{n^{\log n}}$ for input of size n, hence numbers of super-polynomial sizes in n. This comes from the fact that the unfolding of a polynomial size, logarithmic depth circuit with unbounded fan-in may be of size $n^{O(\log n)}$. This means that $\#\mathrm{AC}^1 \nsubseteq \#\mathrm{P}$.

Similarly, one can identify witnesses for acceptance in first-order logic. One possibility is to do this in terms of the two-player model-checking game for FO. In this game, player 1 and player 2 play against each other. The game starts on the whole formula. From there, depending on the outermost operator or quantifier (and the current roles of the players, more on this later) a certain player makes a choice on how to continue.

For a formula without negations the goal of player 1 is to reach an atom that is satisfied by the input structure (we call this role *verifier*) and the goal of player 2 is to reach an atom that is not satisfied by the input structure (we call this role *falsifier*). If negations occur, the players swap their roles whenever a negation is encountered.

We now want to define the game more precisely. Since for our purpose strategies in this game can vary over different occurrences of identical subformulae, we assume a syntax tree representation for formulae, that is, each formula φ is given as $\varphi = (V, E, r, L)$ where (V, E, r) is a rooted directed tree and L is a labeling function assigning to each node in V either one of the Boolean operators $\wedge$, $\vee$ and $\neg$, a quantifier $\exists x$ or $\forall x$ for any variable x or an atom $R(\overline{x})$ where R is a relation symbol from the appropriate vocabulary and $\overline{x}$ is a tuple of variables. The current subformula is given by a node in this syntax tree, allowing a distinction between identical subformulae.

Let $\varphi = (V, E, r, L)$ be an FO-formula over vocabulary σ represented as a syntax tree and $\mathcal{A} \in \mathrm{STRUC}[\sigma]$. Then configurations of the game for $\mathcal{A} \vDash \varphi$ are of the form

$$(\varphi = (V, E, r, L), \mathcal{A}, v, \theta, s)$$

where $v \in V$, θ is an assignment from the free variables in the subformula of φ rooted in v to elements of $\mathrm{dom}(\mathcal{A})$ and s is a bit specifying whether the players have currently swapped roles: If $s = 0$, player 1 has role verifier and player 2 has role falsifier and if $s = 1$ the roles are reversed.

The game starts in config

$$(\varphi = (V, E, r, L), \mathcal{A}, r, \theta_0, 0)$$

where θ_0 is the function with empty domain. The game proceeds as follows: Let

$$(\varphi = (V, E, r, L), \mathcal{A}, v, \theta, s)$$

be a configuration of the game for $\mathcal{A} \vDash \varphi$. Depending on the label $L(v)$, the next move is either fixed or a specific player can make a choice as follows: If $L(v)$ is …

- $\vee$: Player with role verifier chooses child v' of v in (V, E, r), game continues in $(\varphi, \mathcal{A}, v', \theta, s)$
- $\wedge$: Player with role falsifier chooses child v' of v in (V, E, r), game continues in $(\varphi, \mathcal{A}, v', \theta, s)$
- $\neg$: Game continues in $(\varphi, \mathcal{A}, v', \theta, 1 - s)$ where v' is the single child of v in (V, E, r)
- $\exists x$ where x is a variable: Player with role verifier chooses a value $c \in \text{dom}(\mathcal{A})$; game continues in configuration $(\varphi, \mathcal{A}, v', \theta', s)$ where v' is the single child of v in (V, E, r) and θ is the function defined on the set of free variables in the subformula of φ rooted in v' with $\theta'(y) = \theta(y)$ for all variables $y \neq x$ and $\theta'(x) = c$
- $\forall x$ where x is a variable: Player with role falsifier chooses a value $c \in \text{dom}(\mathcal{A})$; game continues in configuration $(\varphi, \mathcal{A}, v', \theta', s)$ where v' is the single child of v in (V, E, r) and θ is the function defined on the set of free variables in the subformula of φ rooted in v' with $\theta'(y) = \theta(y)$ for all variables $y \neq x$ and $\theta'(x) = c$

In this game, player 1 wins if and only if $L(v)$ is an atom and either $s = 0$ and $\mathcal{A} \vDash L(v)$ or $s = 1$ and $\mathcal{A} \nvDash L(v)$.

A strategy for player 1 in this game is a function mapping configurations to specific choices in such a way that a choice is specified for all configurations that give player 1 a choice and are reachable from the starting configuration if player 1 acts according to this strategy and player 2 makes arbitrary choices. This can be made formal by defining the configuration tree of the game. Strategies of player 1 then are subtrees, which contain the root and contain exactly one child of each configuration that gives player 1 a choice and all children of configurations that do not give player 1 a choice.

A winning strategy of player 1 is a strategy which lets player 1 win independently of the choices of player 2.

Now player 1 has a winning strategy in this game if and only if $\mathcal{A} \vDash \varphi$. This means that winning strategies in this game can be seen as witnesses for acceptance in first-order logic, which allows us to define a counting class based on FO.

Definition 2.6. A function $f\colon \{0,1\}^* \to \mathbb{N}$ is in #Win-FO, if there is an FO-formula φ over vocabulary $\tau_{\text{string}} \cup (\text{BIT}^2)$ such that for all $w \in \{0,1\}^*$:

$$f(w) = \#\text{Win}(\varphi, \mathcal{A}_w),$$

where $\#\text{Win}(\varphi, \mathcal{A}_w)$ is the number of winning strategies of player 1 in the model checking game for $\mathcal{A}_w \vDash \varphi$.

As was shown in [11], the counting versions of AC^0 and FO coincide, i.e.: $\#\text{AC}^0 = \#\text{Win-FO}$ (note that the definition of the model checking game was different there, but the resulting counting classes are the same).

For the quantifiers $\exists_\text{b}$ and $\forall_\text{b}$ we define the following rules in the model checking game for FO: Here, the choosing player is restricted to the maximal two elements satisfying the relativization.

