Characterizing Circuit Complexity Classes by Logics with Recursion*

Anselm Haak[1]

1 **Theoretische Informatik, Leibniz Universität Hannover, Appelstraße, D-30167, Germany,** haak@thi.uni-hannover.de

In this submission we introduce a certain recursion scheme into first-order logic variants of which allow us to capture a number of classes from circuit complexity, namely AC^1, SAC^1 and NC^1 as well as their counting analogues. The advantage of our new approach is that it allows for a unified characterization of multiple important classes.

The idea of descriptive complexity is to characterize complexity classes in a model-theoretic way in order to better understand their structure and use logical methods in order to get new insights about the considered classes. The starting point for this strand of research was Fagin's theorem [3]. It tells us that the complexity class NP of problems decidable by non-deterministic polynomial-time Turing machines can be characterized as the class of problems definable in existential second order logic. Many characterizations followed, among them characterizations of the circuit complexity classes AC^0, NC^1 and TC^0, see e.g. [1, 5]. Characterizations of uniform classes from circuit complexity are of special interest, because the definitions of these classes usually build upon two disconnected models: The uniformity condition is defined via some logic or using a machine model, while the main power of computation stems from the name-giving circuits. Model-theoretic characterizations allow to define these classes in purely logical terms.

Another important kind of classes are so called counting classes. These are classes of functions from inputs to natural numbers that are usually defined by counting the number of some kind of witness for membership in a language. Of course, what kind of witness is considered depends on the specific class. For these kind of classes, only very few model-theoretic characterizations are known. The most prominent one is probably the generalization of Fagin's theorem, stating that #P = #FO: Counting the number of accepting paths of NP-machines can be characterized as counting the number of satisfying assignments to free relational variables in FO-formulae [6]. Besides that, we recently showed that Immermans result $AC^0 = FO$ [5] can also be generalized to the counting setting, showing $\#AC^0 = \#\text{Win-FO}$ [4].

Since the area of descriptive complexity for counting classes is not really well developed, more characterizations of such classes are desirable. In this regard we want to focus on classes from circuit complexity and try to give characterizations of classes defined via Boolean circuits that also extend to the counting setting. Towards this goal, the best known characterizations of NC^1 [1] does not seem to be helpful. On the other hand, we feel like the generalization of $AC^0 = FO$ to the whole AC-hierarchy using formulae of logarithmic length does seem a little unnatural. Instead, we sought inspiration from a result by Compton and Laflamme, characterizing NC^1 by FO with a certain kind of linear recursion [2]. This approach does not seem to generalize to the classes SAC^1 and AC^1, though. Instead of focusing on transferring this kind of recursion to other classes, we define a new kind of recursion that more direcly captures the essence of uniform circuit classes with the counting setting in mind. Introducing this kind of recursion into first-order logic allows us to characterize NC^1, SAC^1 and AC^1 in similar ways. Furthermore, the approach also works in the counting

* This submission is based on joint research with Arnaud Durand and Heribert Vollmer.

© Anselm Haak;
licensed under Creative Commons License CC-BY

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

setting. While the recursion scheme is more directly connected to the definition of uniform circuit complexity classes, the approach still bears the advantage of yielding descriptions of the classes in purely logical terms.

We now want to briefly recall the definition of FO-uniform families of Boolean circuits. A Boolean circuit is a directed acyclic graph. The nodes of this graph—also called gates—are marked with "$\wedge$", "$\vee$" or with an index of an input position. Also, one gate is marked as the output gate. To talk about these circuits in first-order logic, we use the vocabulary $\tau_{\text{circ}} = (E^2, G^1_\wedge, G^1_\vee, \text{Input}^2, \text{negatedInput}^2, \text{output}^1)$. The meaning of the predicates $E, G_\wedge, G_\vee$ and output is straightforward. $\text{Input}(x, i)$ holds, if gate x is an input gate and associated with the i-th input position. An FO-uniform circuit family is a family of circuits $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$ for which there is an FO-interpretation (FO-query) $I : \text{STRUC}[\tau_{\text{string}}] \to \text{STRUC}[\tau_{\text{circ}}]$ such that for all $w \in \{0,1\}^*$: $I(\mathcal{A}_w) = C_{|w|}$. Here, τ_{string} is the vocabulary of binary strings, $\mathcal{A}_w$ is the structure representing string w and $C_{|w|}$ is given as a τ_{circ}-structure.

