Model-Theoretic Characterizations of Boolean and Arithmetic Circuit Classes of Small Depth

Arnaud Durand[1], Anselm Haak[2], and Heribert Vollmer[2]

1 **Université Paris Diderot, IMJ-PRG, CNRS UMR 7586, Case 7012, 75205 Paris cedex 13, France**
durand@math.univ-paris-diderot.fr

2 **Theoretische Informatik, Leibniz Universität Hannover, Appelstraße, D-30167, Germany**
(haak|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.

1998 ACM Subject Classification F.1.1 Models of Computation, F.1.3 Complexity Measures and Classes, F.4.1 Mathematical Logic

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

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

The computational power of arithmetic circuits is of current focal interest in computational complexity theory, see the recent surveys [15, 12] or the continuously updated collection of results at [17]. 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: $\mathrm{AC}^0 = \mathrm{FO}$ [10]. 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 [11, 14]. The famous lower bound for AC^0, showing that the parity function cannot be computed by such circuit families [8], 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: $\#\mathrm{AC}^0 = \#\text{Win-FO}$ [9]. 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 [9], 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 aim of this paper is to generalize the theorem $\#\mathrm{AC}^0 = \#\text{Win-FO}$ to larger circuit classes, in particular the classes $\#\mathrm{NC}^1$, $\#\mathrm{SAC}^1$ and $\#\mathrm{AC}^1$, defined by families of arithmetic circuits of polynomial size and logarithmic depth with bounded fan-in addition and multiplication gates (for $\#\mathrm{NC}^1$), unbounded fan-in addition and bounded fan-in multiplication gates ($\#\mathrm{SAC}^1$), and unbounded fan-in addition and multiplication gates ($\#\mathrm{AC}^1$), see [18]. 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 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 [3], and similarly SAC^1 by extending FO by groupoidal quantifiers [13]. 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 [10]. Here it can be said that in Immerman's notation, $\#\mathrm{AC}^1 = \#\text{Win-FO}[\log]$, 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 to define relations by a certain kind of linear recursion [5] (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 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 [5]. In Sect. 5 we characterize the arithmetic classes $\#\mathrm{NC}^1$, $\#\mathrm{SAC}^1$ and $\#\mathrm{AC}^1$ 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., [7]. We consider finite σ-structures where σ is a finite vocabulary consisting of relation and constant symbols. For a structure $\mathcal{A}$, $\mathrm{dom}(\mathcal{A})$ denotes its universe. We will always use structures with universe $\{0, 1, \dots, n-1\}$ 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. When talking about structures with built-in relations, $\vDash$ 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., [11]): 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 $\{0, \dots, n-1\}$, $\leq^2$ 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 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 1.** A language $L \subseteq \{0, 1\}^*$ is in FO if there is an FO-formula φ over vocabulary $\tau_{\text{string}} \cup (\mathrm{BIT}^2)$ such that for all $w \in \{0, 1\}^*$:

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

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

$$(\exists x.\varphi)\ \psi$$

as a shorthand for $\exists x(\varphi \wedge \psi)$ and, respectively,

$$(\forall x.\varphi)\ \psi$$

as a shorthand for $\forall x(\varphi \to \psi) \equiv \forall x(\neg\varphi \vee \psi)$.

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: $\exists_{\mathrm{b}}$, $\forall_{\mathrm{b}}$. Formally, the semantics can be given in FO as follows:

$$(\exists_{\mathrm{b}} x.\varphi(x))\ \psi(x) \equiv \Big(\exists x.\big(\varphi(x) \wedge \forall y \forall z\ (y \neq z \wedge x < y \wedge x < z) \to (\neg\varphi(y) \vee \neg\varphi(z))\big)\Big)\ \psi(x)$$

$$(\forall_{\mathrm{b}} x.\varphi(x))\ \psi(x) \equiv \Big(\forall x.\big(\varphi(x) \wedge \forall y \forall z\ (y \neq z \wedge x < y \wedge x < z) \to (\neg\varphi(y) \vee \neg\varphi(z))\big)\Big)\ \psi(x)$$