3 GPR

We aim to characterize counting classes from circuit complexity beyond $\#\text{AC}^0$ by counting winning strategies in different logics. It has been proved in [6] that NC^1 can be characterized using FO with a certain kind of linear recursion, called relational primitive recursion (RPR). It allows the recursive definition of predicates in the following way:

$$[P(\overline{x}, y) \equiv \theta(\overline{x}, y, P(\overline{x}, y - 1))]$$

where $y - 1$ is a shorthand for a term thas is uniquely determined to be equal to $y - 1$, which is definable since we have arithmetic. Also, by $\theta(\overline{x}, y, P(\overline{x}, y - 1))$ we mean that θ has free variables $\overline{x}$ and y and can contain $P(\overline{x}, y)$ as an atom, but P is not allowed in θ in any other form. Semantically, this means that $P(\overline{x}, y)$ is defined recursively to have the same truth value as $\theta(\overline{x}, y, P(\overline{x}, y - 1))$ for $y > 0$ and $P(\overline{x}, 0)$ is defined to be equivalent to $\theta(\overline{x}, 0, \bot)$. FO(RPR) denotes the class of languages definable by formulae of the form:

$$[P(\overline{x}, y) \equiv \theta(\overline{x}, y, P(\overline{x}, y - 1))]\ \varphi(P)$$

where $\varphi(P)$ is first-order and makes use of the inductively defined P. Over structures with built-in BIT, it holds that $\text{NC}^1 = \text{FO(RPR)}$ [6]. This characterization does not immediately generalize to the classes SAC^1 and AC^1 as well as counting classes. However, inspired by this, we define a different kind of inductive definition called **g**uarded **p**redicative **r**ecursion, GPR for short, that allows us to capture all these classes in a unified way.

Definition 3.1 (GPR)**.** FO(GPR) is the set of all formulae φ of the following form:

$$\varphi ::= [P(\overline{x}, \overline{y}) \equiv \theta(\overline{x}, \overline{y}, P)]\ \varphi(P) \mid \psi$$

where $\psi \in \text{FO}$ and $\theta \in \text{FO}$ with free variables $\overline{x}, \overline{y}$ such that each atomic sub-formula involving symbol P

1. is of the form $P(\overline{x}, \overline{z})$ where $\overline{z}$ is in the scope of a guarded quantification $Q\overline{z}.(\overline{z} < \overline{y}/2 \wedge \xi(\overline{y}, \overline{z}))$ with $Q \in \{\forall, \exists\}$, $\xi \in \text{FO}$ and
2. never occurs in the scope of any quantification not guarded in this way, that is:
   For all $\psi \in \text{SF}(\theta)$ starting with a quantifier not guarded in the above way, it holds that $P(\overline{x}, \overline{z}) \notin \text{SF}(\psi)$.

Similar to the definition of RPR, we use $\overline{z} < \overline{y}/2$ as a readable shorthand for

$$\exists \overline{a}\ 2 \cdot \overline{z} = \overline{a} \wedge \overline{a} < \overline{y}.$$

Here, we use the extensions of order and arithmetic to tuples, which are FO-definable (see, e.g., [13]). We call the part in $[\,\cdot\,]$ a GPR-operator. We define $\text{FO(GPR}_{\text{bound}})$ similarly by allowing only bounded variants for guarded quantifications $Q_b\overline{z}.(\overline{z} < \overline{y}/2 \wedge \xi(\overline{y}, \overline{z}))$ and $\text{FO(GPR}_{\text{semi}})$ for which universal guarded $\forall \overline{z}.(\overline{z} < \overline{y}/2 \wedge \xi(\overline{y}, \overline{z}))$ and bounded existential guarded $\exists_b \overline{z}.(\overline{z} < \overline{y}/2 \wedge \xi(\overline{y}, \overline{z}))$ quantifications are allowed.

This approach is flexible enough to easily express problems computable by small circuit classes.

Example 3.2. The SHORTCAKE problem was defined and proven to be AC^1-complete in [5]. We use a variation of this game which can be shown to still be AC^1-complete with the same proof. Our variation is defined as follows:

Two players, H (or 0) and V (or 1) are alternately moving a token on an $n \times n$ Boolean matrix M. A configuration of the game is a contiguous submatrix of M given by the indices of its first and last rows and columns (i_0, i_1, j_0, j_1) as well as a bit specifying the player whose turn it is. The token is always on the top-left corner of the current submatrix. Due to this we do not need to explicitly store the token location. In the beginning the submatrix is given by $(1, n, 1, n)$ and H starts to play.

In his turn, H tries to move the token horizontally in the submatrix to some entry (i_0, j), $j_0 < j \leq j_1$ satisfying $M_{i_0,j} = 1$. After H's move either all columns to the left of j or all columns to the right of j are removed from the current submatrix, whichever number of columns is greater. The token is then placed on the top-left corner of the current submatrix. This means that the new submatrix is given by

$$\begin{array}{ll} (i_0, i_1, j, j_1), & \text{if } (j_1 + j_0)/2 \leq j \leq j_1 \text{ and} \\ (i_0, i_1, j_0, j), & \qquad\text{otherwise} \end{array}$$

In his turn, V plays similarly but vertically on the rows. The first player with no move left loses.

In order to give an FO(GPR)-formula expressing whether H has a winning strategy in this game, we encode the matrix by a binary word of length n^2. Note that the size of the matrix is reduced to half of the previous size in each turn.