The recursion scheme we introduce allows for the recursive definition of a predicate P. Syntactically, the constructions looks as follows:

$$[P(\overline{x}, \overline{y}) \equiv Q\overline{z}.(\overline{z} < \overline{y}/2 \wedge \psi(\overline{y}, \overline{z}))\theta(\overline{x}, \overline{y}, P(\overline{x}, \overline{z}))].$$

The recursion runs over the parameter $\overline{y}$. Q is a quantifier, which will either be $\exists$ or $\forall$ depending on the parity of $\overline{y}$. The quantification of $\overline{z}$ is guarded by the formula $\overline{z} < \overline{y}/2 \wedge \psi(\overline{y}, \overline{z})$. This means that the value of the second input is at least halved in each step. If Q is an existential quantifier, the semantics is: $P(\overline{x}, \overline{y})$, if there is a $\overline{z}$ among those satisfying $\overline{z} < \overline{y}/2 \wedge \psi(\overline{y}, \overline{z})$ such that $\theta(\overline{x}, \overline{y}, P(\overline{x}, \overline{z}))$ holds. If Q is a universal quantifier, the semantics is: $P(\overline{x}, \overline{y})$, if for all $\overline{z}$ among those satisfying $\overline{z} < \overline{y} \wedge \psi(\overline{y}, \overline{z})$ also $\theta(\overline{x}, \overline{y}, P(\overline{x}, \overline{z}))$ holds. We denote this kind of recursion by GRP(**g**uarded **r**ecursive **p**redicate definition). In this unrestricted way, the recursion branches polynomially. The depth of the recursion is bounded logarithmically due to the condition $\overline{z} < \overline{y}/2$. This intuatively suggests, that this kind of recursion captures the essence of AC^1-computations. The definition can be altered by bounding the quantifier Q in the universal case or in both cases, resulting in characaterizations of SAC^1 and NC^1, respectively. This is done by changing the semantics e.g. in the universal case to: $P(\overline{x}, \overline{y})$, if for all $\overline{z}$ among the maximal two satisfying $\overline{z} < \overline{y}/2 \wedge \psi(\overline{y}, \overline{z})$ also $\theta(\overline{x}, \overline{y}, P(\overline{x}, \overline{z}))$ holds. We call the restricted versions GRP_{semi} and $\text{GRP}_{\text{bounded}}$, respectively.

In order to capture said classes, we add the above kind of recursive definition in FO-formulae. We call the resulting logics FO(GRP), $\text{FO}(\text{GRP}_{\text{semi}})$ and $\text{FO}(\text{GRP}_{\text{bounded}})$, respectively. We define these logics as follows:

▶ **Definition 1.** FO(GRP) is first-order logic with occurrences of GRP. That is, it consists of all formulae of the form

$$[P_1(\overline{x}, \overline{y}) \equiv Q\overline{z}.(\overline{z} < \overline{y}/2 \wedge \psi_1(\overline{y}, \overline{z}))\ \theta_1(\overline{x}, \overline{y}, P(\overline{x}, \overline{z}))] \cdots$$
$$[P_k(\overline{x}, \overline{y}) \equiv Q\overline{z}.(\overline{z} < \overline{y}/2 \wedge \psi_k(\overline{y}, \overline{z}))\ \theta_k(\overline{x}, \overline{y}, P(\overline{x}, \overline{z}))]$$
$$\varphi(P_1, \ldots, P_k),$$

where the semantics of the recursive definitions is as described above.
$\text{FO}(\text{GRP}_{\text{semi}})$ and $\text{FO}(\text{GRP}_{\text{bounded}})$ are defined analogously.

As usual, we also denote by FO(GRP) the class of languages that exactly consist of those words that are encodings of structures with a specific FO(GRP)-definable property. Since we only consider τ_{string}-structures here, this can be stated as

$$L \in \text{FO(GRP)} :\Longleftrightarrow \text{there is a formula } \varphi \in \text{FO(GRP)} \text{ such that } w \in L \Leftrightarrow \mathcal{A}_w \vDash \varphi \text{ for all } w$$

We use $\mathrm{FO}(\mathrm{GRP}_{\mathrm{semi}})$ and $\mathrm{FO}(\mathrm{GRP}_{\mathrm{bounded}})$ to denote classes of languages in the same way.

This leads us to one of our main theorems.