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

▶ **Definition 2.** Let σ, τ be vocabularies, $\tau = (R_1^{a_1}, \ldots, R_r^{a_r})$. A first-order interpretation (or FO-interpretation)

$$I : \mathrm{STRUC}[\sigma] \to \mathrm{STRUC}[\tau]$$

is given by a tuple of FO-formulae $\varphi_0, \varphi_1, \ldots, \varphi_r$ 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 $\mathcal{A} \in \mathrm{STRUC}[\sigma]$, these formulae define the structure

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

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

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

$$R_i^{I(\mathcal{A})} = \{(\langle b_1^1, \ldots, b_1^k\rangle, \ldots, \langle b_{a_i}^1, \ldots, b_{a_i}^k\rangle) \in |I(\mathcal{A})|^{a_i} \mid \mathcal{A} \vDash \varphi_i(b_1^1, \ldots, b_{a_i}^k)\}$$

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 $\mathcal{C} = (C_n)_{n \in \mathbb{N}}$ 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 to 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:

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

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
- $\text{Input}(x, i)$: gate x is an input gate associated with the i'th input bit
- $\text{negatedInput}(x, i)$: gate x is a negated input gate associated with the i'th input bit
- $\text{output}(x)$: gate x is the output gate

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

▶ **Definition 3.** A circuit family $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$ is said to be first-order uniform (FO-uniform) if there is an FO-interpretation

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

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 by [2] this uniformity 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 4.** A language $L \subseteq \{0, 1\}^*$ is in AC^i if there is an FO-uniform circuit family with depth $(\log n)^i$ and polynomial size 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.

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 [3, 11]: $\text{AC}^0 = \text{FO}$.

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, though. 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 5.** A function $f : \{0, 1\}^* \to \mathbb{N}$ is in $\#\text{AC}^i$ ($\#\text{NC}^i$, $\#\text{SAC}^i$) if there is an AC^i (NC^i, SAC^i) circuit family $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$ such that for all $w \in \{0, 1\}^*$,

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

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 [16]) that $\#\mathrm{NC}^1$ and $\#\mathrm{SAC}^1$ are subclasses of $\#\mathrm{P}$. 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 $\#\mathrm{AC}^1 \not\subseteq \#\mathrm{P}$.

Similarly, one can identify witnesses for acceptance in first-order logic. One possibility for this is to do this in terms of the model-checking game defined as follows. The model checking game for FO is the two-player game with the players “verifier” and “falsifier” played recursively on any FO formula φ and input structure $\mathcal{A}$. The verifier tries to reach an atom that is satisfied while the falsifier tries to reach an atom that is not satisfied. For this, the game starts on the whole formula. From there, depending on the outermost operator or quantifier, one of the players makes a choice and the game continues on a certain sub-formula. The rules for this are as follows:

- $\exists x \psi$: verifier chooses a value for x, continue on ψ
- $\forall x \psi$: falsifier chooses a value for x, continue on ψ
- $\alpha \vee \beta$: verifier chooses whether to continue with α or β
- $\alpha \wedge \beta$: falsifier chooses whether to continue with α or β
- $\neg\alpha$: verifier and falsifier swap roles, continue on α
- For any atom : verifier wins if it is true, falsifier wins otherwise

In this game the verifier has a winning strategy—that is, a strategy that lets him win the game independent of the choices of the opponent—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 6.** A function $f\colon \{0,1\}^* \to \mathbb{N}$ is in #Win-FO, if there is an FO-formula φ over vocabulary $\tau_{\mathrm{string}} \cup (\mathrm{BIT}^2)$ such that for all $w \in \{0,1\}^*$:

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

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

As was shown in [9], the counting versions of AC^0 and FO coincide, i.e.: $\#\mathrm{AC}^0 =$ #Win-FO.

