Descriptive Complexity of Circuit-Based Counting Classes

Von der Fakultät für Elektrotechnik und Informatik
der Gottfried Wilhelm Leibniz Universität Hannover
zur Erlangung des akademischen Grades
Doktor rerum naturalium
(abgekürzt: Dr. rer. nat.)
genehmigte

Dissertation

von Herrn

M. Sc. Anselm Haak

2021

1. Referent: Prof. Heribert Vollmer, Leibniz Universität Hannover
2. Referent: Prof. Till Tantau, Universität zu Lübeck
Tag der Promotion: 29.09.2021

Acknowledgements

I wish to thank my supervisor Heribert Vollmer, who introduced me to the beauty of theoretical computer science in his lectures and taught an intuitive and accessible approach to theory. I'm deeply grateful for the support and joint research during my time as a PhD student, and for him being a great boss.

I thank Arnaud Durand and Juha Kontinen, who were co-authors for some of the papers this thesis is based on, for productive and fruitful collaboration. Also, I thank Raghavendra Rao for our recent collaboration, as well as Lauri Hella for interesting discussions on a few occasions (leading in particular to the use of game semantics in Definition 3.1).

I thank my colleagues Arne Meier, Jonni Virtema, Fabian Müller, Yasir Mahmood, Rahel Becker, Timon Barlag and Sabrina Gaube, and former colleagues Martin Lück, Maurice Chandoo, Irina Schindler, and Anca Vais for many discussions and conversations, both theory-related and on other aspects of life. Martin, Maurice and Fabian in particular already accompanied me during undergraduate studies. I'm thankful for valuable feedback and comments on my thesis by Fabian, Timon, Sabrina, Arne, and Gerion.

I deeply thank my parents and sister for always being there for me, and always having an open ear for me.

Danksagung

Ich danke meinem Doktorvater Heribert Vollmer, der mir die Schönheit der Theoretischen Informatik in seinen Vorlesungen nähergebracht und immer ein intuitives und zugängliches Herangehen an die Theorie gelehrt hat. Ich bin ihm sehr dankbar für seine Unterstützung und Zusammenarbeit während meiner Zeit als Doktorand und dafür, dass er ein fantastischer Chef ist.

Ich danke Arnaud Durand und Juha Kontinen, Mitautoren eines Teiles der Paper, die dieser Arbeit zugrunde liegen, für die produktive und ergebnisreiche gemeinsame Forschung. Auch danke ich Raghavendra Rao für unsere kürzliche gemeinsame Forschung, und Lauri Hella für interessante Diskussionen zu verschiedenen Gelegenheiten. (Diese führten insbesondere dazu, Spielsemantik für Definition 3.1 zu verwenden.)

Ich danke meinen Kollegen Arne Meier, Jonni Virtema, Fabian Müller, Yasir Mahmood, Rahel Becker, Timon Barlag und Sabrina Gaube, und meinen ehemaligen Kollegen Martin Lück, Maurice Chandoo, Irina Schindler und Anca Vais für viele Diskussionen und Gespräche, sowohl fachlich als auch zu anderen Aspekten des Lebens. Martin, Maurice und Fabian haben mich insbesondere bereits während meiner Zeit im Studium begleitet. Ich bin Fabian, Timon, Sabrina, Arne und Gerion dankbar für wertvolles Feedback und Kommentare zu meiner Dissertation.

Ganz besonders danke ich meinen Eltern und meiner Schwester dafür, dass sie immer für mich da sind und immer ein offenes Ohr für mich haben.

Abstract

In this thesis, we study the descriptive complexity of counting classes based on Boolean circuits. In descriptive complexity, the complexity of problems is studied in terms of logics required to describe them. The focus of research in this area is on identifying logics that can express exactly the problems in specific complexity classes. For example, problems are definable in ESO, existential second-order logic, if and only if they are in NP, the class of problems decidable in nondeterministic polynomial time. In the computation model of Boolean circuits, individual circuits have a fixed number of inputs. Circuit families are used to allow for an arbitrary number of input bits. A priori, the circuits in a family are not *uniformly* described, but one can impose this as an additional condition, e.g., requiring that there is an algorithm constructing them. For any circuit there is a function counting witnesses (or proofs) for the circuit producing the output 1. Consequently, any class of Boolean circuits has a corresponding class of counting functions.

We characterize counting classes in terms of counting winning strategies in the model-checking game for different logics extending first-order logic, namely the classes $\#AC^0$, $\#NC^1$, $\#SAC^1$, and $\#AC^1$. These classes restrict circuits to constant or logarithmic depth and in some cases also with respect to the fan-in of gates. In the case of the logarithmic-depth classes, this also requires new characterizations of the corresponding classes of Boolean circuit, as known results do not seem suitable for this approach. Our characterization of $\#AC^0$ also leads to a new characterization of the related class TC^0. Our results apply both in the non-uniform as well as the uniform setting.

We put our new characterization of $\#AC^0$ into perspective by studying connections to another logic-based counting class, #FO. This class is known to coincide with #P, the class of functions counting the number of accepting computation paths of NP-machines. We study a variant of #FO and the alternation hierarchy inside this class, as well as its connections to the class $\#AC^0$.

Finally, we observe that often it is easier to characterize non-uniform circuit complexity classes than their uniform counterparts. This lends hope to the idea that characterizations of uniform classes could directly imply corresponding results in the non-uniform setting. We prove that this is true for a wide variety of classes, in particular for all classes we consider in this thesis.

Keywords: descriptive complexity, Boolean circuits, arithmetic circuits, counting complexity, finite model theory

Zusammenfassung

In dieser Arbeit untersuchen wir die deskriptive Komplexität von Zählklassen, die auf Boole'schen Schaltkreisen basieren. In der deskriptiven Komplexität wird die Komplexität von Problemen anhand zur Beschreibung erforderlicher Logiken untersucht. Im Fokus steht dabei die Suche nach Logiken, die genau die Probleme in bestimmten Komplexitätsklassen ausdrücken können. Zum Beispiel lassen sich Probleme genau dann in ESO, existenzieller Prädikatenlogik zweiter Stufe, ausdrücken, wenn sie in der Klasse NP liegen, also nichtdeterministisch in Polynomialzeit entscheidbar sind. Im Berechnungsmodell der Boole'schen Schaltkreise haben einzelne Schaltkreise eine feste Anzahl von Eingängen. Deshalb werden Schaltkreisfamilien verwendet, um beliebige Eingabelängen zu ermöglichen. A priori sind die Schaltkreise einer Familie nicht *uniform* beschrieben, aber dies kann zusätzlich gefordert werden, z. B. in Form eines Algorithmus, der sie konstruiert. Es können zudem Funktionen betrachtet werden, die Zeugen (oder Beweise) dafür zählen, dass ein Schaltkreis seine Eingabe akzeptiert. So ergibt sich für jede Klasse von Boole'schen Schaltkreisen eine entsprechende Klasse von Zählfunktionen.

Wir charakterisieren Zählklassen mittels des Zählens von Gewinnstrategien in Modellprüfspielen verschiedener Erweiterungen der Prädikatenlogik, genauer die Klassen $\#AC^0$, $\#NC^1$, $\#SAC^1$ und $\#AC^1$. Diese sind auf konstante oder logarithmische Tiefe und teils auch in Bezug auf den Eingangsgrad von Gattern beschränkt. Im Falle logarithmischer Tiefe erfordert dies neue Charakterisierungen der entsprechenden Klassen Boole'scher Schaltkreise, da die bekannten Resultate für diesen Ansatz nicht geeignet erscheinen. Unsere Charakterisierung von $\#AC^0$ führt zudem zu einer neuen Charakterisierung der verwandten Klasse TC^0. Unsere Resultate gelten im uniformen wie auch im nicht-uniformen Fall.

Wir setzen unsere Charakterisierung von $\#AC^0$ in Bezug zu bekannten Resultaten. Dazu untersuchen wir Beziehungen zu einer weiteren logikbasierten Zählklasse, #FO. Diese fällt bekanntermaßen mit #P zusammen, der Klasse aller Funktionen, die die Anzahl der akzeptierenden Berechnungspfade von NP-Maschinen zählen. Wir betrachten eine Variante von #FO und die Alternierungshierarchie darin, sowie ihre Verbindungen zur Klasse $\#AC^0$.

Schließlich beobachten wir, dass nicht-uniforme Klassen oft einfacher zu charakterisieren sind als ihre uniformen Gegenstücke. Das führt zu der Hoffnung, dass Charakterisierungen uniformer Klassen direkt entsprechende Resultate für nicht-uniforme Klassen implizieren. Wir beweisen, dass dies für eine Vielzahl von Klassen, insbesondere alle hier betrachteten, der Fall ist.

Schlagworte: Deskriptive Komplexität, Boole'sche Schaltkreise, Arithmetische Schaltkreise, Zählkomplexität, Endliche Modelltheorie

Contents

List of Figures

1 Introduction

How difficult is it to solve different problems on a computer? More specifically, what resources are required to do so, that is, how much time do the computations take and how much storage space is required for them? These and similar questions are at the heart of complexity theory. In other words, the main goal of this area is to understand the computational complexity of problems by determining what resources are required and sufficient to solve them.

Classically, this study focused on decision problems, which are problems that can be answered with either yes or no. Many natural problems are of this form, or can be formulated in this way. With regard to the domain of arithmetic, one could ask whether $a + b = c$ for three numbers a, b and c. In graph theory, one might ask whether there is a path between two given vertices in some directed graph. In logics, the question could be whether a given propositional formula is satisfiable. Formally, decision problems, or languages, are sets of strings over some fixed alphabet, for example sets of binary strings. In order to study the computational complexity of such problems, a computation model is fixed. Most prominently, machine-based models are used, which are abstractions of how computer programs work. Next, relevant complexity measures have to be determined. Classically, the running time and the storage space are mainly considered, but there are many other complexity measures such as the number of processors for parallel computations, or specific complexity measures with regard to randomization or nondeterminism. The latter allows a program to check several possible solutions in parallel without requiring additional time or storage space.

Most fundamentally, tractable and intractable problems are distinguished. Here, the line often is drawn between those problems that can be solved in polynomial time and those that cannot be solved in polynomial time, where the running time is measured with respect to the length of the encoding of the input. More generally, complexity theory studies upper and lower bounds for the complexity of problems. Upper bounds are results stating that problems can be solved using certain resources, and are often directly shown by finding a clever algorithm to solve them. On the other hand, lower bounds are results stating that problems cannot be computed within certain resource bounds. Towards obtaining lower bounds, classes of problems, called *complexity classes* are considered. For example, the class of problems solvable in polynomial time is usually called P. When looking for evidence that certain resource bounds are strictly required to solve a problem, it is necessary to better understand these classes. Here, the central notion of completeness comes into play. Different variants of the notion of problems being complete for a class intuitively capture the concept of problems being among the most difficult problems in that class. Completeness can be used to show that problems cannot be solved within certain resource bounds: A problem complete for a class cannot

be contained in a strict subclass of it, as otherwise the most difficult problems in the class would be contained in the subclass and the two classes would collapse.

While tools like the notion of completeness help to understand how problems fit into different complexity classes, understanding the relations between different classes, and especially proving separations between them, is still a huge challenge. Many questions in this regard are open and we often only gather evidence that a problem cannot be solved within certain resource bounds instead of outright proving it. An example is the famous P-NP-problem, which asks whether the class P mentioned above coincides with the class NP, which allows for the same running time of computations, but additionally allows the usage of nondeterminism, that is, algorithms can check several possibilities in parallel. This problem was first precisely stated in seminal papers by Cook and Levin independently [Coo71, Lev73] and the complexity of many natural computational problems depends on it, see for example an early paper by Karp establishing NP-completeness of 21 natural problems [Kar72]. The P-NP-problem is one of the major open problems in computer science and one of the "Millenium Problems" of the Clay Mathematics Institute, which contains 7 of the most important open problems in mathematics [Cla]. There have been many attempts to solve it, and a continuously updated list of such attempts is maintained by Woeginger [Woe16], but none of the attempts has been successful, yet. Still, P = NP is considered very unlikely by most researchers and would be a major breakthrough, since many NP-complete problems were studied for decades without much progress towards solving them in polynomial time. Consequently, NP-completeness can still be used as evidence that a problem cannot be solved in polynomial time, as otherwise P and NP would be equal. To help with the understanding of complexity classes, both out of theoretical interest and in hopes of obtaining new upper and lower bounds with respect to those classes, another important topic of complexity theory is to study structural properties of the classes. Structural properties are for example closure properties of the classes or characterizations of them in terms of different computation models or using entirely different frameworks, possibly far outside the scope of complexity theory a priori.

The approach of classifying problems by their computational complexity with respect to certain resource bounds on some computation model has been applied to a variety of settings. So how does this thesis fit into this landscape? The classical setting sketched above considers classes of decision problems, defined based on running time or storage space required to algorithmically solve them. In contrast, we study classes of counting problems with respect to certain resource bounds on Boolean circuits (or, equivalently, counting arithmetic circuits). Next, we briefly explain these terms and their relations to classical complexity theory.

Counting problems ask for the number of solutions to a problem, so the answer to such a problem is a natural number. Just like decision problems, counting problems occur naturally in many areas. Even more, many decision problems have a corresponding counting version, which asks for the number of solutions to the decision problem. As examples, recall the decision problems mentioned earlier. The counting version of the problem of determining whether there is a path between two vertices in a directed graph asks for the number of such paths instead, at least if there are no cycles. The counting

version of the satisfiability problem asks how many satisfying assignments there are for a given propositional formula φ. The latter problem is called #Sat and just as its decision counterpart is widely believed to be intractable.

Boolean circuits and (counting) arithmetic circuits are computation models that are an abstraction of actual electronic circuits. Recall that in comparison, machine-based models such as the Turing machine are an abstraction of algorithms in some programming language. Consequently, these circuits can be seen as a model of lower level computation. A different view on circuits also allows to use them as a model for parallel computation.

We study classes of counting problems defined in terms of Boolean circuits with regard to their descriptive complexity. This means that we are interested in logics that can define exactly the problems in such complexity classes. Usually, this approach focuses on characterizations in terms of logics based on first-order logic. Descriptive complexity aims to improve the understanding of complexity classes by providing a different view on them and allowing to use methods from logic to further study them.

We will now cover these topics in more detail. First, we will remind the reader of required foundations of first-order logic. Then we cover counting problems and prominent classes of such problems. We then go into more detail on circuit complexity, also covering counting classes in this context. Having introduced the objects of our study, that is, counting classes in circuit complexity, we then go on to discuss previous research in descriptive complexity. Finally, we list our contributions to the field as well as what parts of this thesis have previously been published.

1.1 First-Order Logic

In first-order logic, properties of first-order structures are expressed. We restrict ourselves to the case of relational structures. Such structures consist of a universe or domain, which is a non-empty set, and relations over that set. First-order formulae can quantify elements, that is, make statements of the form "there is an element" or "for all elements", and talk about elements or tuples of elements being in certain relations. Furthermore, arbitrary Boolean combinations of such statements can be made. In order to have a limited number of symbols usable in these formulae, a vocabulary is fixed in advance. It defines what relation symbols can be used as well as their arities. In doing so, they also restrict the kind of structures under consideration. For example, directed graphs can be seen as first-order structures over the vocabulary (E), where E is a binary relation symbol. A graph is represented as a structure over this vocabulary by using the set of nodes as the universe and interpreting the symbol E as the edge-relation of the graph. Similarly, many kinds of algebraic structures can be seen as first-order structures, albeit relational structures might not always be sufficient. As an example, consider the following first-order formula over vocabulary (E):

$$\forall x \neg \mathsf{E}(x,x) \wedge \forall x \forall y \, \mathsf{E}(x,y) \leftrightarrow \mathsf{E}(y,x).$$

This formula expresses that a graph is undirected and contains no loop. For this, it states that for all nodes, there is no loop on that node, and for all pairs of nodes, there is an edge from the first to the second if and only if there is an edge from the second to the first. Structures satisfying a formula are also called *models* of the formula. Generally, first-order logic is very powerful and solving problems on first-order formulae algorithmically is very difficult, if not impossible. For example, Church's Theorem states that determining whether a given first-order formula is valid is not decidable, see for example the textbook by Boolos and Jeffrey [BJ87]. This changes drastically when we only consider finite structures, bringing us to the area of finite model theory. Here, problems, often have very low complexity, as will become evident for the case of model-checking in Section 1.4.

There are different ways of defining semantics of first-order logic, some of which are of particular value to us. Usually, semantics are defined compositionally, defining whether a structure $\mathbf{A}$ is a model of a formula φ based on the corresponding statement for the direct subformulae of φ. Instead, game semantics can be used, where a structure is a model of a formula, if a certain player has a winning strategy in a suitable two-player game. Skolem functions provide yet another view on semantics of first-order logic: An existential quantifier $\exists y$ expresses the existence of an element depending on all variables quantified to the left of the quantifier. Let these be the variables $x_1, \ldots, x_n$. Then the variable y can be equivalently replaced by a suitable function symbol, getting variables $x_1, \ldots, x_n$ as arguments. Satisfying assignments to these function symbols are called Skolem functions.

1.2 Counting Complexity

As mentioned earlier, while decision problems ask for the existence of a solution for a given input, counting problems ask for the number of solutions. In other words, they ask for the number of *witnesses* for an input being a positive instance of the corresponding decision problem. In the case of propositional formulae, satisfying assignments are witnesses for the satisfiability of a given formula. Consequently, counting witnesses for Sat amounts to counting satisfying assignments, as requested by the problem #Sat. While it is often possible to define and study a related decision problem for a counting problem and this approach has its merits, it is also interesting to study the complexity of the actual counting problems. This is supported by the fact that there are many intractable counting problems whose corresponding decision problems are tractable. For example, we have evidence that the problem of counting satisfying assignments of propositional formulae in disjunctive normal form is intractable [DHK05], while the corresponding decision problem can trivially be solved in polynomial time.

Formally, while decision problems are sets of strings, counting problems are functions mapping such strings to natural numbers. This means that counting problems do not actually need to be defined in terms of counting solutions. For example, the problem of computing the the sum of two numbers given in binary, or the problem of computing the permanent of a matrix, a function defined similarly to the determinant, are also

counting functions. It should be noted though, that the permanent has characterizations in terms of counting certain substructures inside of graphs. Namely, both counting cycle-covers of directed graphs and counting perfect matchings of bipartite graphs are equivalent to computing the permanent of a corresponding matrix.

The area of counting complexity was initiated by Valiant in 1979. In his seminal paper, he studied the complexity of computing the permanent of a matrix [Val79b]. Valiant introduced the class #P, which can be seen as the counting version of the class NP. This class contains all those counting functions that arise as the number of different nondeterministic choices of an NP-machine that lead to the machine accepting an input. Valiant showed that while the determinant can be computed in polynomial time, the permanent is complete for the class #P. As he showed that #Sat and several other well-known problems share this property, this yielded evidence that the permanent cannot be computed in polynomial time. Further research also studied the class #L, which is the restriction of #P to computations that can be done with storage space logarithmic in the input length, and showed that this class actually characterizes the complexity of the determinant [ÀJ93, Vin91, Tod91a, Val92, MV97].

Beside independent interest in a complexity theory for counting problems, the research in this area also yielded results relating back to classical complexity theory, such as Toda's Theorem [Tod91b].

1.3 Circuit Complexity

Circuit complexity, as a subfield of complexity theory, studies the computational complexity of problems on computation models that resemble the structure of actual electronic circuits. This area goes back to work on so-called switching circuits by Shannon, Riordan and Lupanov as early as 1938 [Sha38, RS42, Sha49, Lup58]. The most common type of circuits, which were studied extensively, are Boolean circuits. These give a very low-level view on computations, with circuits consisting of gates computing very simple operations on bits, often just conjunction, disjunction and negation. These gates are connected by wires or edges to form a circuit. In Figure 1.1, an example of such a Boolean circuit can be seen.

While Boolean circuits correspond to lower level computations than machine-based models, they are still a very abstract model. For example, timing-related issues like race conditions as well as restrictions with regard to the fan-out of gates are not considered. From a broader perspective, Boolean circuits are a special case of the model of arithmetic circuits. Here, gates compute addition and multiplication as well as possibly different operations over an arbitrary semiring, which is a certain kind of algebraic structure. Boolean circuits can be seen as arithmetic circuits over the Boolean semiring, with conjunction as multiplication and disjunction as addition on that semiring, and with negation as the only additional operation. Arithmetic circuits are sometimes used as a model for a slightly more high-level view on circuit complexity, by for example allowing addition and multiplication on larger numbers to be computed by individual gates. In contrast, in Boolean circuits these operations already require relatively big circuits, as

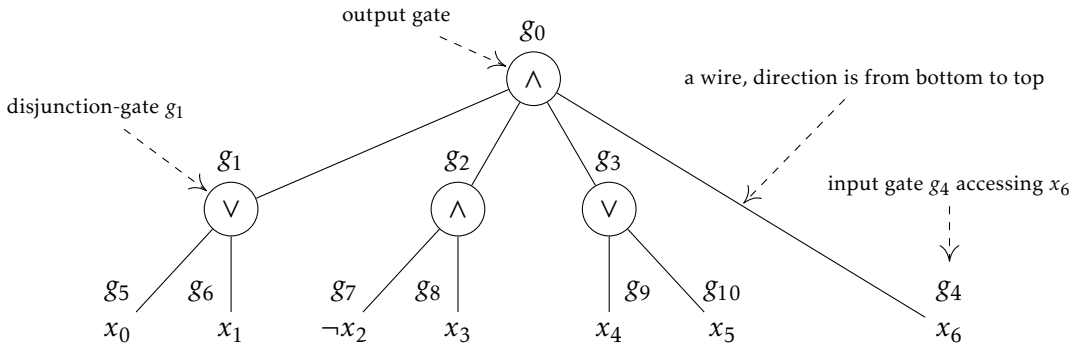

Figure 1.1: An example of a Boolean circuit on 7 inputs.

they have to be implemented using bitwise operations.

In complexity theory, and hence also in the field of circuit complexity, we are mostly interested in the asymptotic complexity of problems, that is, the question of how resource requirements increase for inputs of increasing sizes. The complexity theoretic approach to circuit complexity was heavily influenced by a textbook by Savage [Sav76]. Since circuits are just a finite set of gates connected by wires, they only work on fixed input sizes, though. Consequently, in order to be able to consider asymptotic resource bounds in the models of Boolean and arithmetic circuits, we need to consider families of such circuits. A circuit family is a collection of circuits that contains one circuit for each potential input size, that is, for each number of Boolean inputs. Ultimately, this means that a circuit family is an infinite object, as it contains one circuit for each natural number.

In circuit complexity, the resource bounds we are most interested in are the size of circuits, that is, how many gates they contain, as well as their depth, that is, the length of a longest path from one of their input gates to their output gates. Size and depth of circuit families refer to the size and depth of the circuits in the family measured in relation to the input length. A central complexity class in circuit complexity is AC^0. Here, families of Boolean circuits of polynomial size and only constant depth are allowed, with gates computing conjunctions, disjunctions and negations, and potentially having unbounded fan-in. A typical problem in AC^0 is the decision problem for addition of natural numbers mentioned in the beginning of the introduction. Note that the depth of a circuit relates to the time it takes to do its computation, since operations of gates that are not on the same path to the output gate can be done in parallel. By this argument, addition being in AC^0 can be interpreted as addition being doable in a constant number of steps for numbers with an arbitrary amount of digits. Intuitively, one could expect that this would not be possible, as doing addition using the method learned in school requires to go through the number digit by digit and remember the carry in each step. The actual circuit family showing that addition is possible in constant depth uses a method called carry-lookahead instead. This method looks at all digits in parallel and determines each digit of the outcome of the addition, in a sense, in parallel. As this

method allows to add to numbers with arbitrarily many digits in constant time, this result is also relevant in practice, and gates for addition in microelectronics use more involved variants of the same method, see for example the spanning tree carry lookahead adder by Lynch and Swartzlander [LS92].

Other prominent classes in circuit complexity, which capture the complexity of interesting natural problems such as multiplication and division of binary numbers, or multiplication in a non-solvable group, are TC^0, NC^1, SAC^1, and AC^1. The class TC^0 is defined similarly to AC^0, but allowing for gates that can determine whether the majority of their Boolean inputs are 1. Equivalently, this class can be defined using gates that can determine whether the number of their input bits that are 1 exceeds a certain threshold. In contrast to AC^0 and TC^0, the other three classes allow for circuit families of logarithmic depth, with AC^1 being otherwise defined exactly like AC^0. The classes NC^1 and SAC^1 are defined similarly to AC^1, but with additional restrictions: For NC^1, gates may only have bounded fan-in, that is, each gate can take only 2 inputs. For SAC^1, this restriction only applies to conjunction gates, but disjunction gates can have arbitrary fan-in. The names of these classes go back to Cook, who used NC as an abbreviation for *Nick's class*, as Nick Pippenger first studied this class [Pip79]. Here, NC is the generalization of NC^1 to depth $O(\log n)^i$. Correspondingly, the first letters in AC, TC, and SAC stand for *alternation*, *threshold*, and *semi-unbounded alternation*, respectively. Complete problems for the class TC^0 are problems related to computing summation, multiplication and division, where the latter was long open in the uniform setting, which we will talk about in more detail later, and finally shown by Hesse [Hes01]. It is also known that these operations cannot be done in AC^0 [FSS84]. On the other hand, it is known that all of these operations can also be done in NC^1, but it is unknown, whether NC^1 provides any additional power: The question whether $TC^0 = NC^1$ is one of the central open problems of this area, sometimes called the P-NP-problem of circuit complexity. For SAC^1 and AC^1, natural complete problems are based on logics: The problem of evaluating so-called acyclic Boolean conjunctive queries is complete for SAC^1 [GLS01], while the model-checking problem for intuitionistic propositional logic with one variable is complete for AC^1 [MW14].

All of the above classes are in a sense tractable: When the circuits are provided to us, they can be evaluated sequentially in polynomial time, as they only contain a polynomial number of gates. Even more, as already argued, given enough processors to compute the operations of as many gates as possible in parallel, the depth of circuits determines the time required for the computation. This means that in the case of the above classes, when the circuits are provided to us and with a polynomial number of processors, problems in these classes can be solved in logarithmic time, and potentially even in constant time in case of AC^0 and TC^0—depending on the model of computation. More precisely, if we only allow operations on a bounded number of bits to be computed in a single computation step, AC^0 and TC^0 would still require logarithmic time. This relationship between resource bounds for circuits and for related models of parallel computation were studied by Borodin in 1977 [Bor77]. It also illustrates that beside their importance as a model for low level computations, circuits can also be seen as a model for parallel computation.

For these arguments, we need access to the correct circuit for a given input. Circuit families are an inherently non-uniform model of computation, though. This refers to the fact that they are an infinite collection of individual circuits, with a priori no efficient way to obtain the circuit for a given input size. In contrast, machine-based models are usually inherently uniform, as in their case a single, finite description of the machine or program uniformly describes the behavior for inputs of arbitrary sizes. Of course, studying the complexity of problems with regard to solving them with (non-uniform) circuit families is still interesting: It helps understand the possibilities and limits for the construction of Boolean circuits for a problem—for example designed by humans for specific input sizes.

From a complexity theoretic standpoint, though, the non-uniformity of circuit families in a sense hides the actual power of the circuits as a computation model, because it adds additional power by handling arbitrary properties that depend only on the size of the input. For this reason, uniform variants of the classes are considered, where the actual computational power of the circuits is not overshadowed by the fact that circuits are provided non-uniformly. Uniform circuit families were first introduced for this purpose by Borodin [Bor77]. Here, a uniformity condition is added separately, outside the circuits themselves. For a uniform circuit family, we require that the circuits for specific input sizes can be produced in an efficient manner, for example in polynomial time in a machine-based computation model such as the Turing machine. Depending on the considered classes, different forms of uniformity are adequate. In case of the classes discussed above, our intuition tells us that these classes are close in power to machine-based models with logarithmic running time. To avoid still hiding the actual power of the circuits by too much computational power given to the uniformity, the power of the model for the uniformity should be similar to the expected power of the circuits. Consequently, the usual uniformity conditions in this setting use variants of computation models allowing for logarithmic running times. Note that in this case, instead of requiring that the correct circuit can be computed within certain resource bounds, it is required that relevant information on the circuit can be queried efficiently. For example, it is required that the question whether there is a wire between two specific gates in the circuits can be answered within the desired resource bounds. Such a query-based definition is necessary, as the computation model might not even be powerful enough to even output all gates in the circuits. The different forms of uniformity and their suitability for different circuit complexity classes have been studied extensively [Ruz81, BIS90, BI94]. Also, in the context of discussing suitable uniformity conditions for our setting, we discuss known results in detail, see Section 2.4.

Using uniform circuit families, it was possible to characterize circuit complexity classes in terms of other models of parallel computation, most prominently alternating Turing machines, making the relations between resource bounds of circuit families and other computation models precise. Alternations here refer to alternations in using either existential or universal nondeterministic steps: While usual nondeterminism is existential, as it asks for the existence of a computation with a certain result, universal nondeterminism asks whether all possible computations have a certain result. As an example for such a characterization, the class AC^0 coincides with the class LH of all

decision problems that can be solved by alternating Turing machines in logarithmic time and using a bounded number of alternations [SV84]. Similarly, the class NC^1 coincides with the class ALOGTIME of all decision problems that can be solved by such machines in logarithmic time (but any number of alternations) [Ruz81]. Other classes can be characterized in a similar manner.

Using reasonable uniformity conditions, we also know the following connections between the above circuit complexity classes and classical complexity classes:

$$AC^0 \subsetneq TC^0 \subseteq NC^1 \subseteq L \subseteq NL \subseteq SAC^1 \subseteq AC^1 \subseteq P.$$

Here, L and NL refer to the classes of problems that are decidable in logarithmic space on deterministic and nondeterministic Turing machines, respectively. It is also worth mentioning that important classes from classical complexity theory often have characterizations in terms of circuit complexity. For example, P coincides with the class of problems that can be solved by polynomial-size families of Boolean circuits [Lad75, PF79]. More characterizations of this kind, in particular for nondeterministic classes, were proven by Venkateswaran [Ven92]. These connections to classical complexity theory in combination with lower bound results in circuit complexity, such as $AC^0 \neq TC^0$, also lend some hope to the idea of obtaining new lower bounds in classical complexity theory using lower bound techniques from circuit complexity.

Often, circuit complexity classes also have characterizations in other computation models besides alternating Turing machines and similar models. For example, NC^1 coincides with the class of problems that can be solved by so-called bounded-width branching programs, another relatively low-level model of computation [Bar89]. Moreover, the class SAC^1 coincides with the maybe better known class LOGCFL, which is the class of languages that are in a certain sense similar to context-free languages, that is, to languages that can be accepted by nondeterministic pushdown automata. This class can also be defined in terms of so-called nondeterministic auxiliary pushdown automata [Ven91].

Counting Classes in Circuit Complexity

Due to their success in classical complexity theory, counting classes were also considered in the context of Boolean circuits, at least in case of circuits using only conjunction and disjunction freely as well as negation for input gates. Typically, counting classes are defined by introducing a form of counting witnesses for acceptance into a computation model. In the case of Boolean circuits, the type of witness most commonly considered are so-called proof trees. Intuitively, a proof tree is a simple structure that is minimal with the property that it constitutes proof that a certain circuit accepts its input. The notion of counting proof trees of Boolean circuits leads to counting variants of most of the classes covered earlier, namely we obtain the classes $\#AC^0$, $\#NC^1$, $\#SAC^1$, and $\#AC^1$, which are defined as the classes of functions that arise as the number of proof trees of AC^0, NC^1, SAC^1, and AC^1 circuit families, respectively. The study of counting classes in circuit complexity was initiated by Vinay in 1991, who studied the class $\#SAC^1$ [Vin91]. Papers followed on the counting classes $\#AC^0$

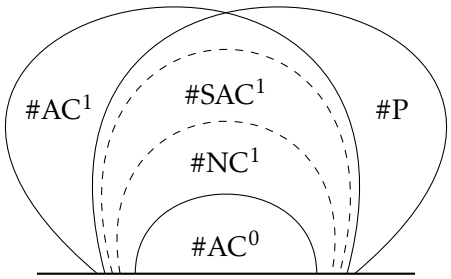

Figure 1.2: Relationship between central counting classes in this thesis, assuming P ≠ NP. Dashed lines indicate that separations are not known.

by Agrawal et al. and Ambainis et al. [AAD00, ABL98], as well as #NC^1 by Buss et al. and Caussinus et al. [BCGR92, CMTV98]. Note that due to connections to arithmetic formulae obtained by Buss et al., the class #NC^1 was in fact considered much earlier, at least as early as 1979 by Valiant [Val79a]. Alternatively, these classes can also be defined in terms of so-called counting arithmetic circuits using the same resource bounds. These are arithmetic circuits as introduced earlier, but the semiring is fixed to be the natural numbers with usual addition and multiplication. Furthermore, inputs are restricted to only 0 and 1, in other words Boolean inputs, but may be negated. Note that #NC^1 and #SAC^1 can be characterized by counting accepting paths in bounded-width branching programs and nondeterministic auxiliary pushdown automata, respectively, corresponding to the characterizations of their classical counterparts [CMTV98, Vin91]. Arithmetic circuits and their computational power are of current focal interest of research in computational complexity theory, see for example recent surveys by Kayal and Saptharishi, and Mahajan [KS14, Mah14] as well as the continuously maintained survey by Saptharishi [Sap15].

In the uniform setting, the same uniformity conditions are considered for the above counting classes as for their classical counterparts. The different forms of uniformities have not been extensively studied in this context, though. The relationships between the central counting complexity classes considered in this thesis, assuming adequate forms of uniformity, are illustrated in the inclusion diagram in Figure 1.2. The assumption #P ≠ NP is required for #P $\not\subseteq$ #AC^1.

The importance of counting classes based on circuits is further emphasized by the fact that classical counting complexity classes often have characterizations in this model. For example, #P can be characterized in terms of counting arithmetic circuit families of polynomial size and additionally bounded degree, which refers to the degree of the polynomial obtained when expanding the arithmetic expression computed by the circuit [Ven92]. Similarly, the class #L has a characterization in terms of so-called skew circuits, where multiplication gates may only have two inputs and one of them needs to be an input gate or a constant [Tod92].

1.4 Descriptive Complexity

In descriptive complexity theory, one studies the complexity of describing problems with formulae, in contrast to computational complexity where the complexity of computing solutions to problems is studied. Here, by a problem being described by a formula, we mean that the set of finite models of the formula is exactly the set of positive instances of the problem. The primary goal of this area is to identify logics that exactly capture complexity classes, that is, logics that can describe exactly those problems that are in a certain complexity class. Such characterizations are often called *model-theoretic*, as complexity classes are characterized as classes of models of certain formulae. Consequently, this field lies in the intersection of finite model theory, the theory of finite models of formulae, and computational complexity theory. The goal of this approach is to better understand the structure of classes, getting a different view on their defining properties. In doing so, the hope is to be able to use tools from finite model-theory to get new insights, especially in the direction of new lower bounds. For example, the famous result that the parity-function cannot be computed in AC^0 [FSS84], separating AC^0 from TC^0, was independently obtained by Ajtal in a purely logical way [Ajt83].

The area of descriptive complexity was started in 1974 by Fagin, who famously showed that existential second-order logic, or ESO, captures the complexity class NP [Fag74]. Here, ESO refers to the logic obtained by extending first-order logic by existential quantifiers for relations, that is, existential second-order quantifiers. But what does it even mean for a logic to capture a complexity class? For this, finite first-order structures are suitably encoded as binary strings and structures are identified with their encodings. This way, properties defined in logics, that is, sets of first-order structures, are associated with languages. More precisely, the set of models of a fixed sentence is associated with the set of encodings of these models. In consequence, a logic $\mathcal{L}$ can be said to capture a complexity class $\mathfrak{C}$, if the set of languages associated with $\mathcal{L}$-formulae is exactly $\mathfrak{C}$. On a technical level, a unique encoding of structures requires to introduce a built-in order on the domains of first-order structures. There are ways to formulate model-theoretic characterizations without built-in order, but in the case of very weak complexity classes this is often not possible. For the purpose of this introduction, assume that a built-in order is always present.

In the following years, much progress was made in the area of descriptive complexity. In 1977, Stockmeyer introduced the polynomial hierarchy, which generalizes the class NP, and pointed out the implications of Fagin's Theorem for its descriptive complexity [Sto76]. In 1982, Immerman and Vardi independently showed that P is captured by first-order logic with an operator computing least fixed points of certain subformulae, denoted by FO(LFP) [Imm82, Imm86, Var82]. This characterization uses built-in order. The question whether P can also be captured by a logic without built-in order (in a suitable sense) is open and an active area of research [Gro08]. Also in 1982, building on Fagin's characterization of NP, Lynch studied the descriptive complexity of languages decidable in nondeterministic polynomial-time in a more fine-grained manner [Lyn82]. Many results followed, and descriptive complexity was also extended to circuit complexity classes. In 1987, Immerman proved characterizations of many

classes, including NL and L, and gave an overview of known model-theoretic characterizations [Imm87]. In 1989, he showed that first-order logic captures AC^0, as well as a generalization of this result to all classes AC^i in the AC-hierarchy. This hierarchy is a generalization of the classes AC^0 and AC^1, with the class AC^i allowing for circuit families of depth $O(\log n)^i$ [Imm89]. For the latter, he introduced the logic FO[$t(n)$] for functions t, in which certain parts are repeated $t(n)$ times before evaluating the formula. After this, two characterizations were obtained for the class NC^1. Based on ideas related to Barrington's characterization of NC^1 in terms of bounded-width branching programs, Barrington et al. showed that NC^1 is captured by first-order logic extended by certain generalized quantifiers for multiplication in a non-solvable group [BIS90]. Generalized quantifiers are a generalization of existential and universal quantifiers well-known in model-theory and finite model-theory. Barrington et al. also obtained a characterization of TC^0 in terms of first-order logic with majority quantifiers, which express that the majority of elements have a certain property. The second characterization of NC^1 was obtained by Compton and Laflamme, who showed that NC^1 is captured by first-order logic with an operator for a variant of primitive recursion for relations [CL90]. Primitive recursion is a concept from computability theory, see for example the monograph by Soare [Soa87]. Using a different approach than the previous characterization of P, Grädel characterized this class and several others by facilitating that the satisfiability problems for certain fragments of propositional logic are complete for said classes [Grä92]. For example, the class P was characterized by the fragment SO-HORN of second-order logic, where the quantifier-free part is a HORN-formula. This means that when treating atoms as literals of propositional logic, the quantifier-free part is a HORN-formula, that is, each clause has at most one positive literal. Similarly, NL was characterized by SO-KROM defined analogously using KROM-formulae, meaning that each clause has only two literals. Note that in the setting of propositional logic, the satisfiability problems for HORN- and KROM-formulae are complete for P and NL, respectively. The former follows from Cook's original proof that SAT is NP-complete, while the latter was shown by Jones et al. [JLL76]. In 2001, Lautemann et al. showed that SAC^1 can be characterized using generalized quantifiers, similar to the earlier characterization of NC^1 by Barrington et al. More precisely, they proved that this class is captured by first-order logic extended by so-called groupoidal quantifiers [LMSV01]. Also, a characterization of the class PP, which can be seen as the decision version of #P, as well as the counting hierarchy based on this class, was obtained by Kontinen in 2009 [Kon09].

There was considerably less progress in descriptive complexity in the setting of counting complexity. In 1994, Frandsen et al. studied the descriptive complexity of the class $\#AC^0$ in terms of arithmetic expressions in the uniform setting [FVB94]. In 1995, Saluja et al. transferred Fagin's Theorem to the counting setting [SST95]. They introduced a counting version of the class ESO by using free relation variables instead of existentially quantified ones, and asking for the number of satisfying assignments to these relation variables. They showed that the resulting class #FO captures the counting complexity class #P. As long as a notion of counting is defined, a logic capturing a class has a similar meaning as in the case of decision problems. By identifying first-order structures with their encodings, each formula of such a logic $\mathcal{L}$ has an associated

counting function. We say that $\mathcal{L}$ captures a counting complexity class $\#\mathfrak{C}$, if the class of counting functions associated with $\mathcal{L}$-formulae is exactly $\#\mathfrak{C}$. Saluja et al. also studied the alternation hierarchy inside of #FO, that is, the hierarchy where the quantifier prefix of formulae is restricted. In 1996, Compton and Grädel [CG96] obtained a characterization of SpanP, another counting complexity class, by an extension of $\#\mathrm{FO}^{\mathrm{rel}}$, and studied the effect of omitting the built-in order both from said extension and from the class $\#\mathrm{FO}^{\mathrm{rel}}$. Recently, there were some further developments, starting in 2016 with another characterization of $\#\mathrm{AC}^0$ by Haak and Vollmer [HV16], which is also part of this thesis. In the following year, Arenas at el. developed an extensive framework for descriptive complexity of counting complexity classes, called quantitative second-order logic or QSO [AMR17, AMR20]. In this framework, they obtained model-theoretic characterizations of many prominent counting classes, among others for the classes #P and #L. It should be noted that the characterization of $\#\mathrm{AC}^0$ in terms of arithmetic expressions mentioned above could also be embedded in the QSO framework. In 2019, Haak et al. [HKM+19] studied the descriptive complexity of counting classes in the range from #P to $\# \cdot \mathrm{NP}$ in terms of team logics.

1.5 Contributions

In this thesis, we study the descriptive complexity of counting classes in circuit complexity. All model-theoretic characterizations we obtain apply both in the non-uniform and in the uniform setting. As a starting point, we transfer Immerman's $\mathrm{AC}^0 = \mathrm{FO}$ [Imm89] to the counting setting. For this, we introduce notions of counting witnesses for first-order logic in Chapter 3. Here, we consider different forms of witnesses, namely winning strategies in Hintikka's model-checking game [Hin82], winning strategies in the full first-order model-checking game obtained by extending Hintikka's game by rules for Boolean connectives, and Skolem functions. We show that all three variants lead to the same counting class when disregarding inputs of length 1. We call the resulting class #Win-FO.

In Chapter 4, we study the descriptive complexity of constant-depth classes, using the logical counting classes introduced in the previous chapter. We show in Section 4.1 that the newly defined counting class captures $\#\mathrm{AC}^0$, that is, $\#\mathrm{AC}^0 = \#\text{Win-FO}$. Here, the difference to the earlier approach to characterizing $\#\mathrm{AC}^0$, that is, in terms of arithmetic expressions, is that our definitions provide an interpretation in terms of counting witnesses. Also, our result is the first characterization of the non-uniform version of $\#\mathrm{AC}^0$. Furthermore, this result leads to a new model-theoretic characterization of TC^0 in Section 4.2, showing that the class coincides with the class of languages definable in first-order logic extended by bitwise excess to #Win-FO-functions.

When viewing functions in our new class #Win-FO as functions counting Skolem-functions, this class is very similar to the class #FO, which captures #P as mentioned earlier. When allowing free function variables instead of free relation variables in the definition of #FO, both classes are defined in terms of counting assignments to certain function variables. For this reason, we investigate the relationship between both

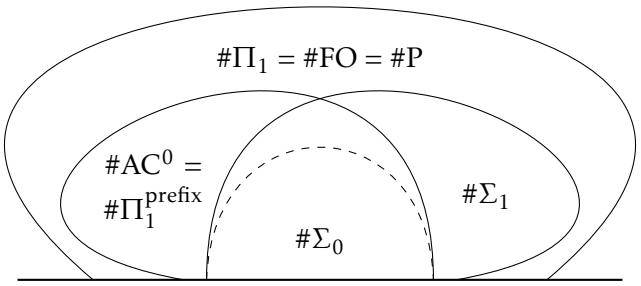

Figure 1.3: Alternation hierarchy in #FO and relationship to $\#AC^0$. Dashed lines indicate that separations are not known.

classes in Chapter 5. It turns out that #Win-FO coincides with a syntactical fragment $\#\Pi_1^{\text{prefix}}$ of $\#\Pi_1 \subseteq \#FO$. Here, $\#\Pi_1$ refers to the fragment of #FO only allowing universal first-order quantifiers. This is shown in Section 5.1. Due to this connection, we study the alternation hierarchy inside our version of #FO in Section 5.2. The results we obtain are illustrated in the inclusion diagram in Figure 1.3, disregarding inputs of length 1. This shows that our classes behave differently to the versions studied by Saluja et al. [SST95]. In Section 5.3, we show that the class $\#\Sigma_1$, while containing some #P-complete problems, can be approximated efficiently in the sense of fully polynomial-time randomized approximation schemes, FPRAS for short. Furthermore, as the whole alternation hierarchy collapses to the class $\#\Pi_1$, we study the hierarchy based on the number of universal quantifiers inside of $\#\Pi_1$ in Section 5.4, showing that this hierarchy is strict. For this result, we build on results by Grandjean and Olive for related classes in the decision setting [GO04]. Additionally, we consider the potential of the original hierarchy by Saluja et al. [SST95] for studying $\#AC^0$ in Section 5.5. Here, we show that $\#AC^0$ is incomparable to all but the smallest and the full class in their framework, indicating that their hierarchy does not help in better understanding the class.

In Chapter 6, we turn our attention to logarithmic-depth classes, namely $\#NC^1$, $\#SAC^1$ and $\#AC^1$. As the known model-theoretic characterizations for the corresponding decision classes do not seem to transfer to the counting setting, we first identify suitable, new characterizations in the decision setting. For this, we introduce a new operator for what we call guarded predicative recursion, GPR for short, in Section 6.1, which allows recursive definitions of predicates. Conceptually, this operator is similar to the relational primitive recursion used by Compton and Laflamme [CL90], and the definition is inspired both by their logic FO+RPR as well as Immerman's logic $FO[t(n)]$. In Section 6.2, we show that first-order logic extended by different variants of GPR captures NC^1, SAC^1 and AC^1. As intended, the new logics resemble the structure of circuits more directly than previous characterizations of the classes. This allows us to transfer the above result to the counting setting by considering functions counting winning strategies in a suitable model-checking game, leading to model-theoretic characterizations of $\#NC^1$, $\#SAC^1$ and $\#AC^1$ in Section 6.3. Together with our results in Section 4.1, this means we

prove model-theoretic characterizations of the counting versions of the central classes in circuit complexity in a unified framework based on counting winning strategies in the first-order model-checking game.

In Chapter 7, we observe that it is often easier to prove model-theoretic characterizations for non-uniform classes than for their uniform counterparts. This leads to the idea that uniform characterizations might generally imply a corresponding result in the non-uniform setting. We prove this to be true for a wide range of classes. For this, we note that the result becomes trivial when providing the non-uniform information in a specific way: Assume that two uniform classes coincide. Consider for both of these classes the non-uniform version obtained by providing an advice in addition to the actual input to a problem, which can hold arbitrary information that only depends on the size of the input. As the non-uniformity is independent of the computation model and uses the uniform classes only in a black-box manner, these non-uniform versions of the classes also coincide. Consequently, we show in Sections 7.1 and 7.2 that for a wide range of circuit complexity classes and logical classes, respectively, the above notion of non-uniformity coincides with the notion usually used in the respective model. As these are general results for the two models, they could potentially also be used to transfer different results from the uniform to the non-uniform setting, apart from model-theoretic characterizations.

1.6 Publications

Chapters 3 and 4 are based on a paper that was published at the *Workshop on Logic, Language, Information and Computation* [HV16] and a corresponding journal article in the *Annals of Pure and Applied Logic* [HV19]. The thorough investigation of different ways of defining the class #Win-FO in Chapter 3 goes beyond the content of those papers: There, what we now call #Win-FO$^{\text{Hintikka}}$ was studied under the name #Win-FO. Also, the observations on differences with regard to inputs of length 1, which also affect other chapters, are new. The results in Chapter 5 have been published at *Computer Science Logic* [DHKV16] and the *Journal of Computer and System Sciences* [DHKV21]. Here, the proofs for our result for the variable hierarchy, covered in Section 5.4 of this thesis, were rewritten, going into more detail especially regarding subtle differences to the framework for related decision classes by Grandjean and Olive. The model-theoretic characterizations of logarithmic-depth classes in the FO[BIT]-uniform setting, which appear in Chapter 6, were published at the *Symposium on Logic in Computer Science* [DHV18]. Here, we complete the picture by proving the corresponding non-uniform characterizations, which were not published before. Finally, the results in Chapter 7 are new and unpublished. This applies both to the results on the correspondence of different forms of non-uniformity in circuit complexity and logics, as well as their implications for the transfer of model-theoretic characterizations from the uniform to the non-uniform setting.

Page left intentionally blank to have matching page numbers with the printed version.

2 Preliminaries

In this chapter we introduce definitions and preliminary results required for descriptive complexity in general and descriptive complexity of counting classes in circuit complexity in particular. Since descriptive complexity is concerned with model-theoretic characterizations of complexity classes, we then need to cover both topics from complexity theory, especially circuit and counting complexity, as well as first-order and in a limited setting second-order logic. We also present known results from descriptive complexity, expanding on the non-technical overview given in the introduction.

We now begin by listing some general notation used throughout the thesis. For a set A and $n \in \mathbb{N}$, we denote by A^n the Cartesian product of n copies of A. For two sets A, B with $A \cap B = \emptyset$, we denote by $A \sqcup B$ the disjoint union of A and B. For a statement A, we denote by $[\![A]\!]$ the truth value of statement A as a value from $\{0,1\}$.

For any function f, we denote by $\mathrm{dom}(f)$ the domain of f. For big O notation, we use the calligraphic letter $\mathcal{O}$, i.e., for $f\colon \mathbb{N} \to \mathbb{N}$, $\mathcal{O}(f)$ is the set of functions growing at most as fast as f, modulo constant factors. Let A, B, C be sets. For any functions $f\colon A \to B$ and $g\colon B \to C$, we denote by $g \circ f$ the composition of f and g, i.e., for any $x \in A$, we have $g \circ f(x) = g(f(x))$. For any function $f\colon A \times B \to C$, we denote by $f(x, \cdot)$ the function $f'\colon B \to C$ with $f'(a) = f(x, a)$ obtained by currying and fixing the first argument to x. This extends to functions with more than two arguments as well as fixing different arguments than the first one.

Let Σ be a finite set, or *alphabet*. A *word* over Σ is a finite sequences of elements, or *symbols*, of Σ. The set of all words over Σ is denoted by Σ^*. We also call sets of words *languages*. For $v, w \in \Sigma^*$, we denote by $v \circ w$ or vw the concatenation of v and w. If $a \in \Sigma$, $|w|_a$ denotes the number of occurrences of a in w. Finally, for any language $L \subseteq \Sigma^*$ we denote by $c_L\colon \Sigma^* \to \{0,1\}$ the characteristic function of L, which is the function with $c_L(x) = 1$ if and only if $x \in L$ for all $x \in \Sigma^*$. In this thesis we will limit ourselves to the alphabet $\{0,1\}$ and languages over this alphabet. For an introduction to formal languages and related notions and notations, see [HMU07].

2.1 Complexity Theory

We assume the reader to be familiar with the standard concepts of complexity theory, in particular with the notion of (nondeterministic) Turing machines and related notions. For an introduction to the area, see the textbook by Arora and Barak [AB09] or Sipser [Sip97]. Classically, decision problems were in the focus of research in complexity theory, that is, languages $L \subseteq \{0,1\}^*$. The resource bounds most commonly considered are time and space, i.e., the runtime and storage space required by algorithms deciding

a language. Furthermore, nondeterminism often helps characterize the complexity of natural problems. A nondeterministic algorithm or machine can *guess* how to continue its computation, and is considered to accepts its input if there is a sequence of guesses that lead to the machine accepting. This can also be seen as verifying multiple possible solutions at once. Nondeterminism can further be generalized to both existential and universal nondeterminism. Existential nondeterminism is the usual kind of nondeterminisnm mentioned above, while universal nondeterminism means that the machine accepts if all possible guesses lead to the machine accepting. Based on this, one can also consider alternating machines, where steps in the computation use existential and universal nondeterminism alternatingly.

We want to recall a few complexity classes that stem from this classical approach, and which are relevant for this work. P (NP) is the class of languages that can be decided by deterministic (nondeterministic) polynomial-time Turing machines. L (NL) is the class of languages that can be decided by deterministic (nondeterministic) logarithmic-space Turing machines. DLOGTIME is the class of languages that can be decided by deterministic logarithmic-time Turing machines. In this case, the machine has random access to the input, as otherwise it would not have any meaningful power. Similarly, ALOGTIME is the class of languages that can be decided by alternating logarithmic-time Turing machines, and LH is the class of languages that can be decided by alternating logarithmic-time Turing machines with a bounded number of alternations.

We now want to briefly recall the notion of completeness of a problem for a class. If $\leq$ is a reducibility, that is, a reflexive and transitive relation between languages, and $\mathfrak{C}$ a complexity class. Then we say that a language L is *complete for* $\mathfrak{C}$ *under* $\leq$ (or *complete for* $\mathfrak{C}$ *under* $\leq$*-reductions*), if $L \in \mathfrak{C}$ and for all $L' \in \mathfrak{C}$, we have $L' \leq L$. We say that $\mathfrak{C}$ is closed under $\leq$, if for all $L \in \mathfrak{C}$ and $L' \leq L$, we have $L' \in \mathfrak{C}$. In this case, completeness formalizes the notion of a language being among the most computationally difficult languages in $\mathfrak{C}$. Whenever we encounter specific reducibilities in this thesis, we will introduce the required notions before.

For the classes mentioned above, we have the following inclusion structure:

$$\text{DLOGTIME} \subseteq \text{L} \subseteq \text{NL} \subseteq \text{P} \subseteq \text{NP}.$$

In the following, when mentioning completeness results, we assume suitable reducibilities. P is often considered the class of tractable problems. Typical complete problems for this class are satisfiability of HORN-formulae as well as reachability in hypergraphs and the circuit value problem, denoted CVP, that is, the problem of evaluating a given Boolean circuit on a given input. Recall that a HORN-formula is a propositional formula in conjunctive normal form with at most one positive literal in each clause. For the hardest problems in NP no sub-exponential time algorithms are known. Typical complete problems are the general propositional satisfiability problem Sat as well as several problems on graphs such as Clique. For NL, typical complete problems are satisfiability of KROM-formulae as well as reachability in directed graphs. Recall that a KROM-formula is a propositional formula in conjunctive normal form with at most two literals in each clause. Typical complete problems for L are the problem to determine

whether an undirected graph is acyclic and the undirected graph accessibility problem, i.e., answering the question whether the is a path between two given vertices in a given undirected graph. Furthermore, many important arithmetic operations such as addition and multiplication can be computed in L. Although DLOGTIME is a very weak class it allows for computations such as computing the length of an input or arithmetic on numbers with logarithmically many bits, which also allows to decode and encode simple pairing functions on binary words.

While not in the focus of this work, we want to briefly mention non-uniformity in the context of classical complexity classes. This notion of non-uniformity will have significance to the proofs in Chapter 7. Intuitively, a computation model is called non-uniform, if it can show different behavior on inputs of different lengths, or in other words, there is no uniform description of the behavior for arbitrary input lengths. As Turing machines are inherently uniform, since there is always a single machine describing the behavior on arbitrary inputs, non-uniformity has to be added separately in this case. In order to add non-uniformity to a Turing machine, apart from the actual input we give the machine access to an additional advice. This advice is described by a function only depending on the input length and provides additional information that is not necessarily computable from the input. Let $\mathfrak{C}$ be a class of languages and $\mathfrak{F}$ be a class of functions from $\{0,1\}^*$ to $\{0,1\}^*$. A language L is in the class $\mathfrak{C}/\mathfrak{F}$ if there is a language $B \in \mathfrak{C}$ and a function $A \in \mathfrak{F}$, the *advice function*, such that $x \in L$ if and only if $\langle x, A(1^{|x|})\rangle \in B$. Here, $\langle\cdot,\cdot\rangle$ denotes a suitable pairing function. The most common example of non-uniform classes are those where $\mathfrak{F}$ is poly, which is the class of all polynomially length-bounded functions, leading to classes like P/poly and NL/poly. For these classes, concatenation is typically used as the pairing function.

An important variant of Turing machines are so-called probabilistic Turing machines. A probabilistic Turing machine can be viewed as a nondeterministic Turing machine, where instead of nondeterministically guessing bits, additional bits are obtained as the result of fair coin tosses. This way, one can assign a probability to the machine accepting. While a multitude of classes arise from this concept, we will only cover the case of classes based on majority here. This means that a machine in this setting accepts an input if it accepts it with probability greater than $\frac{1}{2}$. For a complexity class $\mathfrak{C}$ based on deterministic Turing machines, the corresponding probabilistic class defined in this way is denoted by P$\mathfrak{C}$. The most prominent examples of such classes are PP and PL, which were first studied by Gill [Gil77]. Important problems captured by such classes are, e.g., the problem Majority-Sat.

Before talking about actual counting classes defined by introducing a notion of counting into certain computational models, we want to briefly mention function classes based on deterministic complexity classes. For any complexity class defined via deterministic Turing machines by certain resource bounds, one can also define a functional version of the class. These classes are denoted by adding an F in front of the class and are classes of functions mapping from $\{0,1\}^*$ to $\mathbb{N}$. For example, FP is the class of functions $f\colon \{0,1\}^* \to \mathbb{N}$ computable in deterministic polynomial time.

2.1.1 Counting Classes

A *counting problem* is a function $f : \{0,1\}^* \to \mathbb{N}$. The term counting problem stems from the fact that many of these problems count witnesses for the membership of inputs in some decision problem: For example, the NP-complete problem Sat mentioned above asks whether a given propositional formula is satisfiable. Hence, a satisfying assignment of the formula is called a witness for the formula being in Sat. The corresponding counting problem is the problem #Sat, asking for the number of witnesses for a given formula being in Sat, that is, the number of satisfying assignments of the formula. The study of counting classes was started by Valiant [Val79b]. He introduced #P, the class of counting problems that correspond to the number of accepting computation paths of a nondeterministic polynomial-time Turing machine. For a nondeterministic Turing machine M let $\mathrm{acc}_M(x)$ and $\mathrm{rej}_M(x)$ denote the number of accepting and rejecting computation paths of M on input x, respectively.

Definition 2.1. A function $f : \{0,1\}^* \to \mathbb{N}$ is in #P, if there is a nondeterministic polynomial-time Turing machine M such that for all inputs $x \in \{0,1\}^*$,

$$f(x) = \mathrm{acc}_M(x).$$

Obviously this definition uses a form of witness counting as well: An accepting computation path can be seen as a witness for an NP-machine accepting its input. In a similar vein, counting versions of other classes were introduced, such as #L, the counting version of the class L [ÀJ93, Vin91, Tod91a, MV97].

A typical complete problem for #P (under reasonable reductions) is the problem #Sat while a typical complete problem for #L is the problem, given a directed graph $\mathbf{G} = (V, E)$ and two nodes $s, t \in V$, to count the number of s-t-paths in $\mathbf{G}$. These classes also capture the complexity of important and natural problems that are not necessarily defined in terms of counting of witnesses: The problem of computing the permanent of a matrix is complete for #P [Val79b], while the complexity of computing the determinant is characterized by the class #L [Vin91, Tod91a, Val92, MV97]. For the latter, the closure of #L under subtraction, called GapL, is required, as the determinant function maps to the integers. It should be noted, though, that such counting problems might also have characterizations in terms of counting witnesses. For example, the permanent of a matrix M is exactly the function counting cycle-covers of the directed graph whose adjacency matrix is M, which are disjoint sets of cycles that contain all nodes of the graph.

2.1.2 Circuit Complexity Theory

While machine-based models resemble (potentially high-level) programming languages, circuits model how problems can be solved in hardware. Besides their importance in the context of solving problems in hardware, they are also useful in the context of weak complexity classes and parallel computation. For a thorough introduction to the topic, see the textbook by Vollmer [Vol99]. We begin with the common special case of Boolean circuits, which are circuits that only work with values 0 and 1.

Boolean Circuits

We first need to define a few notions concerning Boolean functions. A *Boolean function* is a function $f : \{0,1\}^n \to \{0,1\}$ for some $n \in \mathbb{N}$. A *family of Boolean functions* is a family $(f_i)_{i\in\mathbb{N}}$, where f_i is an i-ary Boolean function for all $i \in \mathbb{N}$. Typical examples of Boolean functions are the the k-ary disjunction $\vee_k$ and the k-ary conjunction $\vee_k$ for $k \in \mathbb{N}$, the negation $\neg$ as well as the families $\vee = (\vee_k)_{k\in\mathbb{N}}$ and $\wedge = (\wedge_k)_{k\in\mathbb{N}}$, which are the families of Boolean functions corresponding to disjunction and conjunction for an arbitrary number of inputs, respectively. A *Boolean basis* is a finite set of Boolean functions and families of Boolean functions. In the context of Boolean circuits, the bases $\mathcal{B}_0 := \{\wedge_2, \vee_2, \neg\}$, $\mathcal{B}_1 := \{\wedge, \vee, \neg\}$ and $\mathcal{B}_2 := \{\wedge_2, \vee, \neg\}$ are of special importance.

Definition 2.2. Let $\mathcal{B}$ be a Boolean basis and $n \in \mathbb{N}$. A *Boolean circuit over $\mathcal{B}$ on n inputs* is a tuple $(V, E, \alpha, \beta, \text{out})$ with the following properties:

- (V, E) is a directed acyclic graph (or *dag*),
- α is an injective function $E \to \mathbb{N}$,
- β is a function $V \to \{0,\dots,n-1\} \cup \mathcal{B}$,
- if $\beta(g) \in \{0,\dots,n-1\}$ for some $g \in V$, then g has in-degree 0 in the graph (V, E),
- if $\beta(g) \in \mathcal{B}$ for some $g \in V$ and m is the in-degree of g in (V, E), then $\beta(g)$ either is a Boolean function with arity m or a family of Boolean functions, and
- $\text{out} \in V$.

Elements of V are also called *gates* and out is called the *output gate* of the circuit. We say that β labels the gates of the circuit (by the index of an input bit or a function from $\mathcal{B}$). Note that 0-ary functions in the base correspond to constants 0 or 1. Gates labeled with the index of an input bit by β are called *input gates*.

Remark 2.3. Throughout most of this work we use a restricted definition of Boolean circuit adapted for the considered complexity classes, see Remark 2.7 and Definition 2.8.

We now want to define the function computed by a circuit. Intuitively, this function is obtained by evaluating any gate according to its label and returning the value of the output gate. For this, a label $i \in \mathbb{N}$ means that the gate accesses the i-th input bit. The function α is used to get an ordering of the inputs of any gate.

For a formal definition, we first define a function giving the value of an arbitrary gate in a circuit on a given input. Let $n \in \mathbb{N}$, $\mathbf{C} = (V, E, \alpha, \beta, \text{out})$ be a Boolean circuit on n inputs and $x_0,\dots,x_{n-1} \in \{0,1\}$. Let $g \in V$ and $g_1,\dots,g_m$ be the predecessors of g in (V, E) with $\alpha(g_1) < \cdots < \alpha(g_m)$. The *value of g in $\mathbf{C}$ on input $x = x_0 \dots x_{n-1}$*, denoted by $\text{val}(\mathbf{C}, g, x)$, is defined as follows:

- if $\beta(g) \in \{0,\dots,n-1\}$, then $\text{val}(\mathbf{C}, g, x) := x_{\beta(g)}$,

- if $\beta(g)$ is a Boolean function, then the value $\mathrm{val}(\mathbf{C}, g, x)$ is inductively defined as $(\beta(g))(\mathrm{val}(\mathbf{C}, g_1, x), \ldots, \mathrm{val}(\mathbf{C}, g_m, x))$,
- if $\beta(g) = (f_i)_{i\in\mathbb{N}}$ is a family of Boolean functions, then $\mathrm{val}(\mathbf{C}, g, x)$ is inductively defined as $f_m(\mathrm{val}(\mathbf{C}, g_1, x), \ldots, \mathrm{val}(\mathbf{C}, g_m, x))$.