The following formula expresses the existance of a winning strategy. Here, s is an upper bound for the size of the matrix in each step with some padding and p encodes the current player.

$$\Phi := \Big[P(x, s, i_0, i_1, j_0, j_1, p) \equiv$$
$$p = 0 \wedge \theta_H(x, \overline{y}) \vee p = 1 \wedge \theta_V(x, \overline{y}) \Big] \; \varphi(P)$$

with $\varphi(P) \equiv P(n, 2n^2, 1, n, 1, n, 0)$ and θ_H and θ_V specifying the possible moves of the players. For this, θ_H can be chosen as

$$\theta_H(x, \overline{y}) \equiv \exists \overline{z} = (s', i'_0, i'_1, j'_0, j'_1, p').(\overline{z} < \overline{y}/2)$$
$$(s' \leq s/2 - 2) \wedge p' = 1 \wedge i'_0 = i_0 \wedge i'_1 = i_1 \wedge$$
$$\Big[\Big(j'_0 \neq j_0 \wedge j'_1 = j_1 \wedge (j_1 + j_0)/2 \leq j'_0 \leq j_1 \wedge M(i'_0 \cdot n + j'_0) \Big) \vee$$
$$\Big(j'_0 = j_0 \wedge j'_1 \neq j_0 \wedge j_0 \leq j'_1 < (j_1 + j_0)/2 \wedge M(i'_0 \cdot n + j'_1) \Big) \Big] \wedge$$
$$P(x, s', i'_0, i'_1, j'_0, j'_1, p')$$

By using s we can ensure that the numerical value of the tuple decreases to less than half of the previous value in each step. The formula $\theta_V(x, \overline{y})$ is defined analogously with universal guarded quantification instead of the existential one.

In [5] a variant of this game, called SEMICAKE, is shown to be SAC^1-complete. It is easily definable along the same lines in $\mathrm{FO}(\mathrm{GPR}_{\mathrm{semi}})$.

We now introduce a certain normal-form for circuits showing membership in NC^1, SAC^1 and AC^1, which will be needed for our later proofs. Note that for FO-uniform circuits there is an inherent interpretation of gates as natural numbers due to built-in BIT in the logical language and its extension to tuples. This also means there is an order and arithmetic for gates using their encoding in the logic. This allows us to define the following normal-form for circuit families: All tuples of the appropriate size are gates (so $\varphi_{\text{universe}}$ from the FO-interpretation showing uniformity is always true). The $\wedge$-gates are exactly the gates that are odd and neither input nor negated input gates. The $\vee$-gates are exactly the gates that are even and neither input nor negated input gates. Children of gates are smaller than half of each of their parents.

Lemma 3.3. *Let* $\mathfrak{C} \in \{\mathrm{NC}^1, \mathrm{SAC}^1, \mathrm{AC}^1, \#\mathrm{NC}^1, \#\mathrm{SAC}^1, \#\mathrm{AC}^1\}$ *and* $L \in \mathfrak{C}$. *Then there is an* FO*-interpretation*

$$I : \mathrm{STRUC}[\tau_{\text{string}}] \cup (\mathrm{BIT}^2) \longrightarrow \mathrm{STRUC}[\tau_{\text{circ}}]$$

with tuple size $k \in \mathbb{N}$ *that uniformly describes a circuit family showing* $L \in \mathfrak{C}$ *such that for all* $w \in \{0, 1\}^*$*:*

1. $\mathrm{dom}(I(\mathcal{A}_w)) = \mathrm{dom}(\mathcal{A}_w)^k$
2. *for all* $\overline{x} \in \mathrm{dom}(I(\mathcal{A}_w))$*:*
   $G_\wedge^{I(\mathcal{A}_w)}(\overline{x})$ *if and only if*
   $(\neg\mathrm{Input}^{I(\mathcal{A}_w)}(\overline{x}) \wedge \neg\mathrm{negatedInput}^{I(\mathcal{A}_w)}(\overline{x}) \wedge \overline{x} \text{ is odd})$
   and
   $G_\vee^{I(\mathcal{A}_w)}(\overline{x})$ *if and only if*
   $(\neg\mathrm{Input}^{I(\mathcal{A}_w)}(\overline{x}) \wedge \neg\mathrm{negatedInput}^{I(\mathcal{A}_w)}(\overline{x}) \wedge \overline{x} \text{ is even})$
3. *for all* $\overline{x}, \overline{y} \in \mathrm{dom}(I(\mathcal{A}_w))$*:*
   $E^{I(\mathcal{A}_w)}(\overline{x}, \overline{y}) \Rightarrow$ *Numerically,* $\overline{y}$ *is at most half of* $\overline{x}$

Proof. Properties 1 and 2 are straightforward. For property 3, a certain unary encoding of the depth can be added to the encoding of gates in order to halve the numerical value of gates in each step from parent to child.

A formal proof can be found in the full version. □

4 Logical Characterizations of Small Depth Decision Classes

We now show that the newly defined logics characterize the classes NC^1, SAC^1 and AC^1, respectively.

Theorem 4.1.

1. $\mathrm{NC}^1 = \mathrm{FO}(\mathrm{GPR}_{\mathrm{bound}})$
2. $\mathrm{SAC}^1 = \mathrm{FO}(\mathrm{GPR}_{\mathrm{semi}})$
3. $\mathrm{AC}^1 = \mathrm{FO}(\mathrm{GPR})$

Proof. $\underline{\mathrm{AC}^1 \subseteq \mathrm{FO}(\mathrm{GPR})}$: Let $L \in \mathrm{AC}^1$ via the FO-uniform AC^1 circuit family $C = (C_n)_{n \in \mathbb{N}}$ with the properties from Lemma

3.3 and C_n has depth at least 1 for all n. The latter can easily be achieved by adding a new $\wedge$-gate as output-gate with the old output-gate being its only child. Let

$$I = (\varphi_{\text{universe}}, \varphi_{G_\wedge}, \varphi_{G_\vee}, \varphi_{\text{Input}}, \varphi_{\text{negInput}}, \varphi_E, \varphi_{\text{output}})$$

be an FO-interpretation showing that C is uniform. Here, negInput is short for negatedInput. Furthermore, let

$$\varphi_{\text{Literal}}(\overline{x}) := \exists\overline{i}(\varphi_{\text{Input}}(\overline{x}, \overline{i}) \vee \varphi_{\text{negInput}}(\overline{x}, \overline{i})),$$

$$\varphi_{\text{trueLiteral}}(\overline{x}) := \exists\overline{i}(\varphi_{\text{Input}}(\overline{x}, \overline{i}) \wedge S(\overline{i}) \vee \varphi_{\text{negInput}}(\overline{x}, \overline{i}) \wedge \neg S(\overline{i})),$$

$$\psi(\overline{z}, P(\overline{z})) = \big(P(\overline{z}) \wedge \neg\varphi_{\text{Literal}}(\overline{z})\big) \vee \varphi_{\text{trueLiteral}}(\overline{z}).$$