For the quantifiers $\exists_{\mathrm{b}}$ and $\forall_{\mathrm{b}}$ 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 $\#\mathrm{AC}^0$ by counting winning strategies in different logics. It has been proved in [5] 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 intuitively, $P(\overline{x}, y)$ has the same truth value as $\theta(\overline{x}, y, P(\overline{x}, y-1))$ for $y > 0$ and $P(\overline{x}, 0)$ being equivalent to $\theta(\overline{x}, y, \bot)$. Then, FO(RPR) denotes the class of languages definable by formula 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 make use of the inductively defined P. Over structures with built-in BIT, it holds that $\mathrm{NC}^1 = \mathrm{FO(RPR)}$ [5]. This characterization does not immediately generalize to 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 7** (GPR)**.** A formula $\phi \in \mathrm{FO(GPR)}$ if it is of the form:

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

where $\psi \in \mathrm{FO}$ and $\theta \in \mathrm{FO}$ 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 $Q\overline{z}.(\overline{z} \leq \overline{y}/2)$ with $Q \in \{\forall, \exists\}$ and
2. never occur in the scope of any quantification not guarded in this way.

We also call the part in $[\cdot]$ a GPR-operator. We define $\mathrm{FO(GPR_{bound})}$ similary by allowing only bounded variants for guarded quantification $Q_b\overline{z}.(\overline{z} < \overline{y}/2)$ and $\mathrm{FO(GPR_{semi})}$ for which universal guarded $\forall\overline{z}.(\overline{z} \leq \overline{y}/2)$ and bounded existential guarded $\exists_b\overline{z}.(\overline{z} < \overline{y}/2)$ quantifications are allowed.

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

▶ **Example 8.** The SHORTCAKE problem, proved AC^1-complete in [4] 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 that the indices of its first and last lines and columns (i_0, j_0, i_1, j_1). It is given, at each round, with an indication of which corner of this submatrix the token is on and whose turn it is. In the beginning of the game, the configuration is thus $(1, n, 1, n)$, the token is at the $(1, 1)$ corner and H starts to play. In his turn, H tries to move the token horizontally in the submatrix to some entry [1] (i_0, j), $j \neq j_0, j_1$ 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, leaving the token once again on a corner of the current submatrix. I.e. the new configuration is (i_0, j, i_1, j_1) if $j - j_0 \leq j_1 - j$ and (i_0, j_0, i_1, j) if not. In his turn, F plays similarly but vertically, on the rows. The first player with no move left loses.

We encode the matrix by a structure representing a binary word of length n^2. Remark, that the size of the matrix is divided by at least two after each round. The existence of a winning strategy for H is encoded by the following FO(GPR) formula (s is an upper bound for the size of the matrix at each round with some padding, $p = 0, 1$ is for the players),

[1] Supposing j different from both j_0, j_1 allows to forget the precise corner where the token is and simplify in a non essential way the formula

$$\phi := [P(x, \underbrace{s, i_0, i_1, j_0, j_1, p}_{\overline{y}}) \equiv (p = 0 \land \theta_H(x, \overline{y})) \lor (p = 1 \land \theta_V(x, \overline{y}))]\ \varphi(P)$$

with $\varphi(P) \equiv P(n, 2n^2, 1, n, 1, n, 0)$ and $\theta_H(x, \overline{y})$ is

$$\exists \overline{z} = (s', i'_0, i'_1, j'_0, j'_1, p').(\overline{z} < \overline{y}/2)$$
$$\begin{cases} (s' \leq s/2 - 2) \land p' = 1 \land i'_0 = i_0 \land i'_1 = i_1 \land P(x, s, i'_0, j'_0, i'_1, j'_1, p')) \land \\ \big[(M(i'_0 n + j'_0) \land j'_0 \neq j_0, j_1 \land (j_1 - j_0)/2 \leq j'_0 \leq j_1 \land j'_1 = j_1) \lor \\ (M(i'_0 n + j'_1) \land j'_1 \neq j_0, j_1 \land j_0 \leq j'_1 \leq (j_1 - j_0)/2 \land j'_0 = j_0))\big] \end{cases}$$