The *function computed by* $\mathbf{C}$ is $\mathrm{val}(\mathbf{C}, \mathrm{out}, \cdot)$, the function mapping any input to the value of the output gate of $\mathbf{C}$ on that input. By abuse of notation we denote by $\mathbf{C}$ not only the circuit but also the function computed by it. As for a non-input gate g the values of its predecessors are used as arguments for the function $\beta(g)$, we sometimes call its predecessors the *inputs of* g.

By viewing the concatenation of the input bits of a Boolean circuit as a word over $\{0,1\}$, Boolean circuits can compute functions taking binary strings as inputs. As a Boolean circuit has a fixed number of input bits, we need to consider families of such circuits when we want to compute functions on binary strings of arbitrary lengths, for example characteristic functions of languages over $\{0,1\}$.

Definition 2.4. A *family of Boolean circuits over* $\mathcal{B}$ is a family $C = (\mathbf{C}_n)_{n\in\mathbb{N}}$, where $\mathbf{C}_n$ is a Boolean circuit over $\mathcal{B}$ on n inputs for all $n \in \mathbb{N}$.

More generally we also call families of Boolean circuits over any base as well as families of other kinds of circuits, introduced later in this section, *circuit families*. Let $C = (\mathbf{C}_n)_{n\in\mathbb{N}}$ be a family of Boolean circuits. The function computed by C is the Boolean function mapping $x \mapsto \mathbf{C}_{|x|}(x)$ for all $x \in \{0,1\}^*$. Similar is for individual Boolean circuits, we denote this function by C by abuse of notation.

The complexity measures used with regard to families of Boolean circuits are typically the Boolean base as well as size and depth of the circuit families with respect to the input length. The *size* of a circuit is the number of gates and the *depth* is the length of a longest path from any input gate to the output gate. For functions $s, d\colon \mathbb{N} \to \mathbb{N}$, a circuit family $C = (\mathbf{C}_n)_{n\in\mathbb{N}}$ is of *size* s and *depth* d, if the size and depth of $\mathbf{C}_n$ are at most $s(n)$ and $d(n)$, respectively, for all $n \in \mathbb{N}$. Let $\mathcal{B}$ be a Boolean basis and $s, d\colon \mathbb{N} \to \mathbb{N}$. We denote by $\mathrm{CIRCUIT}(\mathcal{B}, s, d)$ the class of circuit families over $\mathcal{B}$ of size $O(s)$ and depth $O(d)$.

Sometimes it is useful to consider Boolean circuits in *input normal form*, which means that negations only occur on the level of inputs. This notion corresponds to the notion of negation normal form in the context of formulae. If $\neg \in \mathcal{B}$, we denote by $\mathrm{CIRCUIT}^{\mathrm{input-nf}}(\mathcal{B}, s, d)$ the class of circuit families in input normal form over $\mathcal{B}$ of size $O(s)$ and depth $O(d)$.

For a Boolean basis $\mathcal{B}$ and sets S and D of functions on the natural numbers, we denote by $\mathrm{CIRCUIT}(\mathcal{B}, S, D)$ the class of circuit families over $\mathcal{B}$ of size $s \in O(S)$ and depth $d \in O(D)$ and similarly for $\mathrm{CIRCUIT}^{\mathrm{input-nf}}(\mathcal{B}, S, D)$ in case that $\neg \in \mathcal{B}$.

We say that a family of Boolean circuits C *accepts* a language L, if $C(x) = c_L(x)$ for all $x \in \{0,1\}^*$. By abuse of notation, we also denote by $\mathrm{CIRCUIT}(\mathcal{B}, s, d)$ the class of languages accepted by circuit families from $\mathrm{CIRCUIT}(\mathcal{B}, s, d)$ and similarly for sets S and D of functions.

We are now in a position to define the usual complexity classes in circuit complexity.

Definition 2.5. For $i \in \mathbb{N}$ we define

- $\mathrm{NC}^i := \mathrm{CIRCUIT}(\mathcal{B}_0 \cup \{f_0, f_1\}, n^{O(1)}, (\log n)^i)$,
- $\mathrm{SAC}^i := \mathrm{CIRCUIT}^{\mathrm{input-nf}}(\mathcal{B}_2 \cup \{f_0, f_1\}, n^{O(1)}, (\log n)^i)$,
- $\mathrm{AC}^i := \mathrm{CIRCUIT}(\mathcal{B}_1 \cup \{f_0, f_1\}, n^{O(1)}, (\log n)^i)$, and
- $\mathrm{TC}^i := \mathrm{CIRCUIT}(\mathcal{B}_1 \cup \{\textsc{Maj}, f_0, f_1\}, n^{O(1)}, (\log n)^i)$,

where f_i is the 0-ary constant function with output i and $\textsc{Maj}$ is the family $(\textsc{Maj}_n)_{n\in\mathbb{N}}$ of majority functions, formally $\textsc{Maj}_n(x_1, \dots, x_n) := [\![\sum_{i=1}^n x_i > \frac{n}{2}]\!]$.

We will also call circuit families with the corresponding restrictions on their resources AC^i, NC^i, SAC^i and TC^i circuit families, respectively.

The classes SAC^i for $i \geq 1$ can equivalently be defined using the base $\{\vee_2, \wedge, \neg\}$ instead of $\mathcal{B}_2$. By De Morgan's law, it is then clear that all of the above classes are closed under complementation, as all other classes already allow the usage of negations arbitrarily. Also note that NC^i, AC^i, and TC^i can equivalently be defined using input normal form for all i. For NC^i and AC^i this follows from De Morgan's law. For TC^i it is easy to see that for all $n \in \mathbb{N}$ and $x_1, \dots, x_n \in \{0, 1\}$:

$$\begin{aligned}
\neg\textsc{Maj}(x_1, \dots, x_n) &= \left[\!\!\left[\neg \sum_{i=1}^n x_i > \frac{n}{2} \right]\!\!\right] \\
&= \left[\!\!\left[\sum_{i=1}^n x_i \leq \frac{n}{2} \right]\!\!\right] \\
&= \left[\!\!\left[\sum_{i=1}^n \neg x_i > \left\lceil \frac{n}{2} \right\rceil - 1 \right]\!\!\right] \\
&= \textsc{Maj}(\neg x_1, \dots, \neg x_n, \underbrace{1, \dots, 1}_{\lfloor \frac{n}{2} \rfloor + 1}, \underbrace{0, \dots, 0}_{\lceil \frac{n}{2} \rceil - 1}).
\end{aligned}$$

Note that $n = \left\lfloor \frac{n}{2} \right\rfloor + 1 + \left\lceil \frac{n}{2} \right\rceil - 1$ for all $n \in \mathbb{N}$. Together with De Morgan's law this shows that TC^i can equivalently be defined using input normal form. In the context of counting in Boolean circuits, which is covered in the following subsection, it is standard to only work with circuits in input normal form.

Remark 2.6. Due to $\wedge_0 = f_0$ and $\vee_0 = f_1$, we could omit f_0, f_1 from the definitions of AC^i and TC^i and f_1 from the definition of SAC^i with only minor changes on the syntactical level. Furthermore, for all the classes the constants are computable using non-constant gates in circuits with at least one input bit.

The most important classes among these both in general as well as for this work are AC^0, NC^1, SAC^1, TC^0 and AC^1. We now want to give some intuition on the power of these classes by stating known connections between them as well as connections to

other complexity classes. For this it should be noted that all of the classes are inherently non-uniform: The computation model uses a different circuit for each input length. Due to this, they are not contained in any of the classical classes. To overcome this one can add a separate uniformity condition, e.g., by requiring that the circuits of a family are computable from the input length (given in unary) in polynomial time. We will cover the topic of uniformity more thoroughly in Subsection 2.1.2 as well as Section 2.4. For the moment we just want to state that the following connections are well known using suitable uniformity conditions, such as FO[BIT]-uniformity introduced later, see, e.g., the monograph by Vollmer [Vol99]:

$$\mathrm{DLOGTIME} \subseteq \mathrm{AC}^0 \subsetneq \mathrm{TC}^0 \subseteq \mathrm{NC}^1 \subseteq \mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{SAC}^1 \subseteq \mathrm{AC}^1 \subseteq \mathrm{P}.$$

Here, it is worth mentioning that P can be characterized in terms of polynomial size families of Boolean circuits. We also know that $\mathrm{TC}^0 \subsetneq \mathrm{PP}$ [All99]. Besides these relations to classical classes, there are also very close relations to other models of parallel computation, most prominently to alternating Turing machines. It is known that AC^0 coincides with LH [SV84] and NC^1 coincides with ALOGTIME [Ruz81]. More generally, depth and size of bounded fan-in circuit families are related to time and space on alternating Turing machines, respectively, with these relations changing slightly when using unbounded fan-in gates, as studied for example in early work by Borodin [Bor77]. Moreover, the class SAC^1 has several characterizations on different models, as was shown by Venkateswaran [Ven91]. Most notable, he showed that $\mathrm{SAC}^1 = \mathrm{LOGCFL}$, the latter being the class of all languages reducible in logarithmic space to a context-free language (a definition of the standard notion of context-free languages can be found in [HMU07]). Furthermore, he gave a characterization of SAC^1 in terms of so-called nondeterministic auxiliary pushdown automata. Separations apart from $\mathrm{AC}^0 \neq \mathrm{TC}^0$ are not known. The question whether the classes TC^0 and NC^1 coincide is one of the central open problems in circuit complexity.

All of the above circuit complexity classes share the following important property: All functions in their bases are commutative. For that reason, we do not need the function α from the definition of Boolean circuits when only dealing with circuits from these classes.

Remark 2.7. Removing the function α from the model of Boolean circuits and limiting all of the above classes to only allow circuits in input normal form does not change the considered classes (for SAC^1 input normal form is already given by definition). In the context of counting (see next section), this model is more natural. Hence, we will generally use this model for the sake of consistency.

For completeness, we give the formal definition of the modified model next.

Definition 2.8. Let $\mathcal{B}$ be a Boolean basis with $\neg \in \mathcal{B}$ such that for for all $f \in \mathcal{B}$, f is commutative. A *Boolean circuit in input normal form over $\mathcal{B}$ on n inputs* is a tuple $(V, E, \beta, \mathrm{out})$ with the following properties:

- (V, E) is a dag,

- β is a labeling function $V \to \{0,\dots,n-1,\neg 0,\dots,\neg n-1\} \cup \mathcal{B} \setminus \{\neg\}$,
- if $\beta(g) \in \{0,\dots,n-1,\neg 0,\dots,\neg n-1\}$ for some $g \in V$, then g has in-degree 0 in the graph (V,E), and
- if $\beta(g) \in \mathcal{B}$ for some $g \in V$ and m is the in-degree of g in (V,E), then $\beta(g)$ either is a Boolean function with arity m or a family of Boolean functions.

Let $n \in \mathbb{N}$, $\mathbf{C} = (V,E,\beta,\text{out})$ be a Boolean circuit in input normal form on n inputs, and $x_0,\dots,x_{n-1} \in \{0,1\}$. Let $g \in V$ and $g_1,\dots,g_m$ be the predecessors of g in (V,E). The *value of g in $\mathbf{C}$ on input $x = x_0 \dots x_{n-1}$*, denoted by $\text{val}(\mathbf{C},g,x)$, is defined as follows: If $\beta(g) = \neg i$ for $i \in \{0,\dots,n-1\}$, then $\text{val}(\mathbf{C},g,x) := \neg x_i$. Otherwise, $\text{val}(\mathbf{C},g,x)$ is defined in the same way as for general Boolean circuits. The *function computed by* $\mathbf{C}$ is again defined as $\mathbf{C} := \text{val}(\mathbf{C},\text{out},\cdot)$. *Families of Boolean circuits in input normal form* are families $C = (\mathbf{C}_n)_{n\in\mathbb{N}}$, where $\mathbf{C}_n$ is a Boolean circuit in input normal form on n inputs for all $n \in \mathbb{N}$. The function computed by C is the function $C(x) := \mathbf{C}_{|x|}(x)$.

To give an overview of the computational power of the different classes of circuits, we will now recall typical problems for each of them, if possible problems that are complete for the classes under suitable reducibilities. In AC^0 it is possible to compute addition and subtraction. Typical complete problems for TC^0 are the problems of computing polynomial sums and products as well as division. This also means that matrix multiplication and similar problems can be solved in TC^0. The breakthrough result that polynomial products and division can be computed in TC^0 even when using very restrictive forms of uniformity, in particular FO[BIT]-uniformity used throughout this thesis, was proven by Hesse in 2001 [Hes01]. As $\text{TC}^0 \subseteq \text{NC}^1$ and it is not known whether these classes coincide, all of the above problems are also in NC^1, but we do not know any problems in NC^1 that are not in TC^0. A complete problem for SAC^1 is the problem of evaluating a given acyclic Boolean conjunctive query in a given database, called ABCQ (for **a**cyclic **B**oolean **c**onjunctive **q**uery), as was shown by Gottlob et al. [GLS01]. To obtain a natural complete problem for AC^1 one can define a variant of the circuit value problem CVP that ensures that the circuit can be evaluated in logarithmic depth. This can be achieved by adding a vector containing the depth of each gate to the input and only allowing monotone, strictly alternating circuits, yielding the problem called AC1CVP, as was shown by Chandra and Tompa [CT90]. Mundhenk and Weiß showed that the model-checking problem for intuitionistic propositional logic with one variable is complete for AC^1 [MW14].

There was also some work on problems based on two-player games complete for SAC^1 and AC^1. Chandra and Tompa introduced a game called Shortcake and showed that determining the existence of winning strategies in this game is AC^1-complete. They also defined a variant that yields an SAC^1-complete problem. These problems nicely illustrate the power of the respective class. We will use a slightly modified variant of the game Shortcake here, using the same name. A similar proof to theirs shows completeness, but the problem is slightly better suited for our purpose.

The game Shortcake is defined as follows. Two players, H (or 0) and V (or 1) alternately move 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 as well as a bit specifying the player whose turn it is. To be more precise, the submatrix is specified by a tuple (i_0, i_1, j_0, j_1), where i_0 and i_1 are the indices of the first and last row of the submatrix, respectively, while j_0 and j_1 are the indices of the first and last column, respectively. 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

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

In his turn, V plays similarly but vertically on the rows. The first player with no move left loses. A straightforward definition of winning strategies (cf. our definition of winning strategies in the first-order model-checking game on page 45) in the above game then yields the AC^1-complete problem ShortCake defined as follows.

Problem:	ShortCake
Input:	Boolean matrix M
Question:	Does player H have a winning strategy in the Shortcake game on M?

In a similar vein, one can also define the game Semicake. Here, the players are called P and C and move alternatingly again, but the game is played on a matrix with values from $\{-1, 0, 1\}$. The main difference is that now P chooses the new position for the token in his move while C chooses which of the possible submatrices the game continues on. The player P wins if the token ever moves to a cell of the matrix containing the entry -1 and loses if there is no legal move left. This results in the SAC^1-complete problem SemiCake asking whether the player P has a winning strategy in the Semicake-game on a given $\{-1, 0, 1\}$-valued matrix.

While there are also probabilistic classes in the vein of PP and PL in circuit complexity, these classes are directly defined in terms of corresponding counting classes. For this reason, we will cover this topic in the following section, where counting classes are defined.

Similar as in classical complexity theory, for all of the classes defined above one can also consider functional versions, that is, classes of functions mapping from $\{0,1\}^*$ to $\mathbb{N}$ computable by the respective computational model. More precisely, in the context of circuit complexity this means that the output can be produced as a sequence of bits by a circuit family with multiple output gates within the required resource bounds. As the outputs are numbers encoded as binary strings, the order of the output gates also needs to be specified. This can be done by using a relation OUT between gates and natural

numbers—instead of having a single element as the output gate—with the intended meaning that $(g,i) \in \mathrm{OUT}$ if and only if g is the i-th output gate of the circuit. As in classical complexity theory, we denote function classes arising in this way by using the prefix F in the name of the class. For example, FAC^0 is the class of functions that can be computed by AC^0 circuit families with multiple output gates.

Counting in Boolean Circuits

As for Turing-machine based classes, counting analogues can also be defined for circuit complexity classes by introducing a process of witness-counting. Let $\mathbf{C}$ be a Boolean circuit and x an input that is accepted by $\mathbf{C}$. A witness for the fact that $\mathbf{C}$ accepts x is a minimal substructure of $\mathbf{C}$ that is sufficient to see that the output gate of $\mathbf{C}$ evaluates to 1 on input x, that is, a set of gates that have to be evaluated to verify this. There are two ways to precisely define witnesses for such circuits: Either a witness is a minimal subcircuit of $\mathbf{C}$ that is sufficient to see that the output gate evaluates to 1 or it is a minimal subtree of $\mathbf{C}$ unrolled into tree-shape with the same property. Here, we work with the latter notion. For a precise definition we now limit ourselves to circuits in input normal form over some basis $\mathcal{B} \subseteq \{\wedge, \vee, \wedge_k, \vee_k, \neg, f_0, f_1 \mid k \in \mathbb{N}\}$.

Let $\mathcal{B} \subseteq \{\wedge, \vee, \wedge_k, \vee_k, \neg, f_0, f_1 \mid k \in \mathbb{N}\}$ and $n \in \mathbb{N}$. Let $\mathbf{C} = (V, E, \beta, \mathrm{out})$ be a Boolean circuit in input normal form over $\mathcal{B}$ on n inputs, $x \in \{0,1\}^n$, and let d be the depth of $\mathbf{C}$. The underlying graph of $\mathbf{C}$ is (V,E) and the underlying graph of $\mathbf{C}$ unrolled into tree-shape is the graph (V_t, E_t) defined as follows:

$$V_t := \bigcup_{i=1}^{d} V^i \text{ and}$$

$$E_t := \bigcup_{i=1}^{d} \left\{ \Big((v_1, \ldots, v_i), (v_1, \ldots, v_i, v_{i+1}) \Big) \;\middle|\; (v_i, v_{i+1}) \in E \right\}.$$

Here, we view elements $v \in V$ as tuples (v) of arity 1. The idea is that, starting from the output gate, for each gate we create one copy for each path leading up to that gate. For simplicity we even create one copy for each potential path leading up to the gate. The output gate in that graph is still the gate out.

Definition 2.9. Let $\mathcal{B} \subseteq \{\wedge, \vee, \wedge_k, \vee_k, \neg, f_0, f_1 \mid k \in \mathbb{N}\}$, $n \in \mathbb{N}$, and $x \in \{0,1\}^n$. Let $\mathbf{C} = (V, E, \beta, \mathrm{out})$ be a Boolean circuit in input normal form over $\mathcal{B}$ on n inputs and let d be the depth of $\mathbf{C}$. Let (V_t, E_t) be the underlying graph of $\mathbf{C}$ unrolled into tree-shape and $\beta_t(v_1, \ldots, v_i) := \beta(v_i)$ for all $1 \leq i \leq d$. A *proof tree of* $\mathbf{C}$ *on input* x (or *proof tree of* $\mathbf{C}(x)$) is an inclusion-minimal induced subtree (V_p, E_p) of (V_t, E_t) with the following properties:

- $\mathrm{out} \in V_p$,
- for any $v \in V_p$ with $\beta_t(v) \in \{\wedge, \wedge_k \mid k \in \mathbb{N}\}$ and for all w with $(w,v) \in E_t$, we also have $w \in V_p$,
- for any $v \in V_p$ with $\beta_t(v) \in \{\vee, \vee_k \mid k \in \mathbb{N}\}$ there is a $w \in V_p$ with $(w,v) \in E_t$,

- for any $v \in V_p$ that has in-degree 0 in (V, E), we have $\mathrm{val}(\mathbf{C}, v, x) = 1$.

Here, (V_p, E_p) being an induced subtree of (V_t, E_t) means that $V_p \subseteq V_t$ and $E_p = E_t \cap V_p \times V_p$.

Let $\mathcal{B} \subseteq \{\wedge, \vee, \wedge_k, \vee_k, \neg, f_0, f_1 \mid k \in \mathbb{N}\}$ be a Boolean basis and $\mathbf{C} = (V, E, \beta, \mathrm{out})$ be a circuit over $\mathcal{B}$ on n inputs for some $n \in \mathbb{N}$. Similar to the definition of $\mathrm{val}(\mathbf{C}, g, x)$, we denote by $\mathrm{proof}(\mathbf{C}, g, x)$ the set of proof trees of (V, E, β, g) on input x for any $x \in \{0,1\}^n$. Furthermore, we write $\mathrm{proof}(\mathbf{C}, x)$ instead of $\mathrm{proof}(\mathbf{C}, \mathrm{out}, x)$. The counting function computed by $\mathbf{C}$ is the function $\#\mathbf{C} := |\mathrm{proof}(\mathbf{C}, \cdot)|$, that is, the function mapping any $x \in \{0,1\}^n$ to the number of proof trees of $\mathbf{C}$ on input x. Accordingly, the counting function computed by a circuit family $C = (\mathbf{C}_n)_{n \in \mathbb{N}}$ over $\mathcal{B}$ is the function $\#C \colon \{0,1\}^* \to \mathbb{N}$ with $\#C(x) := \#\mathbf{C}_{|x|}(x)$ for all $x \in \{0,1\}^*$.

Definition 2.10. Let $\mathcal{B} \subseteq \{\wedge, \vee, \wedge_k, \vee_k, \neg, f_0, f_1 \mid k \in \mathbb{N}\}$ and $s, d \colon \mathbb{N} \to \mathbb{N}$. We denote by $\#\mathrm{CIRCUIT}(\mathcal{B}, s, d)$ the class of functions $f \colon \mathbb{N} \to \mathbb{N}$ that arise as the function $\#C$ for a family C of Boolean circuits in input normal form over $\mathcal{B}$ of size $O(s)$ and depth $O(d)$.

In the same way as for the corresponding classes of Boolean circuits, we also use $\#\mathrm{CIRCUIT}(\mathcal{B}, S, D)$ to denote the class of functions that arise as functions $\#C$ for families C of Boolean circuits over $\mathcal{B}$ of size $s \in O(S)$ and depth $d \in O(D)$ in case of sets of functions S, D.

Since the bases used in the definitions of NC^i, SAC^i and AC^i are of the required form, we can define counting versions of these classes as follows.

Definition 2.11. For $i \in \mathbb{N}$ we define

- $\#\mathrm{NC}^i := \#\mathrm{CIRCUIT}(\mathcal{B}_0 \cup \{f_0, f_1\}, n^{O(1)}, (\log n)^i)$,
- $\#\mathrm{SAC}^i := \#\mathrm{CIRCUIT}(\mathcal{B}_2 \cup \{f_0, f_1\}, n^{O(1)}, (\log n)^i)$ and
- $\#\mathrm{AC}^i := \#\mathrm{CIRCUIT}(\mathcal{B}_1 \cup \{f_0, f_1\}, n^{O(1)}, (\log n)^i)$.

As for the decision versions of these classes, we could omit f_0 and f_1 from the definition of $\#\mathrm{AC}^i$ and f_1 from the definition of $\#\mathrm{SAC}^i$, cf. Remark 2.6.

Similar to the decision case, the most studied classes among these are $\#\mathrm{AC}^0$, $\#\mathrm{NC}^1$, $\#\mathrm{SAC}^1$ and $\#\mathrm{AC}^1$. The following inclusions between these classes are obvious from the definitions:

$$\#\mathrm{AC}^0 \subseteq \#\mathrm{NC}^1 \subseteq \#\mathrm{SAC}^1 \subseteq \#\mathrm{AC}^1.$$

As for classical circuit complexity classes, we want to cover the topic of what is known with regards to connections of these classes to other counting complexity classes. For this, we again assume suitable notions of uniformity (cf. Subsection 2.1.2 and Section 2.4).

Note that, already at the first level, the classes $\#\mathrm{AC}^1$, $\#\mathrm{NC}^1$, $\#\mathrm{SAC}^1$, though based on relatively similar circuit classes, have a quite different computational power. From known results, it can be seen that $\#\mathrm{SAC}^1 \subseteq \#\mathrm{P}$: The class $\#\mathrm{SAC}^1$ can be characterized in terms of multiplicatively disjoint circuits and for such circuits, the degree of the resulting polynomial is always bounded in the size of the circuit, as shown by Malod and

Portier [MP08]. On the other hand, the class #P can be characterized in terms of arithmetic circuits with polynomial degree, which was shown by Venkateswaran [Ven92]. In contrast to functions in $\#\mathrm{SAC}^1$, functions in $\#\mathrm{AC}^1$ can output numbers bigger than $2^{n^{\log n}}$ for inputs of length n, that is, numbers whose encodings have lengths super-polynomial in the input length. The reason is that unfolding a polynomial size, logarithmic depth circuit into tree-shape may result in a graph of size $n^{O(\log n)}$. As #P-functions are bounded by $2^{n^{O(1)}}$, this means that $\#\mathrm{AC}^1 \not\subseteq \#\mathrm{P}$.

Actually, the classes $\#\mathrm{AC}^1$ and #P are incomparable if $\mathrm{P} \neq \mathrm{NP}$. For the remaining separation $\#\mathrm{P} \not\subseteq \#\mathrm{AC}^1$, we first define the operator $\exists$ on function classes. For any class $\mathfrak{F}$ of counting functions, let $\exists \cdot \mathfrak{F}$ be the class of languages L for which there is an $f \in \mathfrak{F}$ such that for all $x \in \{0,1\}^*$, $x \in L \iff f(x) > 0$. Assuming $\#\mathrm{P} \subseteq \#\mathrm{AC}^1$ we get $\mathrm{NP} = \exists \cdot \#\mathrm{P} \subseteq \exists \cdot \#\mathrm{AC}^1 = \mathrm{AC}^1 \subseteq \mathrm{P}$.

A similar argument as that for $\#\mathrm{AC}^1 \not\subseteq \#\mathrm{P}$ shows that, in contrast to the situation in the setting of decision classes, $\#\mathrm{SAC}^i$ cannot be equivalently defined using the base $\{\vee_2, \wedge, \neg\}$: The class $\#\mathrm{CIRCUIT}(\{\vee_2, \wedge, \neg\} \cup \{f_0, f_1\}, n^{O(1)}, (\log n)^i)$ contains the function $2^{(n^{(\log n)^i})}$ (by the same argument showing that $\#\mathrm{AC}^1$ can compute the function $2^{n^{\log n}}$), while this function is not contained in $\#\mathrm{SAC}^i$. This also shows the separation $\#\mathrm{SAC}^1 \neq \#\mathrm{AC}^1$. On the other hand, analogously to the situation in the classical setting, $\#\mathrm{SAC}^1$ can be characterized in terms of counting in nondeterministic auxiliary pushdown automata [Vin91].

We now want to give a bit more context for the power of these classes by covering relevant relations to other complexity classes. Agrawal et al. showed that $\mathrm{FAC}^0 \subseteq \#\mathrm{AC}^0$, so FAC^0 is a lower bound for the counting classes we consider [AAD00]. We also know that multiplication cannot be in FAC^0, as otherwise the completeness of the corresponding decision problem for TC^0 would imply $\mathrm{AC}^0 = \mathrm{TC}^0$. Consequently, we have $\mathrm{FAC}^0 \subsetneq \#\mathrm{AC}^0$.

As polynomial sums and products can be computed in FTC^0, we have $\#\mathrm{AC}^0 \subseteq \mathrm{FTC}^0$. Even more, this inclusion is strict by a similar argument as $\mathrm{FAC}^0 \neq \#\mathrm{AC}^0$: As all $\{0,1\}$-valued functions in $\#\mathrm{AC}^0$ are also in AC^0, $\#\mathrm{AC}^0 = \mathrm{FTC}^0$ would imply $\mathrm{AC}^0 = \mathrm{TC}^0$. On the other end of the spectrum, it is easy to see that $\mathrm{FTC}^0 \subsetneq \#\mathrm{P}$: As FTC^0-circuits can be evaluated in polynomial time, we have $\mathrm{FTC}^0 \subseteq \mathrm{FP} \subseteq \#\mathrm{P}$. For the separation, assume $\mathrm{FTC}^0 = \#\mathrm{P}$ for the sake of contradiction. For any class $\mathfrak{F}$ of counting functions, let $\mathsf{C} \cdot \mathfrak{F}$ be the class of all languages L for which there are $f, g \in \mathfrak{F}$ such that for all $x \in \{0,1\}^*$, $x \in L \Leftrightarrow f(x) > g(x)$. We obtain $\mathrm{PP} = \mathsf{C} \cdot \#\mathrm{P} \subseteq \mathsf{C} \cdot \mathrm{FTC}^0 = \mathrm{TC}^0$, which is a contradiction to $\mathrm{TC}^0 \subsetneq \mathrm{PP}$. Here, $\mathsf{C} \cdot \mathrm{FTC}^0 = \mathrm{TC}^0$ is obvious from the definition: Computing two FTC^0-functions and comparing them can be done in TC^0, while for the converse we can just use the characteristic function of any language in TC^0 and the constant-0 function to show that the corresponding language is also in $\mathsf{C} \cdot \mathrm{FTC}^0$. Regarding the relationship of FTC^0 to the other circuit-based counting classes, we have $\mathrm{FTC}^0 \subseteq \mathrm{FNC}^1 \subseteq \#\mathrm{NC}^1$, where the latter inclusion was shown by Caussinus et al. [CMTV98].

For later reference, we compactly restate the above relations between the central classes of this thesis as well as relevant relations to other classes in the following proposition. These relations are also illustrated in the inclusion diagram shown in

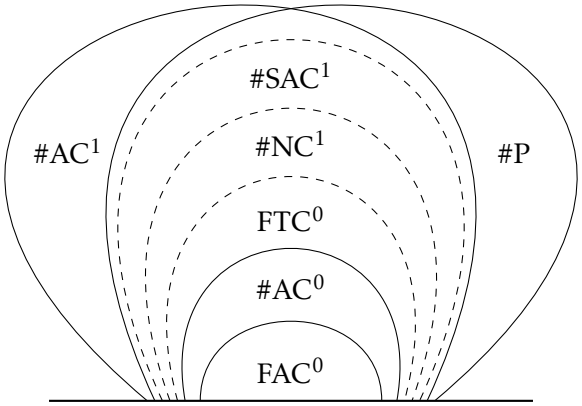

Figure 2.1: Relationship between central counting classes in this thesis and relevant relationships to other classes, assuming P ≠ NP. Dashed lines indicate that separations are not known.

Figure 2.1.

Proposition 2.12. *Using suitable uniformity conditions and assuming $P \neq NP$, we have:*

$$\mathrm{FAC}^0 \subsetneq \#\mathrm{AC}^0 \subsetneq \mathrm{FTC}^0 \subseteq \#\mathrm{NC}^1 \subseteq \#\mathrm{SAC}^1 \begin{array}{c} \subsetneq \\ \subseteq \\ \subseteq \end{array} \begin{array}{l} \#\mathrm{AC}^1 \\ \\ \#\mathrm{P}. \end{array}$$

Also, $\mathrm{FTC}^0 \neq \#\mathrm{P}$. *Here,* $\mathrm{P} \neq \mathrm{NP}$ *is only required for* $\#\mathrm{P} \not\subseteq \#\mathrm{AC}^1$.

There are also characterizations of $\#\mathrm{NC}^1$ and $\#\mathrm{SAC}^1$ in terms of counting accepting paths in different computation models. More precisely, Caussinus et al. showed that $\#\mathrm{NC}^1$ can be characterized in terms of counting paths in bounded width branching programs [CMTV98] and Vinay showed that $\#\mathrm{SAC}^1$ can be characterized in terms of counting accepting paths of nondeterministic auxiliary pushdown automata [Vin91].

As mentioned in the previous section, classes in the vein of PP were also introduced from these counting classes.

Definition 2.13. Let $\#\mathfrak{C}$ be a counting complexity class for a circuit complexity class $\mathfrak{C}$. Then define $\mathrm{P}\mathfrak{C} := \{L \mid \exists f, g \in \#\mathfrak{C} \text{ such that } x \in L \iff f(x) - g(x) > 0 \text{ for all } x\}$.

In this vein, the classes PAC^0 and PNC^1 were studied [AAD00, ABL98, CMTV98]. Note that the above definition is related to so-called Gap- and Diff-classes (cf. GapL, the closure of #L under subtraction) and to the fact that for $\#\mathrm{AC}^0$ and $\#\mathrm{NC}^1$, these classes coincide [ABL98, CMTV98]. Interestingly, Agrawal et al. [AAD00] further showed that the class PAC^0 coincides with TC^0 in the non-uniform setting and Ambainis et al. [ABL98] showed that $\mathrm{TC}^0 \subseteq \mathrm{PAC}^0$ also holds in the $\mathrm{U_D}$-uniform setting. While they were not able

to show the converse, it follows immediately from division (and hence polynomial products) being computable in TC^0. Consequently, $\mathrm{TC}^0 = \mathrm{PAC}^0$ also holds in the $\mathrm{U_D}$-uniform setting.

Regarding complete problems, much less is known for the above counting complexity classes than for their decision counterparts. Certain counting problems on nondeterministic auxiliary pushdown automata are complete for $\#\mathrm{SAC}^1$ [Vin91]. Evaluating arithmetic formulae over the natural numbers and over the integers is complete for $\#\mathrm{NC}^1$ and GapNC^1, respectively [BCGR92]. The latter is the closure of $\#\mathrm{NC}^1$ under subtraction and is one of the Gap-classes mentioned above [CMTV98]. Computing specific entries of the product of a sequence of matrices of constant dimension with integer entries as well as counting paths in bounded-width graphs are also complete for this class. Furthermore, counting accepting paths in so-called visible pushdown automata is complete for $\#\mathrm{NC}^1$ [KLM12]. With regard to constant-depth circuits, there is a problem based on counting paths in a specific family of graphs that is complete for $\#\mathrm{AC}^0$ [AAB+99].

Counting Arithmetic Circuits

We now want to give a different view on the counting classes covered above by pointing out a connection to arithmetic circuits. For many tasks regarding structural analysis, the definitions of counting classes in terms of arithmetic circuits are better suited. It is folklore, though, that these classes can be characterized in terms of a variant of arithmetic circuits, called counting arithmetic circuits. Arithmetic circuits are a generalization of Boolean circuits, where instead of Boolean values (elements of the Boolean semi-ring), elements of an arbitrary semi-ring can occur. Gates then compute functions over that semi-ring.

Counting arithmetic circuits are for the most part just a specific kind of arithmetic circuit: The semi-ring is fixed to be the natural numbers and the basis is a finite subset of $\{+, \times, +_k, \times_k, f_0, f_1 \mid k \in \mathbb{N}\}$, where $+$ and $\times$ are unbounded fan-in addition and multiplication, respectively, and $+_k$ and $\times_k$ are addition and multiplication of fan-in k, respectively. They differ from usual arithmetic circuits in that inputs are not arbitrary elements of the chosen semi-ring, but rather only 0 or 1, i.e., Boolean inputs. Also, input gates can be negated in analogy to Boolean circuits in input normal form. We will now give a formal definition.

Definition 2.14. Let $\mathcal{B} \subseteq \{+, \times, +, +_k, \times_k, f_0, f_1 \mid k \in \mathbb{N}\}$ and $n \in \mathbb{N}$. A *counting arithmetic circuit over $\mathcal{B}$ on n inputs* is a tuple $(V, E, \beta, \mathrm{out})$ with the following properties:

- (V, E) is a dag,
- β is a function $V \to \{0, \dots, n-1, \neg 0, \dots, \neg n-1\} \cup \mathcal{B}$,
- if $\beta(g) \in \{0, \dots, n-1, \neg 0, \dots, \neg n-1, f_0, f_1\}$ for some $g \in V$, then g has in-degree 0 in the graph (V, E), and
- if $\beta(g) \in \mathcal{B}$ for some $g \in V$ and m is the in-degree of g in (V, E), then $\beta(g) \in \{+, \times, +_m, \times_m\}$.

Families of counting arithmetic circuits as well as *size* and *depth* of counting arithmetic circuits and families of such circuits are defined in analogy to the respective notions for Boolean circuits. Let $\mathcal{B} \subseteq \{+, \times, +_k, \times_k, f_0, f_1 \mid k \in \mathbb{N}\}$. The value of a gate g in some counting arithmetic circuit $\mathbf{C}$ over $\mathcal{B}$ on an input x of adequate length is denoted by $\mathrm{val}(\mathbf{C}, g, x)$ in analogy to the case of Boolean circuits. The function computed by a counting arithmetic circuit $\mathbf{C}$ over $\mathcal{B}$, denoted by $\mathbf{C}$, and the function computed by a family C of counting arithmetic circuits over $\mathcal{B}$, denoted by C, are defined accordingly.

There is a close correspondence between Boolean circuits in input normal form and counting arithmetic circuits. More precisely, for any Boolean circuit $\mathbf{C}$ in input normal form that only uses conjunction, disjunction and negation, the corresponding counting arithmetic circuit obtained by replacing conjunction by multiplication and disjunction by addition computes exactly the number of proof trees of $\mathbf{C}$. Venkateswaran proved this fact in high generality [Ven92], and we state it in the following proposition.

Proposition 2.15. *Let $\mathcal{B} \subseteq \{\wedge, \vee, \wedge_k, \vee_k, \neg, f_0, f_1 \mid k \in \mathbb{N}\}$ and $n \in \mathbb{N}$. Let $\mathbf{C} = (V, E, \beta, \mathrm{out})$ be a Boolean circuit in input normal form over $\mathcal{B}$ on n inputs. Then the counting arithmetic circuit $\mathbf{C}' = (V, E, \beta', \mathrm{out})$, where β' is obtained from β by replacing $\vee$ by $+$ and $\wedge$ by $\times$, computes the function $\#\mathbf{C}$. Formally, β' is defined as follows:*

$$\beta'(g) := \begin{cases} \beta(g), & \text{if } \beta(g) \in \{0, \dots, n-1, \neg 0, \dots, \neg n-1, f_0, f_1\} \\ +, & \text{if } \beta(g) = \vee \\ \times, & \text{if } \beta(g) = \wedge \\ +_m, & \text{if } \beta(g) = \vee_m \\ \times_m, & \text{if } \beta(g) = \wedge_m \end{cases} .$$

As the circuit $\mathbf{C}'$ defined in the statement of this proposition has the same size, depth and fan-in as the circuit $\mathbf{C}$ and any counting arithmetic circuit $\mathbf{D}$ is equal to $\mathbf{C}'$ for some Boolean circuit $\mathbf{C}$, this means that we can equivalently define the classes $\#\mathrm{NC}^i$, $\#\mathrm{SAC}^i$, and $\#\mathrm{AC}^i$ for $i \in \mathbb{N}$ in terms of counting arithmetic circuits instead of counting in Boolean circuits. We will use both definitions depending on the context. To illustrate the definition of counting arithmetic circuits, Figure 2.2 sketches an arithmetic circuit family of polynomial size and constant depth computing the function $f(w) = |w|^{\lceil |w|/2 \rceil}$. In other words, this circuit family shows that $f \in \#\mathrm{AC}^0$.

Uniformity

When using circuits as a computational model, circuit families are used to be able to process inputs of arbitrary length. As defined before, a circuit family contains a circuit for each input length $n \in \mathbb{N}$. Since there are no limitations to how these circuits are constructed, this model has unlimited power regarding properties that only depend on the input length. This means that any unary language—in particular any undecidable unary language—is trivially contained in all of the defined circuit complexity classes.

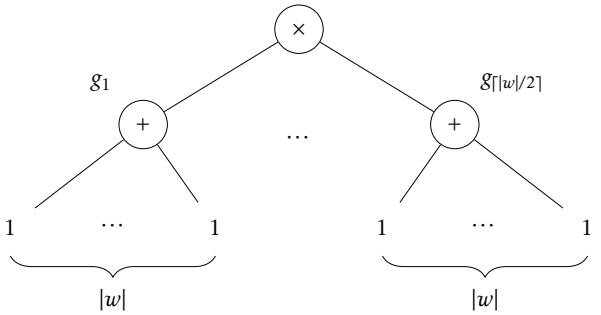

Figure 2.2: Counting arithmetic circuit for the function $f(w) = |w|^{\lceil |w|/2 \rceil}$.

Proposition 2.16. *Let L be a unary language over $\{0,1\}$, that is, there is a set $M \subseteq \mathbb{N}$ such that*

$$L = \{ w \mid |w| \in M \}.$$

L can be accepted by a family $C = (\mathbf{C}_n)_{n\in\mathbb{N}}$ of Boolean circuits, where $\mathbf{C}_n$ is of size 1 and depth 0 for all $n \in \mathbb{N}$.

Proof. For all $n \in \mathbb{N}$, let $\mathbf{C}_n := (\{v\}, \emptyset, \beta, v)$ with $\beta(v) := f_{c_L(1^n)}$, i.e., v is a gate for the constant $c_L(1^n)$. Recall that c_L is the characteristic function of L. By construction $C := (\mathbf{C}_n)_{n\in\mathbb{N}}$ has the desired properties. □

When circuits are used as a model of parallel computation and generally in a complexity theoretic sense, it is desirable to obtain classes of decidable problems. Also, for many classes in circuit complexity, in particular the classes we consider in this thesis, the computational model itself is relatively weak. Hence, to use these classes in a meaningful way in complexity theory—e.g., obtaining connections to other complexity classes—artificially adding power with respect to the length of the input should be avoided.

These issues justify to introduce a form of uniformity to circuit families. A uniform circuit family $(\mathbf{C}_n)_{n\in\mathbb{N}}$ is one where the circuits arise from the respective input length in a uniform manner. This is usually formalized by imposing that the circuit $\mathbf{C}_n$ should be computable from n within reasonable resource bounds. Simple forms of uniformity are P-uniformity and L-uniformity, that is, $\mathbf{C}_n$ can be computed from n in polynomial time or logarithmic space, respectively.

Many of the usual classes from circuit complexity are weak even compared to P or L. For this reason, even weaker forms of uniformity are of interest from a theoretical standpoint. As mentioned above, the goal of this approach is to find a type of uniformity that does not add any power to the model. In this regard, we want to introduce $\mathrm{U_D}$-uniformity, which is based on the class DLOGTIME. Since this uniformity only allows for deterministic computations in logarithmic time, it is standard to not impose that the circuits are computable within these resource bounds. Instead, it should be

possible to query all information needed to specify the circuit within that bound, which is why it can be considered a query-based uniformity condition. Note that L-uniformity mentioned above, where the whole circuits have to be computable in logarithmic space, for polynomial-size circuit families coincides with a suitably defined query-based uniformity where logarithmic-space computations are allowed. To make the notion of $\mathrm{U_D}$-uniformity precise, we first need to make the encoding of circuits precise. Here, we describe the general version including the function α providing an ordering of the edges. When this order is not required the conditions referring to α can simply be omitted. Note that in this context it is standard to assume that circuits have exactly one input gate for each input bit. For the usual classes this is not a restriction, as one can simply use additional conjunction- or disjunction-gates that copy the value of certain input gates.

Definition 2.17. Let $C = (\mathbf{C}_n)_{n\in\mathbb{N}}$ be a family of Boolean circuits over $\mathcal{B}$ of size s. An admissible encoding scheme of C is obtained as follows. Fix a numbering of elements of $\mathcal{B}$. Also, for each n fix a numbering of all gates of $\mathbf{C}_n$ with the following properties:

- Input gates in $\mathbf{C}_n$ are numbered by $0,\dots,n-1$.
- The output gate has number n.
- There is a polynomial p such that the highest number of any gate in $\mathbf{C}_n$ is bounded by $p(s(n))$.

The encoding scheme now describes how individual gates in any circuit $\mathbf{C}_n$ are encoded. Let $\mathbf{C}_n =: (V, E, \alpha, \beta, \mathrm{out})$. Let v be a gate in $\mathbf{C}_n$ with predecessors $v_1,\dots,v_k$ such that $\alpha(v_1,v) < \alpha(v_2,v) < \cdots < \alpha(v_k,v)$. Then v is encoded by $\overline{v} := \langle g, b, g_1,\dots,g_k\rangle$, where g is the number of v, b is the number of $\beta(v)$ and $g_1,\dots,g_k$ are the numbers of $v_1,\dots,v_k$. The circuit $\mathbf{C}_n$ is then encoded by listing the encodings of all non-input gates of $\mathbf{C}_n$ in an arbitrary order: $\langle\overline{v_1},\dots,\overline{v_k}\rangle$, where $V =: \{v_1,\dots,v_k\} \sqcup \{\, v \in V \mid v \text{ is an input gate}\,\}$.

Next, we will define the direct connection language of a circuit family, which describes a family of Boolean circuits by allowing for queries to its relations.

Definition 2.18. Let $C = (\mathbf{C}_n)_{n\in\mathbb{N}}$ be a family of Boolean circuits over $\mathcal{B}$. Fix an admissible encoding scheme of C. The direct connection language (dcl) of C with respect to that encoding scheme is the set $L_{\mathrm{DC}}(C)$ containing for each $n \in \mathbb{N}$ all tuples $\langle y, g, p, b\rangle$ with the following properties:

- $|y| = n$,
- g is the number of a gate v in $\mathbf{C}_n$,
- $p \in \{0,1\}^*$,
- if $p = \varepsilon$: b is the number of $\beta(v)$, and
- if $p = \mathrm{bin}(k)$ for some $k > 0$: b is the number of the k-th predecessor of v respecting the order given by α.

Now a circuit family C is called $\mathrm{U_D}$-*uniform*, if there is an admissible encoding scheme such that the direct connection language of C with respect to that encoding scheme is in DLOGTIME. This notion of uniformity is equivalent to so-called first-order uniformity for many circuit complexity classes, which will be used throughout the paper. As certain concepts from first-order logic are required to introduce first-order uniformity, it will be introduced later (see Definition 2.29). We will defer a discussion of the suitability of different forms of uniformity for the different circuit complexity classes as well as of the choice of the notion of uniformity for this thesis to Section 2.4, since first-order uniformity and some related topics regarding first-order logic are required to adequately cover this topic.

2.2 First-Order Logic and Extensions

In this section we recall important definitions from first-order logic (FO) and some related notions. For a thorough introduction to the topic, we refer the reader to the textbook by Ebbinghaus et al. [EFT94].

2.2.1 Basics of First-order Logic

An *FO-vocabulary* (or *vocabulary*) is a tuple $(\mathsf{R}_1,\dots,\mathsf{R}_k;\mathsf{F}_1,\dots,\mathsf{F}_\ell;\mathsf{c}_1,\dots,\mathsf{c}_m)$ containing *relation symbols* $\mathsf{R}_1,\dots,\mathsf{R}_k$, *function symbols* $\mathsf{F}_1,\dots,\mathsf{F}_\ell$ and *constant symbols* $\mathsf{c}_1,\dots,\mathsf{c}_m$. Each relation symbol $\mathsf{R}\in\{\mathsf{R}_1,\dots,\mathsf{R}_k\}$ and each function symbol $\mathsf{F}\in\{\mathsf{F}_1,\dots,\mathsf{F}_k\}$ have an associated arity, which is a natural number, given by $\mathrm{ar}(\mathsf{R})$ and $\mathrm{ar}(\mathsf{F})$, respectively. A vocabulary is *relational* if it contains no function symbols and no constant symbols. For such vocabularies we omit the semicolons, that is, $(\mathsf{R}_1,\dots,\mathsf{R}_k)$ is a vocabulary containing relation symbols $\mathsf{R}_1,\dots,\mathsf{R}_k$. When defining vocabularies, we often use superscripts to specify arities of relations and functions. For example, $(\mathsf{R}_1^2,\mathsf{R}_2^3)$ refers to a vocabulary containing relation symbols R_1 and R_2 with $\mathrm{ar}(\mathsf{R}_1)=2$ and $\mathrm{ar}(\mathsf{R}_2)=3$. We use $\cup$ to denote the union of two disjoint vocabularies. More precisely, for two vocabularies $\sigma=(\mathsf{R}_1,\dots,\mathsf{R}_{k_1};\mathsf{F}_1,\dots,\mathsf{F}_{\ell_1};\mathsf{c}_1,\dots,\mathsf{c}_{m_1})$, $\tau=(\mathsf{R}_{k_1+1},\dots,\mathsf{R}_{k_1+k_2};\mathsf{F}_{\ell_1+1},\dots,\mathsf{F}_{\ell_1+\ell_2};\mathsf{c}_{m_1+1},\dots,\mathsf{c}_{m_1+m_2})$ we define

$$\sigma\cup\tau := (\mathsf{R}_1,\dots,\mathsf{R}_{k_1+k_2};\mathsf{F}_1,\dots,\mathsf{F}_{\ell_1+\ell_2};\mathsf{c}_1,\dots,\mathsf{c}_{m_1+m_2}).$$

Let $\sigma=(\mathsf{R}_1,\dots,\mathsf{R}_k;\mathsf{F}_1,\dots,\mathsf{F}_\ell;\mathsf{c}_1,\dots,\mathsf{c}_m)$ be a vocabulary. Formally, we use the countable set $\mathrm{Var_{FO}} := \{x_i \mid i\in\mathbb{N}\}$ as the set of first-order variables. In practice, we will also use different names for variables for readability. The set of *terms over* σ (or *σ-terms*) is described by the following grammar:

$$t ::= x \mid \mathsf{c} \mid \mathsf{F}(\underbrace{t,\dots,t}_{\mathrm{ar}(\mathsf{F})\text{-many}}),$$

where $x\in\mathrm{Var_{FO}}$, $\mathsf{c}\in\{\mathsf{c}_i \mid 1\le i\le m\}$ and $\mathsf{F}\in\{\mathsf{F}_i \mid 1\le i\le \ell\}$. Terms that are variables or constant symbols are also called *atomic terms*. Now, the set of *FO-formulae over* σ (or *σ-formulae*) is described by the following grammar:

$$\varphi ::= 0 \mid 1 \mid t_1 = t_2 \mid \mathsf{R}(t_1,\dots,t_{\mathrm{ar}(\mathsf{R})}) \mid \varphi\wedge\varphi \mid \varphi\vee\varphi \mid \neg\varphi \mid \exists x\varphi \mid \forall x\varphi,$$

where $\mathsf{R} \in \{\mathsf{R}_1, \ldots, \mathsf{R}_k\}$, and $t_1, \ldots, t_{\max\{2,\mathrm{ar}(\mathsf{R}_1),\ldots,\mathrm{ar}(\mathsf{R}_k)\}}$ are σ-terms and $x \in \mathrm{Var}_{\mathrm{FO}}$. Formulae of the form $t_1 = t_2$ or $\mathsf{R}(t_1, \ldots, t_{\mathrm{ar}(\mathsf{R})})$ are called *atomic formulae*. We will use $t \neq t'$ instead of $\neg t = t'$ as well as the shorthands $\rightarrow$ and $\leftrightarrow$ for implication and biimplication, respectively. More precisely,

$$\varphi \rightarrow \psi \coloneqq \neg\varphi \vee \psi \quad \text{and} \quad \varphi \leftrightarrow \psi \coloneqq (\varphi \wedge \psi) \vee (\neg\varphi \wedge \neg\psi).$$

Let φ be an FO-formula. If $\exists$ and $\forall$ do not occur in φ, we call the formula *quantifier-free*. We denote by $\mathrm{SF}(\varphi)$ the set of subformulae of φ. Further, the set of variables occurring in φ is denoted by $\mathrm{Vars}(\varphi)$. An occurrence of a variable x in an FO-formula φ is called bound, if it appears in the scope of a quantifier $\exists x$ or $\forall x$. Otherwise it is called free. Similarly, a variable is called free in φ, if there is a free occurrence of the variable in φ and it is called bound if there is a bound occurrence of it in φ. For an FO-formula φ the set of all free variables in φ is denoted by $\mathrm{free}(\varphi)$ while the set of all bound variables in φ is denoted by $\mathrm{bound}(\varphi)$. Note that $\mathrm{bound}(\varphi) \cap \mathrm{free}(\varphi)$ can be non-empty. A formula without free variables is called a *sentence*. A σ-sentence is a σ-formula that is a sentence.

Models in first-order logic are first-order structures, which will be defined next. For σ as above, an *FO-structure* $\mathbf{A}$ over σ (or *σ-structure*) is a tuple

$$(A; \mathsf{R}_1^{\mathbf{A}}, \ldots, \mathsf{R}_k^{\mathbf{A}}; \mathsf{F}_1^{\mathbf{A}}, \ldots, \mathsf{F}_\ell^{\mathbf{A}}; \mathsf{c}_1^{\mathbf{A}}, \ldots, \mathsf{c}_m^{\mathbf{A}}),$$

where A is a non-empty set and $\mathsf{R}_i^{\mathbf{A}}$, $\mathsf{F}_i^{\mathbf{A}}$ and $\mathsf{c}_i^{\mathbf{A}}$ are interpretations of the respective symbols from σ over that set:

$\mathsf{R}_i^{\mathbf{A}} \subseteq A^{\mathrm{ar}(\mathsf{R}_i)}$	is a relation of arity $\mathrm{ar}(\mathsf{R}_i)$ for $1 \leq i \leq k$,
$\mathsf{F}_i^{\mathbf{A}} \colon A^{\mathrm{ar}(\mathsf{F}_i)} \rightarrow A$	is a function of arity $\mathrm{ar}(\mathsf{F}_i)$ for $1 \leq i \leq \ell$, and
$\mathsf{c}_i^{\mathbf{A}} \in A$	is a constant for $1 \leq i \leq m$.

The set A is called the *universe* (or *domain*) of $\mathbf{A}$, denoted by $\mathrm{dom}(\mathbf{A})$. We generally denote the interpretation of a relation symbol R in the structure $\mathbf{A}$ by $\mathsf{R}^{\mathbf{A}}$ and similarly for functions and constants even when not explicitly stated. In this thesis we will only consider finite FO-structures, that is, FO-structures with a finite universe. The set of all finite σ-structures is denoted by $\mathrm{STRUC}[\sigma]$. Similar to the notation used for vocabularies, we use $\cup$ to denote the operation of adding additional interpretations of relation-, function- and constant-symbols to a structure. For a formal definition, let

$$\sigma = (\mathsf{R}_1, \ldots, \mathsf{R}_{k_1}; \mathsf{F}_1, \ldots, \mathsf{F}_{\ell_1}; \mathsf{c}_1, \ldots, \mathsf{c}_{m_1}) \text{ and}$$
$$\tau = (\mathsf{R}_{k_1+1}, \ldots, \mathsf{R}_{k_1+k_2}; \mathsf{F}_{\ell_1+1}, \ldots, \mathsf{F}_{\ell_1+\ell_2}; \mathsf{c}_{m_1+1}, \ldots, \mathsf{c}_{m_1+m_2})$$

be vocabularies, let $\mathbf{A}$ be a σ-structure, and let

$$\mathbf{T} = (R_{k_1+1}, \ldots, R_{k_1+k_2}; F_{\ell_1+1}, \ldots, F_{\ell_1+\ell_2}; c_{m_1+1}, \ldots, c_{m_1+m_2})$$

be a tuple of interpretations of the symbols in τ as relations over $\mathrm{dom}(\mathbf{A})$. We define

$$\begin{aligned}\mathbf{A} \cup \mathbf{T} \coloneqq (\mathrm{dom}(\mathbf{A}); &\mathsf{R}_1^{\mathbf{A}}, \ldots, \mathsf{R}_{k_1}^{\mathbf{A}}, R_{k_1+1}, \ldots, R_{k_1+k_2};\\ &\mathsf{F}_1^{\mathbf{A}}, \ldots, \mathsf{F}_{\ell_1}^{\mathbf{A}}, F_{\ell_1+1}, \ldots, F_{\ell_1+\ell_2};\\ &\mathsf{c}_1^{\mathbf{A}}, \ldots, \mathsf{c}_{m_1}^{\mathbf{A}}, c_{m_1+1}, \ldots, c_{m_1+m_2}).\end{aligned}$$

We also need a notion of assignments to variables, which is especially important in the context of formulae with free variables. An FO-assignment s in a structure $\mathbf{A}$ is a partial function $s\colon \mathrm{Var}_{\mathrm{FO}} \to \mathrm{dom}(\mathbf{A})$. For such an assignment s, we denote by $s[x/a]$ for $x \in \mathrm{Var}_{\mathrm{FO}}$ and $a \in \mathrm{dom}(\mathbf{A})$ the substitution of a for x in s, i.e., the assignment with $s[x/a](x) = a$ and $s[x/a](y) = s(y)$ for $y \neq x$.

Semantics of FO is defined inductively over the compositional structure of the formula. First, we inductively define the value of an arbitrary term t in a structure $\mathbf{A}$ under an FO-assignment s assigning values to all variables occurring in t:

$$\begin{aligned}
x^{(\mathbf{A},s)} &:= s(x), && \text{if } x \in \mathrm{Var}_{\mathrm{FO}},\\
\mathsf{c}^{(\mathbf{A},s)} &:= \mathsf{c}^{\mathbf{A}}, && \text{if } \mathsf{c} \in \{\mathsf{c}_1,\dots,\mathsf{c}_m\}, \text{ and}\\
\mathsf{F}(t_1,\dots,t_r)^{(\mathbf{A},s)} &:= \mathsf{F}^{\mathbf{A}}(t_1^{(\mathbf{A},s)},\dots,t_r^{(\mathbf{A},s)}), && \text{if } \mathsf{F} \in \{\mathsf{F}_1,\dots,\mathsf{F}_\ell\} \text{ and } t_1,\dots,t_r \text{ are terms.}
\end{aligned}$$

The model relation $\models$ assigns a truth value to a formula φ in a given structure $\mathbf{A}$ under a given FO-assignment s assigning a value to all free variables. This relation is defined inductively as follows:

$$\begin{aligned}
&\mathbf{A}, s \not\models 0 && \text{for all } (\mathbf{A}, s),\\
&\mathbf{A}, s \models 1 && \text{for all } (\mathbf{A}, s),\\
&\mathbf{A}, s \models t_1 = t_2 && \text{if } t_1^{(\mathbf{A},s)} = t_2^{(\mathbf{A},s)},\\
&\mathbf{A}, s \models \mathsf{R}(t_1,\dots,t_{\mathrm{ar}(\mathsf{R})}) && \text{if } (t_1^{(\mathbf{A},s)},\dots,t_{\mathrm{ar}(\mathsf{R})}^{(\mathbf{A},s)}) \in \mathsf{R}^{\mathbf{A}},\\
&\mathbf{A}, s \models \varphi \wedge \psi, && \text{if } \mathbf{A}, s \models \varphi \text{ and } \mathbf{A}, s \models \psi,\\
&\mathbf{A}, s \models \varphi \vee \psi, && \text{if } \mathbf{A}, s \models \varphi \text{ or } \mathbf{A}, s \models \psi,\\
&\mathbf{A}, s \models \neg\varphi, && \text{if } \mathbf{A}, s \not\models \varphi,\\
&\mathbf{A}, s \models \exists x\varphi, && \text{if there is an } a \in \mathrm{dom}(\mathbf{A}) \text{ with } \mathbf{A}, s[x/a] \models \varphi, \text{ and}\\
&\mathbf{A}, s \models \forall x\varphi, && \text{if for all } a \in \mathrm{dom}(\mathbf{A}) \text{ it holds that } \mathbf{A}, s[x/a] \models \varphi.
\end{aligned}$$

When $\varphi(x_1,\dots,x_k)$ is a σ-formula for some vocabulary σ, $\mathbf{A}$ a σ-structure, $\overline{a} = (a_1,\dots,a_k)$ a tuple of elements of $\mathrm{dom}(\mathbf{A})$, and $s\colon \{x_1,\dots,x_k\} \to \mathrm{dom}(\mathbf{A})$ with $s(x_i) = a_i$ for all i, we also write $\mathbf{A} \models \varphi(\overline{a})$ instead of $\mathbf{A}, s \models \varphi$. In particular, if φ is a sentence, we also write $\mathbf{A} \models \varphi$.

An important normal form of first-order logic is the so-called prenex normal form. A formula in prenex normal form is of the form

$$\varphi(x_1,\dots,x_k) = Q_1 y_1 \dots Q_\ell y_\ell \, \psi(x_1,\dots,x_k,y_1,\dots,y_\ell),$$

where $Q_1,\dots,Q_\ell \in \{\exists,\forall\}$ and $\psi(x_1,\dots,x_k,y_1,\dots,y_\ell)$ is a quantifier-free formula. We call $Q_1 y_1 \dots Q_\ell y_\ell$ the quantifier prefix of φ. Based on this normal form one can also define fragments Σ_k and Π_k of FO, the classes of the alternation hierarchy. The fragment Σ_k is the set of formulae in prenex normal form that start with an existential quantifier and have exactly k alternations between existential and universal quantifiers. More precisely, a formula in Σ_k is of the form

$$\varphi(\overline{x}) = \exists y_{1,1} \dots \exists y_{1,\ell_1} \forall y_{2,1} \dots \forall y_{2,\ell_2} \dots Q_k y_{k,1} \dots Q_k y_{k,\ell_m} \, \psi(\overline{x},\overline{y}),$$

where Q_k is $\forall$, if k is even, and Q_k is $\exists$ otherwise. The fragment Π_k is defined analogously with the quantifier prefix starting with a universal quantifier.

The following properties are folklore. Formulae in Σ_1 are preserved under extension, that is, for any $\varphi \in \Sigma_1$, and any structure $\mathbf{A}$ and assignment s with $\mathbf{A}, s \models \varphi$, we also have $\mathbf{A}', s \models \varphi$ for any $\mathbf{A}'$ that is obtained by adding new elements to the universe of $\mathbf{A}$ (without changing interpretations of relations). Similarly, formulae in Π_1 are preserved under taking induced substructures, that is, for any $\varphi \in \Pi_1$ the same holds for any $\mathbf{A}'$ that is obtained from $\mathbf{A}$ by removing elements not occurring in s from the universe and all tuples involving those elements from the interpretations of relation symbols.

Another important normal form exists for quantifier-free formulae. In analogy to disjunctive normal form in propositional logic, one can also define disjunctive normal form (DNF) for quantifier-free first-order formulae. Here, atoms and negated atoms are treated as literals so a quantifier-free FO-formula in DNF is of the form

$$\bigvee_{i=1}^{n} \bigwedge_{j=1}^{m} \ell_{i,j},$$

where $\ell_{i,j}$ is an atom or negated atom. Analogously one can define conjunctive normal form (CNF) for quantifier-free first-order formulae.

2.2.2 Encodings of Structures and FO-Definable Languages

In the context of complexity theory and especially when aiming to define complexity classes from classes of logics, it is useful to be able to view structures as strings. For this, we will introduce the standard encoding of structures as binary strings from the literature. As this requires an order on the elements of the domain, we assume that domains are always of the form $\{0, \ldots, n-1\}$ for some $n \in \mathbb{N}$ in this context. This is customary when working with encodings of structures as binary strings, and still allows us to talk about arbitrary structures modulo isomorphisms.

In the context of encodings of structures we restrict ourselves to relational vocabularies, since in this thesis we will only work with relational structures as inputs. Function symbols may occur as free variables (see Subsection 2.2.6) and constants may occur as built-in numerical constants (see Subsection 2.2.3) and hence they might occur in some vocabularies, but they will not be part of structures that are encoded in binary. Furthermore, note that delimiters in the encoding are not necessary, since the positions of certain bits in the encoding are determined by the cardinality of the universe.