Then the following FO(GPR)-formula defines L:

$$\Phi := [P(\overline{y}) \equiv \theta(\overline{y}, P)]\ \exists\overline{o}(\varphi_{\text{output}}(\overline{o}) \wedge P(\overline{o}))$$

with

$$\begin{aligned}\theta(\overline{y}, P) :=& \Big(\text{Even}(\overline{y}) \wedge \\ & \quad \big((\exists\overline{z}.(\overline{z} < \overline{y}/2 \wedge \varphi_E(\overline{y}, \overline{z})))\ \psi(\overline{z}, P(\overline{z}))\big)\Big) \vee \\ & \Big(\text{Odd}(\overline{y}) \wedge \\ & \quad \big((\forall\overline{z}.(\overline{z} < \overline{y}/2 \wedge \varphi_E(\overline{y}, \overline{z})))\ \psi(\overline{z}, P(\overline{z}))\big)\Big).\end{aligned}$$

Even and Odd check the parity of the least significant bit within the least significant variable within tuple $\overline{y}$ using BIT.

Since $\text{Odd}(\overline{y}) \equiv \neg\text{Even}(\overline{y})$, we can write θ as

$$\begin{aligned}\theta(\overline{y}, P) \equiv & \big(Q\overline{z}.(\overline{z} < \overline{y}/2 \wedge \varphi_E(\overline{y}, \overline{z}))\big) \\ & \big(P(\overline{z}) \wedge \neg\varphi_{\text{Literal}}(\overline{z})\big) \vee \varphi_{\text{trueLiteral}}(\overline{z})\end{aligned}$$

where Q is either $\exists$ or $\forall$ depending on the parity of $\overline{y}$.

Let $n \in \mathbb{N}$ and $w \in \{0, 1\}^n$. We now prove that the predicate P in the above formula (as defined by the GPR-operator) is the valuation for the gates in circuit C_n. By definition, on input structure $\mathcal{A}_w$, the formulae from I used above give access to C_n. We prove inductively that for any $k \in \mathbb{N}$, $P(\overline{g})$ gives the value of gate $\overline{g}$ in C_n on input w if all children of $\overline{g}$ have depth $\leq k$.

$k = 0$: Note that $\varphi_{\text{trueLiteral}}(\overline{h})$ gives the value of $\overline{h}$ in C_n on input w if $\overline{h}$ is an input gate. Then for gates $\overline{g}$ all children of which are input gates we have:

$$\begin{aligned}P(\overline{g}) \equiv & \big(Q\overline{z}.(\overline{z} < \overline{g}/2 \wedge \varphi_E(\overline{g}, \overline{z}))\big) \\ & \big(\underbrace{P(\overline{z}) \wedge \underbrace{\neg\varphi_{\text{Literal}}(\overline{z})}_{\text{false}}}_{\text{false}} \vee \varphi_{\text{trueLiteral}}(\overline{z})\big). \qquad (\star)\end{aligned}$$

By assumption, if $\varphi_E(\overline{g}, \overline{z})$ then $\overline{z} < \overline{g}/2$, and thus

$$\overline{z} < \overline{g}/2 \wedge \varphi_E(\overline{g}, \overline{z}) \equiv \varphi_E(\overline{g}, \overline{z}).$$

This yields

$$P(\overline{g}) \equiv \big(Q\overline{z}.\varphi_E(\overline{g}, \overline{z})\big)\ \varphi_{\text{trueLiteral}}(\overline{z})$$

This means that P actually gives the value of $\overline{g}$.

$k \longrightarrow k + 1$: Again, by assumption,

$$\overline{z} < \overline{g}/2 \wedge \varphi_E(\overline{g}, \overline{z}) \equiv \varphi_E(\overline{g}, \overline{z}).$$

We also know that for all children $\overline{z}$ of $\overline{g}$ only two cases can occur:

If $\overline{z}$ is an input gate, then $\neg\varphi_{\text{Literal}}(\overline{z})$ is false and the truth value of $\varphi_{\text{trueLiteral}}(\overline{z})$ is the value held by gate $\overline{z}$.

If $\overline{z}$ is not an input gate, then $\varphi_{\text{trueLiteral}}(\overline{z})$ is false, $\neg\varphi_{\text{Literal}}$ is true and $P(\overline{z})$ gives the value of $\overline{z}$ by induction hypothesis.

By ($\star$) this means that $P(\overline{g})$ actually gives the value of $\overline{g}$.

Since P gives the value of arbitrary non-input gates in C_n on input w for any n and we assumed that the output gate is not an input gate, it is easy to see that the above formula defines L: The formula behind the recursive definition of P simply states that the output gate of the circuit evaluates to true.

$\underline{\text{FO(GPR)} \subseteq \text{AC}^1}$: At first assume that only one occurrence of GPR-operators is allowed. The proof easily extends to the general case. Furthermore, we begin by proving the result without negations in θ. We will explain how to handle arbitrary FO(GPR)-formulae afterwards.

Let $L \in$ FO(GPR) via the formula

$$[P(\overline{x}, \overline{y}) \equiv \theta(\overline{x}, \overline{y}, P)]\ \varphi(P).$$

By definition of FO(GPR), P occurs in θ only in the form $P(\overline{x}, \overline{z})$, where $\overline{z}$ is in the scope of a guarded quantification $Q\overline{z}.(\overline{z} < \overline{y}/2)$ with $Q \in \{\exists, \forall\}$ and not in the scope of any unguarded quantification. Ignoring occurrences of P, φ is an FO-formula. Hence, we can build an AC^0 circuit family evaluating φ except for these occurrences.