▶ **Theorem 2.**
(i) $\mathrm{FO}(\mathrm{GRP}) = \mathrm{AC}^1$
(ii) $\mathrm{FO}(\mathrm{GRP}_{\mathrm{semi}}) = \mathrm{SAC}^1$
(iii) $\mathrm{FO}(\mathrm{GRP}_{\mathrm{bounded}}) = \mathrm{NC}^1$

We will sketch the proof for the unbounded version here.

Proof Sketch for Theorem 2.(i). $\supseteq$: Let $L \in \mathrm{AC}^1$ via the FO-uniform AC^1 circuit family $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$. Let $I = (\varphi_E, \varphi_{G_\wedge}, \varphi_{G_\vee}, \varphi_{\mathrm{Input}}, \varphi_{\mathrm{negatedInput}}, \varphi_{\mathrm{output}})$ be an FO-interpretation witnessing that $\mathcal{C}$ is FO-uniform. We can assume without loss of generality that within all C_n, all $\vee$-gates are even and all $\wedge$-gates are odd. We can also assume that for all gates g_1, g_2 in any C_n it holds that $E(g_1, g_2) \Rightarrow g_2 < \frac{g_1}{2}$. Then we can express acceptance of an input by $\mathcal{C}$ with the FO(GRP)-formula

$$[P(\overline{y}) \equiv Q\overline{z}.(\overline{z} < \overline{y}/2 \wedge \varphi_E(\overline{y},\overline{z}))\ P(\overline{z}) \wedge \neg\varphi_{\mathrm{Literal}}(\overline{z}) \vee \varphi_{\mathrm{trueLiteral}}(\overline{z})] \\ \exists\overline{\mathrm{out}}\ \varphi_{\mathrm{output}}(\overline{\mathrm{out}}) \wedge P(\overline{\mathrm{out}})$$

with $\varphi_{\mathrm{Literal}}(x) := \exists i\ \varphi_{\mathrm{Input}}(x,i) \vee \varphi_{\mathrm{negatedInput}}(x,i)$ and $\varphi_{\mathrm{trueLiteral}} := \exists i\ (\varphi_{\mathrm{Input}}(x,i) \wedge S(i)) \vee (\varphi_{\mathrm{negatedInput}}(x,i) \wedge \neg S(i))$. The predicate P defined recursively gives us exactly the evaluation of the gates in the circuit. Thus, the formula $\exists\overline{\mathrm{out}}\ \varphi_{\mathrm{output}}(\overline{\mathrm{out}}) \wedge P(\overline{\mathrm{out}})$ expresses that the output gate of the circuit evaluates to 1.

$\subseteq$: Let $L \in \mathrm{FO}(\mathrm{GRP})$ via the formula

$$[P(\overline{x},\overline{y}) \equiv Q\overline{z}.(\overline{z} < \overline{y}/2 \wedge \psi(\overline{y},\overline{z}))\ \theta(\overline{x},\overline{y},P(\overline{x},\overline{z}))]\ \varphi(P).$$

For simplicity in this proof sketch, we assume that only one predicate P is defined recursively within the formula. Also, P does not occur inside any negation in θ or φ. Ignoring occurrences of P, φ is an FO-formula. Hence, we can build an FO-uniform AC^0 circuit family evaluating φ except for these occurrences. In order to compute the predicate P, we proceed as follows:

For all $\overline{x},\overline{y},\overline{z}$ build a circuit that computes $\theta(\overline{x},\overline{y},P(\overline{x},\overline{z}))$ from $P(\overline{x},\overline{z})$. We can then for all $\overline{x},\overline{y}$ add a gate computing $P(\overline{x},\overline{y})$ by making it an $\wedge$- or an $\vee$-gate depending on the parity of $\overline{y}$ and connecting to it as inputs the gates computing $\theta(\overline{x},\overline{y},P(\overline{x},\overline{z}))$ for all $\overline{z}$ satisfying $\overline{z} < \overline{y}/2 \wedge \psi(\overline{y},\overline{z})$. Now also connect for all $\overline{x},\overline{y},\overline{z}$ the gate computing $P(\overline{x},\overline{z})$ as input to the circuit computing $\theta(\overline{x},\overline{y},P(\overline{x},\overline{z}))$. This gives us an AC^1 circuit family computing for all $\overline{x},\overline{y}$ the value of $P(\overline{x},\overline{y})$.