We now fix a suitable encoding of structures as binary strings that is standard in the literature.

Definition 2.19. Let $\sigma = (\mathsf{R}_1, \ldots, \mathsf{R}_k)$ be a vocabulary and let $\mathbf{A}$ be a σ-structure. Then the encoding $\mathrm{enc}_\sigma(\mathbf{A})$ of $\mathbf{A}$ is the concatenation of the encodings of its relations. A relation $R \subseteq \mathrm{dom}(\mathbf{A})^m$ for $m \in \mathbb{N}$ is encoded by a string of length $|\mathrm{dom}(\mathbf{A})|^m$. The i-th bit of this encoding states whether the i-th tuple in $\mathrm{dom}(\mathbf{A})^m$ (in lexicographic order) is contained in R.

The case of the empty vocabulary needs to be treated separately, as the encoding of structures over the empty vocabulary would always be the empty string according to the above definition. As a work-around, Immerman treats such structures in the same way as if they had a single, empty, 1-ary relation. Instead, we will simply assume that relational vocabularies are always non-empty. By the above, properties of structures over the empty vocabulary can still be formulated as structures over a vocabulary with a single 1-ary relation symbol. Note that by this definition, the empty string ε is not the encoding of any structure. Furthermore, the following is obvious.

Observation 2.20. *Let $\sigma = (\mathsf{R}_1, \ldots, \mathsf{R}_k)$ be a vocabulary. Then for every $\mathbf{A} \in \mathrm{STRUC}[\sigma]$, the length of the encoding of $\mathbf{A}$ is*

$$|\mathrm{enc}_\sigma(\mathbf{A})| = \sum_{i=1}^{k} |\mathrm{dom}(\mathbf{A})|^{\mathrm{ar}(\mathsf{R}_i)}.$$

It immediately follows that only over vocabularies that contain only a single relation symbol encodings of structures can have length 1. Moreover, among vocabularies with a single relation symbol, only if that symbol is unary there are encodings of length 2 or 3. Consequently, there is only a single vocabulary σ, such that strings of length 1 and 2 (or 3) occur as encodings of σ-structures. This is the vocabulary $\sigma = (\mathsf{S}^1)$.

We are now in a position to define complexity classes from classes of FO-formulae.

Definition 2.21. Let $\mathfrak{C}$ be a class of FO-formulae and σ a vocabulary. A language L is definable in $\mathfrak{C}$ if there is a σ-sentence $\varphi \in \mathfrak{C}$ such that for all $x \in \{0,1\}^*$ we have

$$x \in L \iff \mathrm{enc}_\sigma^{-1}(x) \models \varphi,$$

if x is the encoding of a σ-structure, and $x \notin L$ otherwise. By abuse of notation, we also denote by $\mathfrak{C}$ the class of languages definable in $\mathfrak{C}$.

Intuitively, this means that for L and φ as above, a binary string x is contained in L if and only if it is the encoding of a model of φ.

2.2.3 Built-in Predicates and Classes of Definable Languages

Sometimes it is helpful to allow certain built-in predicates in FO. This is especially helpful when relating known complexity classes to classes definable by formulae. The main reason is that most computational models get an encoding of an input as a string and the bits in these encodings are inherently ordered. Also, for certain classes there are reasons to add predicates beyond an order. Formally, *built-in predicates* (or *numerical predicates*) and *built-in constants* (or *numerical constants*) are relation symbols (and constant symbols, respectively) whose interpretation is implicitly given based on the cardinality of the universe of the input structure. A typical example of a built-in predicate is the natural order on the set $\{0, \ldots, n-1\}$.

Formally, we use relation symbols N_i for $i \in \mathbb{N}$ as numerical predicate symbols and n_i for $i \in \mathbb{N}$ as numerical constant symbols. Each numerical predicate symbol

N_i has an associated arity $\mathrm{ar}(\mathsf{N}_i)$. Now a *(non-uniform) family of interpretations* (of the numerical predicates and constants) is a family $\mathcal{I} = (\mathbf{I}_n)_{n\in\mathbb{N}}$, where $\mathbf{I}_n$ maps each numerical predicate symbol N_i to a relation $\mathbf{I}_n(\mathsf{N}_i) \subseteq \{0,\dots,n-1\}^{\mathrm{ar}(\mathsf{N}_i)}$ and each numerical constant symbol to a number $\mathbf{I}_n(\mathsf{n}_i) \in \{0,\dots,n-1\}$.

Let σ be a vocabulary and φ a σ-sentence using the vocabulary τ of numerical predicate symbols, where $\tau = (\mathsf{N}_{i_1},\dots,\mathsf{N}_{i_k};;\mathsf{n}_{j_1},\dots,\mathsf{n}_{j_\ell})$ for some $k,\ell,i_1,\dots,i_k,j_1,\dots,j_\ell \in \mathbb{N}$. We say that φ holds in a σ-structure $\mathbf{A}$ with a family of interpretations $\mathcal{I} = (\mathbf{I}_n)_{n\in\mathbb{N}}$, denoted by $\mathbf{A} \models_{\mathcal{I}} \varphi$, if and only if $\mathbf{A} \cup (\mathbf{I}_{|\mathrm{dom}(\mathbf{A})|}(\tau)) \models \varphi$. Here, $\mathbf{I}_{|\mathrm{dom}(\mathbf{A})|}(\tau)$ denotes the tuple of interpretations of the symbols in τ by $\mathbf{I}_{|\mathrm{dom}(\mathbf{A})|}$.

We now define classes of languages definable by classes of first-order formulae with additional built-in predicates and constants.

Definition 2.22. Let $\mathfrak{C}$ be a class of FO-formulae, σ a vocabulary, $\mathfrak{R}$ a set of relations over $\mathbb{N}$, and $M \subseteq \mathbb{N}$ a set of constants. A language L is contained in the class $\mathfrak{C}[\mathfrak{R}, M]$ if there is a $\sigma \cup (\mathsf{N}_1,\mathsf{N}_2,\dots) \cup (;;\mathsf{n}_1,\mathsf{n}_2,\dots)$-sentence $\varphi \in \mathfrak{C}$ and a family of interpretations $\mathcal{I} = (\mathbf{I}_n)_{n\in\mathbb{N}}$, where $\mathbf{I}_n(\mathsf{N}_i)$ is the restriction of a relation in $\mathfrak{R}$ to $\{0,\dots,n-1\}$ and $\mathbf{I}_n(\mathsf{n}_i) \in M$ for all $n,i \in \mathbb{N}$, such that for all $x \in \{0,1\}^*$ we have

$$x \in L \iff \mathrm{enc}_\sigma^{-1}(x) \models_{\mathcal{I}} \varphi,$$

if x is the encoding of a σ-structure, and $x \notin L$ otherwise.

For a finite set $\mathfrak{R} = \{R_1,\dots,R_{k_1}\}$ of relations over $\mathbb{N}$ and a finite set $M = \{c_1,\dots,c_{k_2}\} \subseteq \mathbb{N}$, we also write $\mathfrak{C}[R_1,\dots,R_{k_1},c_1,\dots,c_{k_2}]$ instead of $\mathfrak{C}[\mathfrak{R},M]$. The case where just one of $\mathfrak{R}$ and M is finite or even empty is treated similarly. For example, we write $\mathfrak{C}[\mathfrak{R}]$ instead of $\mathfrak{C}[\mathfrak{R},\emptyset]$. Moreover, we use the term $\mathfrak{C}[\mathfrak{R},M]$-sentence to refer to a sentence as it occurs in the definition of the class $\mathfrak{C}[\mathfrak{R},M]$. Similarly, we also use the term FO[$\mathfrak{R}$]-query to refer to FO-queries $\mathcal{I}\colon \mathrm{STRUC}[\sigma] \to \mathrm{STRUC}[\tau]$ where σ contains built-in predicate symbols whose interpretation by relations from $\mathfrak{R}$ is understood from the context.

The most common examples of classes definable by first-order formulae with built-in predicates are the classes FO[Arb], and FO[BIT], and FO[+,×]. Here Arb refers to the set of all relations over $\mathbb{N}$ The BIT-predicate gives bitwise access to the binary representation of numbers from $\mathbb{N}$. More precisely, $(i,x) \in \mathrm{BIT}$ if and only if the i-th bit in the binary representation of x is 1. Here, for an n-bit number, the 0-th bit is the LSB while the $n-1$-th bit is the MSB. The relations + and × are the usual addition and multiplication in $\mathbb{N}$ given as ternary relations. The classes FO[+,×] and FO[BIT] coincide [Imm99]. We will formally use FO[BIT] in the following, which is relevant in the context of fragments for which the different versions do not coincide. In case of the full logic we will sometimes still make use of the predicates for + and × as they are expressible in FO[BIT].

In the case of classes such as FO[BIT] one asks for the existence of an FO[BIT]-sentence and a family of interpretations $\mathcal{I}$ with certain properties. For this reason it can be useful to make the presentation more simple by directly using numerical predicates and constants inside of formulae instead of numerical predicate symbols and numerical constant symbols. While this adds semantics to the syntax to some degree, this is

justified in this context as one can simply choose $\mathcal{I}$ as required as we are only interested in the existence of an appropriate family $\mathcal{I}$. This is the case throughout this thesis: We will use predicates such as $+$, $\times$, and BIT as well as constants such as min and max, referring to the minimal and maximal element of the universe, respectively, inside of formulae with access to these numerical predicates. In this case, the interpretation of the numerical predicates is implicit and clear from the context and the used symbols. We will for example write $\mathbf{A} \models \varphi$ for an FO-structure $\mathbf{A}$ and FO[BIT]-formula φ if and only if $\mathbf{A} \models_{\mathcal{I}} \varphi'$, where φ' is obtained from φ by replacing BIT by a numerical predicate symbol and $\mathcal{I}$ interprets that symbol as the relation BIT restricted to $\mathrm{dom}(\mathbf{A})$.

Note that numerical predicates do not change the encoding of structures as for a given structure $\mathbf{A}$ and family $\mathcal{I}$ of interpretations, the interpretation of the numerical predicate symbols is uniquely determined by $\mathcal{I}$ and $|\mathrm{dom}(\mathbf{A})|$.

2.2.4 Important Vocabularies

Two important vocabularies for the following topics are the vocabulary τ_{string} of binary strings and the vocabulary τ_{circ} of Boolean circuits defined as follows:

$$\tau_{\text{string}} := (\mathsf{S}^1),$$
$$\tau_{\text{circ}} := (\mathsf{E}^2, \mathsf{G}_\wedge^1, \mathsf{G}_\vee^1, \mathsf{Input}^2, \mathsf{negatedInput}^2, \mathsf{out}^1).$$

A τ_{string}-structure $\mathbf{A}$ with universe $\{0, \dots, n-1\}$ for some $n \in \mathbb{N}$ encodes a binary string by using the elements of the universe as positions while $\mathsf{S}^{\mathbf{A}}$ specifies the symbols of the word, more precisely: Let $w = w_0 \dots w_{n-1} \in \{0,1\}^*$ with $w_i \in \{0,1\}$. Then w is encoded by the structure $(\{0, \dots, n-1\}, \{i \mid w_i = 1\})$, which we denote by $\mathbf{A}_w$.

A τ_{circ}-structure with universe $\{0, \dots, n-1\}$ for some $n \in \mathbb{N}$ encodes a Boolean circuit with $\wedge$- and $\vee$-gates in input normal form. For this, the intended meaning of the symbols from τ_{circ} is as follows:

- $\mathsf{E}(x, y)$: gate x is a predecessor of gate y,
- $\mathsf{G}_\wedge(x)$: gate x is an and-gate,
- $\mathsf{G}_\vee(x)$: gate x is an or-gate,
- $\mathsf{Input}(x, i)$: gate x is associated with the i-th input bit,
- $\mathsf{negatedInput}(x, i)$: gate x is associated with the negation of the i-th input bit, and
- $\mathsf{out}(x)$: gate x is the output gate.

Note that we do not allow constants here to keep the definition simple. As mentioned earlier, the classes we consider do not change when disallowing constants.

As we also consider TC^0-circuits, we also require a vocabulary that allows to encode circuits that also use majority-gates. For this, we can simply add an additional predicate stating whether a gate is of this additional type, leading to the following vocabulary:

$$\tau_{\text{maj-circ}} := (\mathsf{E}^2, \mathsf{G}_\wedge^1, \mathsf{G}_\vee^1, \mathsf{G}_{\text{MAJ}}^1, \mathsf{Input}^2, \mathsf{negatedInput}^2, \mathsf{out}^1).$$

All predicates shared with τ_{circ} have the same intended meaning as before, while $x \in \mathsf{G}_{\text{MAJ}}^{\mathbf{C}}$ has the intended meaning that x is a majority-gate in the circuit $\mathbf{C}$.

Furthermore, we already mentioned the concept of classes of counting functions in the context of circuit complexity that are defined in terms of circuits with multiple output gates. When encoding such circuits as FO-structures, we need to replace the predicate out in the vocabulary by a new 2-ary relation symbol OUT, where $(g, i) \in \mathsf{OUT}^{\mathbf{C}}$ has the intended meaning that gate g is the i-th output gate of the circuit $\mathbf{C}$.

2.2.5 FO-Queries

We will also need the notion of *FO-queries* [Imm99], which are sometimes also called *FO-interpretations* [Daw15] or *FO-transductions* [Gro17]. An FO-query is a description of a τ-structure by FO-formulae over σ, where σ and τ are relational vocabularies, which means that it is a way to describe a mapping from σ-structures to τ-structures using FO-formulae. We will give the formal definition next.

Definition 2.23. Let σ, τ be relational vocabularies, $\tau =: (\mathsf{R}_1, \dots, \mathsf{R}_r)$, and $k \in \mathbb{N}$. An *FO-query* is a mapping

$$\mathcal{I} \colon \text{STRUC}[\sigma] \to \text{STRUC}[\tau]$$

given by a tuple $(\varphi_0, \varphi_{\mathsf{R}_1}, \dots, \varphi_{\mathsf{R}_r})$, where $\varphi_0, \varphi_{\mathsf{R}_1}, \dots, \varphi_{\mathsf{R}_r}$ are σ-formulae. The formula φ_0 has k free variables and φ_{R_i} has $k \cdot \text{ar}(\mathsf{R}_i)$ free variables for all $i \geq r$. For any structure $\mathbf{A} \in \text{STRUC}[\sigma]$, these formulae define the structure

$$\mathcal{I}(\mathbf{A}) := (\text{dom}(\mathcal{I}(\mathbf{A})); \mathsf{R}_1^{\mathcal{I}(\mathbf{A})}, \dots, \mathsf{R}_r^{\mathcal{I}(\mathbf{A})}) \in \text{STRUC}[\tau],$$

where $\text{dom}(\mathcal{I}(\mathbf{A}))$ is defined by φ_0 (sometimes also $\varphi_{\text{universe}}$) and the relations are defined by $\varphi_{\mathsf{R}_1}, \dots, \varphi_{\mathsf{R}_r}$ in the following way:

$$\text{dom}(\mathcal{I}(\mathbf{A})) := \{(a_1, \dots, a_k) \in \text{dom}(\mathbf{A})^k \mid \mathbf{A} \models \varphi_0(a_1, \dots, a_k)\} \text{ and}$$
$$\mathsf{R}_i^{\mathcal{I}(\mathbf{A})} := \{(b_1, \dots, b_{\text{ar}(\mathsf{R}_i)}) \in \text{dom}(\mathcal{I}(\mathbf{A}))^{\text{ar}(\mathsf{R}_i)} \mid \mathbf{A} \models \varphi_{\mathsf{R}_i}(b_1, \dots, b_{\text{ar}(\mathsf{R}_i)})\}.$$

Note that the composition of two FO-queries is again an FO-query.

2.2.6 Free Second-Order Variables in First-Order Logic

Second-order logic extends first-order logic by allowing variables for second-order objects, that is, relations and functions, as well as quantifiers for such objects. For this thesis, a more limited notion is sufficient: While we will need second-order variables, we will not need second-order quantifiers. Syntax and semantics for this case can be fully defined in terms of usual first-order syntax and semantics by simply extending the vocabulary and adding the assignments to second-order variables to the structure in the form of additional relations or functions. Let σ be vocabulary not containing the symbols $\mathsf{R}_1, \dots, \mathsf{R}_n, \mathsf{F}_1, \dots, \mathsf{F}_n$. A first-order formula over σ with free relational

variables $\mathsf{R}_1,\dots,\mathsf{R}_n$ and free functional variables $\mathsf{F}_1,\dots,\mathsf{F}_m$ is a first-order formula over $\sigma \cup (\mathsf{R}_1,\dots,\mathsf{R}_n;\mathsf{F}_1,\dots,\mathsf{F}_n;)$. Since the intended use is that we evaluate first-order formulae over σ with free relational and functional variables in σ-structures with separately given assignments to the free second-order variables, we use the following notation: Let φ be such a formula. We write φ as $\varphi(\mathsf{R}_1,\dots,\mathsf{R}_n,\mathsf{F}_1,\dots,\mathsf{F}_m)$. Let $R_1,\dots,R_n,F_1,\dots,F_m$ be interpretations of the corresponding free second-order variables of adequate arity and let $\mathbf{A}$ be a σ-structure. Then we write $\mathbf{A} \models \varphi(R_1,\dots,R_n,F_1,\dots,F_m)$ instead of $\mathbf{A} \cup (R_1,\dots,R_n,F_1,\dots,F_m) \models \varphi$.

2.2.7 The First-Order Model-Checking Game

Another way to view semantics of first-order logic is in terms of a two-player game. We begin with an intuitive explanation of this game and will give formal and precise definitions afterwards. Let φ be a first-order formula over some vocabulary σ, $\mathbf{A}$ a σ-structure and s an assignment to the free variables in φ. The game progresses as follows: Player 1 is the *verifier*. Their goal is to show that in fact $\mathbf{A}, s \models \varphi$. Player 2 on the other hand tries to show the converse and is hence called *falsifier*.

The game builds on the fact that for each operator and for each quantifier either a single witness is sufficient to show that said operator or quantifier evaluates to true or a single witness is sufficient to show that said operator or quantifier evaluates to false. For example, in order to show that a disjunction $\varphi \vee \psi$ evaluates to true in a structure $\mathbf{A}$ under some assignment s, it is sufficient for the verifier to choose between φ and ψ and show that the chosen formula evaluates to true in $\mathbf{A}$ under assignment s. As another example, consider a formula $\forall x\, \varphi$ starting on a universal quantifier, a structure $\mathbf{A}$ and an assignment s. In order to show that $\mathbf{A}, s \not\models \forall x\, \varphi$, it is sufficient for the falsifier to choose an element $a \in \mathrm{dom}(\mathbf{A})$ and show that φ evaluates to false in $\mathbf{A}$ under assignment $s[x/a]$.

The verifier tries to show that the formula evaluates to true in the given structure under the given assignment by making choices whenever a single witness it sufficient for this and the falsifier tries to show the converse by making choices when this is not the case. The goal of the verifier is to reach an atom that is true in the structure under the current assignment; the goal of the falsifier is to reach an atom that is false in the structure under the current assignment. Now $\mathbf{A}, s \models \varphi$ if and only if the verifier can always reach their goal independent of the choices made by the falsifier in the described game.

If φ contains negations, then the players simply switch their roles whenever a negation occurs: the current verifier becomes the new falsifier and the current falsifier becomes the new verifier. Despite the roles changing, we still usually refer to player 1 as the verifier and to player 2 as the falsifier, as their goal with regard to the whole formula stays the same. In cases where it is important to distinguish the players from the current roles, we make this distinction clear.

In the context of the counting classes considered in this thesis, it is necessary that strategies can vary for identical copies of the same subformula, i.e., a strategy can show different behavior for different copies of the same subformula. For this reason we define the model-checking game using a syntax tree representation of the formula. We now

formally defined the game.

For any vocabulary σ, a σ-formula φ represented as a syntax tree is a tuple $\varphi = (V, E, r, \lambda)$ with the following properties. The tuple (V, E, r) is a rooted directed tree, which means that (V, E) is a directed graph and $r \in V$ is a node without out-going edges in that graph such that from any node $v \in V$ there is a path from v to r in (V, E). Moreover, λ is a 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 $x \in \mathrm{Var}_{\mathrm{FO}}$ or an atom $\mathsf{R}(\overline{x})$. Similar as for the function β in circuits, we also say that λ labels the nodes in V. Here, R is a relation symbol from σ and $\overline{x}$ is a tuple of first-order variables of the appropriate arity. Occurrences of subformulae of φ correspond to nodes in this syntax tree, allowing a distinction between identical subformulae.

Let $\varphi = (V, E, r, \lambda)$ be an FO-formula over some vocabulary σ represented as a syntax tree and $\mathbf{A} \in \mathrm{STRUC}[\sigma]$. Let $s\colon \mathrm{free}(\varphi) \to \mathrm{dom}(\mathbf{A})$ be an assignment to the free variables of φ in $\mathbf{A}$. Then *configurations* of the game for $\mathbf{A}, s \models \varphi$ are of the form

$$(v, s', \mathrm{swap})$$

with the following properties:

- $v \in V$,
- s' is an assignment from the free variables in the subformula of φ rooted in v to elements of $\mathrm{dom}(\mathbf{A})$ that agrees with s for variables that are not quantified on the path from v to r, and
- swap is a bit specifying whether the players have currently swapped roles: If swap = 0, player 1 is the current verifier and player 2 is the current falsifier and if swap = 1 the roles are reversed.

The *full first-order model-checking game for* $\mathbf{A}, s \models \varphi$ starts in configuration

$$(r, s, 0).$$

The game proceeds as follows: Let

$$(v, s', \mathrm{swap})$$

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

- $\vee$ or $\wedge$: The current verifier or falsifier, respectively, chooses a child v' of v in (V, E, r), game continues in (v', s', swap),
- $\neg$: Game continues in $(v', s', 1 - \mathrm{swap})$, where v' is the single child of v in (V, E, r), and
- $\exists x$ or $\forall x$, where $x \in \mathrm{Var}_{\mathrm{FO}}$: The current verifier or falsifier, respectively, chooses an element $c \in \mathrm{dom}(\mathbf{A})$; the game continues in configuration $(v', s'[x/c], \mathrm{swap})$, where v' is the single child of v in (V, E, r).

We say that a configuration as above is *terminating*, if $\lambda(v)$ is an atom. It is called a winning configuration of player 1 (or winning configuration of the verifier) if it is terminating and either swap = 0 and $\mathbf{A}, s' \models \lambda(v)$ or swap = 1 and $\mathbf{A}, s' \not\models \lambda(v)$.

A *strategy of player 1* (or a strategy of the verifier) in this game is a function mapping configurations to moves in such a way that a move 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 moves. 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* (or a *winning strategy of the verifier*) is a strategy of player 1 such that every terminating configuration in the strategy is a winning configuration of player 1. Intuitively this means, that using a winning strategy, player 1 wins independent of the choices of player 2.

The above game captures first-order model-checking in the following sense.

Proposition 2.24. *Let σ be a vocabulary, φ a σ-formula, $\mathbf{A} \in \mathrm{STRUC}[\sigma]$ and $s\colon \mathrm{free}(\varphi) \to \mathrm{dom}(\mathbf{A})$. Player 1 has a winning strategy in the first-order model-checking game for $\mathbf{A}, s \models \varphi$ if and only if $\mathbf{A}, s \models \varphi$.*

This can be proven by first observing that our game is a combination of the original first-order model-checking game by Hintikka [Hin82] and the propositional logic model-checking game, which is folklore. Hintikka's model-checking game does not not contain rules for Boolean connectives. Instead, the game is played on formulae in prenex normal form and ends once the quantifier-free part is reached. Note that Hintikka's model-checking game is closely related to earlier work on game semantics for classic logic by Lorenzen and Lorenz [LL78]. As the game presented above is the combination of both these games, it is defined for formulae that are not in prenex normal form. Also the condition that needs to be checked to determine the winner is even simpler than in Hintikka's game. The proposition can then be shown by a simple induction over the compositional structure of the formula using the correctness of the two games. Intuitively, when $\mathbf{A}, s \models \varphi$, whenever player 1 has to make a choice there is a witness, while whenever player 2 has to make a choice there is no witness. Conversely, the choices made by player 1 yield witnesses that φ evaluates to true in $\mathbf{A}$ under assignment s.

We now want to give a simple example to illustrate the full first-order model-checking game.

Example 2.25. We want to express a simple property of Boolean circuits with input in FO and evaluate this formula in an exemplary circuit. As mentioned before, Boolean circuits are represented as first-order structures over the vocabulary τ_{circ}. A natural way to also add an input for the circuit is to use the vocabulary $\sigma := \tau_{\mathrm{circ}} \cup \tau_{\mathrm{string}}$. A circuit on n inputs for $n \in \mathbb{N}$ is now represented by a structure $\mathbf{A} \in \mathrm{STRUC}[\sigma]$ as follows: The universe $\mathrm{dom}(\mathbf{A}) := \{0, \dots, n-1\}^k$ with $k \in \mathbb{N}$ is the set of gates of the circuit represented by tuples over $\{0, \dots, n-1\}$. Potentially, some dummy-gates have to be added to ensure that the universe has this form. The relation symbols from τ_{circ} are interpreted according

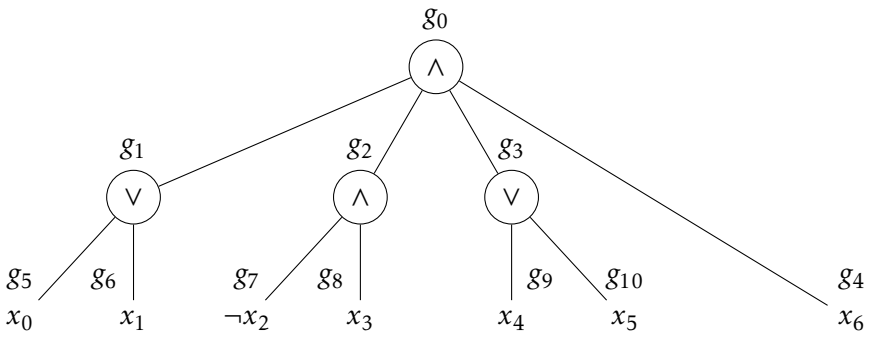

Figure 2.3: The structure **A** in Example 2.25.

to their intended meaning. For example, $g \in \mathsf{G}_\wedge^{\mathbf{A}}$ if and only if g is an $\wedge$-gate. The relation $\mathsf{S}^{\mathbf{A}}$ from τ_{string} only contains elements $(0,\dots,0,i)$ for $i \in \{0,\dots,n-1\}$ with the intended meaning that the i-th input bit is 1 if and only if $(0,\dots,0,i) \in \mathsf{S}^{\mathbf{A}}$.

Now let

$$\varphi := \forall g \Big(\neg \mathsf{G}_\wedge(g) \vee \exists g' \Big(\mathsf{E}(g', g) \wedge \varphi_{\text{trueLiteral}}(g') \Big) \Big),$$

where $\varphi_{\text{trueLiteral}}(x) := \exists i \Big(\mathsf{Input}(x,i) \wedge \mathsf{S}(i) \vee \mathsf{negatedInput}(x,i) \wedge \neg \mathsf{S}(i) \Big)$.

Furthermore, let **A** be the σ-structure representing the Boolean circuit shown in Figure 2.3 with input $x_0 \dots x_6 = 0100011$. Note that this is the same circuit as in Figure 1.1, but additional explanation has been omitted.

First note that **A** satisfies φ as every $\wedge$-gate in the circuit has at least one predecessor that is a literal evaluating to true: The only $\wedge$-gates are g_0 and g_2. Gate g_0 has predecessor g_4 associated with x_6, which is 1. Gate g_2 has predecessor g_7 associated with the negation of x_2, which is 1 (as x_2 is 0).

Now consider the first-order model-checking game for $\mathbf{A}, f_0 \models \varphi$ (where f_0 is the empty assignment). This game starts in configuration $(\varphi_0 := \varphi, s_0 := f_0, 0)$. As no subformula occurs more than once in φ except for the atomic formula $\mathsf{S}(i)$, we do not need to work with the syntax tree representation of the formula in this example and simply specify the current subformula in the first component. By Proposition 2.24, the fact that the formula holds in **A** means that the verifier has a winning strategy in this game.

We now describe how the game proceeds and how the winning strategy of player 1 can be obtained from the above argument for $\mathbf{A} \models \varphi$.

- Configuration $(\varphi_0 = \varphi, s_0 = f_0, 0)$. As the outermost operator or quantifier is $\forall g$, the falsifier moves first. They can choose any element $a \in \text{dom}(\mathbf{A})$ to assign to the variable g.

- Configuration $(\varphi_1, s_1, 0)$, where $\varphi_1 := \neg \mathsf{G}_\wedge(g) \vee \exists g' \Big(\mathsf{E}(g', g) \wedge \varphi_{\text{trueLiteral}}(g') \Big)$ and $s_1 := s_0[g/a]$. Here, the outermost operator or quantifier is the disjunction, so the

verifier chooses which subformula to continue with. Obviously, if $s_1(g) \notin \mathsf{G}_\wedge^{\mathbf{A}}$, the verifier can simply choose the subformula $\neg\mathsf{G}_\wedge(g)$. The game then continues to configuration $(\neg\mathsf{G}_\wedge(g), s_1, 0)$ and as the outermost operator or quantifier now is a negation further to $(\mathsf{G}_\wedge(g), s_1, 1)$. As $\mathbf{A}, s_1 \models \mathsf{G}_\wedge(g)$, this is a winning configuration of the verifier, or more precisely player 1. On the other hand, if $s_1(g) \in \mathsf{G}_\wedge^{\mathbf{A}}$, the verifier has to choose the subformula $\varphi_2 \coloneqq \exists g'\Big(\mathsf{E}(g', g) \wedge \varphi_{\text{trueLiteral}}(g')\Big)$.

- Configuration $(\varphi_2, s_1, 0)$. This only applies for $s_1(g) \in \{g_0, g_2\}$ as observed above. We cover the case where $s_1(g) = g_0$ in detail. Here, the verifier chooses g_4 to assign to the variable g', since g_4 is both a predecessor of g_0 and a (possibly negated) input gate that evaluates to true.

- Configuration $(\mathsf{E}(g', g) \wedge \varphi_{\text{trueLiteral}}(g'), s_2, 0)$ with $s_2 \coloneqq s_1[g'/g_4]$. As the outermost operator or quantifier now is a conjunctions, the falsifier chooses which subformula to continue with. As $s_2(g') = g_4$ is a predecessor of $s_2(g) = g_0$, the falsifier immediately loses when choosing $\mathsf{E}(g', g)$. Hence, we only need to consider the falsifier choosing $\varphi_{\text{trueLiteral}}(g')$ with regard to strategies of the verifier.

- Configuration $(\varphi_{\text{trueLiteral}}(g'), s_2, 0)$. As the outermost operator or quantifier is again an existential quantifier, the verifier chooses an assignment again. Obviously, for a winning strategy, 6 has to be assigned to i, as g_4 is associated to x_6 and in the following step the subformula $\mathsf{Input}(x, i) \wedge \mathsf{S}(i)$ has to be chosen as g_4 is not associated with a negated input bit.

- Configuration $(\mathsf{Input}(x, i) \wedge \mathsf{S}(i), s_3, 0)$, where $s_3 \coloneqq s_2[i/6]$. As the outermost operator or quantifier is a conjunction, the falsifier makes the final choice. As $\mathbf{A}, s_3 \models \mathsf{Input}(x, i)$ and $\mathbf{A}, s_3 \models \mathsf{S}(i)$, no matter what choice is made, a winning configuration of the verifier is reached.

- The case where $s_1(g) = g_2$ is handled similarly.

The first-order model-checking game can also be used in the context of structures with built-in predicates. In case of a logic FO[$\mathfrak{R}$] where $\mathfrak{R}$ is a set of relations over $\mathbb{N}$, a game is defined for any tuple $(\mathbf{A}, \varphi, s)$ again and the interpretations of the numerical predicates are implicit. In particular, this applies to FO[BIT]-sentences and FO[Arb]-sentences.

2.2.8 Skolemization and Skolem Functions

Yet another view on semantics of first-order logic—albeit related to the game-semantic view presented in the previous section—arises from what is called Skolemization. Skolemization is a tool used in the context of first-order satisfiability. Consider a formula in prenex normal form. Skolemization is the process of replacing existential quantifiers in the formula by new function symbols in a certain way, which yields an equisatisfiable formula in Skolem normal form. Here, equisatisfiable means that the new formula is satisfiable if and only if the original formula is satisfiable, and Skolem

normal form means that the formula is in prenex normal form and does does not contain existential quantifiers. For an intuitive explanation consider a σ-formula

$$\varphi = \exists x_1 \forall x_2 \exists x_3 \, \psi(x_1, x_2, x_3)$$

for some vocabulary σ. Now instead of asking for the existence of an x_1 we can simply ask for the existence of a constant a_1 (or equivalently a 0-ary function) with the same properties. Also, instead of asking whether for all x_2 there is an x_3 with property $\psi(x_1, x_2, x_3)$ we can ask for the existence of a 1-ary function f_3 with the property $\psi(x_1, x_2, f_3(x_2))$. Now, the Skolem normal form of φ is obtained by removing the existential quantifiers and replacing every occurrence of x_1 in ψ by a_1 and every occurrence of x_3 in ψ by $f_3(x_2)$, yielding a formula over σ extended by the constant symbol a_1 and the 1-ary function symbol f_3. Since first-order satisfiability asks for the existence of a structure that contains interpretations of a_1 and f_3, this new formula is equisatisfiable to φ. The function symbols replacing existential variables are called Skolem functions (or Skolem function symbols) and assignments to these function symbols correspond to winning strategies of the verifier in Hintikka's first-order model-checking game. Intuitively, this correspondence is due to the fact that Skolem functions directly specify the choices of the verifier for any combination of choices made by the falsifier. This correspondence was observed in a much more general setting, see [Grä13].

For the purposes of this thesis, we describe the process of Skolemization formally using first-order formulae with free second-order variables instead of extending the vocabulary. Let σ be a vocabulary and φ be a σ-formula. If φ does not contain any existential quantifier, then φ is already in Skolem normal form. Otherwise, $\varphi = \forall x_1 \dots \forall x_k \exists y \, \psi$ for some $k \in \mathbb{N}$ and some σ-formula ψ, i.e., $\exists y$ is the outermost existential quantifier in φ. Now let $\varphi' := \forall x_1 \dots \forall x_k \, \psi[y/f(x_1, \dots, x_k)]$, where f is a new free k-ary function variable. The above transformation removes the outermost existential quantifier and replaces it by a new function variable. By applying this procedure for each existential quantifier in φ successively, we obtain a formula without existential quantifiers, that is, a formula in Skolem normal form.

2.3 Descriptive Complexity

In descriptive complexity, the complexity of problems is measured by the logics required to describe them. For this, complexity classes are defined from logics. In Definition 2.22 we have already seen how this can be done for the example of first-order logic, using the fact that first-order structures can be encoded as binary words. Note that due to the definition of the standard encoding of structures (and the fact that the universe of structures is non-empty), the empty string ε is not considered as an input in this context. Characterizations of complexity classes in the sense of descriptive complexity are also called model-theoretic, as languages are viewed as sets of models of certain formulae. In this section we want to give some intuition on how to approach model-theoretic characterizations, explaining the proof ideas for a few examples relevant to us. We will also repeat results from descriptive complexity of circuit complexity classes closely

related to this thesis in more detail compared to the presentation in overview given in the introduction. For more details on the topic we refer the reader to monographs by Immerman and Libkin [Imm99, Lib04].

The area of descriptive complexity was started by Fagin [Fag74] with the following theorem.

Theorem 2.26. NP = ESO.

Here, ESO refers to the class of languages definable in existential second-order logic, which can be defined as follows using the definitions from Subsection 2.2.6.

Definition 2.27. A language L is in ESO if there is a vocabulary σ and a σ-formula with free relational variables $\varphi(\mathsf{R}_1, \ldots, \mathsf{R}_n)$ such that for all $\mathbf{A} \in \mathrm{STRUC}[\sigma]$

$$\begin{aligned} \mathrm{enc}_\sigma(\mathbf{A}) \in L \iff & \text{there are relations } R_1, \ldots, R_n \text{ over } \mathrm{dom}(\mathbf{A}) \text{ with arities} \\ & \mathrm{ar}(\mathsf{R}_1), \ldots \mathrm{ar}(\mathsf{R}_n), \text{ respectively, such that } \mathbf{A} \models \varphi(R_1, \ldots, R_n), \end{aligned}$$

and $x \notin L$, if x is not the encoding of a σ-structure.

We now give a proof sketch for Fagin's Theorem, briefly explaining the intuition behind it.

Proof sketch for Theorem 2.26. A formula showing that a language is in ESO is an FO-formula $\varphi(\mathsf{R}_1, \ldots, \mathsf{R}_n)$ with free relational variables $\mathsf{R}_1, \ldots, \mathsf{R}_n$ such that any $x \in \{0,1\}^+$ is in the language if it encodes a structure in which there is an assignment to the relational variables satisfying φ. An NP-algorithm can simply nondeterministically guess an assignment, that is, a relation for each relation symbol, and then check whether it satisfies φ. This is basically FO model-checking (by extending the input structure by the guessed relations) and can hence be done in P.

On the other hand, a computation path of an NP-machine can be represented by a tableau with polynomially many rows and columns by having the rows contain the configurations in the order of their occurrence during the computation. Both the number of rows as well as the size of configurations is polynomial due to the polynomial runtime of the machine, which allows us to encode the whole tableau in a tuple of relations of high enough arity. This further requires an order on tuples over the universe, which can for example be obtained by adding two more free relational variables: one will be an order on the universe (which can be expressed in FO), the other is the lexicographic order on tuples defined from the first order (also expressible in FO). □

The name ESO stands for **e**xistential **s**econd-**o**rder logic. This refers to second-order logic with only existential second-order quantifiers, but arbitrary first-order quantifiers. While omitting the formal definition, we want to point out that it is easy to see that the complexity class ESO defined above is exactly the class of languages definable in the logic ESO. We also say that the logic ESO *captures* the class NP. More generally, when the class of languages definable in a logic (cf. Definition 2.21) are exactly the languages in a complexity class, we also say that the logic *captures* that class.

Remark 2.28. In all of the model-theoretic characterizations in this work, we allow to use arbitrary vocabularies. This allows for more direct logical definitions of many natural problems. Immerman [Imm99] showed that for any vocabulary σ, there is an FO-query $\mathrm{bin}_\sigma \colon \mathrm{STRUC}[\sigma \cup (\mathrm{BIT})] \to \mathrm{STRUC}[\tau_{\mathrm{string}} \cup (\mathrm{BIT})]$ such that $\mathrm{bin}_\sigma(\mathbf{A}) = \mathbf{A}_{\mathrm{enc}_\sigma(\mathbf{A})}$ and BIT is interpreted in the intended way in $\mathrm{bin}_\sigma(\mathbf{A})$. This means that each structure $\mathbf{A}$ is mapped to the τ_{string}-structure representing the encoding of $\mathbf{A}$. There is also an FO-query for the inverse bin_σ^{-1} of bin_σ for any vocabulary σ.

The model-theoretic classes used in our characterizations are closed under applying partial first-order queries (cf. Definition 7.19). Furthermore, every string occurs as the encoding of some τ_{string}-structure. For this reason, one could equivalently define the model-theoretic classes we use in our characterizations only in terms of the vocabulary τ_{string}. This does not apply to some weaker fragments considered in Chapter 5.

2.3.1 Descriptive Complexity of Circuit Complexity Classes

We will now turn our attention to descriptive complexity of circuit complexity classes. Here, we first introduce a notion of uniformity often used in this context. It is based on logics, more precisely on FO[BIT]-queries, and is therefore especially suitable in the context of descriptive complexity.

Definition 2.29. A circuit family $C = (\mathbf{C}_n)_{n \in \mathbb{N}^+}$ is said to be FO[BIT]-uniform if there is an FO[BIT]-query $\mathcal{I} \colon \mathrm{STRUC}[\tau_{\mathrm{string}}] \to \mathrm{STRUC}[\tau_{\mathrm{circ}}]$ mapping the structure $\mathbf{A}_w$ over τ_{string} to the circuit $\mathbf{C}_{|w|}$ given as a structure over vocabulary τ_{circ} for any $w \in \{0,1\}^+$.

Notice that implicitly, the mapping $\mathcal{I}$ for a FO[BIT]-uniform circuit family only depends on the length of the input and not the specific input bits, as for any $n \in \mathbb{N}$ all input structures $\mathbf{A}_w$ with $|w| = n$ are mapped to the same circuit $C_{|w|}$. Also, for any structure $\mathbf{A}$ and FO[BIT]-query $\mathcal{I}$, the universe of $\mathcal{I}(\mathbf{A})$ is a subset of $\mathrm{dom}(\mathbf{A})^k$ for some $k \in \mathbb{N}$. Consequently, FO[BIT]-uniform circuit families are always of polynomial size. As all of the classes we are interested in are based on polynomial size circuit families, this is not an issue. In the case of FO[BIT]-uniformity, a circuit family is uniformly described by an FO-query. This can be interpreted as the property that queries to the circuits in a circuit family (as τ_{circ}-structures) can be answered by evaluating FO-formulae. In consequence, this uniformity can be seen as another query-based uniformity condition, similar to $\mathrm{U_D}$-uniformity, where answers to similar queries have to be computable in deterministic logarithmic time.

Remark 2.30. By this definition, circuit families in this context do not contain a circuit for $n = 0$. The reason is that there is no structure whose encoding is the empty string. Consequently, all characterizations in this context are with respect to words of length > 0. We will still often talk about circuit families $C = (\mathbf{C}_n)_{n \in \mathbb{N}}$ being FO[BIT]-uniform, implicitly meaning that the circuit $\mathbf{C}_0$ is omitted from the family.

We have already seen classes similar to the complexity class ESO in the context of build-in vocabularies (see Section 2.2.3), namely classes $\mathrm{FO}[\mathfrak{R}]$ for sets $\mathfrak{R}$ of relations

over $\mathbb{N}$. As already mentioned, the important cases for us are those where $\mathfrak{R}$ is either Arb or {BIT}. The predicate BIT has proven essential for many of the descriptive complexity results regarding uniform circuit complexity classes, while Arb is required in the context of non-uniform classes. Note that there was also work on the classes obtained with even weaker built-in predicates, e.g., only allowing the built-in order $\leq$ instead of BIT (cf. results on weaker logics and their relations to presumably smaller, non-standard complexity classes [BL06]).

The first model-theoretic characterization in circuit complexity were proven by Immerman and the result was generalized in the following years [Imm89, BIS90, BI94, Imm99].

Theorem 2.31. *For $i \geq 0$, $\mathrm{AC}^i = \mathrm{FO}[(\log n)^i]$ both in the* FO[BIT]*-uniform and the non-uniform setting.*

Here, $\mathrm{FO}[t(n)]$ denotes the class of languages definable in first-order logic with $t(n)$ iterations of a quantifier block, see [Imm99] for a formal definition. As usual in this context, arbitrary numerical predicates are used for the non-uniform version. While this result yielded a characterization of all classes AC^i, all classes apart from the characterization of AC^0 use an external mechanism of repeating a certain part of the formula. We will now give a short proof sketch for the base case ($i = 0$) of the above theorem, as the proof idea is the foundation for our characterizations of counting classes.

Proof Sketch for Theorem 2.31. We briefly give some intuition on how to prove the case for $i = 0$, that is, FO[BIT]-uniform $\mathrm{AC}^0 = \mathrm{FO}[\mathrm{BIT}]$. The main idea is that existential and universal quantifiers over a polynomial domain correspond to polynomial fan-in disjunctions and conjunctions, respectively.

For the inclusion from left to right, the circuit family is first transformed into a normal form where circuits are layered and each layer has a fixed type of gate (either $\wedge$ or $\vee$). Then we can use the correspondence between Boolean connectives and quantifiers to construct a formula expressing that the circuits evaluate to 1. Uniformity of the circuit family is used to access the circuit for inputs of length n from any input structure $\mathbf{A}_w$ with $|w| = n$.

For the converse direction, the correspondence of Boolean connectives and quantifiers can be used to evaluate the quantifiers. What remains is to evaluate atoms. For this, the numerical predicates of the uniformity can be used to determine the required input bit to get the value of a given atom. □

The following characterization by Barrington et al. [BIS90] is based on the groundbreaking result by Barrington [Bar89] showing that bounded-width branching programs characterize NC^1 and that the word problem for any non-solvable group is NC^1-complete. Here, $\mathrm{FO} + Q_{\mathrm{Mon}}$ refers to first-order logic with so-called monoidal quantifiers. Ultimately, it is sufficient to have monoidal quantifiers for a single non-solvable group. For details on the definition of monoidal quantifiers, see the original paper. The uniformity condition used in this result, $\mathrm{U}^*_{\mathrm{E}}$-uniformity, will be covered in Section 2.4.

Theorem 2.32. $\mathrm{NC}^1 = \mathrm{FO}[\mathrm{Arb}] + Q_{\mathrm{Mon}}$ *and* $\mathrm{U}^*_{\mathrm{E}}$*-uniform* $\mathrm{NC}^1 = \mathrm{FO}[\mathrm{BIT}] + Q_{\mathrm{Mon}}$.

Based on the same ideas, NC^1 can also be characterized by first-order logic extended by the width-5 transitive closure operator [BIS90].

Another characterization of NC^1 in the $\mathrm{U}_{\mathrm{E}}^*$-uniform setting was obtained by Compton and Laflamme [CL90] and adds a kind of recursive definition of predicates to FO, called relational primitive recursion. Their logic FO[BIT] + RPR allows recursive definitions of the form $[\mathsf{P}(\overline{x}, y) \equiv \theta(\overline{x}, y, \mathsf{P}(\overline{x}, y-1))]$ inside of formulae. This definition allows the usage of the predicate symbol P in the formula following afterwards, where the interpretation of $\mathsf{P}(\overline{x}, y)$ is determined recursively by the given equivalence. It should be noted that numerical predicates are required for this operator to make sense since the term $y-1$ is used.

Theorem 2.33. $\mathrm{U}_{\mathrm{E}}^*$-uniform $\mathrm{NC}^1 = \mathrm{FO[BIT]} + \mathrm{RPR}$.

Lautemann et al. [LMSV01] characterized LOGCFL in the spirit of the characterization of NC^1 by FO with group quantifiers. They showed that LOGCFL, or equivalently SAC^1, can be characterized by FO with groupoid quantifiers. Similar to above, $\mathrm{FO} + Q_{\mathrm{Grp}}$ denotes first-order logic with so-called groupoidal quantifiers. We again refer the reader to the original paper for details on the definition of these quantifiers.

Theorem 2.34. L-uniform $\mathrm{SAC}^1 = \mathrm{FO[BIT]} + Q_{\mathrm{Grp}}$.

2.3.2 Descriptive Complexity of Counting Complexity Classes

The study of descriptive complexity of counting complexity classes was started by Saluja et al. [SST95]. They transferred the result NP = ESO to the counting setting, obtaining a characterization of #P by counting the number of satisfying assignments to free second-order variables in an otherwise first-order formula. To emphasize that free relation variables are used in this setting, we denote this class by $\#\mathrm{FO}^{\mathrm{rel}}$. For this result, they considered counting functions that map structures over a fixed vocabulary to natural numbers instead of functions from $\{0,1\}^*$ to $\mathbb{N}$. We adapt the definition to our presentation of descriptive complexity results, defining $\#\mathrm{FO}^{\mathrm{rel}}$ as follows. Note that due the limitations of first-order logic, we restrict our counting functions to inputs of length ≥ 1 in this context. We will sometimes still talk about functions mapping from $\{0,1\}^*$ to $\mathbb{N}$ being in classes such as $\#\mathrm{FO}^{\mathrm{rel}}$, implicitly restricting the function to inputs of length ≥ 1.

Definition 2.35. A function $f : \{0,1\}^+ \to \mathbb{N}$ is in $\#\mathrm{FO}^{\mathrm{rel}}$, if there is a vocabulary σ and an FO[$\leq$]-formula $\varphi(\mathsf{R}_1, \ldots, \mathsf{R}_k, x_1, \ldots, x_\ell)$ over σ with free relation variables $\mathsf{R}_1, \ldots, \mathsf{R}_k$ and free individual variables $x_1, \ldots, x_\ell$ such that for all $\mathbf{A} \in \mathrm{STRUC}[\sigma]$,

$$f(\mathrm{enc}_\sigma(\mathbf{A})) = |\{(S_1, \ldots, S_k, c_1, \ldots, c_\ell) \mid \mathbf{A} \models \varphi(S_1, \ldots, S_k, c_1, \ldots, c_\ell)\}|,$$

and $f(x) = 0$, if x is not the encoding of a σ-structure.

The result by Saluja et al. can now be stated as follows.

Theorem 2.36. $\#\mathrm{P} = \#\mathrm{FO}^{\mathrm{rel}}$.

Proof sketch. This result is obtained similarly to NP = ESO. For the inclusion $\#\mathrm{FO}^{\mathrm{rel}} \subseteq \#\mathrm{P}$, it is again straightforward to guess an assignment to the free relation variables in a fixed formula and check if it is indeed satisfying in polynomial time, yielding a #P-algorithm. For the converse, a more careful analysis is required. First, one has to ensure that there is indeed a 1-1 correspondence between accepting computation paths of the given machine and satisfying assignments to the free relation variables of the constructed formula. Furthermore, an order on the elements is still required to talk about computations and more precisely successive computation steps and adjacent cells of the band. While this was easily quantifiable in the proof for NP = ESO, this is not possible here, as there is not a unique order. Hence, the result only holds over ordered structures. □

2.4 Choice of Uniformity

In descriptive complexity, it is natural and convenient to work with a uniformity condition for circuit families that is based on logic. The usual choice is FO[BIT]-uniformity, which was defined in Definition 2.29. But is this uniformity suitable for the classes we consider? There has been extensive research on different forms of uniformity and their relationships for Boolean circuits in the setting of classes of decision problems. Unfortunately, the same is not true in the setting of counting classes in circuit complexity. When counting classes in circuit complexity were first studied, uniform versions were defined in terms of uniformity conditions that were usually used for the corresponding classes of Boolean circuits: Vinay defined the class $\#\mathrm{SAC}^1$ using L-uniformity [Vin91]. Caussinus et al. used $\mathrm{U_D}$-uniformity for the class $\#\mathrm{NC}^1$ [CMTV98]. Agrawal et al. [AAD00] and Ambainis et al. [ABL98] considered different kinds of uniformity for the class $\#\mathrm{AC}^0$, but mentioned that $\mathrm{U_D}$-uniformity is usually considered suitable in the context of constant depth, referencing results in the decision setting. This is a reasonable approach. Intuitively, this means that these notions of uniformity are not too powerful, as counting classes can in a sense be seen as generalizations of the corresponding decision classes. This is, e.g., evidenced by the fact that $\mathrm{FAC}^0 \subseteq \#\mathrm{AC}^0$ as well as the fact that, using an adequate model of computation, counting arithmetic circuits can be evaluated in the same parallel time as Boolean circuits of the same depth. An adequate model in this context would be able to compute the arithmetic operations computed by gates in the circuits in constant time—instead of Boolean operations on individual bits. On the other hand, the ability of these uniformities to desribe meaningful circuit families in the decision setting is evidence that they are not too restrictive. Similarly, we will base our choice of uniformity on what is known with regard to different uniformity conditions for the decision versions of the classes we consider. For this reason, we will give an overview of what is known about the properties and connections between different forms of uniformity for classes in the NC-, SAC-, AC-, and TC-hierarchies.

We want to begin by going into a bit more detail on what properties a suitable uniformity condition should have. On one hand, a suitable uniformity should not be too restrictive, meaning that it does not make trivial constructions impossible. On the other hand, it should not be too powerful, as this would in a sense hide or obscure the true

power of the class, similar to how non-uniformity allows to solve arbitrary problems that are defined with respect to only the input length (see Proposition 2.16). In consequence, a good choice for a suitable uniformity condition is one that is based on a complexity class characterizing the resulting circuit complexity class. For example, we know that P coincides with the class of languages accepted by P-uniform polynomial-size families of Boolean circuits. Hence, P-uniformity is a natural and suitable uniformity condition for polynomial-size circuits. Another desirable property is a certain robustness, i.e., slight modifications of the uniformity condition should not change the resulting complexity class. In fact, in many cases very restrictive uniformity conditions coincide with more powerful ones, as long as they are not more powerful than the resulting complexity class. Again using the above example, the characterization of the class P in terms of polynomial size families of Boolean circuits holds using either P-uniformity, L-uniformity, or U_D-uniformity.

We will now present results on the relationships between different uniformity conditions in the context of the relevant decision classes. As FO[BIT]-uniform AC^0 = FO[BIT], this uniformity can be seen as a perfect choice for AC^0. It was also shown that U_D-uniform AC^0 coincides with FO[BIT]-uniform AC^0, where a main part of the proof was to show DLOGTIME $\subseteq$ FO[BIT] [Imm89, BIS90]. The suitability of U_D-uniformity for this class can also be seen from the fact that evaluating an U_D-uniform AC^0 circuit family is possible in logarithmic time and with a constant number of alternations, that is, in LH, which is another characterization of AC^0. The above result on U_D- and FO[BIT]-uniformity was generalized to all classes in the AC-hierarchy, potentially extended by an additional operation [BI94]. As TC^i is the class AC^i extended by gates for the majority-operation, this means that for all classes from the AC- and TC-hierarchies, U_D-uniformity and FO[BIT]-uniformity coincide. It should be mentioned that for classes of circuits containing SAC^1, usually L-uniformity is used. The reason is that L $\subseteq$ L-uniform SAC^1 and hence, intuitively, L-uniformity does not give any additional power to these classes. We will cover the relationship of the above, more restrictive, uniformity conditions to L-uniformity for these classes in more detail at the end of this section.

Regarding classes based on bounded fan-in circuits, the situation is slightly different. Here, we know that NC^1 = ALOGTIME. Of course, DLOGTIME $\subseteq$ ALOGTIME, but still, evaluating an U_D-uniform NC^1 circuit family on an alternating Turing machine takes time $O((\log n)^2)$, as in each step from a gate to a predecessor, the machine for the uniformity has to be simulated. For this reason, different uniformity conditions were introduced for the class NC^1 [Ruz81]. Similar to U_D-uniformity they are query-based, but they use a modified version of the direct connection language. Let $C = (\mathbf{C}_n)_{n\in\mathbb{N}}$ be a circuit family of size s and depth d. Recall that the direct connection language $L_{DC}(C)$ of this family consists of tuples $\langle y, g, p, b\rangle$ and allows to query edges of the circuits in C: For $p \in \{0,1\}^+$, we have $\langle y, g, p, b\rangle \in L_{DC}(C)$, if and only if b is the p-th predecessor of g in $\mathbf{C}_{|y|}$ (interpreting p as a natural number). In contrast, the *extended connection language* $L_{EC}(C)$ allows to query paths of length $\leq \log s(n)$: For $p \in \{0,1\}^+$, we have $\langle y, g, p, b\rangle \in L_{EC}(C)$, if and only if b is the gate reached from g using the (reversed) path described by p. Otherwise, $L_{EC}(C)$ is defined in the same way as $L_{DC}(C)$. As this notion is usually only

considered for bounded fan-in circuit families, we have $|p| \leq \log s(|y|)$ and each bit of p specifies whether the first or second predecessor should be chosen from one gate on the path. Based on the extended connection language, two further uniformity conditions are defined. The family C is called U_E-uniform, if $L_{EC}(C)$ can be decided by a deterministic Turing machine in time $O(\log s(n))$. It is called U_E^*-uniform, if $L_{EC}(C)$ can be decided by an alternating Turing machine in space $O(\log s(n))$ and time $O(d(n))$. For the classes of the NC-hierarchy, the following is known regarding U_E- and U_E^*-uniformity.

Theorem 2.37.

1) U_E-uniform $NC^1 = U_E^*$-uniform $NC^1 = ALOGTIME \subseteq$ L-uniform NC^1 *and*

2) U_E-uniform $NC^i = U_E^*$-uniform $NC^i = ATIME\text{-}SPACE((\log n)^i, \log n)$
$=$ L-uniform NC^i *for all* $i \geq 2$.

This means that U_E^*-uniformity in case of NC^1 can be seen as another perfect uniformity condition, as its definition is based on the class ALOGTIME, which at the same time coincides with the resulting class. For this reason, this class is sometimes also called NC^1-uniform NC^1 [BIS90]. Also, this shows that either U_E- or U_E^*-uniformity are suitable for all of these classes. As could be expected due to $L \subseteq NC^2$, for the class NC^2 and even higher classes, L-uniformity also coincides with the two aforementioned notions, and is therefore also suitable.

In case of NC^1, also uniformity conditions based on formulae instead of circuits were considered. Here, Buss used a variant where the so-called formula languages of families of formulae, defined similarly to the direct connection languages of circuit families, are in ALOGTIME [Bus87]. The resulting class again coincides with U_E^*-uniform NC^1. Barrington et al. instead studied the variant, where the formula language is in DLOGTIME [BIS90].

Regarding the usage of the above two uniformity conditions for AC^0 circuits and the usage of U_D-uniformity for bounded fan-in classes, it should be noted that for AC^0, U_D-uniformity (and hence also FO[BIT]-uniformity) coincides with U_E-uniformity, but U_E^*-uniformity is not a reasonable notion in this case, at it would require machines running in constant time. On the other hand, for NC^2 and even higher classes in the NC-hierarchy, U_D-uniformity coincides with U_E-uniformity, and by the above Theorem also with the other mentioned notions.

We now want to shed a bit more light on classes SAC^i and AC^i for $i \geq 1$. Here, L-uniformity is often used in the literature, since $L \subseteq$ L-uniform SAC^1. Actually, as in the case of classes NC^2 for $i \geq 2$, for these classes L-uniformity coincides with more restrictive forms of uniformity such as U_D-uniformity and FO[BIT]-uniformity. This can be seen from results by Ruzzo [Ruz81] and Venkateswaran [Ven91]. Venkateswaran showed LOGCFL = L-uniform SAC^1. It directly follows that U_D-uniform SAC^1 is a subset of LOGCFL, but the converse requires additional arguments. In order to show LOGCFL $\subseteq$ L-uniform SAC^1, Venkateswaran used the notion of machines and circuits with the semi-unboundedness property, where intuitively the lengths of paths of successive universal nodes in computations are bounded, but the time or depth of computations

is not limited (see the original paper for details). We briefly argue that his proof can also be used to obtain LOGCFL $\subseteq$ U_D-uniform SAC^1. First, he showed that LOGCFL-languages can be decided by alternating Turing machines with the semi-unboundedness property in space $O(\log n)$ and with $O(\log n)$ alternations, which does not depend on the uniformity condition. Then, he used a construction of Ruzzo showing that an alternating Turing machine using $O(s(n))$ space and $O(t(n))$ time can be simulated by a U_E-uniform circuit family of size $2^{O(s(n))}$ and depth $t(n)$ [Ruz81]. This allowed him to show that languages decidable by alternating Turing machines with the semi-unboundedness property in space $O(\log n)$ and with $O(\log n)$ alternations can also be accepted by circuit families with the semi-unboundedness property of polynomial size and $O(\log n)$ alternations. Ruzzo already showed that his construction can be made U_E-uniform, which implies U_D-uniformity. Finally, from the above circuit family he constructs an SAC^1 circuit family. His proof can directly be adapted to the U_D-uniform setting, as the uniformity of the SAC^1 circuit family mostly depends on the uniformity of the previous family. The most complex property to check is determining whether there is a path between two disjunction-gates in the circuit using only conjunction-gates, which is possible, as only a constant number of gates have to be considered due to semi-unboundedness. Hence, it can be seen that for SAC^1, the notions of U_D-, and L-uniformity coincide. Also, we now argue that the same holds for all classes $\mathfrak{C}$, where either $\mathfrak{C} = AC^i$ with $i \geq 1$, or $\mathfrak{C} = SAC^i$ with $i \geq 2$. First, note that SAC^1 circuit families are also $\mathfrak{C}$ circuit families by definition, and we have L $\subseteq$ LOGCFL = U_D-uniform SAC^1. Now, instead of using an L-machine to specify edges and types of gates, one can use U_D-uniform SAC^1-subcircuits. In detail, a construction similar to our construction for Theorem 7.4 could be used here, where connections and gate types in circuits are determined by advice bits. In this case, these advice bits could then be computed by U_D-uniform SAC^1-subcircuits. Finally, as DLOGTIME $\subseteq$ FO[BIT] $\subseteq$ L, these arguments show that for these classes, L-uniformity also coincides with FO[BIT]-uniformity.

In conclusion, we have seen that for all the classes in the considered hierarchies, if the class contains L, then U_D-, FO[BIT]-, and L-uniformity coincide for that class. Thus, although L-uniformity is the prevalent choice for these classes in the literature, the FO[BIT]-uniform versions, which are better suited in the context of descriptive complexity, coincide with the corresponding L-uniform versions. In case of AC^0, FO[BIT]-uniformity can even be considered the optimal choice, as it is based on a complexity class exactly capturing the resulting uniform class. The only class where FO[BIT]-uniformity is not optimal is NC^1, where U_E- or U_E^*-uniformity seem to be the most natural choices, and U_E^*-uniformity is in a sense optimal as it is based on ALOGTIME, which coincides with U_E^*-uniform NC^1. Still, the U_D-uniform version of the class $\#NC^1$ was studied in the literature [CMTV98], and this version can be expected to be very close to the FO[BIT]-uniform version. Even more, we will later see that our results clarify the relationship of FO[BIT]- and U_D-uniformity for the considered classes, see Sections 4.3 and 6.4. For these reasons, we argue that FO[BIT]-uniformity is a reasonable choice for all of the considered classes, and we will focus on this uniformity throughout this thesis. It would nonetheless also be interesting to study the descriptive complexity of U_E^*-uniform $\#NC^1$ in the future.

3 Counting Witnesses in First-Order Logic

As advertised in the title, the topic of this thesis is descriptive complexity of counting classes and more specifically counting classes in circuit complexity. As most counting classes are defined in terms of a process of witness-counting, a typical approach is to start with a model-theoretic characterization of the associated decision class and introduce a process of counting witnesses into the model-theoretic class used in that characterization. This was already illustrated by the counting version of Fagin's theorem in Subsection 2.3.2.

We now aim to do the same for the class $\#AC^0$. Its associated decision class is AC^0 and as mentioned before, it is known that $AC^0 = FO$ both in the uniform and the non-uniform case. In consequence, we now want to introduce counting classes based on counting of witnesses in FO-formulae.

We begin by introducing counting classes based on FO and showing some important properties of these classes. We will then compare our choice of model to other alternatives.

3.1 A Counting Class Based on First-Order Logic

Let σ be a vocabulary. For any σ-formula φ, σ-structure $\mathbf{A}$ and assignment s to the free variables of φ, we have $\mathbf{A}, s \models \varphi$ if and only if player 1 has a winning strategy in the first-order model-checking game. Therefore, winning strategies of player 1 can be seen as witnesses for the fact that $\mathbf{A}, s \models \varphi$.

Following this line of thought we will introduce the new class #Win-FO, the class of functions counting winning strategies on first-order formulae. Previously we have already seen other candidates for witnesses of acceptance in first-order logic, namely winning strategies in Hintikka's model-checking game and equivalently Skolem functions. We defer the discussion of the choice of model to the next section, beginning with the definition and some important properties of the resulting classes.