In order to compute the predicate P we proceed as follows: θ is also an FO-formula except for occurrences of P, so we can build for all $\overline{x}, \overline{y}$ an AC^0 circuit that computes $\theta(\overline{x}, \overline{y}, P)$ with certain input gates marked with $P(\overline{x}, \overline{z})$. The circuit can easily be built in a way that $\overline{z}$ is part of the encoding of gates that are marked with $P(\overline{x}, \overline{z})$. Thus, we can remove the marks and instead connect each gate that was marked with $P(\overline{x}, \overline{z})$ to the output gate of the subcircuit computing $\theta(\overline{x}, \overline{z}, P)$. Since occurrences of $P(\overline{x}, \overline{z})$ only occur within guarded quantifications $Q\overline{z}.(\overline{z} < \overline{y}/2)$, there can be at most logarithmically many steps from any $P(\overline{x}, \overline{y})$ before reaching $P(\overline{x}, \overline{0})$, terminating the recursion. By the above, each such step—computing $P(\overline{x}, \overline{y})$, when given values of $P(\overline{x}, \overline{z})$ for certain $\overline{z}$—can be done in constant depth leading to logarithmic depth in total.

The gates computing values of P can now be connected to the AC^0 circuit family evaluating φ as needed. This leads to an AC^1 circuit family evaluating the whole formula.

Next, we talk about the case of θ containing negations. For this, we use the same construction as above, but add a negated version of each gate. We do this by adding a negation-bit to the encoding of all gates (possibly with padding). This is toggled exactly when negations occur in the quantifier-free part. For example, consider a subformula $\alpha = \beta \wedge \neg\gamma$

and assume there was no negation around α. Then we have a gate g, which will compute the truth-value of α, for which the negation-bit is 0. We connect this to the gate for the truth-value of β with negation-bit 0 and—since there is a negation around γ—the gate for the truth-value of γ with negation-bit 1.

Apart from constructing the connections in this way, the negation-bit also changes the gate-type of gates: If a non-negated gate is a $\vee$-gate, the negated version is a $\wedge$-gate and vice versa. Also, negated gates computing the value of literals use the negated version of the respective literal.

In total, this construction only doubles the size of the circuit and does not change its depth, but handles arbitrary negations.

For the case of multiple GPR-operators, we build a circuit for each of them in the above way. In case of nesting, we start from the innermost operator. Adequate connections between the different circuits are easily doable and size and depth of the combination of all those circuits still stays within the desired bounds.

$\underline{\mathrm{NC}^1 \subseteq \mathrm{FO}(\mathrm{GPR}_{\mathrm{bound}})}$ and $\underline{\mathrm{SAC}^1 \subseteq \mathrm{FO}(\mathrm{GPR}_{\mathrm{semi}})}$ can both be shown with the same formula and the same proof as $\mathrm{AC}^1 \subseteq \mathrm{FO}(\mathrm{GPR})$, replacing GPR by $\mathrm{GPR}_{\mathrm{bound}}$ or $\mathrm{GPR}_{\mathrm{semi}}$, respectively.

$\underline{\mathrm{FO}(\mathrm{GPR}_{\mathrm{bound}}) \subseteq \mathrm{NC}^1}$: This can be proven completely analogously to $\mathrm{FO}(\mathrm{GPR}) \subseteq \mathrm{AC}^1$. Instead of AC^0 circuit families for evaluation of φ and θ, we now use NC^1 circuit families. This leads to logarithmic depth for evaluation of θ. In general, repeating this for logarithmically many steps would be a problem. By definition there are no occurrences of P inside any unbounded quantifier, though. For the bounded quantifiers, we still create subcircuits for all possible values for the quantified variable, but we only connect the maximal two satisfying the relativization to the parent. This ensures that gates marked with $P(\overline{x}, \overline{z})$ for some $\overline{x}, \overline{z}$ still only occur in constant depth in the circuit evaluating $\theta(\overline{x}, \overline{y}, P)$, this time with only bounded fan-in gates. Consequently, the construction still only leads to logarithmic depth in total.

$\underline{\mathrm{FO}(\mathrm{GPR}_{\mathrm{semi}}) \subseteq \mathrm{SAC}^1}$: Here, the same trick as for NC^1 can be used. θ can be evaluated using an NC^1 circuit family which is also an SAC^1 circuit family. Also, the semi-unboundedness of the quantifiers around occurrences of P directly corresponds to the semi-unboundedness in SAC^1 circuit families. □

The proof of the inclusion $\mathrm{AC}^1 \subseteq \mathrm{FO}(\mathrm{GPR})$ also immediately gives us the following normal-form for our logical classes.

Corollary 4.2. *Let* $\mathcal{G} \in \{\mathrm{GPR}, \mathrm{GPR}_{\mathrm{bound}}, \mathrm{GPR}_{\mathrm{semi}}\}$. *Then*

$$\mathrm{FO}(\mathcal{G}) = \mathcal{G}\text{-FO},$$

where $\mathcal{G}$-FO *denotes the class of languages definable in first-order logic with one* GPR*-operator in the beginning.*

5 Logical Characterizations of Small Depth Counting Classes

Next, we want to define the game semantics for our new logics. Let φ be an FO(GPR) formula and $\mathcal{A}$ an input structure. The GPR-operators within the formula are global information in the model-checking game for $\mathcal{A} \vDash \varphi$. Configurations of the game are of the form

$$(\psi, \mathcal{A}, \theta, s, H)$$

where ψ is the current formula, $\mathcal{A}$ is the input structure, θ is an assignment to the free variables in ψ, s is a bit specifying whether the players currently swapped roles and H is the complete history of choices made up to this point. The game starts in configuration

$$(\psi, \mathcal{A}, \theta_0, 0, H_0)$$

where ψ is the subformula of φ to the right of the GPR-operators (so $\psi \in \mathrm{FO}$), θ_0 is the function on empty domain and H_0 is the empty history. The game then proceeds in the same way as the model-checking game for FO, but the history is kept in H and occurrences of predicates defined by GPR have to be handled: Whenever a predicate symbol P defined by GPR is reached, the game continues on the recursive definition given by the respective GPR-operator (without changing the input structure, the assignment, the s-bit or the history).

A strategy of player 1 is again a function mapping configurations to specific choices in such a way that a choice is specified for all configurations that give player 1 a choice and are reachable from the starting configuration if player 1 acts according to this strategy and player 2 makes arbitrary choices. Since the history is part of the configuration, choices may differ throughout the recursion. Winning strategies of player 1 are strategies of player 1 that allow him to win independent of the choices made by player 2.