The formula $\theta_V(x, \overline{y})$ associated to V is defined similarly (but with universally guarded quantification for z and (row) i). In [4] 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 due to built-in BIT, we have an order and arithmetic on the gates of circuits from uniform circuit families. Circuit families in our normal-form have the following properties: All tuples of the appropriate size are gates (so $\varphi_{\mathrm{universe}}$ from the FO-interpretation showing uniformity is always true). The $\land$-gates are exactly the gates that are odd and neither input nor negated input gates. The $\lor$-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 9.** *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 $I : \mathrm{STRUC}[\tau_{\mathrm{string}}] \cup (\mathrm{BIT}^2) \to \mathrm{STRUC}[\tau_{\mathrm{circ}}]$ 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. $|I(\mathcal{A}_w)| = |\mathcal{A}|^k$
2. *for all* $\overline{x} \in |I(\mathcal{A}_w)|$*:*
   $G_{\land}^{I(\mathcal{A}_w)}(\overline{x}) \Leftrightarrow (\neg\mathrm{Input}^{I(\mathcal{A}_w)}(\overline{x}) \land \neg\mathrm{negatedInput}^{I(\mathcal{A}_w)}(\overline{x}) \land \overline{x} \text{ is odd})$
   $G_{\lor}^{I(\mathcal{A}_w)}(\overline{x}) \Leftrightarrow (\neg\mathrm{Input}^{I(\mathcal{A}_w)}(\overline{x}) \land \neg\mathrm{negatedInput}^{I(\mathcal{A}_w)}(\overline{x}) \land \overline{x} \text{ is even})$
3. *for all* $\overline{x}, \overline{y} \in |I(\mathcal{A}_w)|$*:*
   $E^{I(\mathcal{A}_w)}(\overline{x}, \overline{y}) \Rightarrow \exists \overline{y}' 2 \cdot \overline{y} = \overline{y}' \land \overline{y}' < \overline{x}$

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 appendix.

◀

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 10.**

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 $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$ with the properties from Lemma 9 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 $\mathcal{C}$ is uniform. Furthermore, let

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

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

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

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

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

Even and Odd check the parity of the least significant bit within the least significant variable within tuple $\overline{y}$ using BIT. Note that $\varphi_E(\overline{y}, \overline{z})$ within the relativization for $\overline{z}$ can be moved outside the relativization, so Φ is equivalent to a FO(GPR)-formula.

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

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

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 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:

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

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 $\overline{g}$.

$k \to k+1$: 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 $\varphi_{\text{trueLiteral}}(\overline{z})$ gives the value of $\overline{z}$.

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 w 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 $L \in \text{FO(GPR)}$ 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 $\overline{x}, \overline{y}$ a 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 $P(\overline{x}, \overline{z})$. 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 also use the negated version of the respective literal compared to the non-negated version.

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.

$\underline{\mathrm{NC}^1 \subseteq \mathrm{FO}(\mathrm{GPR}_{\mathrm{bound}}) \text{ and } \mathrm{SAC}^1 \subseteq \mathrm{FO}(\mathrm{GPR}_{\mathrm{semi}})}$: Can 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 11.** *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 decidable in first-order logic with one* GPR-*operator in the beginning.*

5 Logical Characterizations of Small Depth Counting Classes

Next, we want to define a game semantics for our new logics. The game we define will correspond to model-checking and is defined analogous to the model checking game for FO for the most part. When playing the game on an FO(GPR)-formula

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

it begins on formula φ. The only difference to the model checking for FO is an additional case for atoms of the form $P(\overline{a}, \overline{b})$. In this case, the game continues on the formula $\theta(\overline{x}, \overline{y}, P)$. For all other atoms, the winner is immediately determined as before.