The model-checking game can analogously be defined for formulae with built-in predicates, more precisely: For any set $\mathfrak{R}$ of relations over $\mathbb{N}$ and any first-order vocabulary σ, the *full first-order model-checking game for* $\mathbf{A}, s \models_{\mathcal{I}} \varphi$, where $\mathbf{A}$ is a σ-structure, φ is an $FO[\mathfrak{R}]$-formula over σ and s is an assignment to the free variables in φ, is defined in the same way as the corresponding game for $\mathbf{A}, s \models \varphi$ with the difference that for atoms involving built-in predicate symbols, the family $\mathcal{I}$ is used to evaluate the atom and determine the winner. Let $\#\mathrm{Win}(\mathbf{A}, \mathcal{I}, \varphi, s)$ denote the number of winning strategies of the verifier in that game for $\mathbf{A}, s \models_{\mathcal{I}} \varphi$. As in the case of the modeling relation, we often omit the family $\mathcal{I}$ if it is clear from the context, writing

$\#\mathrm{Win}(\mathbf{A}, \varphi, s)$ instead of $\#\mathrm{Win}(\mathbf{A}, \mathcal{I}, \varphi, s)$, where $\mathcal{I}$ interprets the built-in predicate symbols in the intended way. This for example applies for built-in predicates such as BIT, $+$, and $\times$ that are often used in formulae (instead of built-in predicate symbols) by abuse of notation. We also write $\#\mathrm{Win}(\mathbf{A}, \mathcal{I}, \varphi(a_1, \ldots, a_k))$ (or $\#\mathrm{Win}(\mathbf{A}, \varphi(a_1, \ldots, a_k))$) instead of $\#\mathrm{Win}(\mathbf{A}, \mathcal{I}, \varphi, s)$ (or $\#\mathrm{Win}(\mathbf{A}, \varphi, s)$, respectively) when $\mathrm{free}(\varphi) = \{x_1, \ldots, x_k\}$, $a_1, \ldots, a_k \in \mathrm{dom}(\mathbf{A})$, and $s\colon \mathrm{free}(\varphi) \to \mathrm{dom}(\mathbf{A})$ is defined as $s(x_i) := a_i$ for all i. Accordingly, we simply omit the assignment s in the case that φ is a sentence. We now define the desired counting classes as follows.

Definition 3.1. Let $\mathfrak{R}$ be a set of relations over $\mathbb{N}$. A function $f\colon \{0,1\}^+ \to \mathbb{N}$ is in the class #Win-FO[$\mathfrak{R}$], if there is a vocabulary σ, an FO[$\mathfrak{R}$]-sentence φ over σ, and a non-uniform family $\mathcal{I}$ of interpretations of the built-in predicate symbols in φ by symbols in $\mathfrak{R}$ such that for all $\mathbf{A} \in \mathrm{STRUC}[\sigma]$,

$$f(\mathrm{enc}_\sigma(\mathbf{A})) = \#\mathrm{Win}(\mathbf{A}, \mathcal{I}, \varphi),$$

and $f(x) = 0$ if x is not the encoding of a σ-structure.

We will mainly consider the classes #Win-FO[BIT] and #Win-FO[Arb] in this thesis, where #Win-FO[BIT] is short for #Win-FO[{BIT}], in analogy to our notation for the corresponding logics. In Chapter 4, we will see that these classes actually capture the FO[BIT]-uniform and the non-uniform version of $\#\mathrm{AC}^0$, respectively.

As a preparation for further results first note that the number of winning strategies of the verifier in the full first-order model-checking game can be computed inductively over the compositional structure of formulae as follows.

Proposition 3.2. *Let $\mathfrak{R}$ be a set of relations over $\mathbb{N}$ and σ a vocabulary. Let $\mathbf{A}$ be a σ-structure, φ an FO[$\mathfrak{R}$]-formula over σ, s an assignment to the free variables in φ, and $\mathcal{I}$ a family of interpretations of the built-in predicate symbols in φ by relations in $\mathfrak{R}$.*
Using $\#win(\psi, s')$ as a shorthand for $\#\mathrm{Win}(\mathbf{A}, \mathcal{I}, \psi, s')$ for arbitrary formulae ψ and assignments s', the number of winning strategies of the verifier in the game for $\mathbf{A}, s \models_{\mathcal{I}} \varphi$ arises from the number of winning strategies on subformulae of φ as follows.

- *If $\varphi = \varphi_1 \vee \varphi_2$, then $\#win(\varphi, s) = \#win(\varphi_1, s) + \#win(\varphi_2, s)$.*
- *If $\varphi = \varphi_1 \wedge \varphi_2$, then $\#win(\varphi, s) = \#win(\varphi_1, s) \cdot \#win(\varphi_2, s)$.*
- *If $\varphi = \exists x\, \varphi_1$, then $\#win(\varphi, s) = \sum_{a \in \mathrm{dom}(\mathbf{A})} \#win(\varphi_1, s[x/a])$.*
- *If $\varphi = \forall x\, \varphi_1$, then $\#win(\varphi, s) = \prod_{a \in \mathrm{dom}(\mathbf{A})} \#win(\varphi_1, s[x/a])$.*
- *If $\varphi = \neg\varphi_1$, then $\#win(\varphi, s) = \#win(\varphi', s)$, where φ' is the negation normal form of φ.*

Proof sketch. Exemplary, we cover the cases of existential and universal quantifiers. The remaining cases can be shown similarly.

$\varphi = \exists x\, \varphi_1(x)$: The set of all winning strategies of the verifier in the game for $\mathbf{A}, s \models_{\mathcal{I}} \varphi$ is exactly the set of all strategies that assign a value a to x and then use an arbitrary

winning strategy in the game for $\mathbf{A}, s[x/a] \models_{\mathcal{I}} \varphi_1(a)$. Thus, $\#\mathrm{win}(\varphi, s)$ is the sum of $\#\mathrm{win}(\varphi_1, s[x/a])$ over all elements $a \in \mathrm{dom}(\mathbf{A})$.

$\varphi = \forall x\, \varphi_1(x)$: Each winning strategy of the verifier in the game for $\mathbf{A}, s \models_{\mathcal{I}} \varphi$ has to choose for each assignment a for x a winning strategy in the game for $\mathbf{A}, s[x/a] \models_{\mathcal{I}} \varphi_1(a)$. Arbitrary combinations of these choices are possible. Hence, $\#\mathrm{win}(\varphi, s)$ is the product of $\#\mathrm{win}(\varphi_1, s[x/a])$ over all elements $a \in \mathrm{dom}(\mathbf{A})$. □

Next we show a useful property with regard to the first-order model-checking game: For any formula we can find an equivalent formula on which the verifier has at most one winning strategy in the model-checking game.

Lemma 3.3. *Let $\varphi(z_1, \ldots, z_\ell)$ be an* FO[Arb]*-formula over some vocabulary σ with free variables $\overline{z}$. Then there is an* FO[Arb]*-formula φ' over σ in prenex normal form using the same built-in predicate symbols that is equivalent to φ such that for all $\mathbf{A} \in \mathrm{STRUC}[\sigma]$, $\overline{c} \in \mathrm{dom}(\mathbf{A})^\ell$, and non-uniform families of interpretations $\mathcal{I}$:*

$$\mathbf{A} \models_{\mathcal{I}} \varphi'(\overline{c}) \implies \#\mathrm{Win}(\mathbf{A}, \mathcal{I}, \varphi'(\overline{c})) = 1.$$

Proof. Without loss of generality assume that φ is in prenex normal form. Let

$$\varphi =: Q_1 x_1 \ldots Q_k x_k\, \psi(\overline{z}, \overline{x}),$$

where ψ is quantifier-free. We prove by induction that for all $0 \leq i \leq k$ there are formulae $\varphi_i(z_1, \ldots, z_\ell, x_1, \ldots, x_{k-i})$ and $\varphi'_i(z_1, \ldots, z_\ell, x_1, \ldots, x_{k-i})$ in prenex normal form such that

$$\varphi_i \equiv Q_{k-i+1} x_{k-i+1} \ldots Q_k x_k\, \psi(\overline{z}, \overline{x}),$$

$$\varphi'_i \equiv \neg \varphi_i,$$

and for all $\mathbf{A} \in \mathrm{STRUC}[\sigma]$, $\overline{c} \in \mathrm{dom}(\mathbf{A})^{\ell+k-i}$ and non-uniform families of interpretations $\mathcal{I}$ we have

$$\mathbf{A} \models_{\mathcal{I}} \varphi_i(\overline{c}) \implies \#\mathrm{Win}(\mathbf{A}, \mathcal{I}, \varphi_i(\overline{c})) = 1 \text{ and}$$

$$\mathbf{A} \models_{\mathcal{I}} \varphi'_i(\overline{c}) \implies \#\mathrm{Win}(\mathbf{A}, \mathcal{I}, \varphi'_i(\overline{c})) = 1.$$

For $i = 0$ we have to show that $\psi(\overline{z}, \overline{x})$ and $\neg\psi(\overline{z}, \overline{x})$ have the desired property. Since these formulae are quantifier-free, we can simply construct a DNF for each of them in which each atom that occurs in the formula occurs in every term in the disjunction. This ensures that for all structures $\mathbf{A} \in \mathrm{STRUC}[\sigma]$, all $\overline{c} \in \mathrm{dom}(\mathbf{A})^{\ell+k}$ and all non-uniform families of interpretations at most one term of the disjunction can be satisfied and hence the verifier has at most one winning strategy.

For $i - 1 \to i$: By induction hypothesis we have formulae $\varphi_{i-1}(\overline{z}, x_1, \ldots, x_{k-i+1})$ and $\varphi'_{i-1}(\overline{z}, x_1, \ldots, x_{k-i+1})$ with the above properties. Without loss of generality we can assume that these two formulae use disjoint sets of bound variables. Let

$$\varphi_{i-1}(\overline{z}, \overline{x}) =: Q_{a_1} a_1 \ldots Q_{a_m} a_m\, \xi(\overline{z}, \overline{x}, \overline{a}) \text{ and}$$
$$\varphi'_{i-1}(\overline{z}, \overline{x}) =: Q_{b_1} b_1 \ldots Q_{b_n} b_n\, \xi'(\overline{z}, \overline{x}, \overline{b}),$$

where ξ and ξ' are quantifier-free.

First assume $Q_{k-i+1} = \exists$. Then we have $\varphi_i(\overline{z}, x_1, \ldots, x_{k-i}) \equiv \exists x_{k-i+1}\, \varphi_{i-1}(\overline{z}, \overline{x})$ and $\neg\varphi_i(\overline{z}, x_1, \ldots, x_{k-i}) \equiv \forall x_{k-i+1}\, \varphi'_{i-1}(\overline{z}, \overline{x})$. The formula $\forall x_{k-i+1}\, \varphi'_{i-1}(\overline{z}, \overline{x})$ already has the desired properties, since for all $\mathbf{A} \in \mathrm{STRUC}[\sigma]$, $\overline{c} \in \mathrm{dom}(\mathbf{A})^\ell$, $\overline{d} \in \mathrm{dom}(\mathbf{A})^{k-i}$, and all non-uniform families of interpretations $\mathcal{I}$:

$$\#\mathrm{Win}(\mathbf{A}, \mathcal{I}, \forall x_{k-i+1} \varphi'_{i-1}(\overline{c}, x_{k-i+1}, \overline{d})) = \prod_{d_{k-i+1} \in \mathrm{dom}(\mathbf{A})} \#\mathrm{Win}(\mathbf{A}, \mathcal{I}, \varphi'_{i-1}(\overline{c}, d_{k-i+1}, \overline{d})),$$

and $\#\mathrm{Win}(\mathbf{A}, \mathcal{I}, \varphi'_{i-1}(\overline{c}, d_{k-i+1}, \overline{d})) \leq 1$ for all $d_{k-i+1} \in \mathrm{dom}(\mathbf{A})$.

The formula $\exists x_{k-i+1}\, \varphi_{i-1}(\overline{z}, \overline{x})$ does not yet have the desired properties, since the existential quantifier might have multiple witnesses. Thus, instead of expressing "there is an x_{k-i+1}" we express "there is a minimal x_{k-i+1}". This leads to the following formula:

$$\exists x_{k-i+1} \forall y_{k-i+1}\, \varphi_{i-1}(\overline{z}, \overline{x}) \wedge \Big(y_{k-i+1} \geq x_{k-i+1} \vee \big(\neg(y_{k-i+1} \geq x_{k-i+1}) \wedge \neg\varphi_{i-1}(\overline{z}, x_1, \ldots, x_{k-i}, y_{k-i+1}) \big) \Big).$$

We can replace $\neg\varphi_{i-1}$ in this formula by φ'_{i-1}. The resulting formula is obviously not in prenex normal form. Therefore, we need to transform it to prenex normal form in a way that preserves the desired property. By defining $\xi''(\overline{z}, x_1, \ldots, x_{k-i}, x_{k-i+1}, y_{k-i+1}, \overline{b})$ as the formula

$$\Big((y_{k-i+1} \geq x_{k-i+1} \wedge \bigwedge_{\substack{i \in \{1,\ldots,n\}, \\ Q_{b_i} = \exists}} b_i = 0) \vee (\neg(y_{k-i+1} \geq x_{k-i+1}) \wedge \xi'(\overline{z}, x_1, \ldots, x_{k-i}, y_{k-i+1}, \overline{b}) \Big),$$

we have for all $\mathbf{A} \in \mathrm{STRUC}[\sigma]$, $\overline{c} \in \mathrm{dom}(\mathbf{A})^{\ell+k-(i-1)+1}$, and all non-uniform families of interpretations $\mathcal{I}$:

$$\mathbf{A} \models_{\mathcal{I}} Q_{b_1} b_1 \ldots Q_{b_n} b_n \xi''(\overline{c}, \overline{b}) \implies \#\mathrm{Win}(\mathbf{A}, \mathcal{I}, Q_{b_1} b_1 \ldots Q_{b_n} b_n \xi''(\overline{c}, \overline{b})) = 1.$$

This is, because in ξ'', if $y_{k-i+1} \geq x_{k-i+1}$ is satisfied, the only winning strategy is to set b_i to 0 for all i with $Q_{b_i} = \exists$ independent of the choices by the other player and if $\neg(y_{k-i+1} \geq x_{k-i+1})$ is satisfied, the winning strategies exactly correspond to the winning strategies for $\mathbf{A} \models_{\mathcal{I}} \varphi'_i(\overline{z}, \overline{x})$. Due to the properties of φ'_i, this means that there is at most one winning strategy in this case.

Now,

$$\exists x_{k-i+1} \forall y_{k-i+1} \Big((Q_{a_1} a_1 \ldots Q_{a_m} a_m\, \xi) \wedge (Q_{b_1} b_1 \ldots Q_{b_n} b_n\, \xi'') \Big),$$

is equivalent to φ_i. Since for both subformulae connected by the outermost conjunction we have the property that the number of winning strategies is ≤ 1, we can now simply pull out the quantifiers: In the formula

$$\exists x_{k-i+1} \forall y_{k-i+1} Q_{a_1} a_1 \ldots Q_{a_m} a_m Q_{b_1} b_1 \ldots Q_{b_n} b_n (\xi \wedge \xi''),$$

which is still equivalent to φ_i, the following holds: If the formula is satisfied, there is exactly one way to choose values for the existentially quantified variables among $a_1,\dots,a_m$ in a winning strategy in order to satisfy ξ and exactly one way to choose values for the existentially quantified variables among $b_1,\dots,b_n$ in a winning strategy in order to satisfy ξ'' for any assignment to x_{k-i+1} and y_{k-i+1}. Also, since the formula

$$\forall y_{k-i+1} Q_{a_1} a_1 \dots Q_{a_m} a_m Q_{b_1} b_1 \dots Q_{b_n} b_n (\xi \wedge \xi'')$$

expresses that x_{k-i+1} is the minimal element satisfying a certain formula, there can be at most one choice for x_{k-i+1}. This means that the formula has the desired property.

The case $Q_{k-i+1} = \forall$ can be proven analogously with the roles of φ_i and φ_i' reversed. □

Next, we show that using an FO[BIT]-query $\mathcal{I}\colon \mathrm{STRUC}[\sigma] \to \mathrm{STRUC}[\tau]$, we can for any given τ-sentence φ construct an FO[BIT]-sentence φ' over σ such that the verifier has the same number of winning strategies in the model-checking game for $\mathbf{A} \models \varphi'$ as he has in the game for $\mathcal{I}(\mathbf{A}) \models \varphi$. The intuitive proof idea is to plug in the formulae from the query $\mathcal{I}$ and use Lemma 3.3 to ensure that this does not increase the number of winning strategies. This result will be important later on but can also be used to prove that #Win-FO[BIT] is closed under FO[BIT]-reductions (exact definition follows). From now on, we will denote by BIT_n the restriction of BIT to numbers from $\{0,\dots,n-1\}$.

Lemma 3.4. *Let φ be a τ-sentence for some vocabulary τ, and $\mathcal{I}\colon \mathrm{STRUC}[\sigma \cup (\mathrm{BIT})] \to \mathrm{STRUC}[\tau]$ be an* FO*-query. Then there is an* FO[BIT]*-sentence φ' over σ such that for all structures $\mathbf{A} \in \mathrm{STRUC}[\sigma]$,*

$$\#\mathrm{Win}(\mathbf{A}, \varphi') = \#\mathrm{Win}(\mathcal{I}(\mathbf{A} \cup (\mathrm{BIT}_{|\mathrm{dom}(\mathbf{A})|})), \varphi).$$

Proof. Let φ_0 be the formula in $\mathcal{I}$ defining the universe and let r be the arity of φ_0. By Lemma 3.3 we can assume without loss of generality that the number of winning strategies of the verifier in the model-checking game for $\mathbf{A}, s \models \varphi_0$ is at most 1 for any structure $\mathbf{A}$ and assignment s. Moreover, let φ_0' be a formula equivalent to $\neg\varphi_0$ with the same property.

The idea is to plug in the formulae from $\mathcal{I}$ into φ, replacing all occurrences of symbols from τ. In order to use formulae from $\mathcal{I}$ instead of predicates of τ, we first need to replace each variable x in φ by an r-tuple $(x^1,\dots,x^r)$ of fresh variables. Furthermore, quantifiers should be limited to elements of the universe, i.e., tuples satisfying φ_0. For this, inductively replace $\exists\overline{x}\,\alpha$ by $\exists\overline{x}(\varphi_0(\overline{x}) \wedge \alpha)$ and $\forall\overline{x}\,\alpha$ by $\forall\overline{x}\big((\varphi_0 \wedge \alpha) \vee \varphi_0'\big)$ (starting from the innermost quantifiers). Note that due to the replacement of variables by tuples, quantifiers always occur in blocks of r quantifiers, which can be replaced accordingly. As the number of winning strategies on φ_0 and φ_0' is at most 1, this modification does not change the number of winning strategies.

Now, we can replace each occurrence of a predicate R in τ in φ by the corresponding formula φ_{R} from $\mathcal{I}$. For this, we can again by Lemma 3.3 assume without loss of generality that there is at most one winning strategy of the verifier in the model-checking

game for φ_R in any structure and for any assignment. Let φ' be the resulting formula, which is an FO[BIT]-formula over STRUC$[\sigma]$. It is now easy to see that for all $\mathbf{A} \in$ STRUC$[\sigma]$,

$$\#\mathrm{Win}(\mathbf{A}, \varphi') = \#\mathrm{Win}(\mathcal{I}(\mathbf{A} \cup (\mathrm{BIT}_{|\mathrm{dom}(\mathbf{A})|})), \varphi). \qquad \square$$

As already mentioned, this lemma yields an interesting closure property as a corollary, that is, closure under FO[BIT]-reductions.

Definition 3.5. Let $f, g : \{0,1\}^+ \to \mathbb{N}$. We say that f is (many-one) FO[BIT]-reducible to g, in symbols $f \leq^{\mathrm{fo}} g$, if there are vocabularies σ, τ and an FO-query $\mathcal{I}$: STRUC$[\sigma \cup (\mathrm{BIT})] \to$ STRUC$[\tau \cup (\mathrm{BIT})]$ such that for all $\mathbf{A} \in$ STRUC$[\sigma]$,

- the structure $\mathcal{I}(\mathbf{A} \cup (\mathrm{BIT}_{|\mathrm{dom}(\mathbf{A})|}))$ interprets BIT in the intended way and
- $f(\mathrm{enc}_\sigma(\mathbf{A})) = g(\mathrm{enc}_\tau(\mathcal{I}(\mathbf{A})))$, and $f(x) = 0$, if x is not the encoding of a σ-structure.

Corollary 3.6. *The class* #Win-FO[BIT] *is closed under* FO[BIT]*-reductions, that is: Let* f, g *be functions such that* $g \in$ #Win-FO[BIT] *and* $f \leq^{\mathrm{fo}} g$*. Then* $f \in$ #Win-FO[BIT].

Proof. Let $f \leq^{\mathrm{fo}} g$ via the FO-query $\mathcal{I}$: STRUC$[\sigma_1 \cup (\mathrm{BIT})] \to$ STRUC$[\sigma_2 \cup (\mathrm{BIT})]$. Furthermore, let $g \in$ #Win-FO[BIT] via the sentence φ over vocabulary τ, that is, for all $\mathbf{A} \in$ STRUC$[\tau]$ we have

$$g(\mathrm{enc}_\tau(\mathbf{A})) = \#\mathrm{Win}(\mathbf{A}, \varphi),$$

and $g(x) = 0$ if x is not the encoding of a τ-structure.

As discussed in Remark 2.28 there are FO-queries bin_σ and bin_σ^{-1} for any vocabulary σ, which constitute mappings between the τ_{string}-structure representing the encoding of a structure and the corresponding structure over vocabulary σ, both with built-in BIT. Note that bin_σ^{-1} can behave arbitrarily for structures $\mathbf{A}_w$ where w is not the encoding of a σ-structure. Now, $\mathcal{I}' := \mathrm{bin}_\tau^{-1} \circ \mathrm{bin}_{\sigma_2} \circ \mathcal{I}$: STRUC$[\sigma_1 \cup (\mathrm{BIT})] \to$ STRUC$[\tau \cup (\mathrm{BIT})]$ is an FO-query with the following properties: For any $\mathbf{A} \in$ STRUC$[\sigma_1]$ where $\mathrm{enc}_{\sigma_2}(\mathcal{I}(\mathbf{A}))$ is also the encoding of a τ-structure, $\mathcal{I}'(\mathbf{A})$ is the τ-structure with $\mathrm{enc}_\tau(\mathcal{I}'(\mathbf{A})) = \mathrm{enc}_{\sigma_2}(\mathcal{I}(\mathbf{A}))$. Here, by abuse of notation, we identify structures over $\sigma \cup (\mathrm{BIT})$ with the underlying σ-structure for arbitrary vocabularies σ, as long as BIT is interpreted in the intended way. This is done to simplify the presentation. For example, $\mathrm{enc}_\tau(\mathcal{I}'(\mathbf{A}))$ is the encoding of the underlying τ-structure of $\mathcal{I}'(\mathbf{A})$, although this structure contains an explicit interpretation of BIT. In the same term, $\mathbf{A}$ is implicitly extended by the appropriate interpretation of BIT in order to obtain a structure over $\sigma_1 \cup (\mathrm{BIT})$. We can now apply Lemma 3.4 to obtain a sentence φ' over σ_1 such that for all $\mathbf{A} \in$ STRUC$[\sigma_1]$ where $\mathrm{enc}_{\sigma_2}(\mathcal{I}(\mathbf{A}))$ is also the encoding of a τ-structure, we have

$$\begin{aligned} \#\mathrm{Win}(\mathbf{A}, \varphi') &= \#\mathrm{Win}(\mathcal{I}'(\mathbf{A}), \varphi) \\ &= g(\mathrm{enc}_\tau(\mathcal{I}'(\mathbf{A}))) \\ &= g(\mathrm{enc}_{\sigma_2}(\mathcal{I}(\mathbf{A}))) \\ &= f(\mathrm{enc}_{\sigma_1}(\mathbf{A})). \end{aligned}$$

As the length of the encoding of STRUC[σ_2]- and STRUC[τ]-structures are both given by some polynomial, the property of the encoding of a STRUC[σ_2]-structure also being the encoding of a τ-structure is definable in FO[BIT]. Let ψ be a formula defining for any $\mathbf{A} \in \text{STRUC}[\sigma_1]$ that the length of the encoding of $\mathcal{I}(\mathbf{A})$ is also the length of the encoding of a τ-structure. By Lemma 3.3 we can assume that that for all $\mathbf{A} \in \text{STRUC}[\sigma_1]$, $\#\text{Win}(\mathbf{A}, \psi) \leq 1$. Now, for $\varphi'' := \varphi' \wedge \psi$ we have for all $x \in \{0,1\}^*$

$$\#\text{Win}(\text{enc}_{\sigma_1}^{-1}(x), \varphi' \wedge \psi) = f(x),$$

if x is the encoding of a σ_1-structure and $f(x) = 0$, otherwise. Hence, $f \in$ #Win-FO[BIT]. □

3.2 Counting in First-Order Logic: Choice of Model

Here we discuss the choice of model for the classes #Win-FO and #Win-FO[Arb] and compare the class defined above to the classes arising with different choices. As mentioned in Section 3.1, there are different kinds of objects that can be seen as witnesses for acceptance in the setting of first-order definable languages. For our main definition we use winning strategies in the full first-order model-checking game as witnesses of acceptance. Instead, one could consider the original model-checking game by Hintikka, which is played only on formulae in prenex normal form and terminates as soon as the quantifier-free part of the formula is reached. Further, another candidate for witnesses are Skolem functions instead of winning strategies. We will now compare all three variants and highlight subtle differences between the classes arising from them. Moreover, we will observe the respective properties of the classes we want to characterize, namely non-uniform $\#\text{AC}^0$ and FO[BIT]-uniform $\#\text{AC}^0$, and discuss the consequences for this thesis.

We denote by $\#\text{Win}^{\text{Hintikka}}(\mathbf{A}, \mathcal{I}, \varphi, s)$ the number of winning strategies of the verifier in the original model-checking game of Hintikka for $\mathbf{A}, s \models_{\mathcal{I}} \varphi$, again omitting $\mathcal{I}$ if it is clear from the context, and omitting s in the case that φ is a sentence. We denote by $\#\text{Skolem}(\mathbf{A}, \mathcal{I}, \varphi, s)$ the number of Skolem functions for the formula φ in the structure $\mathbf{A}$ under assignment s and using the non-uniform family $\mathcal{I}$ of interpretations. Define the classes $\#\text{Win-FO[BIT]}^{\text{Hintikka}}$, $\#\text{Win-FO[Arb]}^{\text{Hintikka}}$, #Skolem-FO[BIT] and #Skolem-FO[Arb] analogously to the classes #Win-FO[BIT] and #Win-FO[Arb] based on the corresponding types of witnesses.

We now investigate the relationship between these classes. First note that by the correspondence between Winning Strategies and Skolem functions [Grä13], we get the following identities.

Proposition 3.7.

1. $\#\text{Win-FO[BIT]}^{\text{Hintikka}} = \#\text{Skolem-FO[BIT]}$ *and*
2. $\#\text{Win-FO[Arb]}^{\text{Hintikka}} = \#\text{Skolem-FO[Arb]}$.

Next, we compare the two different definitions based on winning strategies in the first-order model-checking game. Since it is easy to ensure that the verifier has at most one winning strategy for the quantifier-free part of a formula in any structure and for any assignment to the free variables, functions counting the number of winning strategies in Hintikka's model-checking game also arise from the full model-checking game.

Lemma 3.8.

1. $\#\text{Win-FO[BIT]}^{\text{Hintikka}} \subseteq \#\text{Win-FO[BIT]}$ *and*
2. $\#\text{Win-FO[Arb]}^{\text{Hintikka}} \subseteq \#\text{Win-FO[Arb]}$.

Proof. We prove the result for the uniform classes. The proof for the non-uniform classes is completely analogous.

Let $f \in \#\text{Win-FO[BIT]}^{\text{Hintikka}}$ via the formula $\varphi = Q_1 x_1 \dots Q_k x_k \, \psi(x_1, \dots, x_k)$ with $k \in \mathbb{N}$. Since quantifiers are treated the same in both model-checking games and Hintikka's game ends when reaching the quantifier-free part we replace the quantifier-free part by a formula on which the verifier has at most one winning strategy in the full model-checking game using Lemma 3.3. Let ψ' be the formula with this property. Then $Q_1 x_1 \dots Q_k x_k \, \psi'(x_1, \dots, x_k)$ shows $f \in \#\text{Win-FO[BIT]}$. □

For the converse direction we begin by showing that in the definition of the classes #Win-FO[BIT] and #Win-FO[Arb], when ignoring inputs that are encodings of structures with singleton domains (domains of cardinality 1), we can assume prenex normal form and we can simultaneously ensure that for fixed assignments to the quantified variables, the verifier has at most one winning strategy for the quantifier-free part of the sentence in any structure and for any assignment.

Lemma 3.9. *Let φ be an* FO[Arb]*-formula over some vocabulary σ. There is an* FO[Arb]*-formula φ' over σ in prenex normal form such that for all $\mathbf{A} \in \text{STRUC}[\sigma]$ with $|\text{dom}(\mathbf{A})| > 1$, all non-uniform families of interpretations $\mathcal{I}$, and all assignments $s\colon \text{free}(\varphi) \to \text{dom}(\mathbf{A})$ the following hold:*

- $\#\text{Win}(\mathbf{A}, \mathcal{I}, \varphi', s) = \#\text{Win}(\mathbf{A}, \mathcal{I}, \varphi, s)$ *and*
- $\#\text{Win}(\mathbf{A}, \mathcal{I}, \psi, s) \in \{0, 1\}$, *where ψ is the quantifier-free part of φ'.*

Proof. Let σ be a vocabulary. We show inductively that the desired normal form exists for any FO[Arb]-formula φ over σ when we allow constants 0 and 1. It is easy to see that these constants can be replaced by additional existentially quantified variables.

If φ is an atom, $\varphi' := \varphi$ has the desired properties.

If $\varphi = Qx\,\varphi_1$ for some quantifier Q and a formula φ_1 with the desired properties, the formula $\varphi' := Qx\,\varphi_1'$ has the desired properties.

If $\varphi = \neg\varphi_1$ for some formula φ_1 with the desired properties, we move the negation inside the quantifiers in φ_1 changing any existential quantifier to a universal one and vice versa. Finally, the quantifier-free part of the resulting formula is modified to ensure

that the verifier has at most one winning strategy on this part, which yields a formula φ' with the desired properties.

If $\varphi = \varphi_1 \wedge \varphi_2$ for formulae φ_1 and φ_2 with the desired properties, we proceed as follows. Without loss of generality, $\text{bound}(\varphi_1) \cap \text{Vars}(\varphi_2) = \text{bound}(\varphi_2) \cap \text{Vars}(\varphi_1) = \emptyset$, that is, none of the variables quantified in φ_1 occur in φ_2 and vice versa. Let $\varphi_1 =: Q_1 x_1 \dots Q_k x_k \psi_1$ and $\varphi_2 =: O_1 y_1 \dots O_\ell y_\ell \psi_2$, where $Q_1, \dots, Q_k, O_1, \dots, O_\ell$ are quantifiers and ψ_1 and ψ_2 are quantifier-free. Now let

$$\begin{aligned}\varphi'' \coloneqq Q_1 x_1 \dots Q_k x_k O_1 y_1 \dots O_\ell y_\ell \Big(& (z = 0 \wedge \psi_1 \wedge y_1 = 0 \wedge \dots \wedge y_\ell = 0) \vee \\ & (z = 1 \wedge \psi_2 \wedge x_1 = 0 \wedge \dots \wedge x_k = 0) \vee \\ & (z > 1 \wedge x_1 = 0 \wedge \dots \wedge x_k = 0 \wedge \\ & \quad y_1 = 0 \wedge \dots \wedge y_\ell = 0) \Big),\end{aligned}$$

where z is a fresh variable. Intuitively, z acts as a selector such that no two of the terms in the disjunction can be satisfied simultaneously. The formula φ'' is in prenex normal form. Also, as for all $\mathbf{A}, \mathcal{I}, s$ it holds that $\#\text{Win}(\mathbf{A}, \mathcal{I}, \psi_1, s), \#\text{Win}(\mathbf{A}, \mathcal{I}, \psi_2, s) \in \{0, 1\}$, the same holds for the quantifier-free part of φ''.

Let $\mathbf{A} \in \text{STRUC}[\sigma]$ with $|\text{dom}(\mathbf{A})| > 1$, $\mathcal{I}$ be a non-uniform family of interpretations, and $s\colon \text{free}(\varphi) \to \text{dom}(\mathbf{A})$ be an assignment. In the following, we write $\#\text{Win}(\xi, t)$ instead of $\#\text{Win}(\mathbf{A}, \mathcal{I}, \xi, t)$ for any formula ξ and assignment t, as the structure and the family of interpretations do not change. It holds that $\#\text{Win}(\varphi'', s[z/0]) = \#\text{Win}(\varphi_1, s)$. Similarly, $\#\text{Win}(\varphi'', s[z/1]) = \#\text{Win}(\varphi_2, s)$ and $\#\text{Win}(\varphi'', s[z/i]) = 1$ for $i > 1$. Now the desired formula is $\varphi' \coloneqq \forall z \varphi''$, as

$$\begin{aligned}\#\text{Win}(\varphi', s) &= \#\text{Win}(\varphi'', s[z/0]) \cdot \#\text{Win}(\varphi'', s[z/1]) \cdot \prod_{i=3}^{\text{dom}(\mathbf{A})} \#\text{Win}(\varphi'', s[z/i]) \\ &= \#\text{Win}(\varphi'', s[z/0]) \cdot \#\text{Win}(\varphi'', s[z/1]) \\ &= \#\text{Win}(\varphi_1 \wedge \varphi_2, s).\end{aligned}$$

If $\varphi = \varphi_1 \vee \varphi_2$, we proceed analogously. Let $Q_1, \dots, Q_k, O_1, \dots, O_\ell$, ψ_1, and ψ_2 be defined as above. We define

$$\begin{aligned}\varphi'' \coloneqq Q_1 x_1 \dots Q_k x_k O_1 y_1 \dots O_\ell y_\ell \Big(& (z = 0 \wedge \psi_1 \wedge y_1 = 0 \wedge \dots \wedge y_\ell = 0) \\ & \vee (z = 1 \wedge \psi_2 \wedge x_1 = 0 \wedge \dots \wedge x_k = 0) \Big).\end{aligned}$$

Similar to before $\varphi' \coloneqq \exists z \varphi''$ has the desired properties. □

This immediately yields the corollary that ignoring inputs of length 1, the definition based on the full model-checking game and the definition based on Hintikka's model-checking game coincide.

Corollary 3.10. *Let* $f\colon \{0,1\}^+ \to \mathbb{N}$. *Define* $f^*\colon \{0,1\}^+ \to \mathbb{N}$ *by*

$$f^*(x) := \begin{cases} 1, & \text{if } |x| = 1 \text{ and } f(x) > 0, \\ f(x), & \text{otherwise.} \end{cases}$$

Then the following holds:

1. $\{f^* \mid f \in \#\text{Win-FO[BIT]}\} \subseteq \#\text{Win-FO[BIT]}^{\text{Hintikka}}$ *and*
2. $\{f^* \mid f \in \#\text{Win-FO[Arb]}\} \subseteq \#\text{Win-FO[Arb]}^{\text{Hintikka}}$.

Proof. We show statement 1. The non-uniform case can be proven analogously. Let $f \in \#\text{Win-FO[BIT]}$ via the σ-sentence φ. Without loss of generality we can assume that $\sigma = \tau_{\text{string}}$: If f was defined via some formula ξ over a vocabulary $\sigma \neq \tau_{\text{string}}$, we can use the FO-query bin_σ^{-1} (see Remark 2.28) and plug it into a formula defining f over an arbitrary vocabulary, obtaining a τ_{string}-sentence ξ'. This is possible by Lemma 3.4. As the behavior of bin_σ^{-1} is not specified for τ_{string}-structures that do not represent encodings of σ-structures, we further need to modify the behavior of ξ' on such inputs. This is possible as the property that the cardinality of the domain is the length of encodings of σ-structures is definable in FO[BIT].

Let φ' be the sentence arising from φ by applying Lemma 3.9 and let ψ be the quantifier-free part of φ'. Now, let $\mathbf{A} \in \text{STRUC}[\tau_{\text{string}}]$ with $|\text{dom}(\mathbf{A})| > 1$. It holds that $\#\text{Win}(\mathbf{A}, \psi, s) = \#\text{Win}^{\text{Hintikka}}(\mathbf{A}, \psi, s)$ for any assignment s, since $\#\text{Win}(\mathbf{A}, \theta, s') = 0$ if and only if $\#\text{Win}^{\text{Hintikka}}(\mathbf{A}, \theta, s') = 0$ for any structure $\mathbf{A}$, assignment s' and formula θ, and $\#\text{Win}(\mathbf{A}, \psi) \in \{0,1\}$. As quantifiers are treated the same in both model-checking games it follows that $\#\text{Win}(\mathbf{A}, \mathcal{I}, \varphi, s) = \#\text{Win}^{\text{Hintikka}}(\mathbf{A}, \mathcal{I}, \varphi, s)$. Strings 0 and 1 are the only encodings of τ_{string}-structures with domains of cardinality 1, so the statement is already true restricted to inputs of length > 1. The behavior of the formula in structures with domains of cardinality 1 can be adapted as required, finishing the proof. □

On the other hand it is easy to see that the classes are not the same, as for any formula φ, the verifier has at most one winning strategy in Hintikka's model checking game in structures with domains of cardinality 1.

Having identified this difference in models we now observe that the possible outputs of functions in our different classes as well as classes from circuit complexity differ.

Observation 3.11. *Any* FO*-formula can be modified in such a way that the number of winning strategies in the full model-checking game in structures with domains of cardinality* 1 *are any chosen natural numbers without changing the number of winning strategies on other inputs. In contrast, in Hintikka's model-checking game only values* 0 *and* 1 *can occur in structures with domains of cardinality* 1.

Any non-uniform family of counting arithmetic circuits can be modified in such a way that on the input strings 0 *and* 1*, any chosen natural numbers are computed within the same resource bounds. On the other hand,* FO[BIT]*-uniform families of counting arithmetic circuits always map the input strings* 0 *and* 1 *to either* 0 *or* 1*.*

For this reason, in order to exactly capture any counting classes from circuit complexity in terms of counting winning strategies for FO-formulae, the full model-checking game needs to be used for non-uniform circuit classes while Hintikka's model-checking game needs to be used for FO[BIT]-uniform circuit classes. In the remainder of this work we will ignore this subtle difference and say a class is captured when it is captured in the weaker sense that on inputs of length 1, functions can output 1 instead of an arbitrary number > 0, but they have to agree on whether the output is 0 or not. We make this precise in the following definition, also introducing a corresponding kind of inclusion.

Definition 3.12. Let $\mathfrak{C}$, $\mathfrak{D}$ be classes of counting problems. Then $\mathfrak{C} \overset{*}{\subseteq} \mathfrak{D}$ holds if for every $f \in \mathfrak{C}$, either $f \in \mathfrak{D}$ or $f^* \in \mathfrak{D}$, where f^* is defined from f as follows:

$$f^*(x) := \begin{cases} 1, & \text{if } |x| = 1 \text{ and } f(x) > 0 \\ f(x), & \text{otherwise.} \end{cases}$$

If $\mathfrak{C} \overset{*}{\subseteq} \mathfrak{D}$ and $\mathfrak{D} \overset{*}{\subseteq} \mathfrak{C}$, we say that $\mathfrak{C}$ captures $\mathfrak{D}$, denoted by $\mathfrak{C} \overset{*}{=} \mathfrak{D}$.

Using the notion of $\overset{*}{=}$, we can now state the connections between the different counting classes based on first-order logic in the following simplified form.

Corollary 3.13.

1. $\#\text{Win-FO[BIT]} \overset{*}{=} \#\text{Win-FO[BIT]}^{\text{Hintikka}} = \#\text{Skolem-FO[BIT]}$ *and*
2. $\#\text{Win-FO[Arb]} \overset{*}{=} \#\text{Win-FO[Arb]}^{\text{Hintikka}} = \#\text{Skolem-FO[Arb]}$.

3.3 Conclusion

The goal of this chapter was to define a class based on witness counting in first-order logic without introducing second-order variables. We first presented a suitable definition based on counting winning strategies of the verifier in the first-order model-checking game. For this, we used the full model-checking game that ends only after reaching an atom, resulting in the classes #Win-FO[BIT] and #Win-FO[Arb] in the uniform and non-uniform setting, respectively.

We then considered other promising candidates for such a counting class. First, apart from the full model-checking game, there is another game-theoretic characterization of first-order logic: In the original model-checking game of Hintikka, formulae are assumed to be in prenex normal form and the game ends once reaching the quantifier-free part. Secondly, we also considered Skolem functions as witnesses for acceptance in first-order logic. Both notions lead to the definition of corresponding counting classes.

We showed that the class #Win-FO admits a certain robustness by showing that both in the non-uniform as well as the uniform setting, all three classes coincide as long as we disregard inputs of length 1. Note that it is a usual phenomenon in the context of first-order logic that inputs of small size have to be treated differently. For example, recall that FO[BIT]-uniform circuit families generally do not contain a circuit for inputs of length 0. While U_D-uniform classes are capable of handling the empty word ε as input, we still consider both notions of uniformity equivalent on many circuit complexity classes, disregarding the empty word. Hence, it seems natural to also disregard inputs of length 1 in the case of counting classes, as for technical reasons these classes are not robust with respect such inputs.

We briefly observed that there are similar effects when using FO[BIT]-uniformity for counting classes in circuit complexity. Based on these observations and discussions, we argued that to precisely capture non-uniform circuit complexity classes, the definition based on the full model-checking game is suitable. On the other hand, for FO[BIT]-uniform classes the definition based on Hintikka's model-checking game or, equivalently, the definition based on Skolem functions, is better suited.

4 Characterizing Constant-Depth Classes

In this chapter we show that the class #Win-FO, based on counting witnesses in first-order logic and introduced in the previous chapter, captures the class $\#AC^0$ both in the non-uniform and in the FO[BIT]-uniform setting. Furthermore, we will use this result to establish a new characterization of the decision class TC^0 by extending FO with bitwise access to #Win-FO-functions.

4.1 A Model-Theoretic Characterization of $\#AC^0$

Before proving the main theorem we need the following normal form for the class $\#AC^0$.

Lemma 4.1. *Let $f \in \#AC^0$. Then there is some $k \in \mathbb{N}$ and polynomial-size circuit family $C = (\mathbf{C}_n)_{n\in\mathbb{N}}$ of depth k such that $\#C(w) = f(w)$ and for all $n \in \mathbb{N}$ the following holds:*

1. *the underlying undirected graph of $\mathbf{C}_n$ is a tree,*
2. *all input nodes have depth exactly k, that is, the length of all paths from an input node to the output node in $\mathbf{C}_n$ is k, and*
3. *types of gates in $\mathbf{C}_n$ are alternating on any path ending on the output gate and the output gate has type $\wedge$.*

The same normal form applies to FO[BIT]-uniform $\#AC^0$.

Proof. Let $\mathcal{D} = (\mathbf{D}_n)_{n\in\mathbb{N}}$ be a circuit family with $\#\mathcal{D}(x) = f(x)$ for all $x \in \{0,1\}^*$. We successively construct a circuit family with property 1, a circuit with properties 1 and 2 and a circuit with all three properties. Fix $n \in \mathbb{N}$ and let $\mathbf{D} \coloneqq \mathbf{D}_n$ and k be the depth of $\mathbf{D}$.

1.) This can be achieved by constructing a new circuit whose gates are k-tuples of gates of the $\mathbf{D}_n$ or a placeholder element. This way, we can connect gates in such a way that the path from the output gate to any gate is part of the encoding of gates. More precisely, we construct a new circuit family $C'' = (\mathbf{C}''_n)_{n\in\mathbb{N}}$ where the underlying undirected graph of each $\mathbf{C}''_n$ is a tree as follows. Let $n \in \mathbb{N}$. Let $\mathbf{D} =: (V_\mathbf{D}, E_\mathbf{D}, \beta_\mathbf{D}, \text{out}_\mathbf{D})$. Then we define $\mathbf{C}''_n$ as $(V'', E'', \beta'', \text{out}'')$ with the following components. $V'' \coloneqq \{(v_1, v_2, \ldots, v_k) \mid v_i \in V_\mathbf{D} \cup \{\$\}\}$, where $\$ \notin V_\mathbf{D}$ is a placeholder element. Ultimately, only gates of the form $(v_1, \ldots, v_j, \$, \ldots, \$)$ with $v_1 = \text{out}_\mathbf{D}$ are used in the construction, but it does not make a difference to keep all tuples in the set of gates. For a tuple of the above form, v_j determines what gate of the original circuit the tuple corresponds to, while $v_1, \ldots, v_{j-1}$ specify the path from that gate to the output gate. Let $g, h \in V''$ be two gates in the new circuit. Then $(g,h) \in E''$ if and only

if there are a $j \in \{1,\dots,k\}$ and $g_2,\dots,g_j,g_{j+1} \in V_{\mathbf{D}}$ such that $g = (\mathrm{out}_{\mathbf{D}}, g_2,\dots,g_j, \underbrace{\$,\dots,\$}_{k-j \text{ many}})$, $h = (\mathrm{out}_{\mathbf{D}}, g_2,\dots,g_{j+1}, \underbrace{\$,\dots,\$}_{k-j+1 \text{ many}})$, and $(g_j,g_{j+1}) \in E_{\mathbf{D}}$. The gate types are directly obtained from the gate types in $\mathbf{D}$, that is, $\beta''(g_1,\dots,g_j,\$,\dots,\$) \coloneqq \beta_{\mathbf{D}}(g_j)$ for any gate $(g_1,\dots,g_j,\$,\dots,\$) \in V''$. For gates that are not of this form the type can be chosen arbitrarily. The output gate of $\mathbf{C}''_n$ is the gate $\mathrm{out}'' \coloneqq (\mathrm{out}_{\mathbf{D}},\$,\dots,\$)$.

2.) We now construct a circuit C' from C'' that has property 2 in addition to property 1. For this, we can simply add a path of appropriate length only consisting of $\wedge$-gates immediately after any input gate of C''. We now construct $C' = (\mathbf{C}'_n)$ formally. Let d be the depth of C'', $n \in \mathbb{N}$, and let V_{in} be the set of input nodes of $\mathbf{C}''_n$. We will create d copies of each gate of $\mathbf{C}''_n$. Similar is in the proof to ensure the first property, this allows us to include an encoding of the distance to the output gate in the encoding of any gate. We define $\mathbf{C}'_n = (V',E',\beta',\mathrm{out}')$ as follows:

$$
\begin{aligned}
V' &\coloneqq \{v_0,v_1,\dots,v_d \mid v \in V''\},\\
E' &\coloneqq \{(v_{i+1},w_i) \mid (v,w) \in E''\} \cup \{(v_{i+1},v_i) \mid i < d\},\\
\beta'(v_i) &\coloneqq \begin{cases} \beta''(v), & \text{if } v \in V'' \setminus V''_{\mathrm{in}} \\ \beta''(v), & \text{if } v \in V''_{\mathrm{in}} \text{ and } i = d \\ \wedge, & \text{else,} \end{cases}\\
\mathrm{out}' &\coloneqq \mathrm{out}''_1.
\end{aligned}
$$

3.) We now construct the circuit family $C = (\mathbf{C}_n)_{n\in\mathbb{N}}$ with all properties from the statement of this lemma. Intuitively, this can be done by replacing each layer of C' by two layers, an $\wedge$-layer and an $\vee$-layer. This is done for each gate individually: For $\wedge$-gates the new $\vee$-layer will be a dummy-layer while for $\vee$-gates the new $\wedge$-layer will be a dummy-layer. Let $n \in \mathbb{N}$, $\mathbf{C}'_n =: (V',E',\beta',\mathrm{out}')$ and $m \coloneqq |V'|$. We construct $\mathbf{C}_n = (V,E,\beta,\mathrm{out})$ as follows. Let $V \coloneqq \{g_\wedge, g_{\vee,1},\dots,g_{\vee,m} \mid g \in V'\}$. For any internal gate $g \in V'$, we add to E the following edges depending on the type $\beta'(g)$. Let $g^1,\dots,g^\ell$ be the predecessors of g in $\mathbf{C}'_n$. If $\beta'(g) = \wedge$, we add to E the edges $(g_\wedge, g_{\vee,1}),\dots,(g_\wedge, g_{\vee,\ell})$ as well as $(g_{\vee,1}, g^1_\wedge),\dots,(g_{\vee,\ell}, g^\ell_\wedge)$. If $\beta(g) = \vee$, we add to E the edge $(g_\wedge, g_{\vee,1})$ as well as the edges $(g_{\vee,1}, g^1_\wedge),\dots,(g_{\vee,1}, g^\ell_\wedge)$. The construction is illustrated in Figure 4.1.

All of the constructions above are presented in a way that makes it obvious that they can be made FO[BIT]-uniform, since all conditions are FO[BIT]-definable. □

We now prove the non-uniform version of the main theorem of this chapter, i.e., the characterization of $\#\mathrm{AC}^0$ in terms of counting winning strategies in FO-formulae.

Theorem 4.2. $\#\mathrm{AC}^0 = \#\text{Win-FO[Arb]}$

Proof. First note that by Observation 3.11 we do not need to consider inputs of length ≤ 1.

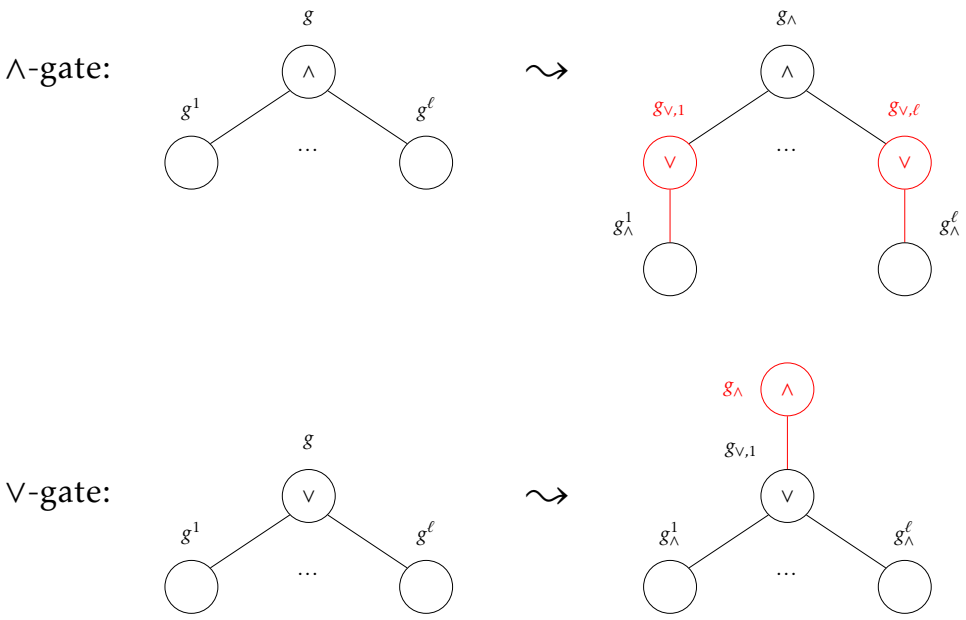

Figure 4.1: Construction of an alternating circuit.

#AC^0 ⊆ #Win-FO[Arb]: Let $f \in \#\mathrm{AC}^0$ via the circuit family $C = (\mathbf{C})_{n\in\mathbb{N}}$ in the normal form of Lemma 4.1 and let k the depth of all circuits in C. Let $w \in \{0,1\}^+$ be an input, out be the output gate of $\mathbf{C}_{|w|}$, and E the edge relation of $\mathbf{C}_{|w|}$. The value $f(w)$ can be written as follows:

$$f(w) = \prod_{\substack{y_1 \text{ with} \\ (y_1,\mathrm{out})\in E}} \sum_{\substack{y_2 \text{ with} \\ (y_2,y_1)\in E}} \cdots \bigwedge_{\substack{y_k \text{ with} \\ (y_k,y_{k-1})\in E}} \mathrm{val}(\mathbf{C}_{|w|}, y_k, w), \qquad (\star)$$

where $\bigwedge$ is a product if k is odd and a sum otherwise.

We will now build an FO-sentence φ over $\tau_{\mathrm{string}} \cup \tau_{\mathrm{circ}}$ such that for any input $w \in \{0,1\}^+$, $\#\mathrm{Win}(\mathbf{A}_w, C', \varphi)$ equals the number of proof trees of the circuit $\mathbf{C}_{|w|}$ on input w. Note that the circuit family C as a family of τ_{circ}-structures can almost directly be used as the non-uniform family of interpretations for the evaluation of φ: We can use the family $C' = (\mathbf{C}'_n)_{n\in\mathbb{N}}$ of interpretations of the symbols from τ_{circ}, which interprets the symbols in analogy to C, but using tuples of elements of the universe as gates. Let $\ell \in \mathbb{N}$ be the arity of these tuples. Now, the set of gates used in $\mathbf{C}'_{|w|}$ is for example the set $\{0,\ldots,|w|-1\}^\ell = \mathrm{dom}(\mathbf{A}_w)^\ell$. Note that this does not work for inputs of length ≤ 1, but these can be handled separately as mentioned before (cf. Observation 3.11). To simplify the presentation, we assume in the following that we do not need tuples—a single element of the universe corresponds to a gate. The proof can be generalized to the case where this assumption is dropped.

Before giving the desired sentence φ over $\tau_{\text{string}} \cup \tau_{\text{circ}}$, we define

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

which in $\mathbf{A}_w$ and using C' to interpret symbols from τ_{circ} expresses that x is an input gate with $\text{val}(\mathbf{C}_{|w|}, x, w) = 1$. Now φ can be given as follows:

$$\varphi := \exists y_0 \forall y_1 \exists y_2 \dots Q_k y_k \left(\mathsf{out}(y_0) \wedge \left(\left(\bigwedge_{1 \le i \le k} \mathsf{E}(y_i, y_{i-1}) \wedge \varphi_{\text{trueLiteral}}(y_k) \right) \vee \bigvee_{\substack{1 \le i \le k, \\ i \text{ odd}}} \left(\left(\bigwedge_{1 \le j < i} \mathsf{E}(y_j, y_{j-1}) \right) \wedge \neg \mathsf{E}(y_i, y_{i-1}) \wedge \bigwedge_{i < j \le k} \mathsf{out}(y_j) \right) \right) \right),$$

where Q_k is an existential quantifier if k is odd, and a universal quantifier otherwise. Notice that for the quantifier-free part of φ, the verifier has at most one winning strategy in any structure and for any assignment to the variables y_i. For this reason, we do not need to argue about the number of winning strategies for the quantifier-free part throughout the proof, but can simply distinguish between this part being satisfied or not.

We will now analyze the number of winning strategies arising from the quantifier prefix of φ. The big disjunction ensures that the number of winning strategies after making a wrong choice for one of the quantified variables (choosing an element that is either not a gate or not a predecessor of the previous gate) is always the neutral element of the arithmetic operation associated with the corresponding quantifier: For existential quantifiers, we sum over all possibilities. Thus, a wrong choice should lead to no winning strategies. For universal quantifiers, we multiply the number of winning strategies for all choices. In this case, any wrong choice should lead to exactly one winning strategy.

We now need to show that the number of winning strategies for $\mathbf{A}_w \models_{C'} \varphi$ is equal to the number of proof trees of the circuit $\mathbf{C}_{|w|}$ on input w. For this, let

$$\varphi^{(n)}(y_1, \dots, y_n) := Q_{n+1} y_{n+1} \dots Q_k y_k \left(\left(\bigwedge_{n+1 \le i \le k} \mathsf{E}(y_i, y_{i-1}) \wedge \varphi_{\text{trueLiteral}}(y_k) \right) \vee \bigvee_{\substack{n+1 \le i \le k, \\ i \text{ odd}}} \left(\left(\bigwedge_{1 \le j < i} \mathsf{E}(y_j, y_{j-1}) \right) \wedge \neg \mathsf{E}(y_i, y_{i-1}) \wedge \bigwedge_{i < j \le k} \mathsf{out}(y_j) \right) \right),$$

where $Q_{n+1}, \dots, Q_{k-1}$ are the quantifiers preceding Q_k. Note that the lower bound of the index i on the big disjunction changed compared to φ. In the following we will use the

abbreviation

$$\#w(\psi(\overline{g})) \coloneqq \#\mathrm{Win}(\mathbf{A}_w, C', \psi(\overline{g}))$$

for tuples $\overline{g}$ of elements of $\mathrm{dom}(\mathbf{A}_w)$ of appropriate arity. Let $\bigwedge_n$ be a product if n is odd and a sum otherwise for any $1 \leq n \leq k$. We now show by induction that for all $n \in \{0, \dots, k\}$

$$\#w(\varphi) = \prod_{\substack{g_1 \text{ with} \\ (g_1,r)\in E}} \sum_{\substack{g_2 \text{ with} \\ (g_2,g_1)\in E}} \cdots \bigwedge_n \underset{\substack{g_n \text{ with} \\ (g_n,g_{n-1})\in E}}{} \#w(\varphi^{(n)}[y_0/\mathsf{out}](g_1, \dots, g_n)).$$

Here, out is used as a constant for the output gate of $C_{|w|}$. While this constant formally does not exist it behaves exactly the same as the variable y_0 in our formula and makes the presentation more succinct and simple in the following.

Induction basis ($n = 0$): The induction hypothesis here simply states

$$\#w(\varphi) = \#w(\varphi^{(0)}[y_0/\mathsf{out}]),$$

which holds by definition.

Induction step ($n \to n+1$): We can directly use the induction hypothesis here:

$$\#w(\varphi) = \prod_{\substack{g_1 \text{ with} \\ (g_1,r)\in E}} \sum_{\substack{g_2 \text{ with} \\ (g_2,g_1)\in E}} \cdots \bigwedge_n \underset{\substack{g_n \text{ with} \\ (g_n,g_{n-1})\in E}}{} \#w(\varphi^{(n)}[y_0/\mathsf{out}](g_1, \dots, g_n)),$$

so it remains to show that for all $g_1, \dots, g_n \in \mathrm{dom}(\mathbf{A})$:

$$\#w(\varphi^{(n)}[y_0/\mathsf{out}](g_1, \dots, g_n)) = \bigwedge_{n+1} \underset{\substack{g_{n+1} \text{ with} \\ (g_n,g_{n+1})\in E}}{} \#w(\varphi^{(n+1)}[y_0/\mathsf{out}](g_1, \dots, g_n, g_{n+1})).$$

We distinguish two cases: Depending on whether $n+1$ is even or odd, the $(n+1)$-st quantifier is either an existential or a universal quantifier. Similarly, all gates of that depth in the circuits from C are either disjunction- or conjunction-gates. As both existential quantifiers and disjunction-gates lead to a sum in the corresponding counting function, and both universal quantifiers and conjunction-gates lead to a product, this as desired.

$n+1$ is odd: This means, the $(n+1)$-st quantifier is a universal quantifier. Thus, from $\#w(\varphi^{(n)}[y_0/\mathsf{out}](\overline{g}))$ we get a product as the outermost operation. We now need to check which assignments g_{n+1} for y_{n+1} are relevant to the product: The big conjunction may only be true if g_{n+1} is a predecessor of the element g_n assigned to y_n. The big disjunction may become true for assignments to g_{n+1} which are no predecessors of g_n only if all variables quantified after g_{n+1} are set to out (we could have chosen any fixed element here that is definable from $\mathbf{A}_w$ in FO[Arb]). This ensures that after choosing a wrong element g_{n+1}, the verifier always has a unique winning strategy. Also, the term for $i = n$ in the big disjunction can only be made true if g_{n+1} is not a predecessor of g_n, so

we can omit it if g_{n+1} is a predecessor of g_n. Since for all elements g_{n+1} that are not predecessors of g_n we fix assignments to all variables quantified afterwards, we get for all $g_1, \dots, g_n \in \mathrm{dom}(\mathbf{A})$:

$$\begin{aligned}
\#w(\varphi^{(n)}[y_0/\mathsf{out}](g_1,\dots,g_n)) &= \prod_{g_{n+1}\in|\mathbf{A}_w|} \#w(\varphi^{(n+1)}[y_0/\mathsf{out}](g_1,\dots,g_{n+1})) \\
&= \prod_{\substack{g_{n+1}\in|\mathbf{A}_w|,\\(g_{n+1},g_n)\in E}} \#w(\varphi^{(n+1)}[y_0/\mathsf{out}](\overline{g})) \cdot \prod_{\substack{g_{n+1}\in|\mathbf{A}_w|,\\(g_{n+1},g_n)\in E}} 1 \\
&= \prod_{\substack{g_{n+1}\in|\mathbf{A}_w|,\\(g_{n+1},g_n)\in E}} \#w(\varphi^{(n+1)}[y_0/\mathsf{out}](\overline{g})).
\end{aligned}$$

Therefore, we get a product only over the predecessors of g_n and the term for $i = n$ in the big disjunction as well as the clause for $i = n$ in the big conjunction are not needed in $\varphi^{(n+1)}$.

***n*+1 is even:** This means, the $(n+1)$-st quantifier is an existential quantifier. Therefore, we get a sum as the outermost operation of $\#w(\varphi^{(n)}[y_0/\mathsf{out}](\overline{g}))$. We now need to check which assignments g_{n+1} for y_{n+1} are relevant for the sum: The big conjunction can only be true if g_{n+1} is a predecessor of the element g_n assigned to y_n. The big disjunction can also only be true if g_{n+1} is a predecessor of g_n. This means that the verifier does not have any winning strategies after choosing gates that are not predecessors of the current gate. Thus, we directly get

$$\#w(\varphi^{(n)}[y_0/\mathsf{out}](g_1,\dots,g_n)) = \sum_{\substack{g_{n+1}\in|\mathbf{A}_w|,\\(g_{n+1},g_n)\in E}} \#w(\varphi^{(n+1)}[y_0/\mathsf{out}](g_1,\dots,g_{n+1})).$$

Here, all terms of the big disjunction in $\varphi^{(n)}$ are still necessary in $\varphi^{(n+1)}$. The big conjunction does neither contain a clause for $i = n$ in $\varphi^{(n)}$ nor in $\varphi^{(n+1)}$, because n is even. This concludes the induction.

For $\varphi^{(k)}(g_1,\dots,g_k) = \varphi_{\mathrm{trueLiteral}}(g_k)$, we get

$$\#w(\varphi^{(k)}(g_1,\dots,g_k)) = \#\mathrm{Win}(\mathbf{A}_w, C', \varphi^{(k)}(g_1,\dots,g_k)) = \mathrm{val}(\mathbf{C}'_{|w|}, g_k, w),$$

which is 1 if g_k is associated with a true literal in $\mathbf{C}'_{|w|}$ on input w, and 0 otherwise. Together with the fact that C and C' are the same modulo renaming of gates as well as Equation $(\star)$ on page 71, this shows $f(w) = \#w(\varphi)$.

#Win-FO[Arb] $\subseteq$ #AC^0: We will show this by showing #Win-FO[Arb]$^{\text{Hintikka}} \subseteq$ #AC^0. By Corollary 3.13 and Observation 3.11, it then follows that #Win-FO[Arb] $\subseteq$ #AC^0. Let $f \in$ #Win-FO[Arb]$^{\text{Hintikka}}$ via the FO[Arb]-sentence φ over some vocabulary σ and the non-uniform family of interpretations $\mathcal{I} = (\mathbf{I}_n)_{n\in\mathbb{N}}$. Let τ be the vocabulary of built-in predicate symbols in φ and let k be the length of the quantifier prefix of φ. We now describe how to construct the circuits $\mathbf{C}_n$ within a circuit family $C = (\mathbf{C}_n)_{n\in\mathbb{N}}$ that shows $f \in$ #AC^0.