Similar to the approach in [11], we can also count the number of winning strategies of the verifier.

Definition 5.1. A function $f \colon \{0,1\}^* \to \mathbb{N}$ is in the class #Win-FO(GPR), if there is an FO(GPR)-formula φ over vocabulary $\tau_{\mathrm{string}} \cup (\mathrm{BIT}^2)$ such that for all $w \in \{0,1\}^*$:

$$f(w) = \#\mathrm{Win}(\varphi, \mathcal{A}_w),$$

where $\#\mathrm{Win}(\varphi, \mathcal{A}_w)$ is the number of winning strategies of player 1 in the model checking game for $\mathcal{A}_w \vDash \varphi$.

$\#\mathrm{Win}\text{-}\mathrm{FO}(\mathrm{GPR}_{\mathrm{bound}})$ and $\#\mathrm{Win}\text{-}\mathrm{FO}(\mathrm{GPR}_{\mathrm{semi}})$ are defined analogously.

This then gives us characterizations of the counting version of the corresponding classes from circuit complexity:

Theorem 5.2.

1. $\#\mathrm{NC}^1 = \#\mathrm{Win}\text{-}\mathrm{FO}(\mathrm{GPR}_{\mathrm{bound}})$
2. $\#\mathrm{SAC}^1 = \#\mathrm{Win}\text{-}\mathrm{FO}(\mathrm{GPR}_{\mathrm{semi}})$
3. $\#\mathrm{AC}^1 = \#\mathrm{Win}\text{-}\mathrm{FO}(\mathrm{GPR})$

Proof. The proof idea is very similar to the one used for the decision version. We again begin by proving the unbounded version.

$\underline{\#\mathrm{AC}^1 \subseteq \#\text{Win-FO(GPR)}}$: Let $f \in \#\mathrm{AC}^1$ via the FO-uniform AC^1 circuit family $C = (C_n)_{n\in\mathbb{N}}$ with the properties from Lemma 3.3 and C_n has depth at least 1 for all n. The latter can easily be achieved by adding a new $\wedge$-gate as output-gate with the old output-gate being its only child. Let

$$I = (\varphi_{\text{universe}}, \varphi_{G_\wedge}, \varphi_{G_\vee}, \varphi_{\text{Input}}, \varphi_{\text{negatedInput}}, \varphi_E, \varphi_{\text{output}})$$

be an FO-interpretation showing that C is uniform. Furthermore, let ψ, φ_{Literal} and $\varphi_{\text{trueLiteral}}$ be defined as in the proof of Theorem 4.1 and θ be defined as follows:

$$\begin{aligned}\theta(\overline{y}, P) :=& \\ &\Big(\text{Even}(\overline{y}) \wedge \big((\exists\overline{z}.(\overline{z} < \overline{y}/2 \wedge \varphi_E(\overline{y},\overline{z})))\psi(\overline{z}, P(\overline{z}))\big)\Big) \vee \\ &\Big(\text{Odd}(\overline{y}) \wedge \big((\forall\overline{z}.(\overline{z} < \overline{y}/2 \wedge \varphi_E(\overline{y},\overline{z})))\psi(\overline{z}, P(\overline{z})) \wedge \\ &(\overline{z} < \overline{y}/2 \wedge \varphi_E(\overline{y},\overline{z}))\big)\Big).\end{aligned}$$

Then

$$\Phi := [P(\overline{y}) \equiv \theta(\overline{y}, P)]\ \exists\overline{o}(\varphi_{\text{output}}(\overline{o}) \wedge P(\overline{o}))$$

defines f. Note that $\varphi_E(\overline{y},\overline{z})$ within the relativization for $\overline{z}$ can be moved outside the relativization without changing the number of winning strategies, leading to an FO(GPR)-formula with the same number of winning strategies. Also, compared to the proof of Theorem 4.1 the guard is added a second time as part of the formula following the quantifier. This is, because relativized quantifications are defined as shorthands for usual quantifications. In the case of universal quantifications, the resulting disjunction possibly adds additional winning strategies. To prevent this, the guard is added after ψ.

Now let $n \in \mathbb{N}$ and $w \in \{0,1\}^n$. We now show that the number of winning strategies in the game for $\mathcal{A}_w \vDash \Phi$ is exactly the number of proof trees of $C_{|w|}$ on input w.

In the following we use $\#\text{Win}(\varphi(\overline{c}))$ where $\overline{c}$ is a tuple of elements of $\text{dom}(\mathcal{A})$, as notation for the number of winning strategies of player 1 in the game for $\mathcal{A} \vDash \Phi$ starting from a configuration with an assignment that assigns variables according to $\overline{c}$ and an arbitrary history. $\text{Odd}(\overline{y})$ can be constructed such that $\#\text{Win}(\text{Odd}(\overline{c})) \in \{0,1\}$ for all $\overline{c}$ and $\text{Even}(\overline{y})$ can be constructed such that for all $\overline{c}$ we have

$$\#\text{Win}(\text{Even}(\overline{c})) = 1 - \#\text{Win}(\text{Odd}(\overline{c})).$$

This means that for all $\overline{c}$ it holds that

$$\#\text{Win}(\theta(\overline{c}))$$

(possibly with occurrences of P in θ) is equal to

$$\begin{aligned}\#\text{Win}\Big((Q\overline{z}.(\overline{z} < \overline{c}/2 \wedge \varphi_E(\overline{c},\overline{z}))) \\ \big(P(\overline{z}) \wedge \neg\varphi_{\text{Literal}}(\overline{z}) \vee \varphi_{\text{trueLiteral}}(\overline{z})\big) \wedge \\ \big(\overline{z} < \overline{c}/2 \wedge \varphi_E(\overline{c},\overline{z})\big)\Big)\end{aligned}$$

where Q is either $\exists$ or $\forall$ depending on the parity of $\overline{c}$.