Now for any $\mathcal{A}$ and $\varphi \in \mathrm{FO}(\mathrm{GPR})$ it holds that

$$\mathcal{A} \vDash \varphi \iff \text{the verifier has a winning strategy for the game on } \mathcal{A} \vDash \varphi.$$

Analogously, we can extend the semantic game for $\mathrm{FO}(\mathrm{GPR}_{\mathrm{bound}})$ and $\mathrm{FO}(\mathrm{GPR}_{\mathrm{semi}})$.

Similar to the approach in [9], we can also count the number of winning strategies of the verifier.

▶ **Definition 12.** A function $f\colon \{0,1\}^* \to \mathbb{N}$ is in #Win-FO(GPR), if there is an FO(GPR)-formula φ over vocabulary $\tau_{\text{string}} \cup (\text{BIT}^2)$ such that for all $w \in \{0,1\}^*$:

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

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

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

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

▶ **Theorem 13.**

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

Proof. The proof consists of carefully counting winning strategies in semantic games for those formulae developed in the decision version (Theorem 10) and is given in the appendix. ◀

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

▶ **Corollary 14.** *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. 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 7 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 $\#\text{AC}^0 = \#\text{Win-FO}$ [9], 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 [9] was used in [6] 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 here presented characterizations of larger arithmetic classes 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. Σ^1_1-formulae on finite structures. *Ann. Pure Appl. Logic*, 24(1):1–48, 1983.

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

3 David A. Mix Barrington, Neil Immerman, and H. Straubing. On uniformity within NC^1. *Journal of Computer and System Sciences*, 41:274–306, 1990.

4 Ashok K. Chandra and Martin Tompa. The complexity of short two-person games. *Discrete Applied Mathematics*, pages 21–33, January 1990.

5 Kevin J. Compton and Claude Laflamme. An algebra and a logic for NC^1. *Inf. Comput.*, 87(1/2):240–262, 1990.

6 Arnaud Durand, Anselm Haak, Juha Kontinen, and Heribert Vollmer. Descriptive complexity of $\#\mathrm{AC}^0$ functions. In Jean-Marc Talbot and Laurent Regnier, editors, *25th EACSL Annual Conference on Computer Science Logic (CSL 2016)*, volume 62 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:16, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

7 Heinz-Dieter Ebbinghaus, Jörg Flum, and Wolfgang Thomas. *Mathematical logic*. Undergraduate texts in mathematics. Springer-Verlag, 1994.

8 Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984.

9 Anselm Haak and Heribert Vollmer. A model-theoretic characterization of constant-depth arithmetic circuits. In Jouko A. Väänänen, Åsa Hirvonen, and Ruy J. G. B. de Queiroz, editors, *Logic, Language, Information, and Computation - 23rd International Workshop, WoLLIC 2016, Puebla, Mexico, August 16-19th, 2016. Proceedings*, volume 9803 of *Lecture Notes in Computer Science*, pages 234–248. Springer, 2016. See also: CoRR, abs/1603.09531.

10 Neil Immerman. Expressibility and parallel complexity. *SIAM J. Comput.*, 18(3):625–638, 1989.

11 Neil Immerman. *Descriptive complexity*. Graduate texts in computer science. Springer, 1999.

12 Neeraj Kayal and Ramprasad Saptharishi. A selection of lower bounds for arithmetic circuits. In Manindra Agrawal and Vikraman Arvind, editors, *Perspectives in Computational Complexity: The Somenath Biswas Anniversary Volume*, pages 77–116. Birkhäuser, 2014.

13 Clemens Lautemann, Pierre McKenzie, Thomas Schwentick, and Heribert Vollmer. The descriptive complexity approach to LOGCFL. *J. Comput. Syst. Sci.*, 62(4):629–652, 2001.

14 Leonid Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

15 Meena Mahajan. Algebraic complexity classes. In Manindra Agrawal and Vikraman Arvind, editors, *Perspectives in Computational Complexity: The Somenath Biswas Anniversary Volume*, pages 51–75. Birkhäuser, 2014.