Let $n \in \mathbb{N}$. If n is not the length of the encoding of some σ-structure, $\mathbf{C}_n$ can simply compute the constant-0 function. Otherwise, let $m \in \mathbb{N}$ be such that $n = |\mathrm{enc}_\sigma(\mathbf{A})|$ for all $\mathbf{A} \in \mathrm{STRUC}[\sigma]$ with $|\mathrm{dom}(\mathbf{A})| = m$. The circuit $\mathbf{C}_n$ consists of two parts. The first part mimics the quantifiers of the formula, while the second part consists of copies of a circuit of constant size evaluating the quantifier-free part of the formula.

The first part is built in analogy to the circuit in Immerman's proof of the inclusion FO[BIT] $\subseteq$ FO[BIT]-uniform AC^0 [Imm99]. The gates of $\mathbf{C}_n$ are of the form $(a_1, \ldots, a_i)$ with $1 \leq i \leq k$ and $a_j \in \{0, \ldots, m-1\}$ for all j. Each such gate is intended to compute the number of winning strategies after the elements $a_1, \ldots, a_i$ are assigned to the first i quantified variables. Therefore, for any $1 \leq i \leq k$ and $a_1, \ldots, a_{i-1} \in \{0, \ldots, m-1\}$, the tuple $(a_1, \ldots, a_{i-1})$ is a conjunction-gate if the i-th quantifier in φ is universal, and a disjunction-gate if the i-th quantifier is existential. Also, if $i \leq k$, there is an edge from the gate $(a_1, \ldots, a_{i-1}, a_i)$ to $(a_1, \ldots, a_{i-1})$ for all $a_i \in \{0, \ldots, m-1\}$. This means that the name of each gate in depth k contains the elements assigned to all quantified variables.

The second part, which evaluates the quantifier-free part of φ based on the elements assigned to the quantified variables, works as follows: As in the proof of Lemma 3.3, we can construct a quantifier-free formula equivalent to the quantifier-free part of φ on which the verifier always has at most one winning strategy in the full model-checking game. We can now directly build a circuit evaluating this formula. As the verifier has at most one winning strategy on the quantifier-free part, this circuit is satisfied by an input if and only if it has exactly one proof tree with that input. This means that when counting proof trees, this second part will only determine the truth value of the quantifier-free part and not increase the number of proof trees. The truth value of the atoms can be determined in the circuit as follows: For predicates of the form $\mathsf{R}(\overline{x})$ for a relation symbol R from σ, we use an input gate associated with the corresponding input bit. What bit has to be accessed is determined by enc_σ. All other predicates are numerical predicates and thus only dependent on n. Therefore, their value can be computed by constants in $\mathbf{C}_n$.

Now, by Lemma 3.2 it is clear that for any $n \in \mathbb{N}$ counting proof trees of $\mathbf{C}_n$ on input $w \in \{0,1\}^n$ leads to the same number as counting winning strategies of the verifier for $\mathrm{enc}_\sigma^{-1}(w) \models_{\mathcal{I}} \varphi$, if w is the encoding of a σ-structure. If w is not the encoding of a σ-structure, $\mathbf{C}_n$ correctly computes the number 0. □

Next we want to transfer this result to the uniform setting. Here, we only show that counting the number of winning strategies in the full model-checking game captures $\#\mathrm{AC}^0$ in the weaker sense of $\overset{*}{=}$. Recall that this means that on inputs of length ≤ 1, functions can output 1 instead of an arbitrary number > 0, but they have to agree on whether the output is 0, as defined in the explanation preceding Corollary 3.13. For the inclusion of the circuit class in the logical class, we will use Lemma 3.4 to replace queries to $\mathbf{C}_{|w|}$ in the FO-sentence constructed in the proof of the non-uniform version above by the corresponding formulae from the FO[BIT]-query showing uniformity of C. For the inclusion of the logical class in the circuit class, we will have to show that the constructed circuit is uniform, which is straightforward.

Theorem 4.3. FO[BIT]-uniform $\#AC^0 \stackrel{*}{=} \#$Win-FO[BIT].

Proof. **FO[BIT]-uniform $\#AC^0 \subseteq$ #Win-FO[BIT]:** Let $f \in$ FO[BIT]-uniform $\#AC^0$ via the circuit family $C = (\mathbf{C}_n)_{n\in\mathbb{N}}$ and the FO[BIT]-query $\mathcal{I}$ showing its uniformity. With the formula φ from the proof of $\#AC^0 \subseteq$ #Win-FO[Arb] this means we have for all w:

$$\begin{aligned} f(w) &= \#\mathbf{C}_{|w|}(w) \\ &= \#\mathrm{Win}(\mathbf{A}' \cup \mathbf{C}_{|w|}, \varphi). \end{aligned}$$

Here, $\mathbf{A}' \cup \mathbf{C}_{|w|}$ is the structure $\mathbf{C}_{|w|}$—whose gates are tuples over the universe of $\mathbf{A}_w$—modified with additional access to the structure $\mathbf{A}_w$ on the adequate subset of the universe. Formally, $\mathbf{A}'$ is obtained from $\mathbf{A}_w$ by changing its universe to $\mathrm{dom}(\mathbf{A}_w)^k$ and adapting the interpretation of S, that is, the i-th element of $\mathrm{dom}(\mathbf{A})^k$ is in $\mathsf{S}^{\mathbf{A}'}$ if and only if the i-th element of $\mathrm{dom}(\mathbf{A}_w)$ is in $\mathsf{S}^{\mathbf{A}_w}$. Let $\mathcal{I}'$ be an FO-query with $\mathcal{I}'(\mathbf{A}_w) = \mathbf{A}' \cup \mathbf{C}_{|w|}$ for all w. This can easily be constructed from $\mathcal{I}$. Furthermore, we can assume that the formula φ_0 in $\mathcal{I}'$ defines a simple upper bound, so it can be extended to a query $\mathcal{I}''$ that also defines BIT. Now, from φ and $\mathcal{I}''$, we get by Lemma 3.4 an FO[BIT]-formula φ' over τ_{string}. As $\mathcal{I}''(\mathbf{A}_w) = \mathbf{A}' \cup \mathbf{C}_{|w|} \cup (\mathrm{BIT}_{\mathbf{C}_{|w|}})$, where $\mathrm{BIT}_{\mathbf{C}_{|w|}}$ denotes the restriction of BIT to the universe of $\mathbf{C}_{|w|}$, we have for all $\mathbf{A}_w \in \mathrm{STRUC}[\tau_{\mathrm{string}}]$:

$$\begin{aligned} \#\mathrm{Win}(\mathbf{A}_w, \varphi') &= \#\mathrm{Win}(\mathcal{I}''(\mathbf{A}_w), \varphi) \\ &= f(w). \end{aligned}$$

#Win-FO[BIT] $\stackrel{*}{\subseteq}$ FO[BIT]-uniform $\#AC^0$: The proof is similar to our proof for $\#$Win-FO[Arb] $\subseteq \#AC^0$. Obviously, the fact that gates can only be encoded as tuples on inputs of length > 1 comes into play here, which is why the two classes only coincide in the weaker sense of $\stackrel{*}{=}$. For inputs of length > 1, the only difference is that we need to show FO[BIT]-uniformity of the circuit. Let $f \in \#$Win-FO[BIT] via the formula φ. We now need formulae φ_0, $\varphi_{\mathsf{G}_\wedge}$, $\varphi_{\mathsf{G}_\vee}$, φ_{E}, φ_{Input}, $\varphi_{\mathsf{negatedInput}}$ and φ_{out} defining a circuit family showing $f \in$ FO[BIT]-uniform $\#AC^0$.

The second part of the circuit from the proof of the non-uniform version is fixed except for the input gates and constants, which depend on the input size. The input gates are defined by giving the index of the input bit they are associated with. For each input gate the index is determined by the assignment to the quantified variables in accordance with the definition of enc_σ. For this reason, the adequate index is definable in FO[BIT] from the tuple of variables assigned to the quantified variables. As we will see in the construction of the first part of the circuit, we will have direct access to this tuple: Each choice made for a quantified variable on the (reversed) path leading from the output gate to an input gate is part of the encoding of said input gate. Constants can be replaced by a gate with x_0 and $\neg x_0$ as its two inputs. Now we just need to uniformly make this gate an $\wedge$-gate for the constant 0 and an $\vee$-gate for the constant 1. This is possible as constants arise from either constants or occurrences of BIT in φ. In both cases, their truth value is definable in FO[BIT].

Now it remains to show, that the first part of the circuit can be made uniform as well. For this, start with an FO[BIT]-query mapping any τ_{string}-structure $\mathbf{A}_w$ to $\mathbf{A}_{0w}$

(the 0 is appended as the new MSB). Together with the fact that the composition of FO[BIT]-queries is still an FO-query [Imm99], it now suffices to construct an FO-query

$$\begin{aligned}\mathcal{I}\colon \mathrm{STRUC}[\tau_{\mathrm{string}}] &\to \mathrm{STRUC}[\tau_{\mathrm{circ}}]\\ \mathbf{A}_{0w} &\mapsto \mathbf{C}_{|w|}.\end{aligned}$$

Let $\varphi =: Q_1 z_1 \dots Q_k z_k \psi(z_1,\dots,z_k)$ with quantifier-free ψ. Then each gate in the circuit can be represented by a sequence of k values in the range $\{0,\dots,n\}$: The root is the sequence consisting of n in every component. For other nodes, each position in the sequence chooses a predecessor in one level of the circuit. For inner nodes we leave a suffix of a tuple set to n, meaning that no predecessors were chosen on those levels. As we can see, having the additional element n as a padding element is quite convenient.

Following the idea above, the formulae in the FO[BIT]-query have to express:

- A tuple $(x_1,\dots,x_k)$ is a gate, if and only if $x_i = n$ implies $x_{i+1} = \dots = x_k = n$ for all i.
- There is an edge from $\overline{x}$ to $\overline{y}$, if and only if in $\overline{x}$ exactly one more component is n than in $\overline{y}$ and both tuples agree on all components that are $< n$ in both of them.
- A gate is an $\wedge$-gate (resp. $\vee$-gate), if and only if its distance to the output gate translates to a $\forall$-quantifier (resp. $\exists$-quantifier) in φ.
- The output gate is the tuple $(n,\dots,n)$.

In detail, the formulae defining $\mathcal{I}$ can be given as follows, with $\overline{x} = (x_1,\dots,x_k)$ and $\overline{y} = (y_1,\dots,y_k)$:

$$\begin{aligned}
\varphi_0(\overline{x}) &:= \bigwedge_{1\le i\le k}\left(x_i = n \to \bigwedge_{i\le j\le k} x_j = n\right),\\
\varphi_{\mathsf{E}}(\overline{x},\overline{y}) &:= \bigwedge_{1\le i\le k} (x_i \neq n \to x_i = y_i) \wedge\\
&\qquad \left((y_1 \neq n \wedge y_2 = n \wedge x_1 = n) \vee\right.\\
&\qquad\quad \left.\bigvee_{2\le i\le k} \left(y_i \neq n \wedge y_{i+1} = n \wedge x_i = n \wedge x_{i-1} \neq n\right)\right),\\
\varphi_{\mathsf{G}_\wedge}(\overline{x}) &:= \bigvee_{\substack{0\le i<k\\ Q_{i+1}=\forall}} \left(\bigwedge_{1\le j\le i} x_i \neq n \wedge \bigwedge_{i+1\le j\le k} x_i = n\right),\\
\varphi_{\mathsf{G}_\vee}(\overline{x}) &:= \bigvee_{\substack{0\le i<k\\ Q_{i+1}=\exists}} \left(\bigwedge_{1\le j\le i} x_i \neq n \wedge \bigwedge_{i+1\le j\le k} x_i = n\right), \text{ and}\\
\varphi_{\mathsf{out}}(\overline{x}) &:= x_1 = n.
\end{aligned}$$

□

By Observation 3.11 and Corollary 3.13, we immediately obtain the following Corollary.

Corollary 4.4. FO[BIT]-uniform $\#AC^0$ = #Skolem-FO[BIT] = #Win-FO[BIT]^{Hintikka}.

4.2 A Model-Theoretic Characterization of TC^0

Next, we will see that the main result of the previous section, $\#AC^0$ = #Win-FO, can be used to also obtain a new characterization of the class TC^0. While there already are model-theoretic characterizations of this class, a new characterization can still give new insights.

Recall that PAC^0 is the class of languages L for which membership of an input x in L can be characterized by the condition $f(x) > g(x)$ for functions $f, g \in \#AC^0$. Also, TC^0 coincides with the class PAC^0 both in the non-uniform [AAD00] and the uniform [ABL98] setting. This leads to the idea to consider Boolean circuits with (bitwise) oracle access to counting functions. As comparing two numbers using bitwise access can be done in AC^0, we will certainly get that PAC^0 is contained in the class of languages accepted by AC^0-circuit families with oracle access to $\#AC^0$-functions. Conversely, as $\#AC^0 \subseteq FTC^0$ and $AC^0 \in TC^0$, one can show that this new class is contained in TC^0, showing that both classes coincide. A model-theoretic characterization of TC^0 is then obtained by combining the characterizations of AC^0 and $\#AC^0$, defining a variant of first-order logic with (bitwise) oracle access to counting functions, namely functions from #Win-FO. We will see that this idea works in both the non-uniform and the uniform setting.

We start by introducing the required notions of oracle circuits and first-order formulae with oracle access to counting functions. Our model of oracle circuits adds a new type of gate for bitwise oracle access to a function oracle, called #-gate. The bits such a gate receives as input are interpreted as an input string to the oracle function, while the uniformity specifies which bit of the output the gate computes. This also means that in general, these gates do not compute commutative functions. Instead of adding the function α as defined in the general definition of Boolean circuits (see Definition 2.2 and the subsequent definition of the language accepted by such a circuit), we only add a numbering of the input bits to #-gates by adding a new ternary predicate that specifies for a given tuple (g, g', i) whether g is the i-th predecessor of g'. We give a formal definition by defining a first-order vocabulary for oracle circuits:

$$\tau_{\text{o-circ}} := (\mathsf{E}^2, \mathsf{G}^1_\wedge, \mathsf{G}^1_\vee, \mathsf{G}^2_\#, \mathsf{Index}^3, \mathsf{Input}^2, \mathsf{negatedInput}^2, \mathsf{out}^1).$$

The intended meaning of the predicate symbols it shares with τ_{circ} stays the same. The meaning of the new predicates is as follows:

- $\mathsf{G}_\#(x, i)$: gate x is an oracle gate accessing the i-th bit of the oracle function and
- $\mathsf{Index}(x, y, i)$: y is an oracle gate and x is the i-th predecessor of y.

Let $\mathbf{C} \in \text{STRUC}[\tau_{\text{o-circ}}]$ be an oracle circuit and let f be a counting function. Further, let $g, i \in \text{dom}(\mathbf{C})$ be a gate and an index (given as an element of the universe) such that

$(g,i) \in \mathsf{G}_{\#}^{\mathbf{C}}$. Then, in the circuit $\mathbf{C}$ with oracle f, the gate g is an oracle gate computing the function

$$\begin{aligned} f_i \colon \{0,1\}^+ &\to \{0,1\} \\ x &\mapsto \mathrm{BIT}(i, f(x)). \end{aligned}$$

Similar as for usual Boolean circuits without oracle, we denote by $\mathbf{C}^f$ the function computed by $\mathbf{C}$ when using the function oracle f. Analogously, in case of a family of oracle circuits C, we denote by C^f the function computed by C when using function oracle f. Furthermore, we use the standard notation for oracle classes: Let $\mathfrak{C}$ be a class of Boolean circuits, A the class of languages decidable by circuits from $\mathfrak{C}$ and B a class of functions $\{0,1\}^+ \to \mathbb{N}$. Then A^B is the class of languages decidable by circuits from $\mathfrak{C}$ with oracle gates, using oracles from B.

As in our proofs we will also need TC^0-circuit families with oracle access to a function oracle, we want to briefly mention that in this case we extend $\tau_{\text{maj-circ}}$ analogously to how we extended τ_{circ} to obtain $\tau_{\text{o-circ}}$. This leads to the vocabulary

$$\tau_{\text{o-maj-circ}} \coloneqq (\mathsf{E}^2, \mathsf{G}_{\wedge}^1, \mathsf{G}_{\vee}^1, \mathsf{G}_{\text{MAJ}}^1, \mathsf{G}_{\#}^2, \mathsf{Index}^3, \mathsf{Input}^2, \mathsf{negatedInput}^2, \mathsf{out}^1),$$

where the intended meaning of the predicates is analogous to their meaning in $\tau_{\text{maj-circ}}$ and $\tau_{\text{o-circ}}$, respectively.

Next, we will introduce an extension of first-order logic that allows for bitwise oracle access to a counting function. We use a two-sorted logic based on the two-sorted logic $\mathrm{FO(Cnt)}_{\mathsf{All}}$ defined by Libkin in Definition 8.1 in [Lib04, p. 142]. In two-sorted first-order logic, structures have two universes (or *sorts*) and can contain relations over either of the universes. One could also allow relations that intertwine both universes, but we do not allow this for our logic. Variables are then either first-sort or second-sort variables and can only be used in relations over the correct sort. The reason we use a two-sorted logic is that we need to specify which bit of the output of the oracle we are interested in, and to do so without requiring built-in predicates, a second sort of numerical values is needed. This means that our definition could also be used for other classes with oracle access to function oracles, in particular independent of built-in predicates.

Access to the oracle function is then given by extending the logic by predicates of the form $\#_\varphi$ for an FO-formula φ. For a two-sorted structure $\mathbf{A}$ and an assignment s for second-sort variables, $\mathbf{A}, s \models \#_\varphi(\bar{j})$ if and only if the $s(\bar{j})$-th bit of $\#\mathrm{Win}(\mathbf{A}, \varphi)$ is 1 (under a suitable encoding of numbers as tuples over the second sort). We call the resulting logic FOCW (for **f**irst-**o**rder with **c**ounting **w**inning strategies) and give a formal definition of this logic and its semantics next.

Let $\sigma = (\mathsf{R}_1, \ldots, \mathsf{R}_k)$ be a relational FO-vocabulary. Beside the set of first-order variables $\mathrm{Var}_{\mathrm{FO}}$, we also need the countable set $\mathrm{Var}[2]_{\mathrm{FO}} \coloneqq \{i_\ell \mid \ell \in \mathbb{N}\}$, the set of second-sort variables. The logic FOCW extends the syntax of FO as follows: A first-sort σ-term is a σ-term as defined for first-order logic. A second-sort σ-term is a second-sort variable $i \in \mathrm{Var}[2]_{\mathrm{FO}}$. The grammar for the construction of FOCW over σ is the extension of the grammar for the construction of FO-formulae by the rules

$$\varphi ::= i = i' \mid +(i_1, i_2, i_3) \mid \times(i_1, i_2, i_3) \mid \exists i\, \varphi(x, i) \mid \#_\psi(\bar{i}),$$

where $i, i', i_1, i_2, i_3 \in \mathrm{Var}[2]_{\mathrm{FO}}$, $\bar{i}$ is a tuple of second-sort variables, and ψ is a first-order sentence over σ.

An FOCW-structure $\mathbf{A}$ over σ is a tuple $(A, \{0,\dots,|A|-1\}; \mathsf{R}_1^{\mathbf{A}},\dots,\mathsf{R}_k^{\mathbf{A}}; +^{\mathbf{A}}, \times^{\mathbf{A}})$, where $(A; \mathsf{R}_1^{\mathbf{A}},\dots,\mathsf{R}_k^{\mathbf{A}}) \in \mathrm{STRUC}[\sigma]$ and $+$ and $\times$ are interpreted as the ternary relations corresponding to addition and multiplication in $\mathbb{N}$ restricted to $\{0,\dots,|A|-1\}$. We define the encoding of an FOCW-structure $\mathbf{A}$ defined as above simply as the encoding of the underlying first-sort structure $\mathbf{A}' := (A; \mathsf{R}_1^{\mathbf{A}},\dots,\mathsf{R}_k^{\mathbf{A}})$, i.e., $\mathrm{enc}_\sigma(\mathbf{A}) := \mathrm{enc}_\sigma(\mathbf{A}')$, as the second-sort domain and the interpretations of the second-sort relation symbols are uniquely determined by $|\mathrm{dom}(\mathbf{A})|$ (similar to built-in predicates).

The semantics of FOCW is clear except for the semantics of $\#_\psi(\bar{i})$. Let $\mathbf{A}$ be an FOCW-structure and $\bar{i}_0$ an assignment to $\bar{i}$. Then

$$\mathbf{A} \models \#_\psi(\bar{i}_0), \quad \text{if the } \mathrm{val}(\bar{i}_0)\text{-th bit of } \#\mathrm{Win}(\mathbf{A}, \varphi) \text{ is } 1.$$

Here, $\mathrm{val}(\bar{i}_0)$ denotes the numeric value of the vector $\bar{i}_0$ under an appropriate encoding of the natural numbers as tuples of elements from the second sort. To be more precise, the tuple $(b_k,\dots,b_0) \in \{0,\dots,n-1\}^k$ encodes the number $\sum_{\ell=0}^{k} n^\ell \cdot b_\ell$. Furthermore, for an n-bit number the 0-th bit is its LSB while the $n-1$-th bit is its MSB, so the bits are numbered from lowest to highest significance. This is in line with how p-adic numbers are written in general for any prime number p.

Our logic FOCW thus extends FO with bitwise access to the number of winning strategies of the verifier in the model-checking game played on arbitrary FO-sentences, i.e., in FOCW we have bitwise access to counting in an exponential range. Libkin's logic $\mathrm{FO(Cnt)}_{\mathsf{All}}$ can count in the range of input positions, i.e., in a linear range. Nevertheless we will show that both logics are equally expressive on finite structures: both correspond to the circuit class TC^0.

Note that we could also define this logic in a way that uses a second-sort universe of exponential size to give direct access to the result of #Win-FO-functions instead of only bitwise access. As FTC^0 can compute (polynomial) sums and products, this class would potentially still capture TC^0. Still, it seems more natural to use a domain of cardinality equal to that of the first-sort domain.

In order to capture TC^0 both in the non-uniform as well as the FO[BIT]-uniform setting, we use FOCW with built-in predicates. These are added in the same way as for FO, giving access to numerical predicates on the first sort both in the formulae themselves as well as in the subscripts of predicates #. The classes we use will be FOCW[Arb] and FOCW[BIT], which are defined accordingly.

The main result of this section can be stated as follows:

Theorem 4.5. $\mathrm{TC}^0 = \mathrm{FOCW} = \mathrm{AC}^{0^{\#\mathrm{AC}^0}}$ *both in the non-uniform and the* FO[BIT]*-uniform setting.*

For the proof of this characterization, we need a bit of additional preparation. We will show this result by the chain of inclusions $\mathrm{TC}^0 \subseteq \mathrm{FOCW} \subseteq \mathrm{AC}^{0^{\#\mathrm{AC}^0}} \subseteq \mathrm{TC}^0$. The rough proof idea for $\mathrm{AC}^{0^{\#\mathrm{AC}^0}} \subseteq \mathrm{TC}^0$ is as follows: Let $L \in \mathrm{AC}^{0^{\#\mathrm{AC}^0}}$ via the AC^0 circuit

family C with oracle access to $f \in \#\mathrm{AC}^0$. We use the fact that AC^0-circuits are also TC^0-circuits and $\#\mathrm{AC}^0 \subseteq \mathrm{FTC}^0$, and then plug in FTC^0-circuits computing f into C to show that $L \in \mathrm{AC}^{0\#\mathrm{AC}^0}$. For this, we will now show that our definitions allow that circuits are plugged in this way in the case of TC^0 both in the non-uniform as well as the FO[BIT]-uniform setting, as long as the oracle is sufficiently weak.

Lemma 4.6. *For any class $\mathfrak{C} \subseteq \mathrm{FTC}^0$, it holds that $\mathrm{TC}^{0\mathfrak{C}} \subseteq \mathrm{TC}^0$ both in the non-uniform and the* FO[BIT]*-uniform setting.*

Proof. We begin by showing the non-uniform version, i.e., $\mathrm{TC}^{0\mathfrak{C}} \subseteq \mathrm{TC}^0$. Let $L \in \mathrm{TC}^{0\mathfrak{C}}$ via the TC^0 oracle-circuit family C and oracle $f \in \mathfrak{C}$. As $\mathfrak{C} \subseteq \mathrm{FTC}^0$, there is an FTC^0 circuit family $\mathcal{D}$ computing f. We now construct a TC^0 circuit family deciding L. This can be done by replacing the oracle gates by subcircuits computing f, which can be taken from $\mathcal{D}$.

As we can non-uniformly choose the circuit for the correct input length and connect input and output gates of that circuit to the surrounding circuit from C as required, we only need to make sure that the size of the circuit that we plug in is only polynomial in the input length. For this, note that the number of inputs of different oracle gates within a single circuit from C can differ. Let q be the polynomial bounding the size of circuits in C depending on the input length. Then the number of inputs to all oracle gates is also bounded by q. There is also a polynomial p bounding the size of circuits in $\mathcal{D}$. Thus, when replacing oracle gates by subcircuits from $\mathcal{D}$, the size of each subcircuit is bounded by $p \circ q$ and hence still polynomial in the input length. This finishes the proof of the non-uniform version.

We next adapt the above proof to the uniform setting, that is, we prove

$$\text{FO[BIT]-uniform } \mathrm{TC}^{0\mathfrak{C}} \subseteq \text{FO[BIT]-uniform } \mathrm{TC}^0.$$

First note that the fact that the size of any circuit from $\mathcal{D}$ that is plugged into C is bounded by $p \circ q$ allows us to fix the tuple length for the representation of gates in the new circuit to be the old tuple length plus $\deg(p \circ q)$, the degree of the polynomial $p \circ q$, plus some constant to deal with small structures (as $p \circ q$ might not be monomial). Similarly to what we do in the proof of Theorem 4.3, we add an additional element n to the universe and use it as padding. We think of the tuples encoding gates as having two parts: For non-oracle gates only the first part of the tuple is used, and connections are only present between gates where all newly added components are n. For oracle gates, the second part is used to represent gates within the respective subcircuit from $\mathcal{D}$ that is plugged in to replace that oracle gate.

For the inner connections, the uniformity of $\mathcal{D}$ is used and the correct circuit from that family is chosen with the help of the uniformity of C: Let $C =: (\mathbf{C}_n)_{n\in\mathbb{N}}$ and $\mathcal{D} =: (\mathbf{D}_n)_{n\in\mathbb{N}}$. Also, let $\mathcal{I}$ and $\mathcal{I}'$ be the FO[BIT]-queries showing uniformity of C and $\mathcal{D}$, respectively. Due to the Index-predicate, the number of inputs of any given oracle gate is FO[BIT]-definable from $\mathbf{C}_{|w|}$ and, using $\mathcal{I}$, also from $\mathbf{A}_w$. Together with the FO[BIT]-query $\mathcal{I}'$, this shows that for any given oracle gate g, the circuit $\mathbf{D}_m$ is FO[BIT]-definable from $\mathbf{A}_w$, where m is the number of inputs of g in $\mathbf{C}_n$.

For the connections between the new subcircuits and the surrounding circuit, gates that were successors of the oracle gate before will be successors of the correct output gate of the corresponding subcircuit afterwards. Here, the correct output gate can be chosen uniformly, as the index of the output bit accessed by an oracle gate as well as the indices of output gates of FTC^0-circuits are specified in the corresponding FO-structures. Gates that were predecessors of the oracle gate will be directly connected to the input gates of the corresponding subcircuit (and these inputs can be made $\wedge$-gates as they only have one predecessor). Similar as for the outputs, connecting predecessors to the correct input gate of the subcircuit can be done uniformly, as both the index of any input gate as well as the index of predecessors of oracle gates are specified by the corresponding FO-structures.

This means that the oracle gates can be uniformly replaced by subcircuits. □

In order to show $\mathrm{FOCW} \subseteq \mathrm{AC}^{0^{\#\mathrm{AC}^0}}$ for the main result of this section, we need a certain closure property of $\#\mathrm{AC}^0$. The class FOCW allows access to several #Win-FO-functions by using occurrences of # indexed with different FO-formulae. In contrast, in $\mathrm{AC}^{0^{\#\mathrm{AC}^0}}$, we can only use a single oracle function. Therefore, we need to combine any constant number of oracle functions from $\#\mathrm{AC}^0$ into a single one such that using oracle access to this one function, we can simulate oracle access to all of the required functions. For this, we show that $\#\mathrm{AC}^0$ is closed under a weak form of concatenation we call *polynomially padded concatenation*.

Definition 4.7. A class $\mathfrak{C}$ of counting functions is closed under polynomially padded concatenation if for all $f, g \in \mathfrak{C}$ there is a polynomial p such that the function

$$h(x) := f(x) \circ 0^{p(|x|)-|g(x)|} \circ g(x)$$

is also in $\mathfrak{C}$.

Lemma 4.8. $\#\mathrm{AC}^0$ *and* FO[BIT]-uniform $\#\mathrm{AC}^0$ *are closed under polynomially padded concatenation.*

Proof. Let $f, g \in \#\mathrm{AC}^0$ via circuit families C_1, C_2 and let p be a polynomial bounding the length of all outputs of the function g depending on the length of the input. The following circuit shows $h(x) := f(x) \circ 0^{p(|x|)-|g(x)|} \circ g(x) \in \#\mathrm{AC}^0$:
Compute $f(x)$ using the circuit family C_1, then shift the result by $p(|x|)$ bits to the left. Compute separately $g(x)$ using the circuit family C_2. Add both results together. Shifting can be done by multiplication with a subcircuit computing $2^{p(|x|)}$. Figure 4.2 illustrates a bit more detailed how this is done with our definition of a circuit (edges have no multiplicities).

Obviously, this circuit has still polynomial size in the input length and computes $h(x)$. It also can be easily seen that if C_1 and C_2 are FO[BIT]-uniform circuit families, then the constructed circuit family is FO[BIT]-uniform. □

We are now in a position to prove the main theorem of this section. We start by showing the non-uniform version, i.e., $\mathrm{TC}^0 = \mathrm{FOCW}[\mathrm{Arb}] = \mathrm{AC}^{0^{\#\mathrm{AC}^0}}$.

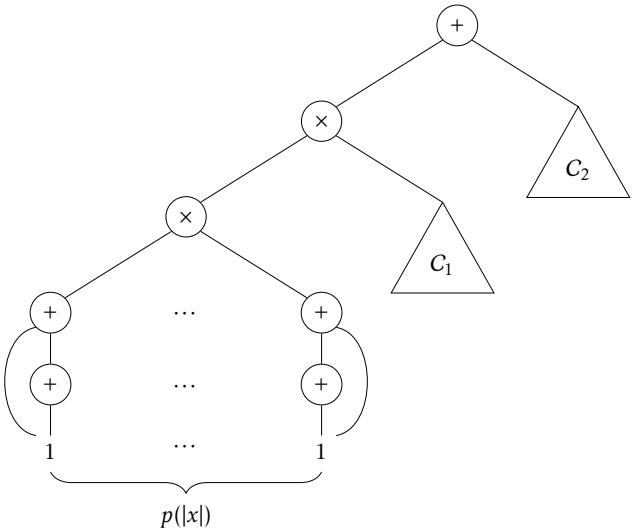

Figure 4.2: $\#AC^0$ is closed under polynomially padded concatenation.

Proof of the non-uniform version of Theorem 4.5. We prove this result by establishing the following chain of inclusions:

$$TC^0 \subseteq PAC^0 \overset{(1)}{\subseteq} FOCW[Arb] \overset{(2)}{\subseteq} AC^{0\#AC^0} \overset{(3)}{\subseteq} TC^0$$

Recall that $TC^0 \subseteq PAC^0$ was shown by Agrawal et al. [AAD00]. We now prove the remaining three inclusions.

Proof of (1): Let $L \in PAC^0$. Then there are $f, g \in \#AC^0$ such that for all x:

$$x \in L \iff f(x) > g(x).$$

Since $\#AC^0 = \#$Win-FO[Arb], we have $f, g \in \#$Win-FO[Arb]. Therefore, we can get the bits of $f(x)$ and $g(x)$ using the #-predicate of FOCW. Let ψ_1, ψ_2 be formulae showing $f \in \#$Win-FO[Arb] and $g \in \#$Win-FO[Arb], respectively. We now need a formula expressing $f(x) > g(x)$. This can be done in the standard way of expressing the natural order on binary numbers using a bit predicate, that is, we express that $f(x)$ has at least as many bits as $g(x)$ (excluding leading zeros) and the first position where the two numbers differ contains a 1 in $f(x)$ and a 0 in $g(x)$. The following formula illustrates how this is expressed in FOCW[Arb]:

$$\exists \overline{p}_0 \Big[f_{\overline{p}_0}(x) = 1 \wedge \forall(\overline{q} > \overline{p}_0)\,(f_{\overline{q}}(x) = 0 \wedge g_{\overline{q}}(x) = 0) \wedge$$
$$\exists(\overline{p}_1 \leq \overline{p}_0)\Big(\forall(\overline{p}_0 \geq \overline{q} > \overline{p}_1)\,(f_{\overline{q}}(x) = g_{\overline{q}}(x)) \wedge f_{\overline{p}_1}(x) > g_{\overline{p}_1}(x)\Big)\Big].$$

Formally, $f_i(x)$ and $g_i(x)$ are accessed using $\#_{\psi_1}(i)$ and $\#_{\psi_2}(i)$, respectively, the input x being implicitly provided as the encoding of the input structure. This means that $f_i(x)$

and $g_i(x)$ are obtained as Boolean values and their values are compared using Boolean connectives in the actual formula. Note that all variables in the formula are second-sort variables.

Proof of (2): Let $L \in \mathrm{FOCW[Arb]}$ via φ, where φ is in prenex normal form. Note that is is straightforward to show that FOCW has a prenex normal form. When evaluating φ using a circuit family, second-sort variables can be treated in the same way as first sort variables. This means that the only difference to an FO[Arb]-formula are occurrences of #-predicates. Hence, we can build a circuit family evaluating φ analogously to the one in the proof for $\mathrm{FO[Arb]} \subseteq \mathrm{AC}^0$. The difference is that within the second part of the circuit, which evaluates the quantifier-free part of φ, #-predicates have to be evaluated. If all occurrences of #-predicates in φ access bits of the same $\#\mathrm{AC}^0$-function, this is easy: In this case, we can use that function as our oracle and use an oracle gate to evaluate the occurrence of the #-predicate. If there are occurrences of #-predicates with different formulae in the subscript, i.e., associated with different counting functions, we first need to combine them into a single counting function that allows to access each of them as needed. For this, we use polynomially padded concatenation: Since there are only constantly many different $\#\mathrm{AC}^0$-functions accessed via occurrences of #-predicates in φ, a polynomially padded concatenation of all of them is still in $\#\mathrm{AC}^0$. Now, in each copy of the subcircuit evaluating the formula, we can use exactly one oracle gate to compute the value of each of the occurrences—only the index needs to be changed according to the construction of the polynomially padded concatenation.

Proof of (3): This can be seen with the following chain of inclusions:

$$\mathrm{AC}^{0\,\#\mathrm{AC}^0} \overset{\text{(i)}}{\subseteq} \mathrm{TC}^{0\,\#\mathrm{AC}^0} \overset{\text{(ii)}}{\subseteq} \mathrm{TC}^{0\,\mathrm{FTC}^0} \overset{\text{(iii)}}{\subseteq} \mathrm{TC}^0.$$

(i) follows from the fact that AC^0 oracle-circuit families are also TC^0 oracle-circuit families. (ii) is due to $\#\mathrm{AC}^0 \subseteq \mathrm{FTC}^0$. (iii) is an application of Lemma 4.6. □

Finally, we show the uniform version of the above theorem, i.e.,

$$\text{FO[BIT]-uniform } \mathrm{TC}^0 = \mathrm{FOCW[BIT]} = \text{FO[BIT]-uniform } \mathrm{AC}^{0\,\text{FO[BIT]-uniform } \#\mathrm{AC}^0}.$$

Proof of the FO[BIT]*-uniform version of Theorem 4.5.* We prove this analogously to the non-uniform version. We explain how the constructions can be made uniform where necessary. Recall the chain of inclusions from before:

$$\mathrm{TC}^0 \subseteq \mathrm{PAC}^0 \overset{(1)}{\subseteq} \mathrm{FOCW} \overset{(2)}{\subseteq} \mathrm{AC}^{0\,\#\mathrm{AC}^0} \overset{(3)}{\subseteq} \mathrm{TC}^0.$$

Recall that the inclusion $\mathrm{TC}^0 \subseteq \mathrm{PAC}^0$ was proven in the FO[BIT]-uniform setting by Ambainis et al. [ABL98]. We will now show that the rest of the inclusions can be made uniform as well.

Proof of (1): The identity $\#\mathrm{AC}^0 = \#\text{Win-FO}$ also holds in the uniform case. As comparing two numbers using bitwise access can obviously also be done in FO[BIT] (using the same formula as in the proof of the non-uniform version), this means that $\mathrm{PAC}^0 \subseteq \mathrm{FOCW}$ also holds in the FO[BIT]-uniform setting.

Proof of (2): We use the same idea as in the non-uniform setting. The first issue in doing so is that we only have #Win-FO[BIT] $\overset{*}{\subseteq}$ #AC^0 instead of an actual inclusion. This can be resolved by letting the uniformity check for each oracle gate whether it has only one predecessor and if this is the case, replace it by a constant-size subcircuit computing the required bit of the function. Furthermore, we need to make the construction using polynomially padded concatenation uniform. Since Lemma 4.8 also holds uniformly, the only problem here is choosing the right indices for the oracle gates. Lets first assume that the formula only uses (possibly multiple bits of) one oracle. In this case, the index is given by a tuple of quantified variables. These are part of the representation of any input gate as a tuple over the universe and can be directly accessed to uniformly describe the index.

If there are two different oracles used within the formula, we use a polynomially padded concatenation of them again. Let $f, g \in$ #AC^0 be the oracle functions and $f(x) \circ 0^{p(|x|)-|g(x)|} \circ g(x|)$ their polynomially padded concatenation. All oracle gates that query g can use the same index as in the original formula. All oracle gates that query f have to add $p(|x|)$ to the index. This can be done in FO[BIT], so the index can be described uniformly. Now, for an arbitrary number of #-predicates, the above can be applied inductively.

Proof of (3): This can be seen with the same chain of inclusions as in the non-uniform version:

$$\mathrm{AC}^{0^{\#\mathrm{AC}^0}} \subseteq \mathrm{TC}^{0^{\#\mathrm{AC}^0}} \subseteq \mathrm{TC}^{0^{\mathrm{FTC}^0}} \subseteq \mathrm{TC}^0,$$

as all of these inclusions also hold in the FO[BIT]-uniform setting with the same arguments as in the proof of the non-uniform version. □

4.3 Conclusion

In this chapter we have seen that both in the non-uniform as well as the FO[BIT]-uniform setting, #Win-FO captures #AC^0. In other words, natural notions of counting models in first-order logic lead to the same counting class as counting proof trees in the corresponding class of families of Boolean circuits. This can be considered an additional argument that counting proof trees is the "right" way of defining counting classes in circuit complexity, as it further confirms the robustness of the class. Note that the earlier characterization of #AC^0 in the $\mathrm{U_D}$-uniform setting in terms of arithmetic expressions [FVB94] can be seen as a translation of our result into the QSO-framework of Arenas et al. [AMR20]. As we will see in the next Chapter, our approach also makes it possible to view #AC^0 as a natural subclass of the class #FO, which captures #P as shown by Saluja et al. [SST95].

Our result also has implications regarding different uniformity conditions for the class #AC^0. Looking at the proof of Theorem 4.3, one can see that evaluating a fixed formula (here in the sense of computing the number of winning strategies of the verifier in a given input structure) can be done by a highly uniform circuit family. It is straightforward to show that this circuit family is not only FO[BIT]-uniform, but also $\mathrm{U_D}$-uniform. As

DLOGTIME $\subseteq$ FO[BIT], the consequence is that both uniformity conditions coincide (when disregarding inputs of length ≤ 1). Furthermore, from these arguments together with Observation 3.11, it follows that #Win-FO exactly captures U_D-uniform $\#AC^0$, including inputs of length 1.

We then used our characterization of $\#AC^0$ and the result that $PAC^0 = TC^0$ both in the non-uniform [AAD00] as well as the FO[BIT]-uniform setting [ABL98] to obtain a new model-theoretic characterization of the class TC^0. For this, we introduce adequate notions of oracle access to counting functions both in circuit complexity as well as first-order logic. For the latter, we use two-sorted logics to obtain a notion that allows to define logics with oracle access to function classes even in the absence of built-in relations on the first sort. This leads to the characterization of TC^0 as first-order logic with oracle access to #Win-FO both in the non-uniform as well as the FO[BIT]-uniform setting.

5 Putting the Characterization of #AC^0 into Perspective

In this chapter we aim to compare our characterization of FO[BIT]-uniform #AC^0 to a model-theoretic characterization of #P similar to that by Saluja et al. [SST95]. For this, we use the characterization of #AC^0 in terms of counting Skolem functions, that is, FO[BIT]-uniform #AC^0 = #Skolem-FO[BIT]. The class #Skolem-FO[BIT] can be viewed as the class of functions counting the satisfying assignments to free function variables in a Π_1[BIT]-formula restricted to formulae that arise from Skolemization. Recall that Saluja et al. [SST95] showed that #P is captured by #FO$^{\text{rel}}$, which is the class of functions counting satisfying assignments to free relational variables and first-order variables in first-order formulae (see Definition 2.35 and Theorem 2.36). This gives rise to the idea to study the class #FO, the variant of #FO$^{\text{rel}}$ where free function variables are used instead of free relational variables. It can be expected that this class still captures #P as first-order logic can express that a function is a relation and vice versa. As #Skolem-FO[BIT] $\subseteq$ #Π_1, where the latter is the subclass of #FO obtained by only allowing Π_1-formulae, it is a natural next step to study the alternation hierarchy inside #FO. Within this framework, we will use $\leq$ as a built-in predicate and min as a built-in constant in addition to BIT, as we consider $\leq$ and min important to define meaningful functions, but for the weaker fragments, $\leq$ and min are not necessarily definable from BIT. We will again define our logical classes using arbitrary vocabularies, as these allow for more direct definitions of many natural problems. Note that the corresponding classes obtained when only allowing vocabulary τ_{string} are not necessarily equivalent to our classes, as we are working with weak fragments of FO. As within the whole chapter we will only consider FO[BIT]-uniform circuit classes, we will omit the uniformity from the names of classes for succinctness, implicitly assuming FO[BIT]-uniformity.

In the case of #FO$^{\text{rel}}$, Saluja et al. also studied the corresponding alternation hierarchy and showed some interesting properties of the classes in this hierarchy. For a class $\mathfrak{C}$ of first-order formulae with free relational variables they denote by #$\mathfrak{C}$ the class of functions definable in the sense of #FO$^{\text{rel}}$ using only $\mathfrak{C}$-formulae. Hence, the classes of the alternation hierarchy are #Σ_i^{rel} and #Π_i^{rel} for $i \in \mathbb{N}$. As mentioned, Saluja et al. [SST95] considered classes of counting functions taking structures over fixed vocabularies as inputs. For this reason, it is not possible that a function is definable over an unintended vocabulary. They showed the following inclusion structure in their framework.

Theorem 5.1. #Σ_0^{rel} = #Π_0^{rel} $\subsetneq$ #Σ_1^{rel} $\subsetneq$ #Π_1^{rel} $\subsetneq$ #Σ_2^{rel} $\subsetneq$ #Π_2^{rel} = #FO$^{\text{rel}}$ = #P *as classes of counting functions taking structures over fixed vocabularies as inputs.*

Furthermore, they showed that #Σ_0^{rel} $\subseteq$ FP. Using our definition in terms of functions

taking strings as inputs, the strictness of the above inclusions does not immediately follow from their results, as it could be possible to define the function used for a separation over some different vocabulary. Still, it seems likely that the results can be transferred, as we will see that in many cases we at least can enforce the vocabulary τ_{string}, see for example the proof of Theorem 5.24, where we do this for the class $\#\Sigma_2^{\text{rel}}$.

To illustrate the power of the above classes, as well as how to define natural counting problems in them, we repeat an example from Saluja et al. that will also be important for us later.

Example 5.2. Consider #3DNF, the problem of counting the number of satisfying assignments of a propositional formula in disjunctive normal form with at most 3 literals per term. We will show that this problem is in the class $\#\Sigma_1^{\text{rel}}$. To do so, we use the vocabulary $\sigma_{\text{3DNF}} \coloneqq (\mathsf{D}_0, \mathsf{D}_1, \mathsf{D}_2, \mathsf{D}_3)$ with $\text{ar}(\mathsf{D}_i) = 3$ for $0 \leq i \leq 3$. Given a 3DNF-formula φ over variables from a set V, we construct a corresponding σ-structure $\mathbf{A}_\varphi$ with universe V such that for any $x_1, x_2, x_3 \in V$, $(x_1, x_2, x_3) \in \mathsf{D}_i^{\mathbf{A}_\varphi}$ if and only if $\bigwedge_{1 \leq j \leq i} \neg x_j \wedge \bigwedge_{i < j \leq 3} x_j$ appears as a term in φ. Now consider the following σ-formula with free relational variable T:

$$\begin{aligned}\Phi_{\#\text{3DNF}}(\mathsf{T}) \coloneqq \exists x \exists y \exists z \Big(&\big(\mathsf{D}_0(x,y,z) \wedge \mathsf{T}(x) \wedge \mathsf{T}(y) \wedge \mathsf{T}(z)\big) \vee \\ &\big(\mathsf{D}_1(x,y,z) \wedge \neg\mathsf{T}(x) \wedge \mathsf{T}(y) \wedge \mathsf{T}(z)\big) \vee \\ &\big(\mathsf{D}_2(x,y,z) \wedge \neg\mathsf{T}(x) \wedge \neg\mathsf{T}(y) \wedge \mathsf{T}(z)\big) \vee \\ &\big(\mathsf{D}_3(x,y,z) \wedge \neg\mathsf{T}(x) \wedge \neg\mathsf{T}(y) \wedge \neg\mathsf{T}(z)\big)\Big).\end{aligned}$$

Observe that $\Phi_{\#\text{3DNF}}$ is a Σ_1-formula. Evaluated on an input structure $\mathbf{A}_\varphi$, it expresses that an assignment to T encodes a satisfying assignment of φ. For this, a propositional assignment is encoded by the set of variables mapped to 1. Hence, the number of assignments T such that $\mathbf{A}_\varphi \models \Phi_{\#\text{3DNF}}(T)$ is equal to the number of satisfying assignments of φ.

In this chapter we will show that in contrast to the case of $\#\text{FO}^{\text{rel}}$, where the alternation hierarchy collapses to the class $\#\Pi_2^{\text{rel}}$, the alternation hierarchy for #FO collapses to the class $\#\Pi_1$, but #FO still captures #P, albeit only in the sense of $\overset{*}{=}$. Furthermore, we show that #Skolem-FO[BIT] coincides with the class of functions definable in the syntactical fragment Π_1^{prefix} of Π_1. We obtain the following inclusion structure between the classes of the hierarchy, $\#\Pi_1^{\text{prefix}}$, and #P:

$$\begin{array}{ccccc} & & \#\text{AC}^0 = \#\Pi_1^{\text{prefix}} & & \\ & \swarrow & & \searrow & \\ \#\Sigma_0 & & \nwarrow\ \ \nearrow & & \#\Pi_1 = \#\text{FO} \overset{*}{=} \#\text{P}. \\ & \searrow & & \swarrow & \\ & & \#\Sigma_1 & & \end{array}$$

This structure is also illustrated in Figure 5.1 on page 97.

Saluja et al. further showed that their class $\#\Sigma_1^{\text{rel}}$ admits fully polynomial-time approximation schemes (FPRAS), showing that all functions in this class are efficiently approximable in this sense. We show that even though the alternation hierarchy in #FO has less levels, which could have hinted at $\#\Sigma_1$ being closer to #P than $\#\Sigma_1^{\text{rel}}$, the class $\#\Sigma_1$ still shares this property.

As the alternation hierarchy in #FO is very shallow, collapsing to the class $\#\Pi_1$, another interesting approach is to study further hierarchies inside the class $\#\Pi_1$. As a first step in this direction, we study the hierarchy arising by restricting the number of universal quantifiers. We show that this hierarchy is strict by using known connections to nondeterministic random access machines [GO04] as well as the corresponding time hierarchy theorem [Coo73].

Finally, we compare the class #AC^0 to the classes introduced by Saluja et al., showing that their hierarchy does not seem to help in better understanding the relation of #AC^0 and #P, as all but the classes $\#\Sigma_0^{\text{rel}}$ and $\#\text{FO}^{\text{rel}}$ are incomparable to #AC^0.

5.1 Relationship Between the Characterizations of #AC^0 and #P

In this section, we will introduce our new framework based on counting satisfying assignments to free function variables in FO-formulae. This idea stems from the intuition that the full class #FO captures #P, just as $\#\text{FO}^{\text{rel}}$, while the class #Skolem-FO[BIT] = #AC^0 better fits in this framework as it is based on counting satisfying assignments to free function variables in Π_1-formulae of a certain form, namely formulae that arise from Skolemization. We now begin by formally defining our framework.

Definition 5.3. A function $f\colon \{0,1\}^+ \to \mathbb{N}$ is in #FO, if there is a vocabulary σ and an FO$[\leq, \text{BIT}, \text{min}]$-formula $\varphi(\mathsf{F}_1, \ldots, \mathsf{F}_k, x_1, \ldots, x_\ell)$ over σ with free function variables $\mathsf{F}_1, \ldots, \mathsf{F}_k$ and free individual variables $x_1, \ldots, x_\ell$ such that for all $\mathbf{A} \in \text{STRUC}[\sigma]$,

$$f(\text{enc}_\sigma(\mathbf{A})) = |\{(F_1, \ldots, F_k, c_1, \ldots, c_\ell) \mid \mathbf{A} \models \varphi(F_1, \ldots, F_k, c_1, \ldots, c_\ell)\}|,$$

and $f(x) = 0$ if x is not the encoding of a σ-structure.

In the same fashion we define counting classes using fragments of FO. For a class $\mathfrak{C}$ of FO-formulae, we denote by $\#\mathfrak{C}$ the class of functions definable in the sense of #FO using only formulae from $\mathfrak{C}$. In this vein we will mainly consider the classes $\#\Sigma_i$ and $\#\Pi_i$ for arbitrary i as well as the class $\#\Pi_1^{\text{prefix}}$ based on a syntactical fragment of $\#\Pi_i$. Note, that the free individual variables could also be seen as free function variables of arity 0.

We stress that we use the built-in relations $\leq$ and BIT and the built-in constant min in the above definition. The predicate BIT is used to ensure that the class #AC^0 = #Skolem-FO[BIT] naturally fits in this framework. The addition of $\leq$ and min is somewhat arbitrary, as the results of this paper could be transferred to the case of BIT as the only built-in predicate. Still, it seems reasonable and natural to add them, as it allows for more interesting definitions in the weaker classes of the hierarchy.

Next, we show that the full class #FO in our framework captures #P.

Theorem 5.4. #FO $\overset{*}{=}$ #P.

Proof. We show this by the inclusions

$$\#\mathrm{FO}^{\mathrm{rel}} \overset{*}{\subseteq} \#\mathrm{FO} \subseteq \#\mathrm{P} \subseteq \#\mathrm{FO}^{\mathrm{rel}}.$$

Recall that $\overset{*}{\subseteq}$ denotes that a class is contained in another class only in the weaker sense of $\overset{*}{=}$. As $\subseteq$ implies $\overset{*}{\subseteq}$, this inclusion chain shows #FO $\overset{*}{=}$ #P.

For the inclusion #FO$^{\mathrm{rel}} \overset{*}{\subseteq}$ #FO, note that in #FO the behavior on inputs of length ≤ 1 can be chosen independently of the the behavior on other inputs. Hence, we prove the inclusion ignoring inputs of length ≤ 1. Let $f \in \#\mathrm{FO}^{\mathrm{rel}}$ via the formula φ containing free relational variables $\mathsf{R}_1, \ldots, \mathsf{R}_k$. We can replace R_i by a function variable F_i of the same arity for all i. We then additionally express that for these functions only 0 and 1, which are first-order definable using the built-in order, are allowed as function values. Note that for this, input length ≥ 2 is required. Then each occurrence $\mathsf{R}_i(\overline{z})$ can be replaced by $\mathsf{F}_i(\overline{z}) = \min$.

The inclusion #FO $\subseteq$ #P is straightforward, as we can simply nondeterministically guess the assignment to the free variables and verify whether it satisfies the formula in polynomial time. The inclusion #P $\subseteq$ #FO$^{\mathrm{rel}}$ was shown in [SST95]. □

The class #Skolem-FO[BIT] = #AC^0 is a subclass of #Π_1 where the occurrence of free function variables in the formula is restricted in a certain way. To show that this class naturally fits in our framework we will now identify a syntactical fragment Π_1^{prefix} of Π_1 that allows us to capture #AC^0 = #Skolem-FO[BIT].

Definition 5.5. A Π_1-formulae φ, possibly with free function variables, is in the class Π_1^{prefix}, if it is of the form $\varphi(\overline{\mathsf{G}}, \overline{x}) = \forall y_1 \ldots \forall y_k \, \psi(\overline{\mathsf{G}}, \overline{x}, y_1, \ldots, y_k)$ for some $k \in \mathbb{N}$, where ψ is quantifier-free and for any function symbol G, all occurrences of G in ψ are of the form $\mathsf{G}(y_1, \ldots, y_{\mathrm{ar}(\mathsf{G})})$.

The term "prefix" in the name of the class refers to the fact that the sequence of arguments of a function symbol in any atom is a prefix of the sequence of bound variables in the order of their occurrence in the quantifier prefix of the formula. The class #Π_1^{prefix} is then defined in analogy to #FO as the class of functions definable in the sense of #FO by a $\Pi_1^{\mathrm{prefix}}[\leq, \mathrm{BIT}, \min]$-formula. We now show that this class coincides with #Skolem-FO[BIT] = #AC^0.

Lemma 5.6. #Skolem-FO[BIT] = #Π_1^{prefix}.

Proof. **#Skolem-FO[BIT] = #Π_1^{prefix}:** Let g be a function in #Skolem-FO[BIT] via the FO[BIT]-formula φ over some vocabulary σ. Furthermore, let $\psi(\overline{\mathsf{F}})$ with free function variables $\overline{\mathsf{F}}$ be the formula obtained by Skolemization from φ. Now, ψ is an Π_1[BIT]-formula over σ and for all $\mathbf{A} \in \mathrm{STRUC}[\sigma]$, we have

$$\#\mathrm{Win}(\mathbf{A}, \varphi) = |\{\, \overline{F} \mid \mathbf{A} \models \psi(\overline{F}) \,\}|.$$

Binary strings that are not encodings of σ-structures are also mapped to 0 by both the function defined by φ and the function defined by ψ. As ψ is prefix-restricted, this shows $g \in \#\Pi_1^{\text{prefix}}$.

$\#\Pi_1^{\text{prefix}} \subseteq$ #Skolem-FO[BIT]: Let $g \in \#\Pi_1^{\text{prefix}}$ via the $\Pi_1^{\text{prefix}}[\leq, \text{BIT}, \min]$-formula φ over some vocabulary σ. Since all function symbols occurring in φ are only applied to a unique prefix of the universally quantified variables, they can be seen as Skolem functions corresponding to suitable existentially quantified variables. Thus, we can replace all occurrences of function symbols by new variables that are existentially quantified at adequate positions between the universally quantified variables. If for example, the input for a function was $x_1, \ldots, x_\ell$, then the new variable is quantified after the part $\forall x_1 \ldots \forall x_\ell$ of the quantifier prefix. Furthermore, we replace occurrences of the built-in predicate $\leq$ by the FO[BIT]-formula defining this predicate. In order to handle the built-in constant min we replace it by the new existentially quantified variable min and force it to be equal to 0, which is again FO[BIT]-definable. Using Lemma 3.3 we can ensure that these last steps do not change the number of Skolem functions. This yields a formula φ' that shows $g \in$ #Skolem-FO[BIT]. □

5.2 An Alternation Hierarchy in #FO

As $\#\text{AC}^0 = \#\Pi_1^{\text{prefix}}$ is a syntactical fragment of $\#\Pi_1$, a natural next step is to study the alternation hierarchy inside #FO. We show that the hierarchy collapses to a quite low level, namely to the class $\#\Pi_1$. Further, we get a clearer picture of how $\#\text{AC}^0 =$ #Skolem-FO[BIT] fits in the hierarchy by studying its relations to the other classes of the hierarchy besides $\#\Pi_1$. In doing so, we determine the complete inclusion structure of the classes in the alternation hierarchy. We begin by showing that the hierarchy collapses to $\#\Pi_1$.

Theorem 5.7. $\#\text{FO} = \#\Pi_1$.

Proof. Let $f \in \#\text{FO}$ via the FO$[\leq, \text{BIT}, \min]$-formula $\varphi(\overline{\mathsf{F}}, \overline{x})$ over some vocabulary σ. By viewing φ as a formula over the vocabulary $\sigma \cup (\overline{\mathsf{F}})$ and noticing that Lemma 3.3 also holds for vocabularies with function symbols, we obtain an FO$[\leq, \text{BIT}, \min]$-formula φ' over σ in prenex normal form with the following properties: For all $\mathbf{A} \in \text{STRUC}[\sigma]$, all assignments $\overline{F}$ for $\overline{\mathsf{F}}$ and all assignments $\overline{c}$ for $\overline{x}$,

$$\mathbf{A} \models \varphi(\overline{F}, \overline{c}) \implies \#\text{Win}(\mathbf{A}, \varphi'(\overline{F}, \overline{c})) = 1 \text{ and}$$

$$\mathbf{A} \not\models \varphi(\overline{F}, \overline{c}) \implies \#\text{Win}(\mathbf{A}, \varphi'(\overline{F}, \overline{c})) = 0.$$

Let $\varphi''(\overline{\mathsf{F}}, \overline{\mathsf{G}}, \overline{x}, \overline{y})$ be the formula obtained from φ' by Skolemization, where $\overline{\mathsf{G}}$ are the Skolem function symbols of arity ≥ 1 and $\overline{y}$ are the Skolem function symbols of arity 0, that is, first-order variables.

The application of Lemma 3.3 above ensures that for any input structure and assignment to the variables $\overline{\mathsf{F}}$ and $\overline{x}$, there is at most a unique assignment to the function

symbols $\overline{\mathsf{G}}$ and first-order variables $\overline{y}$ obtained from Skolemization. In consequence, for all $\mathbf{A} \in \mathrm{STRUC}[\sigma]$,

$$|\{(\overline{F},\overline{c}) \mid \mathbf{A} \models \varphi(\overline{F},\overline{c})\}| = |\{(\overline{F},\overline{G},\overline{c},\overline{d}) \mid \mathbf{A} \models \varphi''(\overline{F},\overline{G},\overline{c},\overline{d})\}|,$$

and binary strings that are not encodings of σ-structures are mapped to 0 by both the function defined by φ and the function defined by φ''. This shows $f \in \#\Pi_1$. □

From the results that $\#\Pi_1^{\mathrm{prefix}} = \#\mathrm{AC}^0 \subseteq \mathrm{FTC}^0$ and $\#\Pi_1 \stackrel{*}{=} \#\mathrm{P}$ and using that $\mathrm{FTC}^0 \subsetneq \#\mathrm{P}$, we now immediately obtain that $\#\Pi_1^{\mathrm{prefix}}$ is strictly contained in the class $\#\Pi_1$ of our hierarchy.

Corollary 5.8. $\#\Pi_1^{\mathrm{prefix}} \subsetneq \#\Pi_1$.

Proof. The inclusion $\#\Pi_1^{\mathrm{prefix}} \subseteq \#\Pi_1$ holds by definition and we have $\#\Pi_1^{\mathrm{prefix}} = \#\mathrm{AC}^0 \subseteq \mathrm{FTC}^0$. Recall that the operator C on complexity classes was introduced before Proposition 2.12. The identity $\mathrm{PP} = \mathsf{C} \cdot \#\Pi_1$ directly follows from $\mathrm{PP} = \mathsf{C} \cdot \#\mathrm{P}$ and $\#\Pi_1 \stackrel{*}{=} \#\mathrm{P}$. The separation then can be shown analogously to $\mathrm{FTC}^0 \neq \#\mathrm{P}$, using $\mathsf{C} \cdot \mathrm{FTC}^0 = \mathrm{TC}^0$ and $\mathrm{TC}^0 \neq \mathrm{PP}$. □

We now want to determine more precisely how the class fits in the alternation hierarchy. As a first step, we compare $\#\Pi_1^{\mathrm{prefix}}$ to the lowest class $\#\Sigma_0$ of our hierarchy, showing that it strictly contains this class.

Theorem 5.9. $\#\Sigma_0 \subsetneq \#\mathrm{AC}^0$.

Proof. We start by showing the inclusion. Certain observations in that proof will then almost directly yield the strictness. Let $f \in \#\Sigma_0$ via the quantifier-free FO-formula $\varphi(\mathsf{F}_1,\ldots,\mathsf{F}_k,x_1,\ldots,x_\ell)$ over some vocabulary σ, where $\mathsf{F}_1,\ldots,\mathsf{F}_k$ are free function variables and $x_1,\ldots,x_\ell$ are free individual variables, that is, for all $\mathbf{A} \in \mathrm{STRUC}[\sigma]$,

$$f(\mathrm{enc}_\sigma(\mathbf{A})) = |\{(F_1,\ldots,F_k,c_1,\ldots,c_\ell) \mid \mathbf{A} \models \varphi(F_1,\ldots,F_k,c_1,\ldots,c_\ell)\}|,$$

and $f(x) = 0$, if x is not the encoding of a σ-structure. Without loss of generality we can assume that in φ no nesting of function symbols occurs. The reason is that for any function symbol H and terms $t_1,\ldots,t_n$, we can replace all occurrences of $\mathsf{H}(t_1,..,t_n)$ by $\mathsf{H}(y_1,..,y_n)$, for fresh variables $y_1,..,y_n$, and express that these variables are equal to the corresponding terms. The latter is expressed by the formula $(\bigwedge_{1\leq i\leq n} y_i = t_i)$. As this formula ensures that in any satisfying assignment, the values assigned to the variables y_i are uniquely determined by the values of the terms t_i, the total number of satisfying assignment of the resulting formula is equal to that of φ.

Let $A := \mathrm{dom}(\mathbf{A})$. For all i, let a_i be the arity of F_i and let m_i be the number of different tuples of variables that occur as inputs to F_i within φ. Let $\overline{e}_{i1},\ldots,\overline{e}_{im_i}$ be those tuples of variables in the order of their occurrence within φ. Now, define $\varphi'(\overline{y}_{11},\ldots,\overline{y}_{km_k},x_1,\ldots,x_\ell)$ as the formula φ after replacing for all i,j all occurrences of $\mathsf{F}_i(\overline{e}_{ij})$ by the new free variable $\overline{y}_{ij}$. Let $m := \sum_i m_i$.