The results 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. ◀

As already mentioned, this approach also works for the corresponding counting classes. The counting versions of classes from circuit complexity are defined via the concept of proof trees: For this, we first unfold the circuit into a tree. Then a proof tree is a minimal subtree witnessing that this unfolded ciruit evaluates to true. This means that a proof tree contains the output gate, for each contained $\vee$-gate exactly one child and for each contained $\wedge$-gate all children, in such a way that all contained leaves evaluate to true. The counting classes are then defined as classes of functions mapping inputs to the number of proof trees of a circuit family. E.g., a function f is in $\#\mathrm{AC}^1$ if there is an AC^1 circuit family $\mathcal{C}$ such that for each input w, $f(w)$ is the number of proof trees of $\mathcal{C}$ on input w.

On the other hand, logical counting classes can be defined as classes of functions counting the number of winning strategies similar of certain formulae, as it is done for #Win-FO. In the definition of said class, we use the game played over the quantifier prefix—the quantifier-free part is only evaluated to determine the winner of the game. In this case, this does not make much sense, because we need to count the number of strategies leading to a certain value of P. Since occurrences of P are operands of Boolean operators, we have to extend the model-checking game to the Boolean part and determine the winner after reaching atomic formulae.

This leads to the following definition of the model-checking game for FO as a turn-based two-player game: The verifier plays against the falsifier. The verifier tries to show that the formula is true, while the falsifier tries to show that it is not. They play over the syntactic structure of the formula, starting with the outermost operator. Depending on which operator they encounter, one player has to make a choice. If the atom they reach in the end is true, the verifier wins and if it is not, the falsifier wins. The operators are handled as follows:

- $\exists x\psi$: verifier chooses a value for x
- $\forall x\psi$: falsifier chooses a value for x
- $\alpha \vee \beta$: 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 and continue on formula α

For FO(GRP), we change this definition for occurrences of any recursively defined predicate P: When reaching this predicate as an atom, instead of directly checking its truth value, the game continues by unrolling the recursive definiton for P on the encountered inputs. We now define the corresponding counting class as the class of functions mapping inputs to the number of winning strategies of the verifier for a FO(GRP)-formula in the above model-checking game.

▶ **Definition 3.** #Win-FO(GRP) is the class of functions f for which there is an FO(GRP)-formula φ such that for all $w \in \{0,1\}^*$:

$$f(w) = \text{number of winning strategies of the verifier for } \mathcal{A}_w \vDash \varphi.$$

#Win-FO($\mathrm{GRP}_{\mathrm{semi}}$) and #Win-FO($\mathrm{GRP}_{\mathrm{bounded}}$) are defined analogously.

This leads us to our second main theorem, which is:

▶ **Theorem 4.**
(i) #Win-FO(GRP) = $\#\mathrm{AC}^1$
(ii) #Win-FO($\mathrm{GRP}_{\mathrm{semi}}$) = $\#\mathrm{SAC}^1$
(iii) #Win-FO($\mathrm{GRP}_{\mathrm{bounded}}$) = $\#\mathrm{NC}^1$

We have characterized some of the most important circuit complexity classes and their counting counterparts in a unified fashion. Also, the characterizations are quite close to the characterization of AC^0 as FO: Here, no recursive definition is allowed leaving only an FO-formula. This also extends to the counting setting.

As further research, it might be interesting to compare logics characterizing TC^0 with the new logic for NC^1 and also to look for different characterizations of TC^0. Another possibility is to check whether the characterization of NC^1 via a linear kind of recursion by Compton and Laflamme can be transferred to the counting setting. One can also try to generalize the ideas to arbitrary classes from the AC-, SAC- and NC-hierarchies. The problem is that in our characterization we can use the quite simple condition "$\overline{z} < \overline{y}/2$" in order to guarantee at most logarithmic depth. It seems plausible that no similarly simple condition works for powers of $\log(n)$. Still, a similar approach might be possible.

References

1 David A. Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within nc1. *Journal of Computer and System Sciences*, 41(3):274 – 306, 1990.

2 Kevin J. Compton and Claude Laflamme. An algebra and a logic for nc1. *Information and Computation*, 87(1):241 – 263, 1990.

3 Ronald Fagin. Generalized first-order spectra and polynomial-time recognizable sets. In Richard Manning Karp, editor, *Complexity of Computation*, volume 7 of *SIAM-AMS Proceedings*, pages 43–73, 1974.

4 Anselm Haak and Heribert Vollmer. *A Model-Theoretic Characterization of Constant-Depth Arithmetic Circuits*, pages 234–248. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

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

6 Sanjeev Saluja, K. V. Subrahmanyam, and Madhukar N. Thakur. Descriptive complexity of #P functions. *Journal of Computer and System Sciences*, 50(3):493–505, 1995.