16 Guillaume Malod and Natacha Portier. Characterizing Valiant's algebraic complexity classes. *Journal of Complexity*, 24(1), February 2008.

17 Ramprasad Saptharishi. A survey of known lower bounds in arithmetic circuits. A continuously updated git survey. https://github.com/dasarpmar/lowerbounds-survey.

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

A Appendix

Proof of Lemma 9. Since $L \in \mathfrak{C}$, there is an FO-interpretation $I : \mathrm{STRUC}[\tau_{\text{string}}] \to \mathrm{STRUC}[\tau_{\text{circ}}]$ that uniformly describes a circuit family showing $L \in \mathfrak{C}$. Let k be the length of tuples encoding gates in this circuit family. We now stepwise construct an FO-interpretation still showing $L \in \mathfrak{C}$ but with properties 1, 2 and 3.

Since we will have to adapt encodings of gates as tuples in order to manipulate certain properties related to the numerical predicates, it is relevant for this proof in what way we represent numbers as tuples. We will always use an *most significant bit first* encoding, meaning that the variable containing the most significant bit of number is the left-most variable in a tuple and significance reduces towards the left.

1.: We first construct an FO-interpretation I' that has property 1. This is done by allowing all tuples as gates, but allowing connections between gates only if both gates were already gates in the original circuit. Analogously, we also have to change the formula determining the output gate—otherwise, multiple tuples could become output gates. The only formulae we have to change for this are those for the universe and the predicates E and output. Let $\varphi_{\text{universe}}, \varphi_E, \varphi_{\text{output}}$ be the respective formulae from I. For I' we instead use

$$\begin{aligned}
\varphi'_{\text{universe}}(\overline{g}) &= \top \\
\varphi'_E(\overline{g}_1, \overline{g}_2) &= \varphi_E(\overline{g}_1, \overline{g}_2) \wedge \varphi_{\text{universe}}(\overline{g}_1) \wedge \varphi_{\text{universe}}(\overline{g}_2) \\
\varphi'_{\text{output}}(\overline{g}) &= \varphi_{\text{output}}(\overline{g}) \wedge \varphi_{\text{universe}}(\overline{g})
\end{aligned}$$

Now, in order to additionally achieve properties 2 and 3 from the statement of the lemma, we proceed as follows: For property 2, we add an additional bit as LSB to the encoding of gates and use only those versions of $\wedge$-gates where this bit is 1 and those versions of $\vee$-gates where this bit is 0. For property 3, we add to the encoding of gates a unary encoding of the depth in a certain way. We then only connect two gates if their corresponding gates from the original circuit were connected and the depth increases by 1 from the parent to the child. (This means a lot of these new versions of the gates are not used.)

For both these approaches we want to add bits to the encoding. Since we can only add additional variables to the tuples encoding gates and each variable increases the number of bits by $\log n$, we pad the number of bits to a multiple of $\log n$ and simply add the according number of additional variables to the tuples.

We now formalize the above ideas.

2.: We construct an FO-interpretation I'' that has property 2 in addition to the previous properties. For this, we increase the tuple size by 1. Then the following formulae can be used for I'':

$$\begin{aligned}
\varphi''_{\text{universe}}(\overline{g}x) &= \top \\
\varphi''_{G_\wedge}(\overline{g}x) &= \mathrm{BIT}(x, 0) \\
\varphi''_{G_\vee}(\overline{g}x) &= \neg\mathrm{BIT}(x, 0) \\
\varphi''_{\text{Input}}(\overline{g}x, i_1 \dots i_k y) &= i_1 = 0 \wedge \varphi'_{\text{Input}}(\overline{g}, i_2 \dots i_k y) \\
\varphi''_{\text{negatedInput}}(\overline{g}x, i_1 \dots i_k y) &= i_1 = 0 \wedge \varphi'_{\text{negatedInput}}(\overline{g}, i_2 \dots i_k y) \\
\varphi''_E(\overline{g}_1 x_1, \overline{g}_2 x_2) &= \varphi'_E(\overline{g}_1, \overline{g}_2) \wedge \bigwedge_{i=1}^{2} \psi_{\text{real}}(\overline{g}_i x_i) \\
\varphi''_{\text{output}}(\overline{g}x) &= \varphi'_{\text{output}}(\overline{g}) \wedge \psi_{\text{real}}(\overline{g}x)
\end{aligned}$$