The value of any tuple $\overline{e}_{ij}$ is uniquely determined by an assignment to the variables $x_1,\dots,x_\ell$. Thus, we can use the free individual variables y_{ij} to count the number of assignments to all terms $\mathsf{F}_i(\overline{e}_{ij})$ for all (i,j). After that, all assignments to the function symbols F_i have to be chosen in accordance with those choices to get the correct number of functions that satisfy the formula. Formally, this can be proven as follows:

$$\begin{aligned} f(\mathrm{enc}_\sigma(\mathbf{A})) &= \sum_{\overline{c}\in A^\ell} \sum_{\substack{(F_1,\dots,F_k)\in \\ A^{A^{a_1}}\times\cdots\times A^{A^{a_k}}}} [\![\mathbf{A}\models \varphi(F_1,\dots,F_k,c_1,\dots,c_\ell)]\!] \\ &= \sum_{\overline{c}\in A^\ell} \sum_{\overline{d}\in A^m} \sum_{(F_1,\dots,F_k)\in G_{\overline{c},\overline{d}}} [\![\mathbf{A}\models \varphi'(\overline{d},\overline{c})]\!], \end{aligned}$$

where $G_{\overline{c},\overline{d}} \coloneqq \{(F_1,\dots,F_k)\in A^{A^{a_1}}\times\cdots\times A^{A^{a_k}} \mid \forall(i,j)\colon\ \mathbf{A}, s_{\overline{c},\overline{d}} \models\ y_{ij} = F_i(\overline{e}_{ij})\}$ and $s_{\overline{c},\overline{d}}$ is the assignment mapping the tuples $\overline{x}$ and $\overline{y}$ to $\overline{c}$ and $\overline{d}$, respectively.

Since $[\![\mathbf{A}\models \varphi'(\overline{d},\overline{c})]\!]$ does not depend on $(F_1,\dots,F_k)$, we can multiply by the cardinality of $G_{\overline{c},\overline{d}}$ instead of summing:

$$f(\mathrm{enc}_\sigma(\mathbf{A})) = \sum_{\substack{\overline{c}\in A^\ell,\\ \overline{d}\in A^m}} [\![\mathbf{A}\models \varphi'(\overline{d},\overline{c})]\!]\cdot|G_{\overline{c},\overline{d}}|.$$

Now we are in a position to show $f\in\#\mathrm{AC}^0$.

The sum only has polynomially many summands and thus can obviously be computed in #AC^0.

For $[\![\mathbf{A}\models \varphi'(\overline{d},\overline{c})]\!]$, the circuit only has to evaluate a quantifier-free formula using the assignment $s_{\overline{c},\overline{d}}$. We can ensure that this assignment is part of the representation of the gate computing the corresponding summand. Then, this is similar to the corresponding part of the proof of FO = AC^0 and thus can be done in FAC$^0\subseteq\#\mathrm{AC}^0$.

For $|G_{\overline{c},\overline{d}}|$ we first note that the total number of possible assignments for $\overline{\mathsf{F}}$ is

$$|A^{A^{a_1}}\times\cdots\times A^{A^{a_k}}| = |A|^{\sum_i |A|^{a_i}}.$$

The definition of $G_{\overline{c},\overline{d}}$ fixes for the interpretation of each function symbol F_i the function value on at most m_i inputs to be equal to some d_{ij}. This means, that the function value on at least $|A|^{a_i}-m_i$ inputs is not determined by the definition of $G_{\overline{c},\overline{d}}$ and can thus be chosen arbitrarily.

If for some (i,j), $\overline{e}_{ij}$ is semantically equal to $\overline{e}_{ij'}$ for some $j'<j$, it has to hold that $d_{ij}=d_{ij'}$. Additionally, if such a j' exists this reduces the number of function values that are fixed by the d_{ij} by 1. To make this formal we define for any $(\overline{c},i,j)$

$$S_{\overline{c}ij} \coloneqq \{j' \mid j'<j \text{ and } \mathbf{A}, s_{\overline{c}} \models\ \overline{e}_{ij} = \overline{e}_{ij'}\},$$

where we use the extension of = to tuples for simplicity and $s_{\overline{c}}$ is the assignment mapping $\overline{x}$ to $\overline{c}$. From the above considerations we get

$$|G_{\overline{c},\overline{d}}| = [\![j'\in S_{ij} \text{ implies } d_{ij}=d_{ij'} \text{ for all } i,j,j']\!]\cdot|A|^{\sum_i |A|^{a_i}-m_i}\cdot|A|^{\sum_{ij}[\![S_{ij}\neq\emptyset]\!]}.$$

Since the a_i and m_i are constants and S_{ij} is FO-definable, $|G_{\overline{c},\overline{d}}|$ can be computed in #AC^0. This concludes the proof for $\#\Sigma_0 \subseteq \#\mathrm{AC}^0$.

Note that for any $\#\Sigma_0$-function f defined using a Σ_0-formula without free second-order variables, $f(w)$ is polynomially bounded in $|w|$ for all inputs w. On the other hand, the above proof shows that for any $\#\Sigma_0$-function f defined using a Σ_0-formula over some vocabulary σ with at least one free second-order variable, there are constants $c_i > 0$ such that $f(\mathrm{enc}_\sigma(\mathbf{A}))$ is divisible by $|\mathrm{dom}(\mathbf{A})|^{\sum_i |\mathrm{dom}(\mathbf{A})|^{c_i} - \mathrm{const}}$ for all $\mathbf{A} \in \mathrm{STRUC}[\sigma]$. It follows that the function f defined by

$$f(w) := |w|^{\lceil |w|/2 \rceil}$$

is not in $\#\Sigma_0$, while it is easy to see that $f \in \#\mathrm{AC}^0$, cf. Figure 2.2 on page 33, where a family of counting arithmetic circuits computing this function was sketched as an example. For the sake of contradiction, assume $f \in \#\Sigma_0$ via a formula $\varphi \in \Sigma_0$ over some vocabulary σ. As $f(w) > 0$ if $|w| \in \{1, 2\}$, we have $\sigma = \tau_{\mathrm{string}}$ by Observation 2.20. For any $\mathbf{A} \in \mathrm{STRUC}[\tau_{\mathrm{string}}]$, the length of the encoding of $\mathbf{A}$ is $|\mathrm{enc}_{\tau_{\mathrm{string}}}(\mathbf{A})| = |\mathrm{dom}(\mathbf{A})|$. Hence, we have $f(\mathrm{enc}_{\tau_{\mathrm{string}}}(\mathbf{A})) = |\mathrm{dom}(\mathbf{A})|^{\lceil |\mathrm{dom}(\mathbf{A})|/2 \rceil}$ for all $\mathbf{A} \in \mathrm{STRUC}[\tau_{\mathrm{string}}]$. This is a contradiction to the fact that $f(\mathrm{enc}_{\tau_{\mathrm{string}}}(\mathbf{A}))$ is divisible by $|\mathrm{dom}(\mathbf{A})|^{\sum_i |\mathrm{dom}(\mathbf{A})|^{c_i} - \mathrm{const}}$ for constants c_i. Consequently, $f \notin \#\Sigma_0$, finishing the proof that $\#\Sigma_0 \neq \#\mathrm{AC}^0$. □

So far we have identified the following inclusion structure:

$$\#\Sigma_0 \subsetneq \#\Pi_1^{\mathrm{prefix}} = \#\mathrm{AC}^0 \subsetneq \#\Pi_1 \stackrel{*}{=} \#\mathrm{P}. \tag{5.1}$$

Next, we want to see the relation between $\#\mathrm{AC}^0 = \#\Pi_1^{\mathrm{prefix}}$ and the remaining class of the hierarchy, i.e., $\#\Sigma_1$. We will see that the two classes are incomparable, which at the same time will complete the picture with regard to the relations between the classes of our hierarchy. We begin by showing that $\#\Pi_1^{\mathrm{prefix}}$ contains a function that is not in $\#\Sigma_1$.

Theorem 5.10. $\#\mathrm{AC}^0 \not\subseteq \#\Sigma_1$.

Proof. Let $f : \{0,1\}^+ \to \mathbb{N}$ be the function defined by

$$f(w) := 2^{|w|_1}.$$

Obviously, $f \in \#\mathrm{AC}^0$ as the circuit family simply has to compute the product $\prod_{i=0}^{n-1} 2^{w_i}$, interpreting the symbols 0 and 1 as numbers.

Now, for the sake of contradiction assume that $f \in \#\Sigma_1$. By definition, there is a formula $\varphi(\overline{\mathrm{F}}, \overline{y}) \in \Sigma_1[\leq, \mathrm{BIT}, \mathrm{min}]$ over some vocabulary σ such that for all $\mathbf{A} \in \mathrm{STRUC}[\sigma]$,

$$f(\mathrm{enc}_\sigma(\mathbf{A})) = |\{(\overline{F}, \overline{a}) \mid \mathbf{A} \models \varphi(\overline{F}, \overline{a})\}|,$$

and $f(x) = 0$ if x is not the encoding of a σ-structure. As f maps both a string of length 1 and a string of length 2 to a number > 0, we have $\sigma = \tau_{\mathrm{string}}$ by Observation 2.20. Fix a word $w \in \{0,1\}^+$ and the corresponding τ_{string}-structure $\mathbf{A}_w$. Now consider the

structure $\mathbf{A}'$ obtained from $\mathbf{A}_w$ by extending the domain $\mathrm{dom}(\mathbf{A}_w) = \{0,\dots,|w|-1\}$ by one additional element, that is, $\mathrm{dom}(\mathbf{A}') := \{0,\dots,|w|\}$ without modifying the interpretation of S, that is, $\mathsf{S}^{\mathbf{A}'} := \mathsf{S}^{\mathbf{A}_w}$. This obviously means that $|w|_1 = |\mathrm{enc}_{\tau_{\mathrm{string}}}(\mathbf{A}')|_1$ and hence $f(w) = f(\mathrm{enc}_{\tau_{\mathrm{string}}}(\mathbf{A}'))$.

To make the presentation simpler, suppose $\overline{\mathsf{F}} = \mathsf{F}$ and that the arity of F is one. Any given interpretation $F\colon \mathrm{dom}(\mathbf{A}_w) \to \mathrm{dom}(\mathbf{A}_w)$ of the symbol F can be extended in several ways on the domain $\mathrm{dom}(\mathbf{A}')$, in particular as the following functions F_1 and F_2:

- $F_1(x) := F(x)$ for all $x \in \mathrm{dom}(\mathbf{A}_w)$ and $F_1(n) := 0$.
- $F_2(x) := F(x)$ for all $x \in \mathrm{dom}(\mathbf{A}_w)$ and $F_2(n) := 1$.

Moreover, the interpretations of the built-in predicate symbols for $\mathbf{A}'$ are supersets of those for $\mathbf{A}_w$. As formulae in Σ_1 are preserved under extension of models, if $\overline{a}$ and F are assignments for F and $\overline{y}$, respectively, such that $\mathbf{A}_w \models \varphi(F,\overline{a})$, then $\mathbf{A}' \models \varphi(F_1,\overline{a})$ and $\mathbf{A}' \models \varphi(F_2,\overline{a})$. Therefore,

$$|\{(\overline{F},\overline{a}) \mid \mathbf{A}' \models \varphi(\overline{F},\overline{a})\}| > |\{(\overline{F},\overline{a}) \mid \mathbf{A}_w \models \varphi(\overline{F},\overline{a})\}|.$$

As $f(\mathrm{enc}_{\tau_{\mathrm{string}}}(\mathbf{A}')) = f(\mathrm{enc}_{\tau_{\mathrm{string}}}(\mathbf{A}_w))$, this means that the assumption $f \in \#\Sigma_1$ has led to a contradiction. □

In order to show that $\#\Sigma_1$ is not contained in $\#\mathrm{AC}^0 = \#\Pi_1^{\mathrm{prefix}}$, we will show that a function that is complete for #P under AC^0-Turing reductions is contained in $\#\Sigma_1$. For two counting functions f and g, f is reducible to g by AC^0-Turing reductions if and only if there is an AC^0 oracle circuit that computes f with oracle gates for g, see Section 4.2 for details on oracle circuits in the FO[BIT]-uniform setting. As this function being in #AC^0 would lead to a contradiction to #AC$^0 \neq$ #P, this will show the desired separation. The function we will use is a modified version of the problem #3DNF, so we begin by showing that the original problem is #P-complete under AC^0-Turing reductions. Note that the related problem #DNF, where any number of literals may occur in each term, was shown to be #P-complete under subtractive reductions by Durand et al. [DHK05].

Lemma 5.11. *The function* #3DNF *is complete for* #P *under* AC^0*-Turing reductions.*

Proof. Valiant showed that #Sat as well as #3CNF are #P-complete under polynomial-time Turing reductions [Val79c]. Here, #3CNF is defined analogously to #3DNF using conjunctive normal form instead of disjunctive normal form. The completeness of #Sat directly follows from Cook's original proof for NP-completeness of Sat [Coo71], and can be modified to show completeness under AC^0-Turing reductions. The completeness of #3CNF is based on a transformation that transforms each clause to a set of clauses defined by simple conditions. Using an adequate encoding of formulae, it is easy to see that this reduction can also be done in AC^0.

Now, #3CNF can be reduced to #3DNF in analogy to the proof of #P-completeness of #DNF under subtractive reductions [DHK05]. Given a 3CNF-formula φ over n variables, we first construct the disjunctive normal form φ' of $\neg\varphi$, which is a 3DNF-formula.

Obviously, the number of satisfying assignments of φ is equal to 2^n minus the number of satisfying assignments of φ'. Since this reduction can be computed in AC^0, #3DNF is complete for #P under AC^0-Turing reductions. □

We will now show that $\#\Sigma_1$ is not contained in $\#\Pi_1^{\text{prefix}}$ by showing that a modified version of #3DNF, which is #P-complete under TC^0-Turing reductions, is contained in $\#\Sigma_1$. TC^0-Turing reductions can be defined analogously to AC^0-Turing reductions, again using oracle circuits.

Theorem 5.12. $\#\Sigma_1 \not\subseteq \#AC^0$.

Proof. We prove this statement by modifying the counting problem #3DNF to get a function in $\#\Sigma_1$ that is #P-complete under TC^0-Turing reductions. As FTC^0 is closed under TC^0-Turing reductions (cf. Lemma 4.6), this function cannot be in $\#\Pi_1^{\text{prefix}} = \#AC^0 \subseteq FTC^0$, because this would contradict $FTC^0 \neq \#P$.

Consider the vocabulary σ_{3DNF} and the formula $\Phi_{\#3DNF}(\mathsf{T})$ from Example 5.2. To get a function in $\#\Sigma_1$, we need to use a free function variable instead of the free relation variable T. Since we cannot use universal quantifiers, relations cannot be represented uniquely as functions of the same arity. In order to still get a #P-complete problem, we want to make sure that compared to #3DNF, the function value of our new counting function only differs by a factor depending on the input length, not on the number of satisfying assignments. To achieve this, we encode any relation T interpreting T as a function F as follows: Interpret for all x an even function value $F(x)$ as $x \notin T$ and an odd function value $F(x)$ as $x \in T$. If the cardinality of the universe is even, this ensures that the numbers of 1's and 0's in a satisfying assignment do not influence the factor by which the new counting function differs from #3DNF.

Following this idea we define for all σ-structures $\mathbf{A}$

$$\#3DNF^{\text{func}}(\text{enc}_\sigma(\mathbf{A})) := |\{F \mid \mathbf{A} \models \Phi_{\#3DNF^{\text{func}}}(F)\}|,$$

where $\Phi_{\#3DNF^{\text{func}}}(\mathsf{F})$ is obtained from $\Phi_{\#3DNF}(\mathsf{T})$ by replacing for all variables x subformulae of the form $\mathsf{T}(x)$ by $\text{BIT}(\min, \mathsf{F}(x))$. By definition, $\#3DNF^{\text{func}} \in \#\Sigma_1$.

We now show that #3DNF is reducible to $\#3DNF^{\text{func}}$ by TC^0-Turing reductions. Since the idea above only works if the universe has even cardinality, the first step of the reduction is doubling the cardinality of the universe. Let $\mathbf{A}$ be a structure and $\mathbf{A}'$ the structure that arises from $\mathbf{A}$ by doubling the cardinality of the universe. Let $A := \{0, \dots, n-1\}$ and $A' := \{0, \dots, 2n-1\}$ with $n := \text{dom}(\mathbf{A})$ be their respective universes. Each satisfying assignment T of $\Phi_{\#3DNF}$, i.e., each T with $\mathbf{A} \models \Phi_{\#3DNF}(T)$, gives rise to the following set of satisfying assignments F for $\Phi_{\#3DNF^{\text{func}}}$, i.e., set of assignments F with $\mathbf{A}' \models \Phi_{\#3DNF^{\text{func}}}(F)$:

$$S_T := \{F \colon A' \to A' \mid \text{for all } x \in A \colon F(x) \equiv 1 \mod 2 \text{ iff } T(x)\}.$$

These sets are disjoint and by definition of $\Phi_{\#3DNF^{\text{func}}}(\mathsf{F})$ their union is equal to $\{F \mid \mathbf{A}' \models \Phi_{\#3DNF^{\text{func}}}(F)\}$.

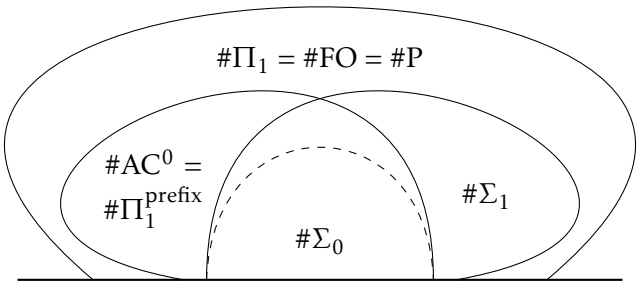

Figure 5.1: Alternation hierarchy in #FO and relationship to #AC^0. Dashed lines indicate that separations are not known.

For each satisfying assignment T, the set S_T contains all functions F mapping each element $x \in A$ to an arbitrary odd element if $x \in T$ and to an arbitrary even element if $x \notin T$, and any element $x \notin A$ to an arbitrary element. This means that there are n possible values for any $x \in A$ and $2n$ possible values for any $x \notin A$. Consequently, we have $|S_T| = |A|^{|A|} \cdot (2 \cdot |A|)^{|A|}$ for all T, meaning that #3DNF and #3DNF$^{\text{func}}$ are related as follows:

$$\#3\text{DNF}(\text{enc}_{\sigma_{3\text{DNF}}}(\mathbf{A})) = \frac{\#3\text{DNF}^{\text{func}}(\text{enc}_{\sigma_{3\text{DNF}}}(\mathbf{A}'))}{|A|^{2|A|} \cdot 2^{|A|}}.$$

Doubling the cardinality of the universe can be done in FTC^0 by adding the adequate number of 0-entries in the encodings of all relations. The term $|A|^{2|A|} \cdot 2^{|A|}$ can be computed in #AC$^0 \subseteq$ FTC^0 and division can be done in FTC^0.

Since #3DNF is #P-complete under AC^0-Turing reductions by Lemma 5.11, this means that #3DNF$^{\text{func}}$ is #P-complete under TC^0-Turing reductions. □

Combining Lemmas 5.10 and 5.12, we get that #Σ_1 and #AC^0 = #Π_1^{prefix} are incomparable. In combination with Equation 5.1, this also separates #Σ_1 from both #Σ_0 and #Π_1. We summarize our results so far in the following Corollary, and illustrate them in the inclusion diagram in Figure 5.1.

Corollary 5.13.

$$\begin{array}{ccccc} & & \#\text{AC}^0 = \#\Pi_1^{\text{prefix}} & & \\ & \nearrow & & \nwarrow & \\ \#\Sigma_0 & & \not\subseteq \not\supseteq & & \#\Pi_1 = \#\text{FO} \stackrel{*}{=} \#\text{P}. \\ & \nwarrow & & \nearrow & \\ & & \#\Sigma_1 & & \end{array}$$

Proof. The equality #Π_1 = #FO $\stackrel{*}{=}$ #P is from Theorems 5.7 and 5.4. From Corollary 4.4 and Lemma 5.6 we get #AC^0 = #Π_1^{prefix}. In Corollary 5.8 we have seen that #$\Pi_1^{\text{prefix}} \subsetneq$ #Π_1. Theorem 5.9 shows that #$\Sigma_0 \subsetneq$ #AC^0. In Lemmas 5.10 and 5.12 it was shown that #AC^0 and #Σ_1 are incomparable. The inclusions #$\Sigma_0 \subseteq$ #$\Sigma_1 \subseteq$ #FO hold by definition.

Finally, $\#\Sigma_0 \subseteq \#AC^0$ together with $\#\Sigma_1 \not\subseteq \#AC^0$ shows $\#\Sigma_1 \neq \#\Sigma_0$ and $\#\Pi_1^{\text{prefix}} \subseteq \#\Pi_1$ together with $\#AC^0 \not\subseteq \#\Sigma_1$ shows $\#\Sigma_1 \neq \#\Pi_1$. □

Note that it is easy to see that all of our separations also separate the respective classes in terms of $\overset{*}{=}$, as both the classes in our hierarchy as well as the class $\#AC^0$ only contain functions mapping inputs of length 1 to either 0 or 1.

5.3 Feasibility of $\#\Sigma_1$

An interesting research direction is identifying feasible subclasses of #P that contain important counting problems, potentially even problems that are complete for #P under some type of reduction. In this vein, one of the main goals of Saluja et al. in their paper [SST95] was to identify such classes in their hierarchy. They showed that functions from $\#\Sigma_1^{\text{rel}}$, a class containing #P-complete problems, and also functions from some syntactic fragment $\#R\Sigma_2^{\text{rel}}$ of their class $\#\Sigma_2^{\text{rel}}$, are subclasses of FPRAS [JVV86]. This class contains all functions that have a so-called fully polynomial-time randomized approximation scheme and will be formally defined in Definition 5.14.

With regard to our hierarchy, we have already seen that $\#\Sigma_0$ is a very weak class, as it is strictly contained in the circuit class $\#AC^0$. At the other end of the spectrum, the class $\#\Pi_1$ is already as powerful as the whole class #FO. In between we have the class $\#\Sigma_1$. The complexity of this class is not clear yet: On one hand we know that it does not contain the whole class $\#AC^0$, on the other hand we have seen that it contains a #P-complete problem.

In this section we show that functions in $\#\Sigma_1$ are tractable in the same sense of approximability as those in $\#\Sigma_1^{\text{rel}}$, that is, $\#\Sigma_1 \subseteq$ FPRAS. As an intermediate step we consider the satisfiability problem for a restricted form of quantifier-free DNF-sentences over functional vocabularies restricted to structures whose domain have a given cardinality. It turns out that functions in $\#\Sigma_1$ are reducible under an adequate type of reduction to counting satisfying assignments of formulae in this setting, more precisely to a problem we call #k-PDNF(m). Finally, we will see that this problem is in FPRAS, showing that $\#\Sigma_1 \subseteq$ FPRAS. This approach is to a degree in analogy to the approach by Saluja et al. showing that every problem in $\#\Sigma_1^{\text{rel}}$ has an FPRAS. As a new tool we introduce the problem #k-PDNF(m).

We now start by formally defining the class FPRAS, which was introduced in 1986 by Jerrum et al. [JVV86]. For this, probabilistic Turing machines are used.

Definition 5.14. A function $f : \{0,1\}^* \to \mathbb{N}$ is in FPRAS, if it has a *fully polynomial-time randomized approximation scheme* (or FPRAS), that is, if there is a probabilistic Turing machine M working on inputs (x, ε) with $x \in \{0,1\}^*$ and $\varepsilon \in \mathbb{Q}_+$ with $0 < \varepsilon < 1$ such that for all such inputs:

- $\mathcal{P}\Big(|M(x,\varepsilon) - f(x)| > \varepsilon \cdot |f(x)|\Big) < \frac{1}{4}$, where $M(x,\varepsilon)$ is the random variable describing the output of M on input (x,ε),
- the running time of M on input (x,ε) is bounded by a polynomial in $|x|, \frac{1}{\varepsilon}$.

Algorithm 1: Algorithm showing that FPRAS is closed under $\leq_{\mathbb{Q}\text{-pr}}$.

Input: (w, ε), where $w \in \{0,1\}^*$ and $\varepsilon \in \mathbb{Q}_+$ with $0 < \varepsilon < 1$
1 Compute in polynomial time $r(w)$
2 $t \leftarrow \varepsilon$-approximation of $g(r(w))$ using the FPRAS for g
3 **return** $h(w) \cdot t$

Here, $\mathcal{P}(A)$ is the probability of event A and $\mathbb{Q}_+$ denotes the set of positive real numbers. A suitable reducibility in this context is $\leq_{\mathbb{Q}\text{-pr}}$, as FPRAS can be shown to be closed under this reducibility.

Definition 5.15. Let f, g be counting problems. We say f *is polynomial-time rational-product reducible to* g, denoted by $f \leq_{\mathbb{Q}\text{-pr}} g$, if there are polynomial-time computable functions $r\colon \{0,1\}^* \to \{0,1\}^*$ and $h\colon \{0,1\}^* \to \mathbb{Q}_+$ such that for all $w \in \{0,1\}^*$:

$$f(w) = h(w) \cdot g(r(w)).$$

While an obvious observation, we include the proof that FPRAS is closed under the above reducibility for the sake of completeness.

Lemma 5.16. *The class* FPRAS *is closed under* $\leq_{\mathbb{Q}\text{-pr}}$*, that is, if* f *and* g *are counting problems,* $f \leq_{\mathbb{Q}\text{-pr}} g$ *and* $g \in$ FPRAS*, then* $f \in$ FPRAS*.*

Proof. Let r and h be as in Definition 5.15 and $f(w) = h(w) \cdot g(r(w))$. Then Algorithm 1 is an FPRAS for f.

The running-time of the FPRAS for g used in this algorithm is polynomial in $|r(w)|$, $1/\varepsilon$ and hence also in $|w|$, $1/\varepsilon$. We now have

$$\begin{aligned} |g(r(w)) - t| \leq \varepsilon \cdot g(r(w)) &\Leftrightarrow h(w) \cdot |g(r(w)) - t| \leq \varepsilon \cdot h(w) \cdot g(r(w)) \\ &\Leftrightarrow |h(w) \cdot g(r(w)) - h(w) \cdot t| \leq \varepsilon \cdot h(w) \cdot g(r(w)) \\ &\Leftrightarrow |f(w) - h(w) \cdot t| \leq \varepsilon \cdot f(w). \end{aligned}$$

This means the error of the new FPRAS for f on input w is bounded by ε if and only if the error of the FPRAS for g on input $r(w)$ is bounded by ε. As Algorithm 1 does not use any additional randomness outside the FPRAS for g, the error probability is still bounded by $\frac{1}{4}$, finishing the proof. □

In order to show that $\#\Sigma_1 \subseteq$ FPRAS, it is helpful to reduce any given $\#\Sigma_1$-problem to a suitable satisfiability problem. Let $f \in \#\Sigma_1$ via some formula φ. As the reduction gets a structure as input, we can already evaluate all parts of φ that only depend on the input in the reduction and replace the existential quantifiers in φ by a large disjunction. What remains is a Boolean combination of atoms involving free function variables. It can be shown that in this way we can arrive at a quantifier-free formula in DNF where atoms are of a very restricted form. This idea is similar to a reduction to restricted DNF-formulae used by Saluja et al. [SST95] to show $\#\Sigma_1^{\text{rel}} \subseteq$ FPRAS. We now introduce a suitable

satisfiability problem (and related counting problem), which is based on satisfiability for a restricted form of quantifier-free DNF-sentences over functional vocabularies.

As the vocabulary will not be fixed, but be implicitly given by the formula, we need to fix a countable set of possible function symbols in advance. Let $\mathfrak{F} := \{\mathsf{F}_i \mid i \in \mathbb{N}\}$ be that countable set of function symbols. The arity of these function symbols is not fixed in advance, but is given implicitly by the formula. In the context of a given formula we use $\mathrm{ar}(\mathsf{F}_i)$ to denote the arity of symbol F_i in that formula. Let pseudo-DNF (or PDNF) be the class of quantifier-free FO-sentences in DNF over any functional vocabulary consisting only of function symbols from $\mathfrak{F}$, using arbitrary constants, and in which all atoms are of the form $\mathsf{F}(\overline{a}) = b$ for some $\mathsf{F} \in \mathfrak{F}$, a tuple of constants $\overline{a}$, and a constant b. As we always assume the universe to be of the form $\{0,\dots,n-1\}$ for some $n \in \mathbb{N}$, the constants are natural numbers. Formally, pseudo-DNF sentences are of the form

$$\varphi = \bigvee_{i=1}^{m_1} \bigwedge_{j=1}^{m_2} \ell_{i,j},$$

where for all i, j, the literal $\ell_{i,j}$ is either of the form $\mathsf{F}_{i,j}(\overline{a}_{i,j}) = b_{i,j}$ or of the form $\mathsf{F}_{i,j}(\overline{a}_{i,j}) \neq b_{i,j}$ for some $\mathsf{F}_{i,j} \in \mathfrak{F}$, $\overline{a}_{i,j} \in \mathbb{N}^{\mathrm{ar}(\mathsf{F}_i)}$ and $b_{i,j} \in \mathbb{N}$.

We now define the satisfiability problem for pseudo-DNF sentences where the cardinality of the universe is given as part of the input in unary, as well as its counting version. As in the rest of this thesis, we assume that the universe of any structure $\mathbf{A}$ with $|\mathrm{dom}(\mathbf{A})| = n$ is $\{0,\dots,n-1\}$ for any $n \in \mathbb{N}$.

Problem:	PDNF-Sat
Input:	$(1^n, \varphi)$, where $n \in \mathbb{N}$, φ is a pseudo-DNF sentence and all constants occurring in φ are bounded by $n-1$
Question:	Is there a model $\mathbf{A}$ of φ with $\mathrm{dom}(\mathbf{A}) = \{0,\dots,n-1\}$?

Problem:	#PDNF
Input:	$(1^n, \varphi)$, where $n \in \mathbb{N}$, φ is a pseudo-DNF sentence and all numbers occurring in φ are bounded by $n-1$
Output:	Number of models $\mathbf{A}$ of φ with $\mathrm{dom}(\mathbf{A}) = \{0,\dots,n-1\}$

For any fixed $\#\Sigma_1$-problem, the arity of functions in a formula showing its membership in the class as well as the number of literals in each term if transformed to a DNF are constant. Thus, we will consider PDNFs with a bounded number of literals in each term and bounded arity of function symbols. Let k-PDNF(m) be the class of PDNFs where each term consists of at most k literals and the arity of function symbols is bounded by m. Furthermore, let k-PDNF(m)-Sat and #k-PDNF(m) be the above problems restricted to the corresponding inputs.

To get some idea of the complexity of the problem #k-PDNF(m), we briefly argue that for $k \geq 2$, #k-PDNF(m) is #P-complete under subtractive reductions (see [DHK05] for a definition of this reducibility): Membership is immediate. For hardness one can introduce the analogous class of Pseudo-CNFs and the related problems. Encoding a Boolean

assignment as a function of arity 1 and ensuring that this function only maps to 0 or 1 in any satisfying assignment, it is straightforward to show that #k-CNF $\leq$ #k-PCNF(1) under parsimonious reductions. By using the same idea as for the reduction from #k-CNF to #k-DNF one can then show that #k-PDNF(1) is #P-hard under subtractive reductions.

We will now show that our new problem #k-PDNF(m) has the desired property, that is, any #Σ_1-problem can be reduced to #k-PDNF(m) for some $k, m \in \mathbb{N}$.

Lemma 5.17. *For all $f \in \#\Sigma_1$ there are $k, m \in \mathbb{N}$ such that $f \leq_{\mathbb{Q}\text{-pr}}$ #k-PDNF(m).*

Proof. Let $f \in \#\Sigma_1$ via $\exists \overline{y}\, \psi(\overline{y}, \overline{z}, \overline{\mathsf{G}})$ over vocabulary σ, where ψ is quantifier-free, that is, for all $\mathbf{A} \in \mathrm{STRUC}[\sigma]$

$$f(\mathrm{enc}_\sigma(\mathbf{A})) = |\{(\overline{G}, \overline{a}) \mid \mathbf{A} \models \exists \overline{y}\, \psi(\overline{y}, \overline{a}, \overline{G})\}|,$$

and $f(x) = 0$ if x is not the encoding of a σ-structure. Inputs that are not encodings of σ-structures can handled by mapping them to a fixed instance without any satisfying assignments. In the following, we cover the case where the input is (the encoding of) a σ-structure. By the introduction of fresh, existentially quantified variables we may, without loss of generality, assume that ψ is in DNF and all occurrences of function symbols in ψ are of the form $\mathsf{F}(\overline{x}_1) = x_2$ or $\mathsf{F}(\overline{x}_1) \neq x_2$, where F is a function symbol, $\overline{x}_1$ a tuple of first-order variables of appropriate arity, and x_2 a first-order variable. Let t be a bound on the number of literals per term in the DNF ψ. Let m be the highest arity among function symbols in $\overline{\mathsf{G}}$. Furthermore, let p be the arity of $\overline{y}$ and q be the arity of $\overline{z}$. Now let $\mathbf{A} \in \mathrm{STRUC}[\sigma]$ and $n := |\mathrm{dom}(\mathbf{A})|$, i.e., $\mathrm{dom}(\mathbf{A}) = \{0, \ldots, n-1\}$. Also, let $\mathrm{dom}(\mathbf{A})^p =: \{y_1, \ldots, y_{n^p}\}$ and $\mathrm{dom}(\mathbf{A})^q =: \{z_0, \ldots, z_{n^q-1}\}$.

Now, for each z_i, write the meaning of the block of existential quantifiers $\exists \overline{y}$ out as a disjunction of polynomial size:

$$\exists \overline{y}\, \psi(\overline{y}, z_i, \overline{\mathsf{G}}) \rightsquigarrow \bigvee_{j=1}^{n^p} \psi(y_j, z_i, \overline{\mathsf{G}}).$$

Define $\psi'_{z_i}(\overline{\mathsf{G}})$ to be this formula after evaluating according to $\mathbf{A}$ and the assignment s mapping $\overline{z}$ to z_i. That means, only a Boolean combination of atomic formulae of the form $\mathsf{G}_i(\overline{b}) = c$ and $\mathsf{G}_i(\overline{b}) \neq c$ remains. Here, $\overline{b}$ is a tuple of constants and c is a constant, as variables have been replaced by constants according to s. All atoms not involving free function variables have been evaluated according to $\mathbf{A}$ and s and the Boolean parts have also been simplified if possible.

We now use a tuple of new function symbols $\overline{\mathsf{H}}$ to sum over the different values for z using the following disjoint disjunction:

$$\theta_{\mathbf{A}} := [\psi'_{z_0}(\overline{\mathsf{G}}) \wedge \overline{\mathsf{H}}(0) = \overline{0}] \vee \ldots \vee [\psi'_{z_{n^q-1}}(\overline{\mathsf{G}}) \wedge \overline{\mathsf{H}}(0) = z_{n^q-1}].$$

Let $\theta'_{\mathbf{A}}$ be the formula obtained by transforming $\theta_{\mathbf{A}}$ into a PDNF in the obvious way. For example, the subformula $[\psi'_{z_0}(\overline{\mathsf{G}}) \wedge \overline{\mathsf{H}}(0) = z_0]$ is replaced by the DNF $\psi'_{z_0}(\overline{\mathsf{G}})$ with

$\overline{\mathsf{H}}(0) = z_0$ added to each term and function symbols are renamed adequately to ensure that all function symbols are from $\mathfrak{F}$. As we are interested in the number of satisfying assignments in $\mathbf{A}$, the desired cardinality of domains given to #k-PDNF(m) should be $n = |\mathrm{dom}(\mathbf{A})|$.

In the formula $\theta'_{\mathbf{A}}$, the assignment to $\overline{\mathsf{H}}$ determines the assignment to the tuple $\overline{z}$, which allows us to sum the number of satisfying assignments to G over all assignments to $\overline{z}$. More precisely, for an assignment mapping $\overline{H}$ to $\overline{\mathsf{H}}$ with $\overline{H}(0) = z_i$, the number of assignments to $\overline{\mathsf{G}}$ that in combination with $\overline{H}$ form a model of $\theta'_{\mathbf{A}}$ is exactly the number of assignments to $\overline{\mathsf{G}}$ satisfying $\exists y\, \psi(\overline{y}, z_i, \overline{G})$, as the atoms $\overline{\mathsf{H}}(0) = z_i$ act as selectors. This means that the number of models with domains of cardinality n for $\theta'_{\mathbf{A}}$ is exactly $f(\mathrm{enc}_\sigma(\mathbf{A}))$ times the number of tuples $\overline{H}$ of functions on $\{0,\dots,n-1\}$ with a fixed output on input 0. From these considerations we get

$$\#\mathrm{PDNF}(\theta'_{\mathbf{A}}, 1^n) = \left(n^{(n-1)\cdot q}\right) \cdot f(\mathrm{enc}_\sigma(\mathbf{A})).$$

Note that the number of different function symbols, the arity of function symbols and the number of literals in each term are bounded as follows: The tuple $\overline{\mathsf{H}}$ is of size q and each function symbol in this tuple has arity 1. The arity of function symbols in $\overline{\mathsf{G}}$ is bounded by m. Furthermore, the number of literals in each term in $\theta'_{\mathbf{A}}$ is bounded by $t+q$ and hence $\theta'_{\mathbf{A}}$ is a $(t+q)$-PDNF(m)-formula. Since $(\theta'_{\mathbf{A}}, 1^n)$ and the function $n^{(n-1)\cdot q}$ (for constant q) are computable in polynomial time, this proves $f \leq_{\mathbb{Q}\text{-pr}} \#(t+q)\text{-PDNF}(m)$. □

Lemma 5.17 and its proof are inspired by a lemma of Saluja et al. [SST95] reducing functions in $\#\Sigma_1^{\mathrm{rel}}$ to a restricted variant of #DNF. However, since they use satisfiability of propositional formulae and a Boolean encoding of the different values of z, they need logarithmic clause width (#$k \cdot \log$DNF).

In order to show #k-PDNF(m) $\in$ FPRAS for all $k, m \in \mathbb{N}$, we first need the following Chernoff bound by Mitzenmacher and Upfal [MU17].

Lemma 5.18. *Let $X_1,\dots,X_n$ be independent Poisson trials such that $\mathcal{P}(X_i = 1) = p_i$. Let $X := \sum_{i=1}^n X_i$ and $\mu := \mathbb{E}[X]$. For $0 < \delta < 1$,*

$$\mathcal{P}(|X - \mu| \geq \delta\mu) \leq 2e^{-\mu\delta^2/3}.$$

Here, $\mathbb{E}[X]$ is the expected value of the random variable X.

Finally, we will prove that the number of models of a k-PDNF(m) whose domain has the given cardinality can be approximated by an FPRAS for any $k, m \in \mathbb{N}$.

Lemma 5.19. #k-PDNF(m) $\in$ FPRAS *for all $k, m \in \mathbb{N}$.*

Proof. Let $k, m \in \mathbb{N}$ and let $(\varphi, 1^n)$ over vocabulary σ be an input for #k-PDNF(m). This means that all numbers occurring in φ are bounded by $n-1$. Let ℓ be the number of function symbols occurring in φ. Without loss of generality, let the function symbols occurring in φ be the symbols F_i with arities $a_i := \mathrm{ar}(\mathsf{F}_i)$, where $1 \leq i \leq \ell$. The total number of σ-structures with domains of cardinality n is

$$\prod_{i=1}^{\ell} n^{n^{a_i}} = n^{\sum_{i=1}^{\ell} n^{a_i}}.$$

Now, if φ is satisfiable, then there is a satisfiable term θ in φ. The term θ is a conjunction of at most k atoms or negated atoms and each of them fixes at most the output of one function on one input. This means that at least $n^{\sum_i n^{a_i}-k}$ σ-structures with domains of cardinality n satisfy θ, which is at least a $\frac{1}{n^k}$ fraction.

Let X be the $\{0,1\}$-valued random variable obtained by picking a σ-structure with domain of cardinality n uniformly at random and returning 1 if and only if that structure satisfies φ. Let p be the probability of X being 1. Then $\mathbb{E}[X] = p$. We now apply Lemma 5.18: Let $t \in \mathbb{N}$ and $X_1, \ldots, X_t$ be independent instances of X. By Lemma 5.18, for every $1 > \varepsilon > 0$ it holds that

$$\mathcal{P}\left(\left|\sum_{i=1}^{t} X_i - \mathbb{E}[\sum_{i=1}^{t} X_i]\right| > \varepsilon \cdot \mathbb{E}[\sum_{i=1}^{t} X_i]\right) \leq 2e^{-\mathbb{E}[\sum_{i=1}^{t} X_i]\varepsilon^2/3}.$$

Dividing the inequality inside $\mathcal{P}(\cdot)$ by t and using $\frac{\mathbb{E}[\sum_{i=1}^{t} X_i]}{t} = \mathbb{E}[X] = p$, we obtain

$$\mathcal{P}\left(\left|\frac{\sum_{i=1}^{t} X_i}{t} - p\right| > \varepsilon \cdot p\right) \leq 2e^{-pt\varepsilon^2/3}.$$

This means that we can approximate p by $\frac{\sum_{i=1}^{t} X_i}{t}$ with the desired error probability by choosing t such that the right-hand side is bounded by $\frac{1}{4}$. This yields $t > \frac{3 \cdot \ln 8}{p\varepsilon^2}$. Since for $p > 0$ we have $p \geq \frac{1}{n^k}$, we choose

$$t := \left\lceil \frac{3\lceil \ln 8 \rceil \cdot n^k}{\varepsilon^2} \right\rceil + 1.$$

For $p = 0$ there will never be an error with any number of trials.

Now t is polynomially bounded in $\frac{1}{\varepsilon}$ and n and computable in polynomial time. We can now approximate the number of models by the total number of σ-structures with domain of cardinality n times the above approximation of p. By these considerations, Algorithm 2 is an FPRAS for #k-PDNF(m).

□

Combining the previous results we immediately get that each function in #Σ_1 can be approximated by an FPRAS.

Corollary 5.20. #$\Sigma_1 \subseteq$ FPRAS.

5.4 Hierarchy Based on the Number of Universal Quantifiers

We have seen in the previous section that the strict subclass #Σ_1 of #FO is feasible in the sense of FPRAS. In order to potentially identify more interesting subclasses of #FO one can investigate hierarchies in #FO based on different syntactic restrictions besides limiting the number of alternations. In Section 5.2, we have seen that the alternation

Algorithm 2: FPRAS for #k-PDNF(m).

Input: $(\varphi, 1^n, \varepsilon)$, where φ is a k-PDNF(m) and $\varepsilon \in \mathbb{Q}_+$ with $0 < \varepsilon < 1$

```
sat ← 0
t ← ⌈ 3⌈ln 8⌉·n^k / ε^2 ⌉ + 1
for i ← 1 to t do
    Pick a σ-structure A with dom(A) = {0,...,n−1} uniformly at random
    if A ⊨ φ then
        sat ← sat + 1
Let {F_1,...,F_ℓ} be the function symbols occurring in φ
return n^(Σ_{i=1}^{ℓ} n^ar(F_i)) · sat/t
```

hierarchy in #FO $\stackrel{*}{=}$ #P collapses to the class #Π_1. For this reason, we focus our attention on a hierarchy inside #Π_1 in this section: We study the hierarchy obtained by restricting the number of universal variables.

Let $\Pi_1(k\forall)$ denote the class of Π_1 formulae of the form

$$\forall x_1 \cdots \forall x_k \, \psi,$$

where ψ is a quantifier-free formula. In line with our notation, the function class corresponding to $\Pi_1(k\forall)$ is denoted by #$\Pi_1(k\forall)$. It is clear from the definition that $\Pi_1(k\forall) \subseteq \Pi_1((k+1)\forall)$, and consequently also #$\Pi_1(k\forall) \subseteq$ #$\Pi_1((k+1)\forall)$, for all $k \in \mathbb{N}$. We will initiate the study of the hierarchy of classes #$\Pi_1(k\forall)$ by showing that these inclusions are actually strict, i.e.,

$$\#\Pi_1(k\forall) \subsetneq \#\Pi_1((k+1)\forall) \text{ for all } k \in \mathbb{N}. \tag{5.2}$$

Grandjean and Olive considered a hierarchy of decision classes closely related to the hierarchy above [GO04]. They studied classes ESO$^\sigma(k\forall)$ for $k \in \mathbb{N}$ and vocabulary σ. Their classes are similar to the decision version of our classes #$\Pi_1(k\forall)$, but allow to quantify both functions and relations and are restricted to a single vocabulary.

Definition 5.21. We denote by ESO$^\sigma(k\forall)$ the class of ESO-sentences over vocabulary σ of the form

$$\exists \mathsf{R}_1 \ldots \exists \mathsf{R}_n \forall x_1 \ldots \forall x_k \, \psi,$$

where $n \in \mathbb{N}$, R_i is a relation or function variable for all i, x_i is a first-order variable for all i, and ψ is a quantifier-free formula over σ using the additional relation and function symbols $\mathsf{R}_1, \ldots, \mathsf{R}_n$. Here, function variables can also be 0-ary.

Note that no built-in predicates are used in this definition. As usual, we use ESO$^\sigma(k\forall)$ to also denote the class of languages definable by ESO$^\sigma(k\forall)$-sentences. Grandjean and Olive showed that for every vocabulary σ and all $k \geq 1$,

$$\text{ESO}^\sigma(k\forall) = \text{NTIME}^\sigma_{\text{RAM}}(n^k),$$

where $\mathrm{NTIME}^{\sigma}_{\mathrm{RAM}}(n^k)$ is the class of σ-structures that can be recognized by a nondeterministic RAM in time $O(n^k)$ [GO04]. The hierarchy theorem for nondeterministic time complexity by Cook [Coo73] applies to these classes, yielding

$$\mathrm{NTIME}_{\mathrm{RAM}}(n^k) \subsetneq \mathrm{NTIME}_{\mathrm{RAM}}(n^{k+1}) \text{ for all } k.$$

We will now use these results to show that the hierarchy based on the number of universal variables is also strict in our setting (see Equation (5.2)). For all $k \in \mathbb{N}$, we denote by $\mathrm{ESO}^{\sigma}_{\mathrm{f}}(k\forall)$ the class of $\mathrm{ESO}^{\sigma}(k\forall)$-formulae where all second-order variables are function variables, as well as the class of languages definable by such formulae. We use our usual notation for built-in predicates. Note that the class $\mathrm{ESO}^{\sigma}_{\mathrm{f}}(k\forall)[\leq, \mathrm{BIT}, \min]$ is exactly the decision version of $\#\Pi_1(k\forall)$ limited to vocabulary σ. Next, we show that this class coincides with $\mathrm{ESO}^{\sigma}(k\forall) = \mathrm{NTIME}^{\sigma}_{\mathrm{RAM}}(n^k)$, if $k \geq 1$.

Lemma 5.22. $\mathrm{ESO}^{\sigma}_{\mathrm{f}}(k\forall)[\leq, \mathrm{BIT}, \min] = \mathrm{ESO}^{\sigma}(k\forall)$ *for all vocabularies* σ *and* $k \geq 1$.

Proof. Let $k \in \mathbb{N}$ and $k \geq 1$. The inclusion $\mathrm{ESO}^{\sigma}_{\mathrm{f}}(k\forall) \subseteq \mathrm{ESO}^{\sigma}(k\forall)$ holds by definition. As $\mathrm{ESO}^{\sigma}(k\forall) = \mathrm{NTIME}^{\sigma}_{\mathrm{RAM}}(n^k)$, it is clear that the built-in constant min can be handled. Furthermore, the class $\mathrm{NTIME}^{\sigma}_{\mathrm{RAM}}(n^k)$ is robust in the sense that additional commands computable in linear time on multitape Turing machines can be added without changing the class (when a bound for the values the machine works with is used), cf. the discussion of robustness of their classes by Grandjean and Olive [GO04]. Hence, the built-in predicates $\leq$ and BIT can be handled by adding corresponding commands to the computational model, showing $\mathrm{ESO}^{\sigma}_{\mathrm{f}}(k\forall)[\leq, \mathrm{BIT}, \min] \subseteq \mathrm{ESO}^{\sigma}(k\forall)$.

For $\mathrm{ESO}^{\sigma}(k\forall) \subseteq \mathrm{ESO}^{\sigma}_{\mathrm{f}}(k\forall)[\leq, \mathrm{BIT}, \min]$, let $L \in \mathrm{ESO}^{\sigma}(k\forall)$ via the $\mathrm{ESO}^{\sigma}(k\forall)$-formula φ. Without loss of generality, we can assume that φ is of the form

$$\varphi = \exists \mathsf{R}_1 \ldots \exists \mathsf{R}_n \exists \mathsf{F}_1 \ldots \exists \mathsf{F}_m \forall x_1 \ldots \forall x_\ell \, \psi$$

for some $k, m, \ell \in \mathbb{N}$, where the R_i are relation variables, the F_i are function variables and ψ is a quantifier-free formula over σ. We can also assume that in ψ, arguments of relations and functions are only variables by replacing any argument of a function by a new existentially quantified 0-ary function that is forced to be equal to the respective term (cf. proof of Lemma 5.17).

Now, for input structures with domains of cardinality > 1, we can simulate relations by functions by replacing occurrences of atoms $\mathsf{R}(\overline{x})$ for some relation variable R and tuple of variables $\overline{x}$ in the formula by an atom $\mathsf{F}_{\mathsf{R}}(\overline{x}) > \min$ for a new existentially quantified function symbol F_{R} of the same arity as R.

There remains the case of input structures with domains of cardinality 1. For this we show that for any formula in $\varphi \in \mathrm{ESO}^{\sigma}_{\mathrm{f}}(k\forall)[\leq, \mathrm{BIT}, \min]$, there is a formula $\varphi' \in \mathrm{ESO}^{\sigma}_{\mathrm{f}}(k\forall)[\leq, \mathrm{BIT}, \min]$ that is equivalent to φ on inputs with domains of cardinality > 1 but has any desired behavior on inputs with domains of cardinality 1. We show this for $\sigma = \tau_{\mathrm{string}}$, but the proof easily generalizes to arbitrary vocabularies.

Let $\varphi \in \mathrm{ESO}^{\tau_{\mathrm{string}}}_{\mathrm{f}}(k\forall)[\leq, \mathrm{BIT}, \min]$, that is, φ is of the form

$$\varphi = \exists \overline{\mathsf{F}} \forall x_0 \forall \overline{x} \, \psi,$$

where $\overline{\mathsf{F}}$ is a tuple of function variables, x_0 is a first-order variable, $\overline{x}$ is a (potentially empty) tuple of first-order variables, and ψ is a quantifier-free formula. In order to determine whether the domain of the current structure is of cardinality 1, we introduce a new existentially quantified 0-ary function variable that will be forced to be assigned to the maximal element of the domain. We can then compare min and max to determine whether the domain is of cardinality 1. Now, the desired formula is

$$\varphi' := \exists\overline{\mathsf{F}}\exists\mathsf{max}\forall x_0\forall\overline{x}\Big(x_0 \leq \mathsf{max} \wedge (\mathsf{min} = \mathsf{max} \rightarrow \theta) \wedge (\neg\,\mathsf{min} = \mathsf{max} \rightarrow \psi)\Big),$$

where θ is a quantifier-free formula expressing that the input structure has the desired properties if its domain has cardinality 1. For example, if we want the formula to be satisfied by the τ_{string}-structure $\mathbf{A}_1$, but not $\mathbf{A}_0$, we define $\theta := \mathsf{S}(\mathsf{min})$. □

We are now in a position to prove the strictness of our hierarchy.

Theorem 5.23. *$\#\Pi_1(k\forall) \subsetneq \#\Pi_1((k+1)\forall)$ for all $k \in \mathbb{N}$.*

Proof. The case where $k = 0$ can be handled using properties of $\#\Sigma_0$ observed in the proof of Theorem 5.9: For any $f \in \#\Pi_1(0\forall) = \#\Sigma_0$, we know that $f(w)$ is either polynomially bounded in $|w|$ for all w, or f can be defined using a Σ_0-formula over some vocabulary σ and $f(\text{enc}_\sigma(\mathbf{A}))$ is divisible by $|\text{dom}(\mathbf{A})|^{\sum_i |\text{dom}(\mathbf{A})|^{c_i}-\text{const}}$ for all STRUC$[\sigma]$. Using this property, we can show that $g(w) := 2^{|w|} \notin \#\Sigma_0$ with similar arguments to those used for $f(w) := |w|^{\lceil |w|/2 \rceil} \notin \#\Sigma_0$ in the proof of Theorem 5.9. The same can be shown for g^* defined by $g^*(w) := 1$, if $|w| = 1$, and $g^*(x) := g(x)$ otherwise. On the other hand, $g^* \in \#\Pi_1(1\forall)$ via the τ_{string}-formula $\varphi(\mathsf{F}, \mathsf{max}) := \forall y\Big(y \leq \mathsf{max} \wedge (\mathsf{F}(y) = \mathsf{min} \vee \mathsf{F}(y) = \mathsf{max})\Big)$.

Now, let $k \geq 1$. Let $L \in \text{ESO}_{\text{f}}^{\tau_{\text{string}}}((k+1)\forall)[\leq, \text{BIT}, \text{min}] \setminus \text{ESO}_{\text{f}}^{\tau_{\text{string}}}(k\forall)[\leq, \text{BIT}, \text{min}]$. Such a language exists by Lemma 5.22 and $\text{ESO}^{\tau_{\text{string}}}(k\forall) \subsetneq \text{ESO}^{\tau_{\text{string}}}((k+1)\forall)$. Without loss of generality we can assume that at least one input of length 1 and at least one input of length 2 are contained in L, as the behavior of both $\text{ESO}^{\tau_{\text{string}}}((k+1)\forall)$- as well as $\text{ESO}^{\tau_{\text{string}}}(k\forall)$-formulae on a bounded number of inputs can be adjusted without changing the behavior on other inputs. This can, e.g., be seen from the fact that $\text{ESO}_{\text{f}}^{\tau_{\text{string}}}(\ell\forall) = \text{NTIME}^{\sigma}_{\text{RAM}}(n^\ell)$ for all ℓ. Let φ be a formula showing $L \in \text{ESO}_{\text{f}}^{\tau_{\text{string}}}((k+1)\forall)$. Now let f be the function defined by φ in the sense of $\#\Pi_1((k+1)\forall)$.

For the sake of contradiction, assume $f \in \#\Pi_1(k\forall)$. As f maps at least one input of length 1 and one input of length 2 to an output > 0, any formula defining f in the sense of $\#\Pi_1(k\forall)$ has to be a formula over vocabulary τ_{string} by Observation 2.20. Let $\varphi'(\overline{\mathsf{F}}, x_1, \ldots, x_n) \in \Pi_1(k\forall)[\leq, \text{BIT}, \text{min}]$ be such a formula. Then the formula $\exists\overline{\mathsf{F}}\exists\mathsf{F}_{x_1}\ldots\exists\mathsf{F}_{x_n}\varphi'[x_1/\mathsf{F}_{x_1}]\ldots[x_n/\mathsf{F}_{x_n}]$ with 0-ary function symbols F_{x_i} for $1 \leq i \leq n$ shows $L \in \text{ESO}_{\text{f}}^{\tau_{\text{string}}}(k\forall)[\leq, \text{BIT}, \text{min}]$ leading to a contradiction. □

5.5 #AC^0 Compared to the Classes of Saluja et al.

In Sections 5.1 and 5.2 we have seen that #AC^0 naturally fits in our hierarchy by showing that $\#\Sigma_0 \subsetneq \#\text{AC}^0 = \#\Pi_1^{\text{prefix}} \subsetneq \#\Pi_1$ for the syntactical subclass $\#\Pi_1^{\text{prefix}}$ of $\#\Pi_1$.

It is apparent that definitions in this context are very sensitive to changes of built-in predicates. For example, already in the setting of decision classes, $\#AC^0 = FO$ is only known with built-in predicate BIT. For weaker fragments there is also a difference between using free relation variables or free function variables, as is evidenced by the different inclusion structures obtained in our alternation hierarchy compared to that obtained by Saluja et al. for the alternation hierarchy in $\#FO^{rel}$ at least for fixed vocabularies. For structures of cardinality 1, this difference is even present for stronger classes as we have for example seen that $\#FO \stackrel{*}{=} \#FO^{rel} = \#P$, but $\#FO \neq \#FO^{rel}$.

For this reason we next investigate the relationship of $\#AC^0$ to classes from the alternation hierarchy in $\#FO^{rel}$ to see what implications the difference in the definitions have. It turns out that while $\#AC^0$ strictly contains $\#\Sigma_0$, it does not seem to naturally fit in this hierarchy as it is incomparable to the classes $\#\Sigma_1^{rel}$, $\#\Pi_1^{rel}$, and $\#\Sigma_2^{rel}$.

Theorem 5.24.

1) $\#\Sigma_0^{rel} \subsetneq \#AC^0$.

2) Let $\mathfrak{C} \in \{\#\Sigma_1^{rel}, \#\Pi_1^{rel}, \#\Sigma_2^{rel}\}$. *Then the following holds:* $\#AC^0 \not\subseteq \mathfrak{C}$ *and* $\mathfrak{C} \not\subseteq \#AC^0$.

Proof. The proof of the inclusion $\#\Sigma_0^{rel} \subsetneq \#AC^0$ is analogous to the proof of Theorem 5.9 and is thus omitted.

For the second statement recall from Theorem 5.1 that $\#\Sigma_1^{rel} \subsetneq \#\Pi_1^{rel} \subsetneq \#\Sigma_2^{rel}$. The claim $\mathfrak{C} \not\subseteq \#AC^0$ for $\mathfrak{C} \in \{\#\Sigma_1^{rel}, \#\Pi_1^{rel}, \#\Sigma_2^{rel}\}$ can be proven as follows: From Example 5.2 we know that $\#3DNF \in \mathfrak{C}$ and from Lemma 5.11 we know that #3DNF is #P-complete under AC^0-Turing reductions. In analogy to the argument for $\#3DNF^{func} \notin \#AC^0$ in the proof of Lemma 5.12, one can show that $\#3DNF \notin \#AC^0$. In consequence, $\mathfrak{C} \not\subseteq \#AC^0$.

It remains to show $\#AC^0 \not\subseteq \mathfrak{C}$. Define the counting function f as follows: $f(w) := 1$, if $|w| = 1$ or $|w|$ is even, and $f(w) := 0$ otherwise. Obviously $f \in \#AC^0$. We will now see that $f \notin \#\Sigma_2^{rel}$. We show this by an argument similar to the proof that #HAMILTONIAN is not in $\#\Sigma_2^{rel}$, used by Saluja et al. to separate $\#\Sigma_2^{rel}$ from #FO [SST95].

For the sake of contradiction, assume that $f \in \#\Sigma_2^{rel}$ via a formula $\varphi(\overline{R}, \overline{x}) \in \Sigma_2^{rel}$ over some vocabulary σ, where

$$\varphi(\overline{R}, \overline{x}) = \exists \overline{y} \forall \overline{z}\, \psi(\overline{R}, \overline{x}, \overline{y}, \overline{z}),$$

for a quantifier-free formula ψ. By Observation 2.20 we have $\sigma = \tau_{string}$, as f maps inputs of length 1 and 2 to the output 1. Let s and t be the lengths of the tuples $\overline{x}$ and $\overline{y}$, respectively. Let $n > s + t$ be an even number and $w = w_0 \dots w_{n-1} \in \{0,1\}^n$ with $w_i \in \{0,1\}$. By assumption, there exist $\overline{R}$, $\overline{a}$, and $\overline{b}$ such that

$$\mathbf{A}_w \models \forall \overline{z}\, \theta(\overline{R}, \overline{a}, \overline{b}, \overline{z}).$$

Let $\mathbf{A}$ be the structure over vocabulary $\tau_{string} \cup (\leq^2)$ obtained by explicitly adding the built-in relation $\leq$ restricted to $\{0, \dots, n-1\}$ to the structure $\mathbf{A}_w$. By the choice of n, there is an $i \in \{0, .., n-1\}$ such that i does not appear in the tuples $\overline{a}$ and $\overline{b}$. Let $\mathbf{A}'$ be the induced substructure of $\mathbf{A}$ obtained by removing the element i. Let $\overline{R}'$ be the tuple of

relations obtained by removing tuples involving the element i from $\overline{R}$. As Π_1-formulae are preserved under taking substructures, it follows that

$$\mathbf{A}' \models \forall \overline{z}\, \theta(\overline{R}^*, \overline{a}, \overline{b}, \overline{z}).$$

Moreover, the structure $\mathbf{A}'$ is isomorphic to the $\tau_{\text{string}} \cup (\leq^2)$-structure $\mathbf{A}_{w'}$ with $w' = w_0 \dots w_{i-1} w_{i+1} \dots w_{n-1}$ and the intended interpretation of $\leq$ by the mapping

$$j \mapsto \begin{cases} j, & \text{if } j < i \\ j-1, & \text{if } j > i. \end{cases}$$

This means that

$$\mathbf{A}_{w'} \models \forall \overline{z}\, \psi(\overline{R}'', \overline{a}', \overline{b}', \overline{z}),$$

where $\overline{R}''$, $\overline{a}'$ and $\overline{b}'$ are obtained by modifying $\overline{R}'$, $\overline{a}$, and $\overline{b}$, respectively, according to the above mapping. This implies $f(\mathbf{A}_{w'}) \geq 1$, which is a contradiction to the definition of f, as $|w'|$ is odd. □

5.6 Conclusion and Outlook

In this chapter, we put the model-theoretic characterization of #AC^0 obtained in Chapter 4 in perspective to the characterization of #P by Saluja et al. [SST95], using that both can be seen as classes of functions counting assignments to free function variables in first-order formulae. This led to a slightly modified definition of the class #FO. By showing that #Skolem-FO can be seen as a syntactical fragment $\#\Pi_1^{\text{prefix}}$ of #FO, we established that #AC^0 fits into our framework. This motivated further study of the structure of the class #FO in our setting. Studying the alternation hierarchy inside of #FO, we have seen that in our setting the hierarchy collapses to the Π_1-level and generally has a different structure, different properties and contains different problems compared to the hierarchy studied by Saluja et al. [SST95]. We then further studied the structure of the classes of our hierarchy beyond the relationship to #AC^0. Here, we first proved that all problems in the Σ_1-level of our hierarchy can be efficiently approximated in the sense of FPRAS. In the process, we introduced the problem #k-PDNF(m), a restricted form of first-order satisfiability based on DNF-formulae with a limited number of literals per term, that is complete for the Σ_1-level of our hierarchy and is contained in FPRAS. Next, we further studied the fine structure of the class #FO by considering the hierarchy based on the number of universal quantifiers inside the Π_1-level of our hierarchy. Using results by Grandjean and Olive for related classes in the decision setting [GO04], we were able to show that this hierarchy is strict. Finally, we have seen that the class #AC^0 does not seem to naturally fit in the hierarchy by Saluja et al. [SST95], as the class is incomparable to all but the lowest and the highest level of their hierarchy.