We now prove that the number of winning strategies $\#\text{Win}(P(\overline{g}))$ is exactly the number of proof trees of the subcircuit of C_n rooted in $\overline{g}$. By definition, on input structure $\mathcal{A}_w$, the formulae from I used above give access to C_n. We prove inductively that for any $k \in \mathbb{N}$, $\#\text{Win}(P(\overline{g}))$ gives the number of proof trees of the subcircuit of C_n rooted in $\overline{g}$ on input w if all children of $\overline{g}$ have depth $\leq k$.

$k = 0$: Note that $\varphi_{\text{trueLiteral}}(\overline{h})$ gives the value of $\overline{h}$ in C_n on input w if $\overline{h}$ is an input gate and that for these gates the value of the gate is equal to the number of proof trees of the subcircuit rooted in them. This means that for gates $\overline{g}$ all children of which are input gates we have:

$$\begin{aligned}\#\text{Win}(P(\overline{g})) &= \#\text{Win}(Q\overline{z}.(\overline{z} < \overline{g}/2 \wedge \varphi_E(\overline{g},\overline{z})) \\ &\qquad \big(P(\overline{z}) \wedge \neg\varphi_{\text{Literal}}(\overline{z}) \vee \varphi_{\text{trueLiteral}}(\overline{z})\big) \\ &= \bigcirc_{\substack{\overline{z}\in\text{dom}(\mathcal{A}_w)^k, \\ \overline{z}<\overline{g}/2\wedge\varphi_E(\overline{g},\overline{z})}} \#\text{Win}(P(\overline{z}) \wedge \neg\varphi_{\text{Literal}}(\overline{z}) \vee \varphi_{\text{trueLiteral}}(\overline{z})),\end{aligned} \tag{$\star\star$}$$

where $\bigcirc$ is either summation or multiplication depending on the parity of $\overline{g}$. Note that this summation or multiplication always occurs here, since strategies can differ depending on the history.

We can assume that there is exactly one winning strategy showing φ_{Literal}, respectively $\varphi_{\text{trueLiteral}}$, if it is true (and none otherwise). Since $k = 0$, we know that for all $\overline{z}$ that meet the conditions, $\varphi_{\text{Literal}}(\overline{z})$ is true yielding

$$\#\text{Win}(P(\overline{g})) = \bigcirc_{\substack{\overline{z}\in\text{dom}(\mathcal{A}_w)^k, \\ \overline{z}<\overline{g}/2\wedge\varphi_E(\overline{g},\overline{z})}} \#\text{Win}(\varphi_{\text{trueLiteral}}(\overline{z})).$$

By assumption, if $\varphi_E(\overline{g},\overline{z})$ then $\overline{z} < \overline{g}/2$, and thus $\overline{z} < \overline{g}/2 \wedge \varphi_E(\overline{g},\overline{z}) \equiv \varphi_E(\overline{g},\overline{z})$.

This means that $\#\text{Win}(P(\overline{g}))$ is exactly the number of proof trees of the subcircuit of C_n rooted in $\overline{g}$.

$k \longrightarrow k+1$: Again, by assumption, $\overline{z} < \overline{g}/2 \wedge \varphi_E(\overline{g},\overline{z}) \equiv \varphi_E(\overline{g},\overline{z})$. We also know that for all children $\overline{z}$ of $\overline{g}$ only two cases can occur:

If $\overline{z}$ is an input gate, then we have that $\#\text{Win}(\neg\varphi_{\text{Literal}}(\overline{z})) = 0$ and $\#\text{Win}(\varphi_{\text{trueLiteral}}(\overline{z}))$ is exactly the number of proof trees of the subcircuit of C_n rooted in $\overline{z}$.

If $\overline{z}$ is not an input gate, then by assumption we have that $\#\text{Win}(\varphi_{\text{trueLiteral}}(\overline{z})) = 0$, $\#\text{Win}(\neg\varphi_{\text{Literal}}) = 1$ and by induction hypothesis $\#\text{Win}(P(\overline{z}))$ is exactly the number of proof trees of the subcircuit of C_n rooted in $\overline{z}$.

By ($\star\star$) this means that $\#\text{Win}(P(\overline{g}))$ is equal to the number of proof trees of the subcircuit of C_n rooted in $\overline{g}$.

Now for every $w \in \{0,1\}^*$ and non-input gate $\overline{g}$ in $C_{|w|}$ we have: $\#\text{Win}(P(\overline{g}))$ is the number of proof trees of the subcircuit of $C_{|w|}$ on input w that is rooted in $\overline{g}$. Thus, it is easy to see that the above formula defines f: It can be seen almost immediately that the number of winning strategies of the formula behind the recursive definition of P is the number of winning strategies for $P(\underline{\text{output}})$, where $\underline{\text{output}}$ is the unique element satisfying φ_{output}. By the induction above this is equal to the number of proof trees of circuit $C_{|w|}$.

$\underline{\#\text{Win-FO(GPR)} \subseteq \#\text{AC}^1}$: This can be proven completely analogously to the decision version. Counting proof trees of the constructed circuit family leads exactly to the function given by the number of winning strategies of the formula we started with.

The inclusions $\#\text{NC}^1 \subseteq \#\text{Win-FO}(\text{GPR}_{\text{bound}})$ and $\#\text{SAC}^1 \subseteq \#\text{Win-FO}(\text{GPR}_{\text{semi}})$, that is, the bounded and the semi-unbounded case, can—as for the decision version—both be shown with the same formula as the unbounded case by changing the GPR-operator to a $\text{GPR}_{\text{bound}}$- or GPR_{semi}-operator, respectively.

The converse directions, that is, $\#\text{Win-FO}(\text{GPR}_{\text{bound}}) \subseteq \#\text{NC}^1$ and $\#\text{Win-FO}(\text{GPR}_{\text{semi}}) \subseteq \#\text{SAC}^1$, can also be shown analogoulsy, using again the restriction that P occurs only within bounded quantifiers within θ. □

Analogously to the decision version, the proof again allows us to establish a normal-form for our new logical classes.