with

$$\psi_{\mathrm{real}}(\overline{g}x) = (\varphi'_{G_\wedge}(\overline{g}) \wedge x = 1) \vee (\varphi'_{G_\vee}(\overline{g}) \wedge x = 0) \vee \exists\overline{i}(\varphi'_{\mathrm{Input}}(\overline{g},\overline{i}) \vee \varphi'_{\mathrm{negatedInput}}(\overline{g},\overline{i})).$$

3.: We construct an FO-interpretation I''' that additionally has property 3. For this, let $\ell \cdot \log n$ be a bound for the depth of the circuit family described by I''. For I''', we increase the tuple size by 2ℓ. The idea is to create for each gate of the old circuit a duplicate for each possible depth within the circuit. This will make the circuit layered. The depth is encoded in the following way: Depth 0 is encoded by the sequence only consisting of 1s. Depth $i+1$ is then encoded by the same sequence as i only the first two 1s are made 0s. This kind of unary encoding is possible since the circuits have logarithmic depth. In the final circuit, gates are only connected if their versions in the old circuit were connected and the child's depth is 1 higher than the parent's. This leads to the following formulae for I''':

$$\begin{aligned}
\varphi'''_{\mathrm{universe}}(\overline{h}\overline{g}x) &= \top \\
\varphi'''_{G_\wedge}(\overline{h}\overline{g}x) &= \varphi''_{G_\wedge}(\overline{g}x) \\
\varphi'''_{G_\vee}(\overline{h}\overline{g}x) &= \varphi''_{G_\vee}(\overline{g}x) \\
\varphi'''_{\mathrm{Input}}(\overline{h}\overline{g}x,\overline{j}\overline{i}y) &= \overline{j} = \overline{0} \wedge \varphi''_{\mathrm{Input}}(\overline{g}x,\overline{i}y) \\
\varphi'''_{\mathrm{negatedInput}}(\overline{h}\overline{g}x,\overline{j}\overline{i}y) &= \overline{j} = \overline{0} \wedge \varphi''_{\mathrm{negatedInput}}(\overline{g}x,\overline{i}y) \\
\varphi'''_{E}(\overline{h}_1\overline{g}_1x_1,\overline{h}_2\overline{g}_2x_2) &= \varphi''_{E}(\overline{g}_1x_1,\overline{g}_2x_2) \wedge \exists\overline{i}\Big((\forall\overline{j} \le \overline{i}\mathrm{BIT}(\overline{h}_1,\overline{j})) \wedge (\forall\overline{j} > \overline{i}\neg\mathrm{BIT}(\overline{h}_1,\overline{j})) \\
&\qquad (\forall\overline{j} \le (\overline{i}-2)\mathrm{BIT}(\overline{h}_2,\overline{j})) \wedge (\forall\overline{j} > (\overline{i}-2)\neg\mathrm{BIT}(\overline{h}_2,\overline{j}))\Big)
\end{aligned}$$

The circuit family described by I''' has properties 1, 2 and 3. ◀

Proof of Theorem 13. The proof idea is very similar to the one used for the decision version. We again begin by proving the unbounded version.