We have only started the investigation of our framework and will now discuss several directions for future research, offering many open problems. It would be interesting

to further study the connection between the classes #AC^0 and #Π_1^{prefix} by investigating connections between natural hierarchies inside both classes. In 1983, Sipser proved a depth hierarchy within the Boolean class AC^0 [Sip83]. This hierarchy can be transferred to the setting of arithmetic circuits: There is an infinite depth hierarchy within #AC^0. We expect that this depth hierarchy is connected to the hierarchy obtained by restricting the number of different arities of Skolem functions in #Π_1^{prefix}. Similarly, Rossman proved a size hierarchy within AC^0 over ordered graphs [Ros10]. Again, this hierarchy can be transferred to the setting of arithmetic circuits. It would be interesting to identify a related hierarchy inside #Π_1^{prefix}. A starting point could be to restrict #Win-FO to formulae in prenex normal form with quantifiers for tuples, but enforcing strict alternations. The hierarchy in the resulting class obtained by restricting the arity of tuples that can be quantified seem to be connected to the size hierarchy in #AC^0, but the details are not clear. It would be interesting to further study these hierarchies inside #Π_1^{prefix} and make the connections to known hierarchies precise.

With regard to the alternation hierarchy inside #FO, we have seen that considerable differences arise from the different definitions used in our framework and the one of Saluja et al. [SST95], that is, the difference between using free relation or free function variables as well as different sets of built-in predicates and constants. In Section 5.5, we have established the connections between #AC^0 and the hierarchy of Saluja et al. Here, it could be interesting to study connections between our classes and the classes in the hierarchy of Saluja et al. further. For example, it can be shown that the Σ_1-level of their hierarchy is not contained in the Σ_1-level of our hierarchy by showing that the latter does not contain the function counting the number of satisfying assignments of a propositional formula in 3DNF. Moreover, we know very little about how differences between the two hierarchies are influenced by subtle modifications. As an example, one can show that using two built-in constants as well as built-in BIT, the hierarchy studied by Saluja et al. changes considerably, collapsing to the Π_1-level. It could be interesting to systematically study the differences arising in the alternation hierarchy inside #FO based on different sets of built-in predicates, built-in constants and potentially built-in functions in combination with allowing free function variables, free relation variables or possibly both.

While outside the scope of this thesis, it could also be interesting to further investigate the problem #k-PDNF(m) and related problems. In the setting of propositional formulae, the problem of counting satisfying assignments of formulae in DNF, namely #DNF, is known to be in FPRAS even with an unbounded number of literals in each term [KLM89]. Is this also true in the setting of pseudo-DNF sentences with bounded arity of function symbols?

With regard to the hierarchy based on the number of universal quantifiers within the Π_1-level of our hierarchy, studied in Section 5.4, a lot more is known about this hierarchy in the decision setting. For example, Grandjean and Olive have shown that the same classes as those obtained by restricting the number of universal quantifiers also arise by either restricting the arity of free relation symbols and the number of universal quantifiers, or the number of variables (without requiring the formula to be in prenex

normal form) [GO04]. It would be interesting to find out whether similar connections exist between the corresponding hierarchies in the counting setting. Additionally, it could be interesting to study the relation of #AC^0 to these classes and attempt show hat #AC^0 is not contained in any finite level of the hierarchy.

On a broader scope, one could also study classes arising by counting satisfying assignments to free relation or function variables for classes of formulae beyond first-order logic, in particular fragments of second-order logic. Here, the hope is to obtain characterizations of complexity classes beyond #P. As mentioned before, one step in this direction was recently made by Haak et al. [HKM$^+$19] from the perspective of counting complexity of team logics.

6 Descriptive Complexity of Logarithmic-Depth Circuit Complexity Classes

In Chapter 4 we have seen that #Win-FO captures $\#AC^0$ both in the non-uniform and in the FO[BIT]-uniform setting. Still, the characterization of other counting classes in circuit complexity remained open. In this chapter we turn our attention to the most prominent classes remaining, i.e., $\#NC^1$, $\#SAC^1$, and $\#AC^1$. Our goal is to obtain characterizations for these classes using an approach similar to that used to characterize $\#AC^0$ in Chapter 4: Study logical characterizations of the corresponding classical complexity classes NC^1, SAC^1, and AC^1, and attempt to characterize the counting classes by counting winning strategies in the respective logics.

We will now briefly recall the different known characterizations of the corresponding classical classes, which were already stated in Section 2.3.1, and discuss their value in obtaining characterizations for the counting classes $\#NC^1$, $\#SAC^1$, and $\#AC^1$. Both the result that first-order logic with monoidal quantifiers captures NC^1 [BIS90] as well as the result that first-order logic with groupoid quantifiers captures SAC^1 [LMSV01] do not seem to directly provide ideas for a logical characterization of the corresponding counting classes, as it is unclear how to introduce the concept of winning strategies for the additional types of quantifiers. Also, this approach would have the downside that it would not help in characterizing $\#AC^1$.

The characterization of U_E^*-uniform NC^1 as first-order logic with relational primitive recursion [CL90] does not seem to be helpful, as proof trees of the circuit evaluating a given formula do not correspond to winning strategies of that formula. Furthermore, this would not help in obtaining characterizations of $\#SAC^1$ and $\#AC^1$.

Finally, we know that AC^1 is captured by $FO[\log n]$ [Imm99]. Here, the very restrictive structure of the quantifier-block, which consists of individual quantifiers each with a quantifier-free relativization (a formula restricting the elements considered for the quantifier), prevents a transfer of this result to the counting setting. While it might be possible to remedy this issue, we would first need a modified version of the class $FO[\log n]$ characterizing AC^1. Also, this would not help in characterizing the classes NC^1 and SAC^1.

Since by the above considerations, directly transferring the known characterizations to the counting setting does not seem possible, we instead use a combination of the idea of the classes $FO[\log n]$ and $FO + RPR$ characterizing AC^1 and NC^1, respectively. In order to capture $\#AC^1$ we use a kind of recursive definition of predicates similar to RPR that more directly replicates the structure of circuits, which we will call GPR. An issue

in coming up with a recursive definition replicating the behavior of logarithmic depth circuits is that the condition that a circuit has logarithmic depth is in a sense a global condition of the circuit family and in the uniform setting can be seen as a semantic condition of the formulae showing uniformity. To overcome this, we show that this condition can be made syntactical when assuming built-in order on the gates of the circuits: We will present a normal form of circuit families where for any two gates g_1 and g_2, if g_2 is a predecessor of g_1, then $g_2 < \frac{1}{2} \cdot g_1$. This leads to the definition of GPR, where we use a syntactic restriction to ensure that the depth of the recursion is logarithmically bounded. More precisely, we will introduce a certain form of relativized quantifiers that make use of built-in predicates in order to ensure that the numerical value reduces to less than half of the previous numerical value in each step of the recursion. We will see that variants of this idea allow us to also capture the classes NC^1 and SAC^1, and these characterizations work both in the non-uniform and the FO[BIT]-uniform setting. Furthermore, our model-theoretic classes are in a sense "close enough" to the respective circuit classes to allow us to transfer the characterizations to the counting setting (both non-uniformly and uniformly). For this, we generalize the first-order model-checking game to first-order logic with occurrences of GPR and consider classes of functions counting winning strategies in this game.

We will start by giving the definition of the class FO + RPR in more detail. Then, we will introduce the new operators GPR, $\mathrm{GPR}_{\mathrm{bound}}$, and $\mathrm{GPR}_{\mathrm{semi}}$ and the resulting definitions of first-order logic extended by the different operators. To illustrate our new definitions, we will define the AC^1-complete problem ShortCake (see Section 2.1.2) in FO + GPR. We will then go on to prove that FO + GPR captures AC^1 and that the variants $\mathrm{FO+GPR}_{\mathrm{bound}}$ and $\mathrm{FO+GPR}_{\mathrm{semi}}$ capture the classes NC^1 and SAC^1, respectively (both in the non-uniform and the FO[BIT]-uniform setting). Finally, we will transfer these results to the counting setting, obtaining model-theoretic characterizations of $\#\mathrm{AC}^1$, $\#\mathrm{NC}^1$, and $\#\mathrm{SAC}^1$ by counting winning strategies in the model-checking game for FO + GPR-formulae.

6.1 Guarded Predicative Recursion

We will now present in more detail the logic FO + RPR, which was shown to capture $\mathrm{U}_{\mathrm{E}}^*$-uniform NC^1 by Compton and Laflamme [CL90], as we take inspiration from the notion of relational primitive recursion used in this logic. As the name suggests, relational primitive recursion (RPR) is a recursive definition of a relation in the vein of primitive recursion. Since this recursion is to be used inside first-order logic, the recursive definition is given by a first-order formula. Primitive recursion necessitates an order of elements. For this reason, RPR-operators can only be defined if the universe is assumed to be $\{0, \dots, n-1\}$, which is mostly reasonable in the presence of built-in predicates such as $\leq$ or BIT. Following the definitions of Compton and Laflamme, we only define this operator and the resulting logic with built-in BIT, using the notation FO[BIT] + RPR in the following in line with our usual notation.

Syntactically, an RPR-operator is a formula of the following form:

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

where $y-1$ is a shorthand for a term that is uniquely determined to be equal to $s(y)-1$ under any assignment s with $y \in \mathrm{dom}(s)$. Note that $y-1$ is (uniquely) definable from y in FO[BIT]. Having $\mathsf{P}(\overline{x}, y-1)$ as an argument of θ has the meaning that the predicate symbol P may occur in θ, but only in the form of occurrences of the atom $\mathsf{P}(\overline{x}, y-1)$. Semantically, the meaning of this operator is that the interpretation of P is defined recursively such that for any $\overline{a}, b$ with $b > 0$, $\mathsf{P}(\overline{a}, b)$ is equivalent to $\theta(\overline{a}, b, \mathsf{P}(\overline{a}, b-1))$, and $\mathsf{P}(\overline{a}, 0)$ is equivalent to $\theta(\overline{a}, 0, \bot)$.

Now, FO[BIT] + RPR is the class of first-order formulae with built-in BIT and arbitrary occurrences of RPR-operators, even allowing simultaneous definitions of multiple predicates using RPR. As usual, we also denote by FO[BIT] + RPR the class of languages definable by FO[BIT] + RPR-formulae.

We will now introduce a new type of recursive definition of relations that more closely resembles the structure of Boolean circuits. We call this new kind of recursive definition **g**uarded **p**redicative **r**ecursion, abbreviated as GPR. First, we need a bit of additional notation, starting with relativized quantifiers. A relativization of a quantifier is a formula restricting the domain of elements considered for that quantifier. We also use relativizations for the quantification of tuples. More precisely, we write

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

as a shorthand for $\exists x_1 \ldots \exists x_k (\varphi \wedge \psi)$ and, similarly,

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

as a shorthand for $\forall x_1 \ldots \forall x_k (\neg\varphi \vee \psi) \equiv \forall x_1 \ldots \forall x_k (\varphi \rightarrow \psi)$. For better readability and simplicity, we will use the extensions of $=$ and $<$ to tuples in the following definition. Note that these extensions are easily definable from $=$ and $<$ with quantifier-free formulae. We also generally use extensions of numerical predicates to tuples throughout this chapter, as long as they are definable. Furthermore, for an FO-formula φ and a relation variable P, we write $\varphi(\mathsf{P}^+)$ if P does not occur inside any negation in φ, i.e., for all $\psi \in \mathrm{SF}(\varphi)$ starting with a negation, P does not occur in ψ. We now define the GPR-operator as well as the logic FO + GPR.

Definition 6.1. Let $\mathfrak{R}$ be a set of relations over $\mathbb{N}$ such that the predicates $\times$ and $<$ are definable in FO[$\mathfrak{R}$].

Then the set of FO[$\mathfrak{R}$] + GPR-formulae over σ is the set of formulae generated by the grammar for FO[$\mathfrak{R}$]-formulae over σ extended by the rule

$$\varphi ::= [\mathsf{P}(\overline{x}, \overline{y}) \equiv \theta(\overline{x}, \overline{y}, \mathsf{P}^+)]\ \varphi(\mathsf{P}^+),$$

where θ is an FO[$\mathfrak{R}$]-formula over σ, $\overline{x}$ and $\overline{y}$ are tuples of variables, P is a relation variable and each atomic sub-formula involving P in θ

1. is of the form $\mathsf{P}(\overline{x},\overline{z})$, 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 $Q \in \{\forall,\exists\}$, $\xi \in \mathrm{FO}[\mathfrak{R}]$ and

2. never occurs in the scope of any quantification not guarded in this way, that is: For all $\psi \in \mathrm{SF}(\theta)$ starting with a quantifier not guarded in the above way, P does not occur in ψ.

Here, $\overline{z} < \overline{y}/2$ is a shorthand for a formula equivalent to $\exists\overline{u}\,(\overline{z}+\overline{z} = \overline{u} \wedge \overline{u} < \overline{y})$. We call the part in $[\cdot]$ a GPR-operator. The relation variable P is considered a bound variable in formula $[\mathsf{P}(\overline{x},\overline{y}) \equiv \theta(\overline{x},\overline{y},\mathsf{P}^+)]\,\varphi(\mathsf{P}^+)$, as its interpretation is given by the GPR-operator. The class $\mathrm{FO}[\mathfrak{R}]+\mathrm{GPR}$ is the class of all $\mathrm{FO}[\mathfrak{R}]+\mathrm{GPR}$-formulae over any vocabulary σ.

Semantics are defined as follows: Let $\psi = [\mathsf{P}(\overline{x},\overline{y}) \equiv \theta(\overline{x},\overline{y},\mathsf{P}^+)]\,\varphi(\overline{z},\mathsf{P}^+)$ be an $\mathrm{FO}[\mathfrak{R}]+\mathrm{GPR}$-formula with a single GPR-operator over σ for some set $\mathfrak{R}$ of relations over $\mathbb{N}$ and vocabulary σ. Let $\mathbf{A} \in \mathrm{STRUC}[\sigma]$ and $\mathcal{I}$ be a non-uniform family of interpretations of built-in predicate symbols in ψ by relations in $\mathfrak{R}$. Then in $\mathbf{A}$ and using the family $\mathcal{I}$ of interpretations, the operator $[\mathsf{P}(\overline{x},\overline{y}) \equiv \theta(\overline{x},\overline{y},\mathsf{P}^+)]$ defines the interpretation P of the relation variable P with the following properties: For all tuples $\overline{a}$ and $\overline{b}$ of elements of $\mathrm{dom}(\mathbf{A})$ with the same arity as $\overline{x}$ and $\overline{y}$, respectively,

$$(\overline{a},\overline{b}) \in P \text{ if and only if } \mathbf{A} \models_{\mathcal{I}} \theta(\overline{a},\overline{b},P).$$

Accordingly $\mathbf{A} \models_{\mathcal{I}} [\mathsf{P}(\overline{x},\overline{y}) \equiv \theta(\overline{x},\overline{y},\mathsf{P}^+)]\,\varphi(\overline{c},\mathsf{P}^+)$ holds for a tuple $\overline{c}$ of elements of $\mathrm{dom}(\mathbf{A})$ with the same arity as $\overline{z}$, if and only if $\mathbf{A} \models_{\mathcal{I}} \varphi(\overline{c},P)$, where P is the interpretation of P as defined by the GPR-operator. As usual, if the family $\mathcal{I}$ is clear from the context, e.g., in case of the built-in relation BIT, we will omit $\mathcal{I}$ and write $\mathbf{A} \models \varphi$ instead of $\mathbf{A} \models_{\mathcal{I}} \varphi$. Semantics for formulae with multiple GPR-operators is defined analogously.

We will also refer to the class of languages definable by $\mathrm{FO}[\mathfrak{R}]$+GPR formulae over any vocabulary σ by $\mathrm{FO}[\mathfrak{R}]+\mathrm{GPR}$. In the following, we will only make use of $\mathrm{FO}[\mathfrak{R}]+\mathrm{GPR}$ where $\mathfrak{R}$ is either the set Arb of all relations over $\mathbb{N}$ or $\mathfrak{R}$ only contains BIT. Note that in both cases, $<$ and $\times$ are definable in $\mathrm{FO}[\mathfrak{R}]$.

We next illustrate this new definition and show its flexibility for defining natural problems by expressing the $\#\mathrm{AC}^1$-complete problem ShortCake, defined on page 26, in $\mathrm{FO}[\mathrm{BIT}]+\mathrm{GPR}$.

Example 6.2. We will now define the problem ShortCake in $\mathrm{FO}[\mathrm{BIT}]+\mathrm{GPR}$. For this, we will use the vocabulary $\sigma := (\mathsf{M}^2)$ that contains one binary predicate, representing the Boolean input matrix: An $n \times n$-matrix is represented by the structure $\mathbf{A}$ with domain $\{0,\dots,n-1\}$ and $\mathsf{M}^{\mathbf{A}}$ is exactly the input matrix, using numbers 0 to $n-1$ to index both rows and columns. We will use two formulae θ_H and θ_V expressing valid moves of players H and V, respectively, as well as a formula θ_{win} expressing that the current configuration is winning for player H. Then, a GPR-operator is used to recursively define a predicate on configurations of the game given by the indices i_0, i_1, j_0, j_1 specifying the current submatrix and a variable p specifying the current player (we use 0 for player H and 1 for player V).

In a single step of the game, we only reduce either the number of rows or the number of columns, but not both. Furthermore, the size, i.e., the number of entries, of the current submatrix might only reduce to half of the previous size rounded up. Both of these properties lead to issues when simply encoding configurations by the indices i_0, i_1, j_0, j_1, as the tuple of indices would not decrease to less than half in each step. In order to remedy this, we introduce auxiliary variables to the encoding of configurations that allow us to enforce a decrease to less than half in each step.

This leads to the formula

$$\varphi_{\text{shc}} \coloneqq \Big[\mathsf{P}(\overline{y}) \equiv \theta_{\text{win}}(\overline{y}) \vee p = 0 \wedge \theta_H(\overline{y}, \mathsf{P}^+) \vee p = 1 \wedge \theta_V(\overline{y}, \mathsf{P}^+)\Big] \mathsf{P}\Big(\overline{\text{maxsteps}}, 0, n-1, 0, n-1, 0\Big),$$

where $\overline{y} \coloneqq (\overline{s}, i_0, i_1, j_0, j_1, p)$, $\overline{s}$ is a tuple of auxiliary variables and $\overline{\text{maxsteps}}$ is the starting value of $\overline{s}$. We will define it later and ensure that it is FO[BIT]-definable. Similarly, 0 and $n-1$ are FO[BIT]-definable.

To simplify the presentation in the following, we both identify natural numbers with the binary strings encoding them, as well as tuples of natural numbers that are $\leq n$ with natural numbers. For the latter, tuples of numbers $\leq n$ encode natural numbers in accordance with the lexicographic order on those tuples, i.e., the leftmost component has the highest significance. We encode in the tuple $\overline{s}$ the current step of the game using a special unary encoding. Fix some $n \in \mathbb{N}$. Let $s \in \mathbb{N}$ be a bound for the number of steps in the Shortcake game on $n \times n$ input matrices and let $m \coloneqq 2s$. We will later determine a sufficient bound for s. A number $\ell \in \mathbb{N}$ is encoded in the step counter by the tuple $\overline{s}$ that represents the number with binary encoding $0^{2\ell}1^{m-2\ell}$. As mentioned before, we also write $\overline{s} = 0^{2\ell}1^{m-2\ell}$ in this case. We will now argue that this ensures that the step counter decreases to less than half in each step of the game. Let $\ell \in \mathbb{N}$ and $\overline{s}$ be the encoding of ℓ in the step counter, i.e., $\overline{s} \coloneqq 0^{2\ell}1^{m-2\ell}$. Then the next step, $\ell + 1$, is encoded in the step counter by $\overline{s}' \coloneqq 0^{2\ell+2}1^{m-2\ell-2}$. First note that we have $\overline{s}' = \frac{\overline{s}}{4}$ by construction. We now show that for any two assignments $\overline{\text{config}}$, $\overline{\text{config}}'$ for the tuple (i_0, i_1, j_0, j_1, p), we have

$$\Big(\overline{s}', \overline{\text{config}}'\Big) < \frac{1}{2} \cdot \Big(\overline{s}, \overline{\text{config}}\Big).$$

The arity of $\overline{\text{config}}$ and $\overline{\text{config}}'$ is 5, as these tuples are of the form (i_0, i_1, j_0, j_1, p). Hence,

if $\overline{s} \geq 2$ (which is true in all relevant cases by construction) we have

$$\begin{aligned}\left(\overline{s}', \overline{\text{config}}'\right) &\leq \overline{s}' \cdot n^5 + n^5 - 1 \\ &= \frac{\overline{s}}{4} \cdot n^5 + n^5 - 1 \\ &\leq \left(\frac{\overline{s}}{2} - 1\right) \cdot n^5 + n^5 - 1 \\ &< \frac{\overline{s} \cdot n^5}{2} \\ &\leq \frac{\overline{s} \cdot n^5 + \overline{\text{config}}}{2} \\ &= \frac{\left(\overline{s}, \overline{\text{config}}\right)}{2}.\end{aligned}$$

It is pretty obvious that we can find a bound s and define m, the tuple length of $\overline{s}$, and $\overline{\text{maxsteps}}$ such that this construction allows for at least s steps in the recursion. For completeness, we explain how to choose these numbers. We do this for the case that the input structure has a domain of cardinality at least 2. The case of domains of cardinality 1 can be handled separately.

First, determine a bound s for the number of steps. We know that if the current submatrix has size ℓ, its size will decrease to at most $\left\lceil \frac{\ell}{2} \right\rceil$ in the next step. Hence, for an $n \times n$ input matrix the number of steps is bounded by $\lceil \log_2(n^2) \rceil$. This means we can set $s := \lceil \log_2(n^2) \rceil$, resulting in $m = 2\lceil \log_2(n^2) \rceil$. We can now define $\overline{\text{maxsteps}} := 2^{2 \cdot \lceil \log_2(n^2) \rceil} - 1$, which is FO[BIT]-definable. Consequently, an arity of 6 for the tuple $\overline{s}$ is sufficient, as $2^{2 \cdot \lceil \log_2(n^2) \rceil} - 1 \leq (2n^2)^2 - 1 = 4n^4 - 1$ by construction and the highest number that can be encoded in a tuple of length 6 is $n^6 - 1 \geq 4 \cdot n^4 - 1$ (recall that we assumed $n \geq 2$).

Finally, we define the formula θ_H expressing valid moves of the player H:

$$\begin{aligned}\theta_H(\overline{y}) := \Big(\exists \overline{z} = (\overline{s}', i_0', i_1', j_0', j_1', p').(\overline{z} < \overline{y}/2)\Big) \\ \overline{s}' = \frac{\overline{s}}{4} \wedge p' = 1 \wedge i_0' = i_0 \wedge i_1' = i_1 \wedge \\ \Big(\Big(j_0' \neq j_0 \wedge j_1' = j_1 \wedge (j_1 + j_0)/2 \leq j_0' \leq j_1 \wedge \mathsf{M}(i_0', j_0')\Big) \vee \\ \Big(j_0' = j_0 \wedge j_1' \neq j_0 \wedge j_0 \leq j_1' < (j_1 + j_0)/2 \wedge \mathsf{M}(i_0', j_1')\Big)\Big) \wedge \\ \mathsf{P}(\overline{s}', i_0', i_1', j_0', j_1', p').\end{aligned}$$

Here, we use shortcuts for several FO[BIT]-definable properties such as $\overline{s} = \frac{\overline{s}}{4}$ for readability.

The formula $\theta_V(\overline{y}, \mathsf{P}^+)$ is defined analogously with universal guarded quantification for $\overline{z}$ and the definition of $\theta_{\text{win}}(\overline{y})$ is straightforward.

We next want to define variants of GPR replicating the properties of families of NC^1- and SAC^1-circuits, respectively. For this, the guarded quantifiers have to be restricted such that they do not quantify over a polynomial domain. We do this by introducing *bounded quantifiers*, which are relativized quantifiers where we only consider the maximal two elements meeting the condition expressed by the relativization. Bounded existential and bounded universal quantifiers are denoted by $\exists_b$ and $\forall_b$, respectively. Formally, the semantics of $\exists_b$ can be given by an equivalent FO-formula as follows:

$$\left(\exists_b(\overline{x}).\varphi(\overline{x})\right)\psi(\overline{x}) \equiv \left(\exists\overline{x}.\left(\varphi(\overline{x}) \wedge \forall\overline{y}\forall\overline{z}\left((\overline{y} \neq \overline{z} \wedge \overline{x} < \overline{y} \wedge \overline{x} < \overline{z}) \rightarrow (\neg\varphi(\overline{y}) \vee \neg\varphi(\overline{z}))\right)\right)\right)\psi(\overline{x}).$$

Semantics of $\forall_b$ is defined analogously.

We extend the first-order model-checking game to formulae with occurrences of bounded quantifiers $\exists_b$ and $\forall_b$ by treating them analogously to $\exists$ and $\forall$, respectively, restricting the possible choices of the choosing player to the maximal two elements satisfying the relativization. This is in accordance with the semantics of the quantifiers.

We define the bounded variant of GPR, denoted by GPR_{bound}, analogously to GPR by allowing only bounded guarded quantifications $Q_b\overline{z}.(\overline{z} < \overline{y}/2 \wedge \xi(\overline{y},\overline{z}))$, and the semi-bounded variant GPR_{semi}, where bounded universal guarded quantifications $\forall_b\overline{z}.(\overline{z} < \overline{y}/2 \wedge \xi(\overline{y},\overline{z}))$ and existential guarded quantifications $\exists\overline{z}.(\overline{z} < \overline{y}/2 \wedge \xi(\overline{y},\overline{z}))$ are allowed. This leads to the definition of the logics $FO[\mathfrak{R}] + GPR_{bound}$ and $FO[\mathfrak{R}] + GPR_{semi}$ in analogy to $FO[\mathfrak{R}] + GPR$.

6.2 Model-theoretic Characterizations of Small Depth Decision Classes

We now want to prove that our new logics FO[BIT] + GPR and the variants using GPR_{bound} and GPR_{semi} capture AC^0, NC^1, and SAC^1, respectively. As a first step, we need a normal form for the circuit complexity classes that translates the condition of having logarithmic depth to a local, syntactic condition. Assuming an order on the set of gates (and arithmetic operations defined accordingly), this condition will be that for any two gates g_1, g_2 in a circuit, if g_2 is a predecessor of g_1, then $g_2 < \frac{1}{2} \cdot g_1$. In order to replicate the operations computed by the gates by the correct quantifiers inside a GPR-operator, we also make the condition for a gate having type $\wedge$ or $\vee$ a local condition.

In the FO[BIT]-uniform case, there is an inherent interpretation of gates as natural numbers due to the built-in relation BIT in the logical language of the uniformity and its extension to tuples. In order to be able to define the extension of BIT to tuples in FO[BIT], we further need that the formula φ_0 of the FO[BIT]-query showing uniformity is somewhat simple.

This leads to the following normal form in the FO[BIT]-uniform setting: All tuples of the appropriate arity are gates (so φ_0 from the FO-query 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. For any gate, all predecessors of the gate is smaller than half of that gate.

In the non-uniform setting, we have to additionally state that we can assign each gate a natural number such that the above condition holds. This way, we can use the same normal form both in the uniform- and the non-uniform setting.

Lemma 6.3. *Let $\mathfrak{C} \in \{\mathrm{NC}^1, \mathrm{SAC}^1, \mathrm{AC}^1\}$ and $C = (\mathbf{C}_n)_{n\in\mathbb{N}}$ be a $\mathfrak{C}$ circuit family in input normal form. Then there is some $k \in \mathbb{N}$ and a $\mathfrak{C}$ circuit family $C' = (\mathbf{C}'_n)_{n\in\mathbb{N}}$ in input normal form with $\#C' = \#C$, where the $\mathbf{C}'_n$ are given as τ_{circ}-structures, and for all $n \in \mathbb{N}$, $\mathbf{C}'_n$ has the following properties:*

1. $\mathrm{dom}(\mathbf{C}'_n) = \{0, \dots, n-1\}^k$,
2. *for all $\overline{x} \in \mathrm{dom}(\mathbf{C}'_n)$,*
   $\overline{g} \in \mathsf{G}_{\wedge}^{\mathbf{C}'_n}$ if and only if $\overline{g} \notin \mathsf{Input}^{\mathbf{C}'_n}$, $\overline{g} \notin \mathsf{negatedInput}^{\mathbf{C}'_n}$ and $\overline{g}$ is odd, and
   $\overline{g} \in \mathsf{G}_{\vee}^{\mathbf{C}'_n}$ if and only if $\overline{g} \notin \mathsf{Input}^{\mathbf{C}'_n}$, $\overline{g} \notin \mathsf{negatedInput}^{\mathbf{C}'_n}$ and $\overline{g}$ is even, and
3. *for all $\overline{g}, \overline{h} \in \mathrm{dom}(\mathbf{C}'_n)$,*
   if $(\overline{g}, \overline{h}) \in \mathsf{E}^{\mathbf{C}'_n}$, then $\overline{g}$ is less than half of $\overline{h}$ numerically.

If C is FO[BIT]*-uniform, then there is such a C' that is* FO[BIT]*-uniform.*

In the above, statements about numerical values of tuples are made with respect to the lexicographic order of tuples based on the natural order on $\mathbb{N}$.

Proof. We prove the FO[BIT]-uniform version of the statement. The non-uniform version can be shown with the same construction without using uniformity. Let $\mathcal{I}\colon \mathrm{STRUC}[\tau_{\mathrm{string}}] \to \mathrm{STRUC}[\tau_{\mathrm{circ}}]$ be an FO[BIT]-query showing the uniformity of C. Let k be the number of free variables of φ_0 in $\mathcal{I}$, i.e., the arity of tuples encoding gates in this circuit family. We now construct an FO[BIT]-query $\mathcal{I}'''$ describing a circuit family C' with the desired properties, including the fact that $\#C = \#C'$.

For the details of this proof, it is relevant in what way we represent numbers as tuples. As mentioned before we use a lexicographic order based on the natural order on $\mathbb{N}$. Just as we did in Example 6.2, we will assume that the leftmost component has the highest significance in this encoding.

1.) We first construct an FO[BIT]-query $\mathcal{I}'$ such that the circuit family described by $\mathcal{I}'$ 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 out. Let $\varphi_0, \varphi_{\mathsf{E}}, \varphi_{\mathsf{out}}$ be the respective formulae from $\mathcal{I}$. For $\mathcal{I}'$ we instead use

$$\begin{aligned} \varphi'_0(\overline{g}) &:= 1, \\ \varphi'_{\mathsf{E}}(\overline{g}_1, \overline{g}_2) &:= \varphi_{\mathsf{E}}(\overline{g}_1, \overline{g}_2) \wedge \varphi_0(\overline{g}_1) \wedge \varphi_0(\overline{g}_2), \text{ and} \\ \varphi'_{\mathsf{out}}(\overline{g}) &:= \varphi_{\mathsf{out}}(\overline{g}) \wedge \varphi_0(\overline{g}). \end{aligned}$$

The gate type of the new dummy gates does not matter, so we do not need to change the formulae for gate types. As this step only potentially adds new dummy gates from

which the output gate is not reachable, it does not change the number of proof trees on any inputs.

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 component to the encoding of gates and only define edges to and from those versions of $\wedge$-gates where this component is 1, and those versions of $\vee$-gates where this component is 0. Also, input gates are only connected to the rest of the circuit if this additional component is 0. The interpretation of out is changed accordingly, depending on the type of that gate. This allows us to make all odd non-input gates $\wedge$-gates and all even non-input gates $\vee$-gates.

For property 3, we add to the encoding of gates a unary encoding of the depth—in the sense of "distance to the output gate"—of the same form as the unary encoding used in Example 6.2. We then only connect two gates g_1 and g_2 by an edge (g_1, g_2), if there was an edge between the corresponding gates in the original circuit and the depth encoded in g_2 is 1 less than the depth encoded in g_1. This ensures that the numerical value of gates reduces to less than half in each step from a gate to one of its predecessors. Notice that by this construction, our circuits contain many dummy gates, i.e., gates from which the output gate is not reachable or which are not even connected to any other gates. We now formalize the above ideas.

2.) We now define the formulae for the FO[BIT]-query $\mathcal{I}''$, which ensures property 2 in addition to the previous properties. For this, we increase the arity of tuples by 1, adding a new component to the right of the encoding of gates. Note that, as we add a new component to the right, encodings of indices of input gates implicitly change, so we need to modify the formulae accordingly. The query $\mathcal{I}''$ is defined by the following formulae:

$$\begin{aligned}
\varphi''_0(\overline{g}x) &:= 1,\\
\varphi''_{\mathsf{Input}}(\overline{g}x, i_1 \dots i_k y) &:= \varphi'_{\mathsf{Input}}(\overline{g}, i_2 \dots i_k y),\\
\varphi''_{\mathsf{negatedInput}}(\overline{g}x, i_1 \dots i_k y) &:= \varphi'_{\mathsf{negatedInput}}(\overline{g}, i_2 \dots i_k y),\\
\varphi''_{\mathsf{G}_\wedge}(\overline{g}x) &:= \forall \overline{i}\left(\neg\varphi''_{\mathsf{Input}}(\overline{g}x, \overline{i}) \wedge \neg\varphi''_{\mathsf{negatedInput}}(\overline{g}x, \overline{i})\right) \wedge \mathrm{BIT}(0, x),\\
\varphi''_{\mathsf{G}_\vee}(\overline{g}x) &:= \forall \overline{i}\left(\neg\varphi''_{\mathsf{Input}}(\overline{g}x, \overline{i}) \wedge \neg\varphi''_{\mathsf{negatedInput}}(\overline{g}x, \overline{i})\right) \wedge \neg\mathrm{BIT}(0, x),\\
\varphi''_{\mathsf{E}}(\overline{g}_1 x_1, \overline{g}_2 x_2) &:= \varphi'_{\mathsf{E}}(\overline{g}_1, \overline{g}_2) \wedge \psi_{\mathrm{real}}(\overline{g}_1 x_1) \wedge \psi_{\mathrm{real}}(\overline{g}_2 x_2), \text{ and}\\
\varphi''_{\mathsf{out}}(\overline{g}x) &:= \varphi'_{\mathsf{out}}(\overline{g}) \wedge \psi_{\mathrm{real}}(\overline{g}x),
\end{aligned}$$

where $\overline{g}$ is a tuple of arity k and ψ_{real} expresses that a gate is not a dummy gate as follows:

$$\begin{aligned}
\psi_{\mathrm{real}}(\overline{g}x) := (\varphi'_{\mathsf{G}_\wedge}(\overline{g}) \wedge x = 1) \vee (\varphi'_{\mathsf{G}_\vee}(\overline{g}) \wedge x = 0) \vee\\
\exists \overline{i}\left((\varphi'_{\mathsf{Input}}(\overline{g}, \overline{i}) \vee \varphi'_{\mathsf{negatedInput}}(\overline{g}, \overline{i})) \wedge x = 0\right).
\end{aligned}$$

3.) We construct an FO[BIT]-query $\mathcal{I}'''$ that additionally ensures property 3. We add a unary encoding of the depth to the encoding of all gates, analogously to the construction used in Example 6.2. As in that example, inputs of length 1 are handled separately. Also,

we again sometimes identify natural numbers with binary strings in the following. Let $c \cdot \lfloor \log_2 n \rfloor$ with $c \in \mathbb{N}$ be a bound for the depth of the circuit family described by $\mathcal{I}''$. The idea is to create for each gate of the old circuit a duplicate for each possible depth within the circuit. In analogy to the construction in Example 6.2, the depth is encoded in the following way: Let $m := c \cdot \lfloor \log_2 n \rfloor$. Depth i is then encoded by the number with binary encoding $0^{2i}1^{2m-2i}$ for all $i \leq m$. This means that it is sufficient to increase the arity of tuples by $\ell := 2c$, as the largest number we need to encode is the number with binary encoding $1^{2c \cdot \lfloor \log_2 n \rfloor}$, i.e., $2^{2c \cdot \lfloor \log_2 n \rfloor} - 1$, and we have

$$2^{2 \cdot c \cdot \lfloor \log_2 n \rfloor} - 1 \leq n^{\ell} - 1$$

for all $n \in \mathbb{N}$. For the output gate we need to express that the encoding of its depth is 1^{2m}.

In the final circuit, gates are only connected if their versions in the old circuit were connected and the predecessor's depth is 1 higher than the successor's (both encoded as described above). The fact that this construction ensures property 3 can be proven analogously to the corresponding proof in Example 6.2. This leads to the following formulae for $\mathcal{I}'''$:

$$\begin{aligned}
\varphi'''_0(\overline{h}\overline{g}x) &:= 1,\\
\varphi'''_{\mathsf{G}_\wedge}(\overline{h}\overline{g}x) &:= \varphi''_{\mathsf{G}_\wedge}(\overline{g}x),\\
\varphi'''_{\mathsf{G}_\vee}(\overline{h}\overline{g}x) &:= \varphi''_{\mathsf{G}_\vee}(\overline{g}x),\\
\varphi'''_{\mathsf{Input}}(\overline{h}\overline{g}x, \overline{j}\overline{i}y) &:= \overline{j} = \overline{0} \wedge \varphi''_{\mathsf{Input}}(\overline{g}x, \overline{i}y),\\
\varphi'''_{\mathsf{negatedInput}}(\overline{h}\overline{g}x, \overline{j}\overline{i}y) &:= \overline{j} = \overline{0} \wedge \varphi''_{\mathsf{negatedInput}}(\overline{g}x, \overline{i}y), \text{ and}\\
\varphi'''_{\mathsf{out}}(\overline{h}\overline{g}x) &:= \forall \overline{j}\Big((\overline{j} \leq 2c \cdot \lfloor \log_2 n \rfloor) \rightarrow \mathrm{BIT}(\overline{j}, \overline{h}) \wedge\\
&\qquad (\neg\overline{j} \leq 2c \cdot \lfloor \log_2 n \rfloor \rightarrow \neg\mathrm{BIT}(\overline{j}, \overline{h}))\Big),
\end{aligned}$$

where $\overline{h}$ and $\overline{j}$ are tuples of length ℓ, $\overline{g}$ and $\overline{i}$ are tuples of length k, and x and y are individual variables. Notice that the condition $\overline{j} \leq 2c \cdot \lfloor \log_2 n \rfloor$ is FO[BIT]-definable. Moreover,

$$\begin{aligned}
\varphi'''_{\mathsf{E}}(\overline{h}_1\overline{g}_1x_1, \overline{h}_2\overline{g}_2x_2) := \varphi''_{\mathsf{E}}(\overline{g}_1x_1, \overline{g}_2x_2) \wedge\\
\exists\overline{i}\Big(\Big(\forall(\overline{j} \leq \overline{i})\ \mathrm{BIT}(\overline{j}, \overline{h}_2)\Big) \wedge \Big(\forall(\overline{j} > \overline{i})\ \neg\mathrm{BIT}(\overline{j}, \overline{h}_2)\Big) \wedge\\
\Big(\forall(\overline{j} \leq (\overline{i} - 2))\ \mathrm{BIT}(\overline{j}, \overline{h}_1)\Big) \wedge \Big(\forall(\overline{j} > \overline{i} - 2)\ \neg\mathrm{BIT}(\overline{j}, \overline{h}_1)\Big)\Big),
\end{aligned}$$

where $\overline{h}_1$, $\overline{h}_2$, $\overline{i}$ and $\overline{j}$ are tuples of length ℓ, $\overline{g}_1$ and $\overline{g}_2$ are tuples of length k, and x_1 and x_2 are individual variables. Here, $\overline{j} \leq \overline{i} - 2$ is FO[BIT]-definable and we use quantifiers of the form $\forall(\overline{j} \leq \overline{i})\ \varphi$ as a shorthand for $\forall\overline{j}\,(\overline{j} \leq \overline{i} \wedge \varphi)$ for readability.

The circuit family described by $\mathcal{I}'''$ has properties 1, 2 and 3. Furthermore, the modifications made in steps 2 and 3, just as those made in step 1, do not change the

structure of the component of gates from which the output gate is reachable. Therefore, the number of proof trees of circuits in the family described by $\mathcal{I}'''$ is the same as the number of proof trees of the corresponding circuits in the family described by $\mathcal{I}$. □

We are now in a position to show the decision version of the main theorem of this chapter: The newly defined logical classes FO + GPR, FO + $\text{GPR}_{\text{bound}}$, and FO + GPR_{semi} capture the classes NC^1, SAC^1, and AC^1, respectively, both in the non-uniform and the FO[BIT]-uniform setting.

Theorem 6.4.

1) $\text{FO[Arb]} + \text{GPR} = \text{AC}^1$ *and* $\text{FO[BIT]} + \text{GPR} = \text{FO[BIT]}$-uniform AC^1.

2) $\text{FO[Arb]} + \text{GPR}_{\text{bound}} = \text{NC}^1$ *and* $\text{FO[BIT]} + \text{GPR}_{\text{bound}} = \text{FO[BIT]}$-uniform NC^1.

3) $\text{FO[Arb]} + \text{GPR}_{\text{semi}} = \text{SAC}^1$ *and* $\text{FO[BIT]} + \text{GPR}_{\text{semi}} = \text{FO[BIT]}$-uniform SAC^1.

Proof. We begin by proving the non-uniform version of $\text{FO} + \text{GPR} = \text{AC}^1$. We will then argue that the constructions can be made uniform. Afterwards, we explain what needs to be changed for the bounded and semi-bounded variants.

$\text{AC}^1 \subseteq \text{FO[Arb]} + \text{GPR}$: Let $L \in \text{AC}^1$ via the non-uniform circuit family $C = (\mathbf{C}_n)_{n\in\mathbb{N}}$ in the normal form of Lemma 6.3. Without loss of generality we can assume that $\mathbf{C}_n$ has depth at least 1 for all $n \in \mathbb{N}$, as we can simply add a new $\wedge$-gate that has the old output gate as its only predecessor and make it the new output gate. Let $k \in \mathbb{N}$ be such that $\text{dom}(\mathbf{C}_n) = \{0,\dots,n-1\}^k$ for all $n \in \mathbb{N}$ and let $\mathcal{I} = (\mathbf{I}_n)_{n\in\mathbb{N}}$ be a non-uniform family of interpretations such that $\mathbf{I}_n$ interprets the symbols of τ_{circ} according to $\mathbf{C}_n$.

We will define L in FO[Arb] + GPR over vocabulary τ_{string} with τ_{circ} as the vocabulary for build-in predicate symbols, and use the family $\mathcal{I}$ to get for all $n \in \mathbb{N}$ access to the relations of $\mathbf{C}_n$ from any input structure $\mathbf{A}_w$ with $|w| = n$. For the construction of the desired FO[Arb] + GPR formula we need the following auxiliary formulae:

$$\begin{aligned}
\varphi_{\text{Literal}}(\overline{x}) &:= \exists \overline{i}\,(\mathsf{Input}(\overline{x},\overline{i}) \vee \mathsf{negatedInput}(\overline{x},\overline{i})),\\
\varphi_{\text{trueLiteral}}(\overline{x}) &:= \exists \overline{i} = (i_1,\dots,i_k)\,(\mathsf{Input}(\overline{x},\overline{i}) \wedge \mathsf{S}(i_k) \vee \mathsf{negatedInput}(\overline{x},\overline{i}) \wedge \neg\mathsf{S}(i_k)),\\
\psi(\overline{z},\mathsf{P}(\overline{z})) &:= (\mathsf{P}(\overline{z}) \wedge \neg\varphi_{\text{Literal}}(\overline{z})) \vee \varphi_{\text{trueLiteral}}(\overline{z}).
\end{aligned}$$

Then the following sentence Φ defines L:

$$\Phi := [\mathsf{P}(\overline{y}) \equiv \theta(\overline{y},\mathsf{P}^+)]\ \exists\overline{x}(\mathsf{out}(\overline{x}) \wedge \mathsf{P}(\overline{x})),$$

where

$$\begin{aligned}
\theta(\overline{y},\mathsf{P}) := \Big(\neg\,\text{BIT}(0,\overline{y}) \wedge \exists\overline{z}.(\overline{z} < \overline{y}/2 \wedge \mathsf{E}(\overline{z},\overline{y}))\ \psi(\overline{z},\mathsf{P}(\overline{z}))\Big) \vee\\
\Big(\text{BIT}(0,\overline{y}) \wedge \forall\overline{z}.(\overline{z} < \overline{y}/2 \wedge \mathsf{E}(\overline{z},\overline{y}))\ \psi(\overline{z},\mathsf{P}(\overline{z}))\Big).
\end{aligned}$$

Here, $\text{BIT}(0,\overline{y})$ is used to determine whether $\overline{y}$ is even or odd and hence the type of gate $\overline{y}$.

Now in any structure $\mathbf{A}_w$ and under any assignment s for the free variables $\overline{y}$ and P, the formula θ is equivalent to

$$\Big(Q\overline{z}.(\overline{z} < \overline{y}/2 \wedge \mathsf{E}(\overline{z},\overline{y}))\Big)\Big(\mathsf{P}(\overline{z}) \wedge \neg\varphi_{\text{Literal}}(\overline{z})\Big) \vee \varphi_{\text{trueLiteral}}(\overline{z}),$$

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

We now prove the correctness of our construction, that is, we prove that Φ defines L. Let $n \in \mathbb{N}$, $w \in \{0,1\}^n$ and let P be the interpretation of P defined by the GPR-operator in the above formula when using $\mathcal{I}$ to interpret symbols from τ_{circ}, that is, interpreting those symbols according to $\mathbf{C}_n$. We will prove that P is the valuation of the gates in the circuit $\mathbf{C}_n$. More precisely, we prove inductively that for any $d \in \mathbb{N}$, $P(\overline{g})$ gives the value of gate $\overline{g}$ in $\mathbf{C}_n$ on input w if all predecessors of $\overline{g}$ have depth $\leq d$. In the following, we will not talk about general equivalence of formulae, but instead about equivalence and truth of formulae with respect to the input structure $\mathbf{A}_w$ and the interpretation $\mathbf{I}_n$ of symbols in τ_{circ}.

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

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

Since C is in the normal form of Lemma 6.3, if $\mathsf{E}(\overline{h},\overline{g})$ is true for some gate $\overline{h}$, then $\overline{h} < \overline{g}/2$, and thus

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

This yields

$$P(\overline{g}) \equiv \Big(Q\overline{z}.\mathsf{E}(\overline{z},\overline{g})\Big)\,\varphi_{\text{trueLiteral}}(\overline{z}),$$

which means that $P(\overline{g})$ is true if and only if $\text{val}(\mathbf{C}_n,\overline{g},w) = 1$.

$d \to d+1$: Again, as C is in the normal form of Lemma 6.3,

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

We also know that for all predecessors $\overline{h}$ of $\overline{g}$ only two cases can occur: If $\overline{h}$ is an input gate, then $\neg\varphi_{\text{Literal}}(\overline{h})$ is false, and $\varphi_{\text{trueLiteral}}(\overline{h})$ is true if and only if $\text{val}(\mathbf{C}_n,\overline{h},w) = 1$. If $\overline{h}$ is not an input gate, then $\varphi_{\text{trueLiteral}}(\overline{h})$ is false, $\neg\varphi_{\text{Literal}}(\overline{h})$ is true, and $P(\overline{h})$ is true if and only if $\text{val}(\mathbf{C}_n,\overline{h},w) = 1$ by induction hypothesis. By Equation ($\star\star$) above, this means that in fact $P(\overline{g})$ is true if and only if $\text{val}(\mathbf{C}_n,\overline{g},w) = 1$.

We showed that the truth value of P on arbitrary non-input gates in $\mathbf{C}_n$ corresponds to their value in the circuit when it takes input w. Also, we assumed that the output gate of $\mathbf{C}_n$ is not an input gate. As these properties hold for all $n \in \mathbb{N}$ and all $w \in \{0,1\}^n$, this

means 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 1.

FO[Arb] + GPR $\subseteq$ AC^1: We assume that only one occurrence of a GPR-operator is allowed. The proof easily extends to the general case. Let $L \in \mathrm{FO[Arb]} + \mathrm{GPR}$ via some formula φ over some vocabulary σ and additional vocabulary τ for built-in predicates, where

$$\varphi =: [\mathsf{P}(\overline{x},\overline{y}) \equiv \theta(\overline{x},\overline{y},\mathsf{P}^+)]\ \psi(\mathsf{P}^+).$$

Without loss of generality, we can assume that θ is in negation normal form. By definition of GPR, P occurs in θ only in the form of atoms $\mathsf{P}(\overline{x},\overline{z})$, 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 quantification not guarded in this way.

Now, build an AC^0 circuit family evaluating ψ except for the occurrences of P. For this, notice that ψ is an FO[Arb]-formula over $\sigma \cup (\mathsf{P})$. Next, we explain how to construct an AC^1 circuit which contains for any tuples of elements $\overline{a},\overline{b}$ of appropriate arity a gate computing whether $(\overline{a},\overline{b}) \in P$, where P is the interpretation of P as defined by the GPR-operator.

The formula θ is, similar to ψ, an FO[Arb]-formula over vocabulary $\sigma \cup (\mathsf{P})$, so we can build an AC^0 circuit family that computes for all tuples of elements of adequate arity $\overline{a},\overline{b}$ the value of $\theta(\overline{x},\overline{y},P)$ with certain input gates marked with $P(\overline{a},\overline{c})$ for some tuple of elements $\overline{c}$. We cannot assume that θ is in prenex normal form, though. Hence, we need to evaluate an FO[Arb]-formula in negation normal form, but not necessarily prenex normal form, using an AC^0 circuit family. This can be done similarly to the proofs of FO[Arb] $\subseteq$ AC^0 by Immerman [Imm99] or our proof of #Win-FO[Arb] $\subseteq$ #AC^0 in the proof of Theorem 4.2. The main difference is that we do not have two separate parts, one for the quantifier-prefix, which branches for all possible values that can be assigned to the quantified variables, and one for the quantifier-free part, where the circuit replicates the structure of the formula. Instead, the circuit replicates the structure of the formula in general, each gate evaluating a node of the syntax tree representation of the formula. For this, we can simply have the corresponding node in the syntax tree as part of the encoding of gates in the circuit. Despite the difference, we still ensure that the assignment to the quantified variables is part of the encoding of gates by using the tuple of assigned elements as part of the encoding. This way, we can directly evaluate a formula that is not in prenex normal form.

We then simply connect each gate that was marked with $P(\overline{a},\overline{c})$ to the output gate of the subcircuit computing $\theta(\overline{a},\overline{c},P)$. Since occurrences of $\mathsf{P}(\overline{x},\overline{z})$ only occur within guarded quantifications $Q\overline{z}.(\overline{z} < \overline{y}/2)$, there can be at most logarithmically many steps from any $P(\overline{a},\overline{b})$ before reaching $P(\overline{a},\overline{0})$, terminating the recursion. By the above, each such step—computing $P(\overline{a},\overline{b})$, when given values of $P(\overline{a},\overline{c})$ for certain $\overline{c}$—is done by an AC^0 circuit family and hence in constant depth. Consequently, the construction leads to logarithmic depth in total.

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

For the case of multiple GPR-operators, we build a circuit for each of them in the above way and connect them to the AC^0 circuit family evaluating the FO[Arb]-formula following those operators as needed.

Next, we will argue that the above constructions can be made uniform.

FO[BIT]-uniform $AC^1 \subseteq$ FO[BIT] + GPR: The main difference to the proof of the non-uniform version is that we get access to the circuit family C not by using built-in predicates for the symbols from τ_{circ}, but instead use the formulae from the FO[BIT]-query $\mathcal{I}$ showing uniformity of C. The remaining built-in predicates in the constructed formula, that is, predicates other than the relations of circuits in C and the BIT-predicate, such as the order $<$, are FO[BIT]-definable. Thus, they can be replaced by adequate subformulae and the resulting formula shows $L \in \text{FO[BIT]} + \text{GPR}$.

FO[BIT] + GPR $\subseteq$ FO[BIT]-uniform AC^1: For inputs w of length > 1, we can encode the current node in the syntax tree representation of an FO[BIT]-formula as a tuple of elements of the universe of $\mathbf{A}_w$. Hence, the direct evaluation of an FO[BIT]-formula that is not in prenex normal form can be done in FO[BIT]-uniform AC^0. Further, we can construct the circuit family in such a way that the tuple representing a gate contains $\overline{a}$ and $\overline{b}$, if the gate is supposed to evaluate an atom $\mathsf{P}(\overline{a}, \overline{b})$ for tuples of elements $\overline{a}, \overline{b}$ of adequate arity.

Furthermore, we can construct the subcircuits computing the value of $P(\overline{a}, \overline{b})$, where P is the interpretation of P defined by the GPR-operator, in such a way that the tuple representing the output gate of such a subcircuit contains the tuples $\overline{a}$ and $\overline{b}$. By constructing the subcircuits in this way, it is easy to define the required connections between subcircuits in FO[BIT].

The case of inputs of length 1 has to be handled separately. For this, simply modify the behavior of the resulting circuit family on the inputs 0 and 1.

Finally, we turn to the bounded and semi-bounded cases. We will see that these follow almost immediately from the proofs of the unbounded versions. Both in the non-uniform and the FO[BIT]-uniform setting, $NC^1 \subseteq \text{FO} + \text{GPR}_{\text{bound}}$ and $SAC^1 \subseteq \text{FO} + \text{GPR}_{\text{semi}}$ can be shown with the same formula and the same proof as $AC^1 \subseteq \text{FO(GPR)}$, replacing GPR by $\text{GPR}_{\text{bound}}$ or GPR_{semi}, respectively. The reason is that in this case, the circuit family showing membership of L already has the property that all gates have bounded fan-in in the case of NC^1 and all $\wedge$-gates have bounded fan-in in the case of SAC^1. Therefore, the bounded and semi-bounded quantifiers quantify over all predecessors of gates.

FO + $\text{GPR}_{\text{bound}} \subseteq NC^1$: This can be proven completely analogously to FO + GPR $\subseteq AC^1$ both in the non-uniform and in the FO[BIT]-uniform setting. Instead of AC^0 circuit families evaluating ψ 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 successor. This ensures that gates marked with $P(\overline{a}, \overline{c})$ for some $\overline{c}$ still only occur in constant depth in the circuit evaluating $\theta(\overline{a}, \overline{b}, P)$ for any $\overline{a}, \overline{b}$, this time with only bounded fan-in gates. For this, notice that generally an FO[BIT]-formula without unbounded quantifiers can be evaluated in NC^0. Consequently, the construction still

only leads to logarithmic depth in total.

$\mathbf{FO + GPR_{semi} \subseteq SAC^1}$: Here, we can proceed similarly as for the bounded case. The formula θ can be evaluated using an NC^1 circuit family except for unbounded guarded quantifiers $\exists \overline{z}.(\overline{z} < \overline{y}/2 \wedge \xi)$. For these guarded quantifiers we use unbounded fan-in $\vee$-gates. This results in an SAC^1 circuit family. Note that the semi-unbounded case is the reason we need the condition that P only occurs positively anywhere. If we dropped this condition, we would possibly need to evaluate negations of unbounded existential quantifiers, which would lead to unbounded fan-in $\wedge$-gates. □

The proofs that the circuit complexity classes are contained in the corresponding model-theoretic classes also immediately gives us the following normal form for our model-theoretic classes.

Corollary 6.5. *Let* $\mathcal{G} \in \{\mathrm{GPR}, \mathrm{GPR}_{\mathrm{bound}}, \mathrm{GPR}_{\mathrm{semi}}\}$. *Then*

$$\mathrm{FO}[\mathrm{Arb}] + \mathcal{G} = \mathcal{G}\text{-FO}[\mathrm{Arb}] \quad \textit{and} \quad \mathrm{FO}[\mathrm{BIT}] + \mathcal{G} = \mathcal{G}\text{-FO}[\mathrm{BIT}],$$

where $\mathcal{G}$-FO *denotes the class of formulae in* $\mathrm{FO} + \mathcal{G}$ *with a single* $\mathcal{G}$*-operator in the beginning.*

6.3 Model-theoretic Characterizations of Small Depth Counting Classes

Next, we want to define the game semantics for our new logics. Let $\mathfrak{R}$ be a set of relations over $\mathbb{N}$. We will adapt the full model-checking game for first-order logic to FO[$\mathfrak{R}$]+GPR-formulae. The full model-checking game is chosen as the game is basically played on the formula obtained after unrolling recursive definitions given by GPR-operators. The resulting formula is not in prenex forms, so alternative types of witnesses considered in Chapter 3 are not suitable. Let φ be an FO[$\mathfrak{R}$] + GPR-formula over some vocabulary σ, $\mathbf{A}$ a σ-structure, $s\colon \mathrm{free}(\varphi) \to \mathrm{dom}(\mathbf{A})$, and $\mathcal{I}$ a non-uniform family of interpretations of the built-in predicate symbols in φ by relations in $\mathfrak{R}$. In the definition, we mainly want to ensure that strategies for the game can vary for different subformulae and even different copies of the same subformula that arise when evaluating the recursive definition described by a GPR-operator. There are different ways to achieve this. One could for example unroll the GPR-operator based on the current input structure to obtain a formula of polynomial size and play the model-checking game on this formula. Instead, we choose to simply add the history of all previous formulae and decisions to any configuration. Strategies can then naturally differ for different copies of the same subformula, as they can differ for different histories. This means we also do not need to use a syntax tree representation of the formula and that we can simply add the GPR-operators occurring in φ as global information to the game for $\mathbf{A}, s \models_{\mathcal{I}} \varphi$. Notice that in order to be able to distinguish two occurrences of the same subformula that have the same history leading to them except for the last decision, e.g., in case of a formula $\psi \wedge \psi$, decisions for Boolean connectives have to be specified by encoding whether the first or second subformula was chosen, not by stating the chosen subformula.

This results in the following definition of the model-checking game for $\mathbf{A}, s \models_{\mathcal{I}} \varphi$. Configurations of the game are of the form

$$(\psi, s', \text{swap}, H),$$

where ψ is the current formula, s' is an assignment to the free variables in ψ, swap is a bit specifying whether the players currently swapped roles and H is the complete history of formulae and corresponding choices up to this point. The game starts in configuration

$$(\psi, s, 0, H_0),$$

where ψ is the subformula of φ to the right of the GPR-operators (so $\psi \in \text{FO}[\mathfrak{R}]$) and H_0 is the empty history. Just as the first-order model-checking game, the new game is played by the *verifier* (or *player 1*) and the *falsifier* (*player 2*). The verifier starts and wants to show that the formula evaluates to true in $\mathbf{A}$ with the family $\mathcal{I}$ of interpretations and under assignment s. The game then proceeds in the same way as the model-checking game for $\text{FO}[\mathfrak{R}]$, 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 corresponding GPR-operator (without changing the assignment, the swap-bit or the history).

A *strategy of the verifier* is again a function mapping configurations to specific choices in such a way that a choice is specified for all configurations that give the verifier a choice and are reachable from the starting configuration if they act according to this strategy and the falsifier makes arbitrary choices. Since configurations contain a full history, choices may differ for copies of the same occurrence of a subformula that arise when evaluating the recursive definition described by a GPR-operator. *Winning strategies of the verifier* are strategies of the verifier that allow them to win independent of the choices made by the falsifier.

Similar to the definition of #Win-FO in Definition 3.1, we can define counting classes in terms of counting winning strategies in the model-checking game for FO + GPR-formulae.

Definition 6.6. Let $\mathfrak{R}$ be a set of relations over $\mathbb{N}$ such that the predicates $\times$ and $<$ are definable in $\text{FO}[\mathfrak{R}]$.

Then a function $f : \{0,1\}^+ \to \mathbb{N}$ is in the class #Win-FO[$\mathfrak{R}$]+GPR, if there is an FO[$\mathfrak{R}$]+GPR-sentence φ over some vocabulary σ and a non-uniform family of interpretations $\mathcal{I}$ of the built-in predicate symbols in φ by relations from $\mathfrak{R}$ such that for all $\mathbf{A} \in \text{STRUC}[\sigma]$:

$$f(\text{enc}_\sigma(\mathbf{A})) = \#\text{Win}(\mathbf{A}, \mathcal{I}, \varphi),$$

and $f(x) = 0$ if x is not the encoding of a σ-structure. Here, $\#\text{Win}(\mathbf{A}, \mathcal{I}, \varphi)$ is the number of winning strategies of the verifier in the model checking game for $\mathbf{A} \models_{\mathcal{I}} \varphi$, omitting the assignment as φ is a sentence. If the family $\mathcal{I}$ of interpretations is clear from the context, for example in case of built-in BIT, we also write $\#\text{Win}(\mathbf{A}, \varphi)$ instead of $\#\text{Win}(\mathbf{A}, \mathcal{I}, \varphi)$.

The complexity classes #Win-FO[$\mathfrak{R}$] + $\text{GPR}_{\text{bound}}$ and #Win-FO[$\mathfrak{R}$] + GPR_{semi} are defined analogously, restricting the moves on quantifiers $\exists_b$ and $\forall_b$ to only allow the two choices that are potential witnesses for the quantifier.

We will now show that the newly defined counting classes capture the classes $\#\text{AC}^0$, $\#\text{SAC}^1$, and $\#\text{NC}^1$, respectively, both in the non-uniform and the FO[BIT]-uniform setting.

Theorem 6.7.

1) $\#\text{Win-FO[Arb]} + \text{GPR} = \#\text{AC}^1$ *and*
$\#\text{Win-FO[BIT]} + \text{GPR} \stackrel{*}{=} \text{FO[BIT]-uniform } \#\text{AC}^1$.

2) $\#\text{Win-FO[Arb]} + \text{GPR}_{\text{bound}} = \#\text{NC}^1$ *and*
$\#\text{Win-FO[BIT]} + \text{GPR}_{\text{bound}} \stackrel{*}{=} \text{FO[BIT]-uniform } \#\text{NC}^1$.

3) $\#\text{Win-FO[Arb]} + \text{GPR}_{\text{semi}} = \#\text{SAC}^1$ *and*
$\#\text{Win-FO[BIT]} + \text{GPR}_{\text{semi}} \stackrel{*}{=} \text{FO[BIT]-uniform } \#\text{SAC}^1$.

Proof. Similar to the structure of the proof of Theorem 6.4, we prove the non-uniform version of #Win-FO + GPR = $\#\text{AC}^1$ first. We will then argue how the construction can be made uniform and finally turn to the bounded and the semi-bounded variant. The proof idea is very similar to the one used for the decision version.

$\#\text{AC}^1 \subseteq$ #Win-FO[Arb] + GPR: Let $f \in \#\text{AC}^1$ via the AC^1 circuit family $\mathcal{C} = (\mathbf{C}_n)_{n\in\mathbb{N}}$ in the normal form of Lemma 6.3. Without loss of generality we can again assume that $\mathbf{C}_n$ has depth at least 1 for all $n \in \mathbb{N}$ by the same argument as for the decision version. Let $k \in \mathbb{N}$ be such that $\text{dom}(\mathbf{C}_n) = \{0,\dots,n-1\}^k$ for all $n \in \mathbb{N}$ and let $\mathcal{I} = (\mathbf{I}_n)_{n\in\mathbb{N}}$ be a non-uniform family of interpretations such that $\mathbf{I}_n$ interprets the symbols of τ_{circ} according to $\mathbf{C}_n$.