Corollary 5.3. *Let $\mathcal{G} \in \{\text{GPR}, \text{GPR}_{\text{bound}}, \text{GPR}_{\text{semi}}\}$. Then*

$$\#\text{Win-FO}(\mathcal{G}) = \#\text{Win-}\mathcal{G}\text{-FO},$$

where $\#\text{Win-}\mathcal{G}\text{-FO}$ denotes the class of functions that can be described as the number of winning strategies for first-order formulae with one GPR*-operator in the beginning.*

Remark 2. *To further show the robustness of our classes, we want to mention certain variations of our logics that do not change the resulting complexity classes. For all decision classes, we can drop condition 2 from Definition 3.1 without changing the class. For* #Win-FO(GPR) *the same holds.*

For $\#\text{Win-FO}(\text{GPR}_{\text{bound}})$ *and* $\#\text{Win-FO}(\text{GPR}_{\text{semi}})$*, condition 2 cannot be dropped but can be replaced by the following weaker version: "never occur in the scope of any universal quantification not guarded in this way".*

6 Conclusion

We extended the only so-far known logical characterization of an arithmetic circuit class, namely $\#\text{AC}^0 = \#\text{Win-FO}$ [11], to arithmetic classes defined by circuits of logarithmic depth. In order to achieve this, we first had to develop logical characterizations of the corresponding Boolean classes.

The result from [11] was used in [7] to place $\#\text{AC}^0$ in a strict hierarchy of counting classes within #P. In this way, lower bounds for several logically-defined arithmetic classes were obtained. Our hope is that the characterizations of larger arithmetic classes presented here will also lead to new insights about these and hopefully spur development of new upper and lower bounds, e.g., is $\#\text{NC}^1 \subseteq \text{NC}^1$? Is $\#\text{NC}^1 \neq \#\text{P}$? Is $\text{NC}^1 \neq \text{PP}$?

References

[1] Miklós Ajtai. 1983. Σ_1^1-Formulae on Finite Structures. *Ann. Pure Appl. Logic* 24, 1 (1983), 1–48.

[2] Marcelo Arenas, Martin Muñoz, and Cristian Riveros. 2017. Descriptive Complexity for counting complexity classes. In *LICS*. IEEE Computer Society, 1–12.

[3] David A. Mix Barrington and Neil Immerman. 1994. Time, hardware, and uniformity. In *Proceedings 9th Structure in Complexity Theory*. IEEE Computer Society Press, 176–185.

[4] David A. Mix Barrington, Neil Immerman, and H. Straubing. 1990. On uniformity within NC^1. *J. Comput. System Sci.* 41 (1990), 274–306.

[5] Ashok K. Chandra and Martin Tompa. 1990. The complexity of short two-person games. *Discrete Applied Mathematics* (Jan. 1990), 21–33.

[6] Kevin J. Compton and Claude Laflamme. 1990. An Algebra and a Logic for NC^1. *Inf. Comput.* 87, 1/2 (1990), 240–262. https://doi.org/10.1016/0890-5401(90)90063-N

[7] Arnaud Durand, Anselm Haak, Juha Kontinen, and Heribert Vollmer. 2016. Descriptive Complexity of $\#\text{AC}^0$ Functions. In *CSL 2016 (LIPIcs)*, Vol. 62. 20:1–20:16. https://doi.org/10.4230/LIPIcs.CSL.2016.20

[8] Heinz-Dieter Ebbinghaus, Jörg Flum, and Wolfgang Thomas. 1994. *Mathematical logic*. Springer-Verlag.

[9] Gudmund Skovbjerg Frandsen, Mark Valence, and David A. Mix Barrington. 1994. Some Results on Uniform Arithmetic Circuit Complexity. *Mathematical Systems Theory* 27, 2 (1994), 105–124.

[10] Merrick L. Furst, James B. Saxe, and Michael Sipser. 1984. Parity, Circuits, and the Polynomial-Time Hierarchy. *Mathematical Systems Theory* 17, 1 (1984), 13–27. https://doi.org/10.1007/BF01744431

[11] Anselm Haak and Heribert Vollmer. 2016. A Model-Theoretic Characterization of Constant-Depth Arithmetic Circuits. In *WoLLIC (Lecture Notes in Computer Science)*, Vol. 9803. Springer, 234–248.

[12] Neil Immerman. 1989. Expressibility and Parallel Complexity. *SIAM J. Comput.* 18, 3 (1989), 625–638. https://doi.org/10.1137/0218043

[13] Neil Immerman. 1999. *Descriptive complexity*. Springer.

[14] Neeraj Kayal and Ramprasad Saptharishi. 2014. A Selection of Lower Bounds for Arithmetic Circuits. In *Perspectives in Computational Complexity: The Somenath Biswas Anniversary Volume*. Birkhäuser, 77–116.

[15] Clemens Lautemann, Pierre McKenzie, Thomas Schwentick, and Heribert Vollmer. 2001. The Descriptive Complexity Approach to LOGCFL. *J. Comput. Syst. Sci.* 62, 4 (2001), 629–652.

[16] Leonid Libkin. 2004. *Elements of Finite Model Theory*. Springer. https://doi.org/10.1007/978-3-662-07003-1

[17] Meena Mahajan. 2014. Algebraic Complexity Classes. In *Perspectives in Computational Complexity: The Somenath Biswas Anniversary Volume*, Manindra Agrawal and Vikraman Arvind (Eds.). Birkhäuser, 51–75.

[18] Guillaume Malod and Natacha Portier. 2008. Characterizing Valiant's algebraic complexity classes. *Journal of Complexity* 24, 1 (Feb. 2008).

[19] Sanjeev Saluja, K. V. Subrahmanyam, and Madhukar N. Thakur. 1995. Descriptive Complexity of #P Functions. *J. Comput. System Sci.* 50, 3 (1995), 493–505.

[20] Ramprasad Saptharishi. 2017. A survey of known lower bounds in arithmetic circuits. A continuously updated git survey. (2017). https://github.com/dasarpmar/lowerbounds-survey.

[21] Heribert Vollmer. 1999. *Introduction to Circuit Complexity - A Uniform Approach*. Springer.

Acknowledgments

We thank the anonymous reviewers for helpul comments.