$\#\mathrm{AC}^1 \subseteq \#\mathrm{Win}\text{-}\mathrm{FO}(\mathrm{GPR})$: Let $f \in \#\mathrm{AC}^1$ via the FO-uniform AC^1 circuit family $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$ with the properties from Lemma 9 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

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

be an FO-interpretation showing that $\mathcal{C}$ is uniform. Furthermore, let θ and its subformulae ψ, $\varphi_{\mathrm{Literal}}$ and $\varphi_{\mathrm{trueLiteral}}$ be defined as in the proof of Theorem 10. Then

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

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.

In the following we use $\#\mathrm{Win}(\varphi,\mathcal{A})$ as notation for the number of winning strategies of the verifier for $\mathcal{A} \vDash \varphi$. $\mathrm{Odd}(\overline{y})$ can be constructed such that for all $\mathcal{A}$ we have $\#\mathrm{Win}(\mathrm{Odd}(\overline{y}),\mathcal{A}) \in \{0,1\}$ and $\mathrm{Even}(\overline{y})$ can be constructed such that for all $\mathcal{A}$ we have $\#\mathrm{Win}(\mathrm{Even}(\overline{y}),\mathcal{A}) = 1 - \#\mathrm{Win}(\mathrm{Odd}(\overline{y}),\mathcal{A})$. This means that for all $\mathcal{A}$ it holds that

$$\#\mathrm{Win}(\theta(\overline{y},P),\mathcal{A}) = \#\mathrm{Win}\Big(\big(Q\overline{z}.(\overline{z} < \overline{y}/2 \wedge \varphi_E(\overline{y},\overline{z}))\big)P(\overline{z}) \wedge \neg\varphi_{\mathrm{Literal}}(\overline{z}) \vee \varphi_{\mathrm{trueLiteral}}(\overline{z}), \mathcal{A}\Big)$$

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 number of winning strategies verifying that $P(\overline{g})$ is true 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 $k \in \mathbb{N}$, $\#\mathrm{Win}(P(\overline{g}))$ 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 $\varphi_{\mathrm{trueLiteral}}(\overline{h})$ 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:

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

where $\bigcirc$ is either summation or multiplication depending on the parity of $\overline{g}$.

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

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

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

$k \to k+1$: 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 $\#\mathrm{Win}(\neg\varphi_{\mathrm{Literal}}(\overline{z})) = 0$ and $\#\mathrm{Win}(\varphi_{\mathrm{trueLiteral}}(\overline{z}))$ 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 $\#\mathrm{Win}(\varphi_{\mathrm{trueLiteral}}(\overline{z})) = 0$, $\#\mathrm{Win}(\neg\varphi_{\mathrm{Literal}}) = 1$ and by induction hypothesis $\#\mathrm{Win}(P(\overline{z}))$ is exactly the number of proof trees of the subcircuit of C_n rooted in $\overline{z}$.

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

Since $\#\mathrm{Win}(P(\overline{g}))$ gives the number of proof trees of the subcircuit of $C_{|w|}$ rooted in $\overline{g}$ for arbitrary non-input gates $\overline{g}$ in C_n on input w for any w, it is easy to see that the above formula defines f: The number of winning strategies of the formula behind the recursive definition of P is almost immediately the number of winning strategies for $P(\underline{\mathrm{output}})$, where $\underline{\mathrm{output}}$ is the unique element satisfying $\varphi_{\mathrm{output}}$. By the induction above this is equal to the number of proof trees of circuit C_n.

$\underline{\#\text{Win-FO(GPR)} \subseteq \#\mathrm{AC}^1}$: 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.

Both $\#\mathrm{NC}^1 \subseteq \#\text{Win-FO}(\mathrm{GPR}_{\mathrm{bound}})$ and $\#\mathrm{SAC}^1 \subseteq \#\text{Win-FO}(\mathrm{GPR}_{\mathrm{semi}})$ can—as for the decision version—be shown with the same formula as $\#\mathrm{AC}^1 \subseteq \#\text{Win-FO(GPR)}$ by changing the GPR-operator to a $\mathrm{GPR}_{\mathrm{bound}}$- or $\mathrm{GPR}_{\mathrm{semi}}$-operator, respectively.

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