In analogy to the proof for the decision version, we will construct a #Win-FO[Arb] + GPR-sentence Φ over vocabulary τ_{string} such that the number of winning strategies of the verifier in the model-checking game for $\mathbf{A}_w \models_{\mathcal{I}} \Phi$ is equal to $f(w)$ for all $w \in \{0,1\}^+$. For the construction, we again need the auxiliary formulae φ_{Literal}, $\varphi_{\text{trueLiteral}}$, and ψ defined as in the proof of Theorem 6.4. We can now define the desired sentence Φ as follows:

$$\Phi := [\mathsf{P}(\overline{y}) \equiv \theta(\overline{y}, \mathsf{P}^+)]\, \exists \overline{x}\, (\mathsf{out}(\overline{x}) \wedge \mathsf{P}(\overline{x}))$$

with

$$\theta(\overline{y}, \mathsf{P}) := \Big(\neg \text{BIT}(0,\overline{y}) \wedge \big(\exists \overline{z}.(\overline{z} < \overline{y}/2 \wedge \mathsf{E}(\overline{z},\overline{y}))\big)\, \psi(\overline{z}, \mathsf{P}(\overline{z})\Big) \vee$$
$$\Big(\text{BIT}(0,\overline{y}) \wedge \big(\forall \overline{z}.(\overline{z} < \overline{y}/2 \wedge \mathsf{E}(\overline{z},\overline{y}))\big)\, \big(\psi(\overline{z}, \mathsf{P}(\overline{z})) \wedge \overline{z} < \overline{y}/2 \wedge \mathsf{E}(\overline{z},\overline{y})\big)\Big).$$

Note that θ is defined similarly to the definition in the proof of the decision version, with the difference that now the subformula $(\overline{z} < \overline{y}/2 \wedge \mathsf{E}(\overline{z},\overline{y}))$ not only occurs in the relativizations, but is added again to the formula following the guarded quantifier in the

universal case. The reason is that 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 again.

We will now show that Φ together with $\mathcal{I}$ defines the function f, i.e., that the number of winning strategies of the verifier in the game for $\mathbf{A}_w \models_{\mathcal{I}} \Phi$ is equal to $f(w)$, and hence to the number of proof trees of $\mathbf{C}_{|w|}$ on input w, for all $w \in \{0,1\}^+$. Let $n \in \mathbb{N}$ and $w \in \{0,1\}^n$. In the following, we use $\#\mathrm{Win}(s, \varphi)$, φ being an FO[Arb] + GPR-formula, as notation for the number of winning strategies of verifier in the game for $\mathbf{A}_w, s \models_{\mathcal{I}} \varphi$. The same number arises when adding an arbitrary history, as the number of winning strategies is independent of the history. Now under any assignment s for the free variables $\overline{y}$ and P, we have

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

where Q is either $\exists$ or $\forall$ depending on the parity of $s(\overline{y})$. Notice that in the case of existential quantifiers, the number of winning strategies does not change by adding the guard again in the end.

We now prove that for any non-input gate $\overline{g}$, the number of winning strategies $\#\mathrm{Win}(\mathsf{P}(\overline{g}))$ for atoms $\mathsf{P}(\overline{g})$ in the formula following the GPR-operator is exactly the number of proof trees of the subcircuit of $\mathbf{C}_n$ rooted in $\overline{g}$. Here, the assignment is omitted from $\#\mathrm{Win}(\mathsf{P}(\overline{g}))$, as it is given by $\overline{g}$. More precisely, we inductively prove that for any $d \in \mathbb{N}$, $\#\mathrm{Win}(\mathsf{P}(\overline{g}))$ is the number of proof trees of the subcircuit of $\mathbf{C}_n$ rooted in $\overline{g}$ on input w if all predecessors of $\overline{g}$ have depth $\leq d$.

$d = 0$: We immediately get that $\#\mathrm{Win}(\varphi_{\mathrm{trueLiteral}}(\overline{h}))$ is the value of $\overline{h}$ in $\mathbf{C}_n$ on input w if $\overline{h}$ is an input gate. As for any input gate, the value of the gate is equal to the number of proof trees of the subcircuit rooted in it, we have for gates $\overline{g}$ all predecessors of which are input gates:

$$\begin{aligned}\#\mathrm{Win}(\mathsf{P}(\overline{g})) &= \#\mathrm{Win}\Big(\Big(Q\overline{z}.(\overline{z} < \overline{g}/2 \wedge \mathsf{E}(\overline{z},\overline{g}))\Big) \\ &\qquad \Big(\Big(\mathsf{P}(\overline{z}) \wedge \neg\varphi_{\mathrm{Literal}}(\overline{z})) \vee \varphi_{\mathrm{trueLiteral}}(\overline{z})\Big) \wedge \overline{z} < \overline{g}/2 \wedge \mathsf{E}(\overline{z},\overline{g})\Big)\Big) \\ &= \bigwedge_{\substack{\overline{h} \in \mathrm{dom}(\mathbf{A}_w)^k, \\ \overline{h} < \overline{g}/2 \text{ and } (\overline{h},\overline{g}) \in \mathsf{E}^{\mathbf{I}_{|w|}}}} \#\mathrm{Win}\Big(\mathsf{P}(\overline{h}) \wedge \neg\varphi_{\mathrm{Literal}}(\overline{h}) \vee \varphi_{\mathrm{trueLiteral}}(\overline{h})\Big), \qquad (\star\star\star)\end{aligned}$$

where $\bigwedge$ is either summation or multiplication depending on the parity of $\overline{g}$. Recall that universal relativized quantifiers are defined such that $(\forall x.\varphi)\ \psi = \forall x(\neg\varphi \vee \psi)$. Hence, for a relativized quantifier of the form $(\forall x.\varphi)\ (\psi \wedge \varphi) = \forall x(\neg\varphi \vee (\psi \wedge \varphi))$, the number

of winning strategies is exactly the product of the number of winning strategies of $\psi(a)$ over all assignments a for x that satisfy the relativization φ. For this reason, it could be beneficial in the counting setting to modify the definition of $(\forall \overline{x}.\varphi)\ \psi$ for any formulae φ and ψ such that this formula translates to $\forall x(\neg\psi \vee (\psi \wedge \varphi))$.

Note that the number of winning strategies of the verifier on both φ_{Literal} and $\varphi_{\text{trueLiteral}}$ is ≤ 1. Since $d = 0$, we know that for all assignments for $\overline{z}$ satisfying the relativization, $\varphi_{\text{Literal}}(\overline{z})$ is true, yielding

$$\#\text{Win}(\mathsf{P}(\overline{g})) = \bigwedge_{\substack{\overline{h}\in \text{dom}(\mathbf{A}_w)^k,\\ \overline{h}<\overline{g}/2 \text{ and } (\overline{h},\overline{g})\in \mathsf{E}^{\mathbf{I}_{|w|}}}} \#\text{Win}(\varphi_{\text{trueLiteral}}(\overline{h})).$$

By assumption, $(\overline{h},\overline{g}) \in \mathsf{E}^{\mathbf{I}_{|w|}}$ implies $\overline{h} < \overline{g}/2$. Therefore, $\#\text{Win}(\mathsf{P}(\overline{g}))$ is exactly the number of proof trees of the subcircuit of $\mathbf{C}_n$ rooted in $\overline{g}$.

$d \to d+1$: Again by assumption, $\#\text{Win}(s, \overline{z} < \overline{g}/2 \wedge \mathsf{E}(\overline{z},\overline{g})) = \#\text{Win}(s, \mathsf{E}(\overline{z},\overline{g}))$ under any assignment s for $\overline{z}$. We also know that for all predecessors $\overline{h}$ of $\overline{g}$ only two cases can occur: If $\overline{h}$ is an input gate, then we have $\#\text{Win}(\neg\varphi_{\text{Literal}}(\overline{h})) = 0$ and $\#\text{Win}(\varphi_{\text{trueLiteral}}(\overline{h}))$ is exactly the number of proof trees of the subcircuit of $\mathbf{C}_n$ rooted in $\overline{h}$. If $\overline{h}$ is not an input gate, then by assumption we have $\#\text{Win}(\varphi_{\text{trueLiteral}}(\overline{h})) = 0$, $\#\text{Win}(\neg\varphi_{\text{Literal}}(\overline{h})) = 1$ and by induction hypothesis $\#\text{Win}(\mathsf{P}(\overline{h}))$ is exactly the number of proof trees of the subcircuit of $\mathbf{C}_n$ rooted in $\overline{h}$. By Equation $(\star\star\star)$ above, this means that $\#\text{Win}(\mathsf{P}(\overline{g}))$ is equal to the number of proof trees of the subcircuit of $\mathbf{C}_n$ rooted in $\overline{g}$.

As we have assumed that all circuits $\mathbf{C}_n$ in the family C have depth at least 1, 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 verifier on the formula following the GPR-operator is the number of winning strategies on $\mathsf{P}(\overline{\text{output}})$, where $\overline{\text{output}}$ is the unique output gate of $\mathbf{C}_{|w|}$. By the induction above this is equal to the number of proof trees of circuit $\mathbf{C}_{|w|}$ on input w.

#Win-FO[Arb] + GPR $\subseteq$ #AC^1: This can be proven completely analogously to the decision version. For this, note that θ can also be assumed to be in negation normal form in the counting setting. Counting proof trees of the constructed circuit family leads exactly to the desired function.

Next, we will argue that the above constructions can be made uniform.

FO[BIT]-uniform #AC^1 $\subseteq$ #Win-FO[BIT] + GPR: As in the decision version, we access the circuit family C not by using built-in predicates for the symbols in τ_{circ}, but instead use the formulae from the FO[BIT]-query $\mathcal{I}$ showing uniformity of C. In order to keep the number of winning strategies the same despite replacing atoms by formulae in this way, we need to make sure that on all formulae from $\mathcal{I}$, the verifier has at most one winning strategy. This can be assumed by Lemma 3.3. The remaining built-in predicates used in the formula, that is, predicates other than the relations of circuits in C and the BIT-predicate, such as the order $<$, are again FO[BIT]-definable. We can again apply Lemma 3.3 to ensure that replacing occurrences of these predicates by formulae does not change the number of winning strategies. By these considerations, we can obtain an FO[BIT] + GPR-formula defining the desired function.

#Win-FO[BIT] + GPR $\overset{*}{\subseteq}$ FO[BIT]-uniform #AC^1: As in the proof of the decision version, this can be proven analogously to the non-uniform version. Let f be a function in #Win-FO[BIT] + GPR via the formula φ. For inputs w of length > 1, the current position in the formula φ and the current assignment to the variables can again be encoded in a tuple of elements of $\mathrm{dom}(\mathbf{A}_w)$.

As inputs of length 1 can only be mapped to 0 or 1 by FO[BIT]-uniform #AC^1 circuit families (cf. Observation 3.11), we can only modify the behavior of the resulting circuit family on inputs 0 and 1 to ensure that the circuit computes the function

$$f^*(x) := \begin{cases} 1, & \text{if } |x| = 1 \text{ and } f(x) > 0, \\ f(x), & \text{otherwise.} \end{cases}$$

Both in the non-uniform and in the FO[BIT]-uniform setting, the inclusions #NC$^1 \subseteq$ #Win-FO + GPR$_{\text{bound}}$ and #SAC$^1 \subseteq$ #Win-FO + GPR$_{\text{semi}}$ can—as in the proof of the decision version—be shown with the same formula as the unbounded case by changing the GPR-operator to a GPR$_{\text{bound}}$- or GPR$_{\text{semi}}$-operator, respectively.

The converse directions, that is, #Win-FO[Arb] + GPR$_{\text{bound}} \subseteq$ #NC^1, #Win-FO[BIT] + GPR$_{\text{bound}} \overset{*}{\subseteq}$ #NC^1, #Win-FO[Arb] + GPR$_{\text{semi}} \subseteq$ #SAC^1, and #Win-FO[BIT] + GPR$_{\text{semi}} \overset{*}{\subseteq}$ #SAC^1 can also be proven with the same arguments as the decision versions, using again the restriction that P occurs only within bounded quantifiers within θ and, in the case of #SAC^1, also the fact that it does not occur inside of negations. □

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

Corollary 6.8. *Let* $\mathcal{G} \in \{\mathrm{GPR}, \mathrm{GPR}_{\text{bound}}, \mathrm{GPR}_{\text{semi}}\}$. *Then*

$$\#\text{Win-FO[Arb]} + \mathcal{G} = \mathcal{G}\text{-WinFO[Arb]} \quad \textit{and} \quad \#\text{Win-FO[BIT]} + \mathcal{G} = \mathcal{G}\text{-WinFO[BIT]},$$

where $\mathcal{G}$-WinFO[$\mathfrak{R}$] *denotes the class of functions arising as the number of winning strategies of the verifier in the model-checking game for* $\mathcal{G}$-FO*-formulae with built-in predicates from* $\mathfrak{R}$.

Remark 6.9. 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 6.1 without changing the class. For #Win-FO[Arb] + GPR and #Win-FO[BIT] + GPR the same holds.

For #Win-FO(GPR$_{\text{bound}}$) and #Win-FO(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". In the case of GPR$_{\text{bound}}$, this requires showing that the occurrences of P can be moved out of unbounded existential quantifiers. For universal quantifiers this cannot be possible, as they can result in counting functions that grow faster than any function in #SAC^1 by the same argument that shows #AC$^1 \not\subseteq$ #SAC^1.

6.4 Conclusion and Outlook

In this chapter, we have combined ideas from known characterizations of the circuit complexity classes NC^1 and AC^1 to obtain unified characterizations of the classes NC^1, SAC^1 and AC^1. In contrast to previous classes characterizing the above circuit complexity classes, our new model-theoretic classes are in a sense close enough to the circuit classes to allow us to transfer our results to the counting setting. For this, we generalized the full first-order model-checking game to the underlying logics of our new classes, defining the corresponding counting classes in terms of counting of winning strategies analogously to #Win-FO. We show that all of these characterizations hold both in the non-uniform as well as the FO[BIT]-uniform setting.

Similar to our characterization of $\#AC^0$ by #Win-FO in Theorem 4.3, our new characterizations also have implications regarding the choice of uniformity. As the circuit families in our proofs are highly uniform, it is straightforward to show that they are actually U_D-uniform. It follows that U_D- and FO[BIT]-uniformity coincide for NC^1 as well as the counting versions of NC^1, SAC^1 and AC^1, when disregarding inputs of length 1. Even more, our logics capture the U_D-uniform counting classes precisely, while they only capture their FO[BIT]-uniform counterparts in the sense of $\overset{*}{=}$.

We hope that our characterizations lead to new insights into structural properties of and connections between the studied classes, hopefully resulting in new upper or lower bounds related to separations such as $\#NC^1 \neq FNC^1$, $\#NC^1 \neq \#P$, or $NC^1 \neq PP$. In this direction, it could be interesting to put the characterizations of the logarithmic-depth classes into perspective in a similar manner as we did for #Win-FO in Chapter 5. A step towards this goal would be to characterize #P in terms of a logic with a recursion operator similar to the operators GPR and RPR, possibly building on the known characterization of the class in terms of circuit complexity.

As discussed in Section 2.4, FO[BIT]-uniform NC^1 might not be the "right" definition of uniform NC^1, the more natural one being U_E^*-uniform NC^1. Still, obtaining new upper or lower bounds for this class would be interesting. In particular, lower bounds regarding this class would of course be even stronger than for U_E^*-uniform NC^1. Nonetheless, a model-theoretic characterization of U_E^*-uniform $\#NC^1$ would be desirable. Here, a possible approach for future research could be to take a deeper look at the characterization of U_E^*-uniform NC^1 by Compton and Laflamme [CL90]. While this characterization does not seem to easily transfer to the counting setting, it would be interesting to better understand where it fails and whether this can be remedied, keeping intact the simpler structure of their recursion scheme to hopefully allow for an U_E^*-uniform NC^1 circuit family to evaluate the recursive definitions.

Page left intentionally blank to have matching page numbers with the printed version.

7 Transferring Characterizations from the Uniform to the Non-uniform Setting

Consider a complexity class $\mathfrak{C}$ defined in terms of circuit families with certain resource bounds, e.g., one of the levels of the AC-hierarchy. Assume there is a superlogic $\mathcal{L}$ of FO capturing $\mathfrak{C}$ in the FO[BIT]-uniform setting, that is, FO[BIT]-uniform $\mathfrak{C} = \mathcal{L}$[BIT]. Intuitively, it seems pretty obvious that this result for uniform classes should transfer to the corresponding non-uniform classes. The reason is that in both cases, non-uniformity means that some additional information, an advice, is given to the respective model of computation (circuits and formula) and this information only depends on the length of the input and is bounded by a polynomial in the length of the input. This intuition is further evidenced by the fact that usually it is easier to prove characterizations of non-uniform classes than proving their uniform counterparts. Proving the uniform version then requires additional work and the proof needs to be refined. This can for example seen for our characterizations in Chapters 4 and 6.

The goal of this chapter is to formalize this intuition and obtain a reasonably general result stating that uniform characterizations can be transferred to the non-uniform setting. The generalizations made are motivated and potential ways to further generalize the result are discussed.

Note that the converse, that is, directly being able to obtain uniform characterizations from non-uniform characterizations, cannot hold in a general form. The reason is that non-uniformity hides differences between classes with respect to properties that only depend on input lengths. This is also reflected in the fact that usually the non-uniform version of a characterization is proven first and then the uniform version utilizes a similar construction, but requires additional work to show that it can be made uniform.

When attempting to show for $\mathfrak{C}$ and $\mathcal{L}$ as above that $\mathfrak{C} = \mathcal{L}$[Arb] using the assumption that FO[BIT]-uniform $\mathfrak{C} = \mathcal{L}$[BIT], for both inclusions one starts with a non-uniform description of a language. In order to facilitate the uniform characterization result in a black-box manner, one obviously needs a uniform description instead. Here, non-uniformity by an additional advice function comes into play. This concept adds non-uniformity externally to an otherwise uniform computation model. Hence, our goal becomes to show that $\mathfrak{C} =$ (FO[BIT]-uniform $\mathfrak{C}$)/poly, that is, non-uniformity of circuit families can be replaced by adding a polynomially length-bounded advice to a uniform circuit family, and similarly that $\mathcal{L}$[Arb] $= \mathcal{L}$[BIT]/poly. From the uniform characterization result we immediately get (FO[BIT]-uniform $\mathfrak{C}$)/poly $= \mathcal{L}$[BIT]/poly, which then yields the desired result.

In this chapter, we show that both for a wide range of classes in circuit complexity as well as in logics over first-order structures, the usual kind of non-uniformity used in the

respective setting coincides with non-uniformity given by a separate advice function. In case of circuit complexity, this requires a bit of additional work, as we need to construct a general, uniform circuit family taking additional advice bits as input such that by setting the advice bits adequately, the family computes any desired function computable by a non-uniform circuit family with the same resource bounds. In first-order logic, non-uniformity is given by built-in numerical predicates, which are much more similar to an additional advice function. Here, whether the two types of non-uniformity coincide comes down to whether the logic is powerful enough to decode and encode non-uniform information given either by a non-uniform family of interpretations or in the form of an advice concatenated to the encoding of the input structure.

In Section 7.1 we show that under reasonable assumptions for $\mathfrak{C}$, even for classes using non-standard bases, it holds that $\mathfrak{C} = (\text{FO[BIT]-uniform } \mathfrak{C})/\text{poly}$. This actually means that for any class $\mathfrak{D}$ coinciding with FO[BIT]-uniform $\mathfrak{C}$, we immediately get $\mathfrak{C} = \mathfrak{D}/\text{poly}$. We then go on to generalize this result to the counting setting. In Section 7.2 we show that in a very general setting, it holds that $\mathcal{L}[\text{Arb}] = (\mathcal{L}[\text{BIT}])/\text{poly}$ for logics $\mathcal{L}$. Again, we generalize the result to the counting setting. Here, we do so by considering a general form of quantitative logics. As our framework is somewhat unusual, we also briefly go over several examples of logics and quantitative logics to which our results apply, in particular the model-theoretic classes introduced in this thesis. Finally, we will give some concluding remarks regarding the utility of our results and potential further generalizations in Section 7.3.

7.1 Non-uniformity Via Advice Functions in Circuit Complexity

In this section, we will show that non-uniformity of circuit families can equivalently be replaced by non-uniformity given by an advice function under reasonable assumptions. We will first show this for decision classes and then argue that the results transfer to the counting setting. As we will consider more general bases beyond $\mathcal{B}_0$, $\mathcal{B}_1$, and $\mathcal{B}_2$, we need to say a few words on how to describe circuit families over such bases FO[BIT]-uniformly. Different gate types can simply be allowed by using one predicate symbol G_f for each function f in the basis in analogy to the symbols $\mathsf{G}_\wedge$ and $\mathsf{G}_\vee$ from τ_{circ}. In case of non-commutative functions we also need to encode the order of edges. This can be done in the same way as for oracle gates in the definition of the vocabulary $\tau_{\text{o-circ}}$ on page 78, where a predicate for the index of inputs is used.

7.1.1 Decision Classes

Let $\mathfrak{C}$ be a class of Boolean circuits. The fact that $\mathfrak{C} \supseteq (\text{FO[BIT]-uniform } \mathfrak{C})/\text{poly}$ can intuitively be shown by non-uniformly plugging in the correct advice bits into the circuit family with advice. The only issue here is that for the latter class, the advice is considered part of the input, leading to increased resource requirements with respect to the actual input. Using this idea we obtain the following lemma.

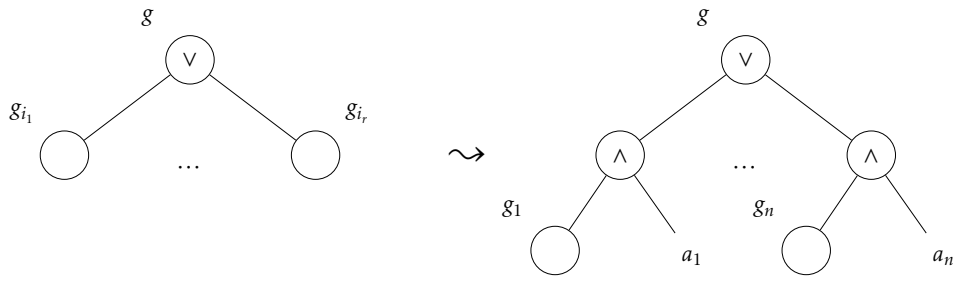

Figure 7.1: Using advice bits to determine the inputs to a disjunction gate.

Lemma 7.1. *Let $\mathcal{B}$ be a Boolean basis, and S and D be sets of functions on $\mathbb{N}$ such that for any $s \in O(S)$, $d \in O(D)$, and for any polynomial p, we have $s \circ p \in O(S)$ and $d \circ p \in O(D)$. Let $\mathfrak{C} := \mathrm{CIRCUIT}(\mathcal{B}, S, D)$. Then* (FO[BIT]-uniform $\mathfrak{C}$)/poly $\subseteq \mathfrak{C}$.

Proof. Let $L \in$ (FO[BIT]-uniform $\mathfrak{C}$)/poly via the FO[BIT]-uniform circuit family $C = (\mathbf{C}_n)_{n\in\mathbb{N}}$ and the polynomially length-bounded advice function $A\colon \mathbb{N} \to \{0,1\}^*$, that is, for any $x \in \{0,1\}^+$ we have

$$x \in L \iff C(x \circ A(|x|)) = 1.$$

Let $C' = (\mathbf{C}'_n)_{n\in\mathbb{N}}$ be the circuit family where $\mathbf{C}'_n$ is defined as $\mathbf{C}_{n+|A(n)|}$ after replacing each gate accessing a bit of the advice by the corresponding constant. Obviously, C' is a (non-uniform) circuit family accepting L. Let $s \in S$ and $d \in D$ be the size and depth of C, respectively, and let p be a polynomial such that $n + |A(n)| \leq p(n)$. Then the size and depth of C' are bounded by $s \circ p$ and $d \circ p$, respectively. By assumption, this means that C' is a $\mathfrak{C}$ circuit family. □

For the converse, from a non-uniform circuit family we need to construct an FO[BIT]-uniform circuit family that takes additional advice. A first idea is to simply connect any pair of gates by an edge and use the advice bits to, in a sense, activate and deactivate these edges. As a simple example, consider a disjunction-gate g with a set $V_g = \{g_1, \ldots, g_n\}$ of potential inputs of g, later this will be the set of gates in the subsequent layer. Let r be the fan-in of g and let $i_1, \ldots, i_r$ be indices such that $g_{i_1}, \ldots, g_{i_r}$ are the actual inputs of g. By introducing new conjunction gates $c_1, \ldots, c_n$, where c_i takes as inputs the gate g_i and a fresh advice bit a_i, and using the new gates c_i as inputs to g instead of the gates g_{i_j}, the advice bits a_i now act in the desired way, activating or deactivating the edges (g_i, g). Now, when setting $a_i = 1$ if and only if $i \in \{i_1, \ldots, i_r\}$, the gate computes the same function as in the original circuit. In Figure 7.1, this construction is illustrated.

There are two issues with this idea: First, connecting every pair of gates by an edge would lead to an (undirected) clique, so the graph would not be a dag. Second, this construction only works in a very specific setting: It uses an unbounded fan-in gate independent of the fan-in of the original gate, and it uses bounded fan-in

conjunction. Furthermore, the construction makes use of both the fact that disjunction is a commutative function and the fact that there is a neutral element for disjunction.

The first issue can be handled by a weaker notion of layered circuits, which ensures that edges are only present between subsequent layers. This allows us to add all possible edges, with the goal of later activating and deactivating them using advice bits, and still preserve the property that the graph is a dag.

Regarding the second issue, note that some important functions such as conjunction and exclusive disjunction share the first two properties. Therefore, we could use the proof idea illustrated in Figure 7.1 for many interesting classes of circuit families. To get a more general result, we will instead use additional gates and advice bits to also determine the order of inputs as well as, in case of unbounded fan-in gates, the actual fan-in by advice bits. While this leads to slightly bigger circuits, it works in much greater generality. Namely, we show this for all classes of circuit families that have access to unbounded fan-in disjunction and bounded fan-in conjunction and a certain robustness with respect to resource bounds. Depending on applications, the more direct idea applicable to functions such as disjunction could be advantageous. We will also adapt the proof idea to classes NC^i to make our results applicable to all classes from the AC-, NC-, SAC-, and TC-hierarchies except for the very restrictive class NC^0.

We now begin by defining a notion of weakly layered circuits, as mentioned above, and showing that for all classes of circuit families with a certain robustness with respect to resource bounds, weakly layered circuits are a normal form. Note that, in contrast to usual notions of layered circuits, we do not require that gates on the same layer have the same type for weakly layered circuits.

Definition 7.2. Let $\mathbf{C} = (V, E, \alpha, \beta, \mathrm{out})$ be a Boolean circuit and let d be the depth of $\mathbf{C}$. We call $\mathbf{C}$ *weakly layered* if V is the disjoint union of sets $L_0, \dots, L_d$ such that $\mathrm{out} \in L_d$ and for all $0 \leq i \leq d$, $v \in L_i$, and $(v, w) \in E$, we have $w \in L_{i+1}$. A circuit family is called *weakly layered* if all circuits in the family are weakly layered.

Lemma 7.3. *Let $C = (\mathbf{C}_n)_{n\in\mathbb{N}}$ be a Boolean circuit family of size s and depth d over some Boolean basis $\mathcal{B}$. Then there is a weakly layered circuit family $C' = (\mathbf{C}'_n)_{n\in\mathbb{N}}$ of size $s \cdot (d+1)$ and depth d over $\mathcal{B}$ that accepts the same language as C.*

Proof. This can easily be achieved by adding a depth-counter to the encoding of gates. Edges then only are present if they were present in the original circuit and the depth-counter is respected. More precisely, for any $n \in \mathbb{N}$ define $\mathbf{C}'_n$ from $\mathbf{C}_n$ as follows: Let $\mathbf{C}_n = (V, E, \alpha, \beta, \mathrm{out})$ and d be the depth of $\mathbf{C}_n$.

Define $\mathbf{C}'_n := (V', E', \alpha', \beta', \mathrm{out}')$, where the components are defined as follows:

$$
\begin{aligned}
V' &:= \{(g, i) \mid g \in V, 0 \leq i \leq d\},\\
E' &:= \{((g_1, i_1), (g_2, i_2)) \mid (g_1, g_2) \in E, i_2 = i_1 + 1\},\\
\alpha'((g_1, i_1), (g_2, i_2)) &:= \alpha(g_1, g_2) \cdot (d+1)^2 + i_1 \cdot (d+1) + i_2,\\
\beta'(g, i) &:= \beta(g), \text{ and}\\
\mathrm{out}' &:= (\mathrm{out}, d).
\end{aligned}
$$

The family $C' := (\mathbf{C}'_n)_{n\in\mathbb{N}}$ obviously has the desired properties. □

We are now in a position to prove that for any class $\mathfrak{C}$ of circuit families with access to unbounded fan-in disjunction and bounded fan-in conjunction and a certain robustness with respect to resource bounds, it holds that $\mathfrak{C} \subseteq (\text{FO[BIT]-uniform } \mathfrak{C})/\text{poly}$.

Theorem 7.4. *Let $\mathcal{B}$ be a Boolean basis, D a set of functions on the natural numbers and $\mathfrak{C} := \text{CIRCUIT}(\mathcal{B}, n^{O(1)}, D)$ such that*

- $O(1) \subseteq O(D)$,
- *for any function $d \in O(D)$ there is a $d' \in O(D)$ with $d \leq d'$ such that $d'(|w|)$ is definable in* FO[BIT] *from* $\mathbf{A}_w \in \text{STRUC}[\tau_{string}]$,
- *unbounded fan-in disjunction is in* FO[BIT]-uniform $\text{CIRCUIT}(\mathcal{B}, n^{O(1)}, O(1))$, *and*
- *bounded fan-in conjunction can be computed by a circuit over $\mathcal{B}$.*

Then $\mathfrak{C} \subseteq (\text{FO[BIT]-uniform } \mathfrak{C})/\text{poly}$.

Proof. We can assume without loss of generality that $O(D) \subseteq n^{O(1)}$, as depth is generally bounded by size. Let $L \in \mathfrak{C}$ via the circuit family $C = (\mathbf{C}_n)_{n\in\mathbb{N}}$ of size s and depth d. Also, we can assume that s and d are FO[BIT]-definable, as we can increase both size and depth to an FO[BIT]-definable upper bound by adding dummy-gates. We proceed as explained earlier by first transforming C to a weakly layered circuit family and then constructing a family with all possible edges present, using advice bits to activate or deactivate edges. Let $C' = (\mathbf{C}'_n)_{n\in\mathbb{N}}$ be a weakly layered circuit family accepting L obtained from C by applying Lemma 7.3. Note that C' has size $O(s \cdot (d+1))$ and depth $O(d)$, and each layer contains exactly $s(n)$ nodes.

Let $n \in \mathbb{N}$ and g be a gate in $\mathbf{C}'_n$. Furthermore, let $L_0, \ldots, L_{d(n)}$ be the layers of $\mathbf{C}'_n$ and let $0 \leq k \leq d(n)$ be such that $g \in L_k$. Assume that $k \geq 1$ for now. Consider the induced subcircuit of $\mathbf{C}'_n$ rooted in g restricted to gates from layers L_k and L_{k-1}. Let $\mathbf{C}'$ be this subcircuit. We now construct from $\mathbf{C}'$ the new circuit $\mathbf{C}$ with output gate g that takes as inputs all input gates of the circuit $\mathbf{C}'_n$, all gates in L_{k-1}, and an advice $a \in \{0,1\}^*$, with the following properties: For any $f \in \mathcal{B}$ and for any combination of gates $g_1, \ldots, g_r \in L_{k-1}$, where r is the arity of f if f has bounded fan-in, and $1 \leq r \leq s(n)$ if f has unbounded fan-in, there is an advice $a \in \{0,1\}^*$ such that $\text{val}(\mathbf{C}, g, x \circ y \circ a) = f(\text{val}(\mathbf{C}, g_1, x \circ y \circ a), \ldots, \text{val}(\mathbf{C}, g_r, x \circ y \circ a))$ for all $x \in \{0,1\}^n$ and $y \in \{0,1\}^{|L_{k-1}|}$. Also, for any input position $0 \leq i \leq n-1$ there is an advice $a \in \{0,1\}^*$ such that $\text{val}(\mathbf{C}, g, x \circ y \circ a) = x_i$ for all $x = x_0 \ldots x_{n-1} \in \{0,1\}^n$ and $y \in \{0,1\}^{|L_{k-1}|}$. Finally, for any constant $b \in \{0,1\}$, there is an advice $a \in \{0,1\}^*$ such that $\text{val}(\mathbf{C}, g, x \circ y \circ a) = b$ for all $x \in \{0,1\}^n$ and $y \in \{0,1\}^{|L_{k-1}|}$. We want to again stress that in $\mathbf{C}$, the gates from L_{k-1} are additional input gates, which is also why the strings y above have length $|L_{k-1}| = s(n)$. Intuitively, the above means that the advice can be used to determine

- the type of gate g and
- the associated input bit, if it is an input gate,
- the required constant, if it is a constant gate, and otherwise

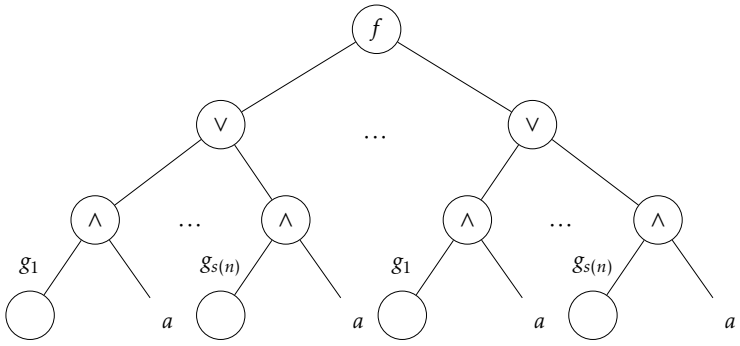

Figure 7.2: Construction to replace non-uniformity by advice bits. Subcircuit $\mathbf{D}_{f,r}$ for a gate g in the original circuit, where $f \in \mathcal{B}$, $1 \leq r \leq s(n)$, and $g_1, \ldots, g_{s(n)}$ are the gates on the previous layer in the original circuit.

- the function from $\mathcal{B}$ the gate g computes,
- the arity r of the gate in case of unbounded fan-in functions, and
- its inputs, chosen among input gates of $\mathbf{C}'_n$ and gates on the previous layer.

For simplicity, we use unbounded fan-in disjunction and bounded fan-in conjunction in the construction. These ultimately have to replaced by subcircuits over $\mathcal{B}$, which exist by assumption.

First, for any function $f \in \mathcal{B}$ and any possible fan-in $1 \leq r \leq s(n)$ of f we construct a subcircuit $\mathbf{D}_{f,r}$ that computes the value of gate g, if g has type f and fan-in r, provided the correct advice bits are chosen. For this, $\mathbf{D}_{f,r}$ takes as inputs all gates from L_{k-1} as well as advice bits $a_{\mathbf{D}_{f,r},1,1}, \ldots, a_{\mathbf{D}_{f,r},r,s(n)}$, where $a_{\mathbf{D}_{f,r},i,j}$ determines whether the i-th input of g is the j-th gate from L_{k-1}, if g computes the function f and has fan-in exactly r. This can be done with a gate of type f getting r inputs $g_{f,1}, \ldots, g_{f,r}$ that are disjunction gates. The gate $g_{f,i}$ determines the i-th input to g by getting as inputs one gate for any gate of the previous layer. Each of these gates is the conjunction of a gate from L_{k-1} and the corresponding advice bit. More precisely, the j-th conjunction gate takes as inputs the j-th gate from L_{k-1} and the advice bit $a_{\mathbf{D}_{f,r},i,j}$. We illustrate this construction in Figure 7.2. Note that we omit indices of advice bits in our illustrations and simply label all advice bits by a. All these advice bits are distinct and as they are provided non-uniformly we can fix any ordering of them to complete the construction.

Next, we will construct for any function $f \in \mathcal{B}$ a subcircuit $\mathbf{D}_f$, which computes the output of g, if g computes the function f and provided the correct advice bits are chosen. For any bounded fan-in function $f \in \mathcal{B}$, we can simply choose $\mathbf{D}_f \coloneqq \mathbf{D}_{f,r}$, where r is the arity of f. For unbounded fan-in functions we use a disjunction over all possible fan-ins $1 \leq r \leq s(n)$ and additional advice bits $a_{\mathbf{D}_f,r}$ for $f \in \mathcal{B}$, $1 \leq r \leq s(n)$ to choose the correct circuit $\mathbf{D}_{f,r}$. This construction is illustrated in Figure 7.3, again omitting indices of advice bits.

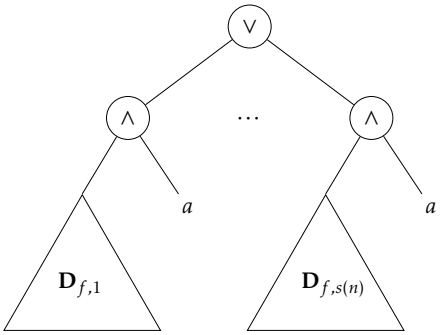

Figure 7.3: Construction to replace non-uniformity by advice bits. Subcircuit $\mathbf{D}_f$ for a gate g in the original circuit, where $f \in \mathcal{B}_{\text{unb}}$.

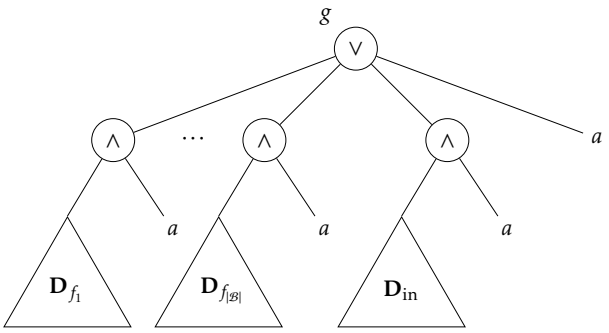

Figure 7.4: Construction to replace non-uniformity by advice bits. Subcircuit $\mathbf{C}$ for a gate g in the the original circuit.

Similarly, we can construct a circuit $\mathbf{D}_{\text{in}}$, which computes a disjunction over n conjunction gates, one for each input position. The i-th of these conjunction gates takes as inputs an input gate for the i-th input bit and an input gate for the additional advice bit $a_{\mathbf{D}_{\text{in}},i}$.

Finally, the circuit $\mathbf{C}$ can be constructed using the same idea as for $\mathbf{D}_f$ for unbounded fan-in gates (and hence also for $\mathbf{D}_{\text{in}}$), using a disjunction over the subcircuits $\mathbf{D}_f$ for all $f \in \mathcal{B}$, $\mathbf{D}_{\text{in}}$, and a gate for the case that g computes a constant, and again using additional advice bits, namely $a_{\mathbf{C},f}$ for $f \in \mathcal{B}$, $a_{\mathbf{C},i}$ for $1 \leq i \leq n$, as well as a_{const}, selecting the correct subcircuit and in case of a constant gate also providing the value for that gate. The construction, using $\mathcal{B} =: \{f_1, \ldots, f_{|\mathcal{B}|}\}$, can be seen in Figure 7.4, indices of advice bits again being omitted.

It is obvious from the construction that, when choosing adequate assignments to the advice bits and using the actual gates from L_{k-1} in $\mathbf{C}'_n$ instead of the input bits in $\mathbf{C}$, the circuit $\mathbf{C}$ computes the value of gate g in the original circuit. Hence, the in-going edges of g can be replaced by the circuit $\mathbf{C}$ in $\mathbf{C}'_n$. This construction has to be repeated for all gates in $\mathbf{C}'_n$.

The size of the circuit $\mathbf{C}$ and thereby also the number of advice bits in this circuit, including the required circuits $\mathbf{D}_f$, $\mathbf{D}_{f,r}$, and $\mathbf{D}_{\text{in}}$, is in

$$O\left(|\mathcal{B}| + \sum_{f \in \mathcal{B}_{\text{bound}}} (\text{ar}(f) \cdot s(n)) + |\mathcal{B}_{\text{unb}}| \cdot s(n)^2 + n\right),$$

where $\text{ar}(f)$ is the arity of f, if f is a Boolean function with bounded fan-in, and $\mathcal{B}_{\text{bound}}$ and $\mathcal{B}_{\text{unb}}$ are the sets of bounded and unbounded fan-in functions in $\mathcal{B}$, respectively. As $|\mathcal{B}|$ and $\text{ar}(f)$ for all $f \in \mathcal{B}_{\text{bound}}$ are constant, this means that both size and number of advice bits are in $O(s(n)^2 + n)$. Furthermore, the depth of $\mathbf{C}$, again including the subcircuits $\mathbf{D}_f$, $\mathbf{D}_{f,r}$ and $\mathbf{D}_{\text{in}}$, is 7 and hence constant.

The case that $k = 0$, that is, $g \in L_0$, can be handled analogously. Here, the subcircuits $\mathbf{D}_f$ and the corresponding advice bits are not required, as we know that such gates can only be input gates or constant gates.

In total, this construction yields a circuit family C'' with the required properties of size $O(s^3 \cdot d + s^2 \cdot d \cdot n)$ and depth $O(d)$. Finally, we need to argue that the constructed circuit family is FO[BIT]-uniform. This is easy to see, as the constructed circuits have a really simple structure: Simply construct a circuit with $d(n) + 1$ layers each of which have $s(n)$ gates and are connected by copies of the subcircuit $\mathbf{C}$ as defined above. Here, it is important that s and d are FO[BIT]-definable, and that the unbounded fan-in disjunction can be computed by a FO[BIT]-uniform circuit family. It follows that $L \in (\text{FO[BIT]-uniform CIRCUIT}(\mathcal{B}, n^{O(1)}, D))/\text{poly}$, finishing the proof. □

The above theorem applies to many natural classes with access to unbounded fan-in disjunction and bounded fan-in conjunction, in particular it is immediately applicable to all classes from the AC-, SAC-, and TC-hierarchies. Among classes we consider in this thesis, the only remaining case are classes from the NC-hierarchy. For this reason, we focus on these classes next, more precisely on classes NC^i with $i \geq 1$.

The issue with using the proof idea of Theorem 7.4 for classes from the NC-hierarchy is that in order to compute unbounded disjunctions over $\mathcal{B}_0$, we need circuits of logarithmic depth. Applying this construction to each gate hence would increase the depth by a logarithmic factor. To handle this, we group operations in NC^i circuit families into batches of logarithmic depth. By using a certain normal form, we can ensure that in all of the batches only copies of the same subcircuit with different inputs occur. Then, all outputs computed by a batch can instead be computed by an unbounded fan-in gate for a fixed unbounded fan-in Boolean function that is not necessarily commutative, more precisely the function computed by the aforementioned subcircuit. This allows us to apply Theorem 7.4 in order to obtain an FO[BIT]-uniform circuit family with advice. Finally, by replacing unbounded fan-in gates by circuits of logarithmic depth, we then obtain the desired circuit family.

As a first step, we need a more restrictive normal form for NC^i circuits than the weakly layered circuits used earlier, which will allow us to group operations into batches as planned. In the following, to simplify the presentation, we do not consider negation gates to be separate gates for non-uniform circuit families in input normal form. Instead,

we consider negated input gates to be input gates that gives access to a negated input bit the same way as we already did in the FO[BIT]-uniform setting.

Lemma 7.5. *Let $i \in \mathbb{N}$, $i \geq 1$ and let $C = (\mathbf{C}_n)_{n\in\mathbb{N}}$ be an NC^i circuit family. Then there is an NC^i circuit family $C' = (\mathbf{C}'_n)_{n\in\mathbb{N}}$ of some depth d accepting the same language as C with the following properties:*

1. *C' is in input normal form,*
2. *C' is weakly layered and alternating with the output gates being conjunction gates,*
3. *for all n, $d(n)$ is divisible by the smallest even number that is at least $\log_2 n$, and*
4. *for all n and all input gates g in $\mathbf{C}'_n$, either there is no path from g to the output gate of $\mathbf{C}'_n$, or the length of any such path is exactly $d(n)$.*

Proof. By Lemma 7.3 we can assume without loss of generality that C is weakly layered. We can also assume it to be in input normal form. In order to make C alternating and have the required properties with respect to its depth we can use similar ideas to those used in the proof of Lemma 4.1.

First, we replace each layer by two layers, a $\wedge$-layer followed by a $\vee$-layer, using the corresponding construction from the proof of Lemma 4.1. More precisely, if g is a disjunction gate with predecessors g_1 and g_2, we change its type to $\wedge$ and add an additional gate $g_\vee$ of type $\vee$. We then replace the edges from g_1 and g_2 to g by the edges $\{(g_1, g_\vee), (g_2, g_\vee), (g_\vee, g_\wedge)\}$. Similarly, for a conjunction gate g, we add two new gates $g_{\vee,1}$ and $g_{\vee,2}$ of type $\vee$ and replace the edges from g_1 and g_2 to g by the edges $\{(g_1, g_{\vee,1}), (g_2, g_{\vee,2}), (g_{\vee,1}, g), (g_{\vee,2}, g)\}$. Recall that this construction was illustrated in Figure 4.1. As in the bounded fan-in case all $\wedge$- and $\vee$-gates need to have fan-in 2, we can then add additional constants as predecessors if needed, in a way that does not change the outputs of the respective gates. Let C' be the resulting circuit family.

We then need to ensure that there is a function $d\colon \mathbb{N} \to \mathbb{N}$ such that for all $n \in \mathbb{N}$, all paths from an input gate to the output gate in $\mathbf{C}'_n$ have length exactly $d(n)$ and that $d(n)$ is divisible by the smallest even number that is at least $\log_2 n$. First note that due to the circuit being layered, all paths from a fixed gate to the output gate have the same length. Now, let $d'(n)$ be the depth of C' and $d(n) \geq d'(n)$ be the next multiple of the smallest even number that is at least $\log_2 n$. For any input gate g, if the distance from g to the output gate is $\leq d(n)$, simply replace g by an alternating path of $\wedge$- and $\vee$-gates of adequate length followed by an input gate accessing the same input bit that g originally accessed. To ensure a fan-in of two for gates on these new paths, add constants again as necessary. □

Theorem 7.6. $\mathrm{NC}^i \subseteq (\text{FO[BIT]-uniform } \mathrm{NC}^i)/\text{poly}$ *for all* $i \geq 1$.

Proof. Let $L \in \mathfrak{C} = \mathrm{NC}^i$ via the circuit family $C = (C_n)_{n\in\mathbb{N}}$ of depth d. By Lemma 7.5 we can assume that C has properties (1)-(4) from the statement of that lemma. Fix some $n \in \mathbb{N}$ and the corresponding circuit $\mathbf{C}_n$ from C. Our goal now is to group the operations in $\mathbf{C}_n$ into subcircuits of depth $d_{\text{batch}}(n)$ for all $n \in \mathbb{N}$, where $d_{\text{batch}}(n)$ is defined as the

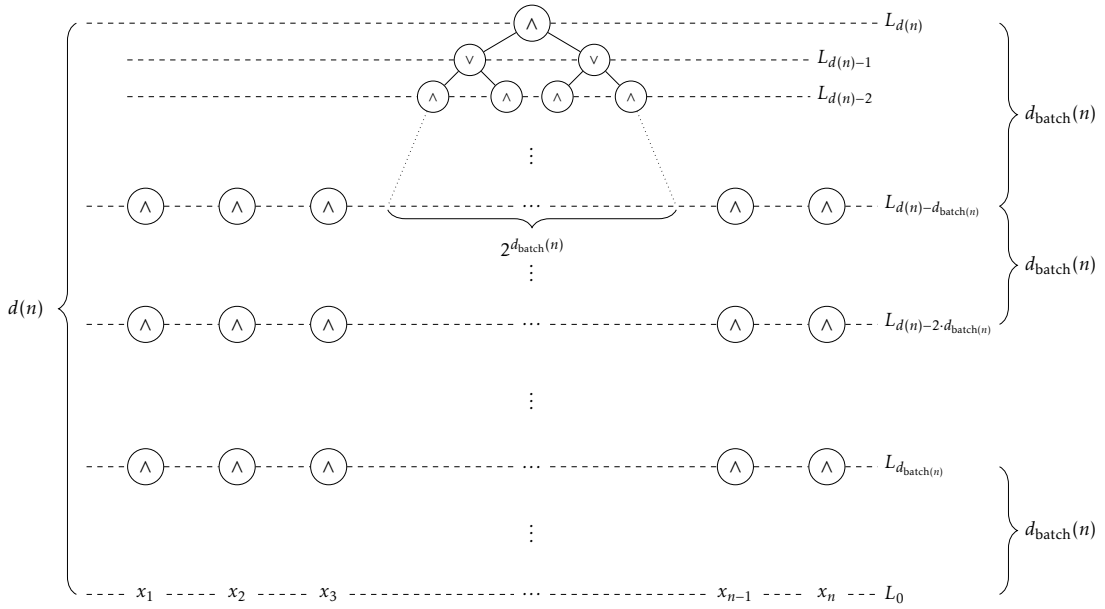

Figure 7.5: Batching construction for classes NC^i. Circuit from an NC^i circuit family in the normal form of Lemma 7.5. Layers marked by dashed lines, layers inside batches of depth $d_{\text{batch}}(n)$ mostly omitted. For simplicity, no constant gates are shown.

smallest even number $\geq \log_2 n$. To make this more precise, let $L_0, \ldots, L_{d(n)}$ be the layers of $\mathbf{C}_n$. Then for $0 \leq j \leq d(n)/d_{\text{batch}}(n)$, the j-th batch is the substructure of $\mathbf{C}_n$ induced by the gates in $L_{j\cdot d_{\text{batch}}(n)} \cup \cdots \cup L_{(j+1)\cdot d_{\text{batch}}(n)}$, i.e., the substructure obtained by restricting the set of gates to the specified set and restricting the relations and function defining the circuit accordingly. This is possible as by assumption, $d(n)$ is divisible by $d_{\text{batch}}(n)$. In the following we will call these subcircuits *batches*. We will call gates in the last layer of a batch, i.e., the layer closest to the output gate of $\mathbf{C}_n$, output gates of that batch. Note that the batches are not necessarily circuits, as there may be non-input gates with fan-in 0. Also, the batches overlap: Batch j and batch $j+1$ both contain all gates from $L_{(j+1)\cdot d_{\text{batch}}(n)}$. This is intended, as we will for each j replace all paths from the first to the last layer in batch $j+1$ by direct edges, and change the gate type of the output gates of the batch accordingly. Here, it can be necessary to add additional gates copying the value of output gates of batch j, as the new output gates of batch $j+1$ might require several connections to the same output gate of batch j. As $\mathbf{C}_n$ is alternating, all paths in each of the batches are alternating between $\wedge$- and $\vee$-gates. Since the output gate is an $\wedge$-gate and $d_{\text{batch}}(n)$ is even, the output gates of each batch are $\wedge$-gates as well. The structure of $\mathbf{C}_n$ including grouping of layers into batches is illustrated in Figure 7.5.

Now consider some $0 \leq j \leq d(n)/d_{\text{batch}}(n)$ and a non-constant gate $g \in L_{(j+1)\cdot d_{\text{batch}}(n)}$, i.e., g is a non-constant output gate of batch j. Let $\mathbf{C}_g$ be the subcircuit of $\mathbf{C}_n$ rooted in g

and let $\mathbf{C}'_g$ be the substructure of $\mathbf{C}_g$ induced by $L_{j\cdot d_{\text{batch}}(n)} \cup \cdots \cup L_{(j+1)\cdot d_{\text{batch}}(n)}$. If $j > 0$, the structure $\mathbf{C}'_g$ is not a circuit, as the gates in its lowest layer are not input gates but still have no predecessors. We can remedy this by simply treating the gates on the lowest layer as input gates, associating the ℓ-th gate in $L_{j\cdot d_{\text{batch}}(n)}$ with the ℓ-th input bit. Now by assumption, $\mathbf{C}'_g$ is a weakly layered, alternating circuit with exactly $d_{\text{batch}}(n)$ layers. Let $\mathbf{C}''_g$ be the circuit $\mathbf{C}'_g$ unfolded into tree-shape and extended to a full binary tree of depth $d_{\text{batch}}(n)$. For the latter, constant gates are replaced by full binary trees of adequate depth. This transformation does not increase the depth of the circuit, so the size of $\mathbf{C}''_g$ is $2^{d_{\text{batch}}(n)}$, which is still polynomial in n when applying the transformations to all circuits in C. This structure is partially illustrated for the output gate of $\mathbf{C}_n$ in Figure 7.5. As the underlying graph of $\mathbf{C}''_g$ now is a full binary tree, the output gate is an $\wedge$-gate, and $\mathbf{C}''_g$ is still alternating, $\mathbf{C}''_g$ now actually is a fixed circuit apart from the input gates. This means that the function computed by g is the fixed function h_n computed by such a circuit, with inputs that are either constants or chosen from $L_{j\cdot d_{\text{batch}}(n)}$.

This means that we can replace for all $1 \leq j \leq d(n)/d_{\text{batch}}(n)$ all gates in $L_{j\cdot d_{\text{batch}}(n)}$ by gates of type h_n with inputs chosen among new constant gates or gates in $L_{(j-1)\cdot d_{\text{batch}}(n)}$, and remove all layers $L_{j'}$ with $(j-1)\cdot d_{\text{batch}}(n) < j' < j \cdot d_{\text{batch}}$. The only issue is that this construction may introduce multiple edges for the same pair of gates. To prevent this, we create a number of copies of each of those gates. As the total number of inputs for each of the gates of type h_n is polynomial, this requires only a polynomial number of copies. This yields a circuit $\mathbf{C}'_n$ of depth $d(n)/d_{\text{batch}}(n)$ using gates computing the function h_n. By applying this construction for each n, we obtain a circuit family C' of polynomial size and depth $O(d(n)/\log n)$ over basis $\{(h_n)_{n\in\mathbb{N}}, f_0, f_1\} \subseteq \{\vee, \wedge_2, (h_n)_{n\in\mathbb{N}}, f_0, f_1\}$ such that for all $x \in \{0,1\}^*$:

$$x \in L \iff C'(x) = 1.$$

Let $\mathfrak{C}' := \text{CIRCUIT}(\{\vee, \wedge_2, (h_n)_{n\in\mathbb{N}}, f_0, f_1\}, n^{O(1)}, O((\log n)^{i-1}))$. Obviously, $C' \in \mathfrak{C}'$. As $\mathfrak{C}'$ has the required properties, Theorem 7.4 now yields an FO[BIT]-uniform $\mathfrak{C}'$ circuit family C'' and a polynomially length-bounded advice function $A\colon \mathbb{N} \to \{0,1\}^*$ such that for all $x \in \{0,1\}^+$:

$$x \in L \iff C''(x \circ A(|x|)).$$

Finally, we replace any gate g in C'' computing one of the unbounded fan-in functions $\vee$ and $(h_n)_{n\in\mathbb{N}}$ by a subcircuit of logarithmic depth, obtaining the circuit family C'''. This is possible as both $\vee$ and $(h_n)_{n\in\mathbb{N}}$ can be computed in FO[BIT]-uniform NC^1. Now, C''' is a circuit family over $\mathcal{B}_0$. As C'' has polynomial size and depth $O((\log n)^{i-1})$, C''' has polynomial size and depth $O((\log n)^i)$, which means that C''' is an FO[BIT]-uniform NC^i circuit family. As this last step does not change the advice bits, we immediately have for all $x \in \{0,1\}^+$,

$$x \in L \iff C'''(x \circ A(|x|)) = 1,$$

finishing the proof. □

Summarizing, we have obtained the following result for the relationship of non-uniform classes based on different forms of non-uniformity.

Corollary 7.7. *Let $\mathcal{B}$ be a Boolean basis, D be a non-empty set of functions on $\mathbb{N}$. Let $\mathfrak{C}$ be either NC^i for some $i \geq 1$, or $\mathrm{CIRCUIT}(\mathcal{B}, n^{O(1)}, D)$ with the following properties:*

- *$O(1) \subseteq O(D)$, and for any $d \in O(D)$ and polynomial p, we have $d \circ p \in O(D)$,*
- *for any $d \in O(D)$ there is a $d' \in D$ with $d \leq d'$ such that $d'(|w|)$ is definable in $\mathrm{FO[BIT]}$ from $\mathbf{A}_w \in \mathrm{STRUC}[\tau_{string}]$,*
- *unbounded fan-in disjunction is in $\mathrm{FO[BIT]}$-uniform $\mathrm{CIRCUIT}(\mathcal{B}, n^{O(1)}, O(1))$, and*
- *bounded fan-in conjunction can be computed by a circuit over $\mathcal{B}$.*

Then $\mathfrak{C} = (\mathrm{FO[BIT]}\text{-uniform } \mathfrak{C})/\mathrm{poly}$.

We formulated the above results in terms of FO[BIT]-uniformity, as this is the type of uniformity used throughout this thesis. An immediately arising question is whether the results also apply to other forms of uniformity. We will briefly discuss this next. Lemma 7.1 directly transfers to any reasonable uniformity, as the uniformity condition is not relevant for the proof (we could even use the class $\mathfrak{C}/\mathrm{poly}$ instead of $(\mathrm{FO[BIT]}\text{-uniform } \mathfrak{C})/\mathrm{poly}$). For Theorems 7.4 and 7.6 it is obvious that the result transfers to less restrictive types of uniformity. Regarding more restrictive forms of uniformity, there are not many options commonly used. For most classes, $\mathrm{U_D}$-uniformity coincides with FO[BIT]-uniformity. Regarding NC^1, and possibly all classes using at least logarithmic depth, we expect that the proofs transfer to the $\mathrm{U_E^*}$-uniform setting, but we will not proof this here. As other very weak forms of uniformity such as FO[<]-uniformity are not the focus of this thesis, we do not attempt to generalize the results to this setting. It could be interesting to do this in the future.

7.1.2 Counting Classes

We will now show that the results of the previous section also apply to counting classes from circuit complexity, i.e., that $\#\mathfrak{C} = (\mathrm{FO[BIT]}\text{-uniform } \#\mathfrak{C})/\mathrm{poly}$ for a wide variety of classes $\mathfrak{C}$. For the most part, the proofs of the previous section directly transfer to the counting setting. We will argue why they also apply to counting and how to handle the few points where the constructions have to be modified.

First, note that it would be natural to only cover classes of circuit families over bases $\mathcal{B} \subseteq \{\wedge, \vee, \wedge_k, \vee_k, \neg, f_0, f_1 \mid k \in \mathbb{N}\}$ in this section, since we defined counting classes only over such bases. The proof of Theorem 7.6 used Theorem 7.4 with a basis containing a different family of Boolean functions, though. Hence, if we only prove the counting version of Theorem 7.4 for bases $\mathcal{B} \subseteq \{\wedge, \vee, \wedge_k, \vee_k, \neg, f_0, f_1 \mid k \in \mathbb{N}\}$, we will not be able to use it for the counting version of Theorem 7.6. For this reason, we show a more general version of the former, i.e., we show it for a general form of counting arithmetic circuits. For completeness, we will also state the counting version of Lemma 7.1 in this general setting. We begin by defining the required generalization of counting arithmetic

circuits, allowing arbitrary functions and families of functions over $\mathbb{N}$, but still only using (possibly negated) bits as inputs.

Definition 7.8. Let $\mathcal{B}$ be a finite set of functions $f : \mathbb{N}^k \to \mathbb{N}$ for some $k \in \mathbb{N}$ and families $(f_i)_{i\in\mathbb{N}}$ of functions $f_i : \mathbb{N}^i \to \mathbb{N}$, and $n \in \mathbb{N}$. We call $\mathcal{B}$ a *generalized counting arithmetic basis*. A *generalized counting arithmetic circuit over $\mathcal{B}$ on n inputs* is a tuple $(V, E, \alpha, \beta, \text{out})$ with the following properties:

- (V, E) is a dag,
- α is an injective function $E \to \mathbb{N}$,
- β is a function $V \to \{0, \dots, n-1, \neg 0, \dots, \neg n-1\} \cup \mathcal{B}$,
- if $\beta(g) \in \{0, \dots, n-1, \neg 0, \dots, \neg n-1, f_0, f_1\}$ for some $g \in V$, then g has in-degree 0 in the graph (V, E), and
- if $\beta(g) \in \mathcal{B}$ for some $g \in V$ and m is the in-degree of g in (V, E), then $\beta(g)$ is a function of arity m or a family of functions.

Families of generalized counting arithmetic circuits, *size* and *depth* of such families, as well as the function $\mathbf{C}$ or C computed by a circuit $\mathbf{C}$ or circuit family C, respectively, are defined in analogy to the respective notions for Boolean circuit families.

As usual, we say that β labels the gates in the circuit (by the function or input bit they compute). We use $\text{CIRCUIT}(\mathcal{B}, s, d)$ to refer to the class of families of generalized counting arithmetic circuits of size s and depth d over $\mathcal{B}$ as well as the class of functions computed by such circuits. This generalizes to the case where, instead of s and d, sets of functions S and D are given. Note that the above is a very general notion of counting arithmetic circuits as we do not even require that there are Boolean functions that result in the counting functions used in the basis of a generalized counting arithmetic circuit.

In order to show (FO[BIT]-uniform $\#\mathfrak{C}$)/poly $\subseteq \#\mathfrak{C}$, the proof idea of simply plugging in the advice bits directly works in the counting setting.

Lemma 7.9. *Let $\mathcal{B}$ be a generalized counting arithmetic basis, and S and D be sets of functions on $\mathbb{N}$ such that for any $s \in O(S)$, $d \in O(D)$, and for any polynomial p, we have $s \circ p \in O(S)$ and $d \circ p \in O(D)$. Let $\mathfrak{C} := \text{CIRCUIT}(\mathcal{B}, S, D)$. Then* (FO[BIT]-uniform $\#\mathfrak{C}$)/poly $\subseteq \#\mathfrak{C}$.

Proof. This can be proven completely analogously to Lemma 7.1: Input gates that access parts of the advice can again be replaced by the corresponding constant gates, and the arguments concerning size and depth of the circuit family also apply in the counting setting. □

We will now prove the counting version of Theorem 7.4 for classes of functions defined in terms of the above notion of generalized counting arithmetic circuits. The construction in the proof of the decision version ultimately only uses arbitrary gates from the basis with inputs that compute exactly the values of their original inputs, and the remainder of the construction is based on $\wedge$- and $\vee$-gates that have exactly one

witness if they evaluate to 1. Hence, the proof can directly be transferred to this setting, using gates for $\times$ and $+$ instead of $\wedge$ and $\vee$, respectively. Regarding the required normal form, note that the notion of weakly layered circuits directly transfers to generalized counting arithmetic circuits, as it is based on a property of the underlying graphs of the circuits in a family. As a first step, we now show that weakly layered circuit families are a normal form in this setting, too.

Lemma 7.10. *Let $C = (\mathbf{C}_n)_{n\in\mathbb{N}}$ be a family of generalized counting arithmetic circuits over some basis $\mathcal{B}$ of size s and depth d. Then there is a weakly layered family $C' = (\mathbf{C}'_n)_{n\in\mathbb{N}}$ of generalized counting arithmetic circuits of size $s \cdot (d+1)$ and depth d over basis $\mathcal{B}$ that computes the same function as C.*

Proof. The construction from the proof of Lemma 7.3 is applicable for arbitrary gate types, as only the underlying graph is modified. For all circuits in the family, the subcircuit induced by the set of gates for which there is a path to the output gate is isomorphic to the the respective subcircuit of the original circuit from C. Hence, the construction directly works in the counting setting. □

We are now in a position to prove the counting version of Theorem 7.4.

Theorem 7.11. *Let $\mathcal{B}$ be a generalized counting arithmetic basis, D be a set of functions on $\mathbb{N}$, and $\mathfrak{C} := \mathrm{CIRCUIT}(\mathcal{B}, n^{O(1)}, D)$ such that*

- *$O(1) \subseteq O(D)$,*
- *for any $d \in D$ there is a $d' \in D$ with $d \leq d'$ such that $d'(|w|)$ is definable in* FO[BIT] *from $\mathbf{A}_w \in \mathrm{STRUC}[\tau_{string}]$,*
- *unbounded fan-in summation is in* FO[BIT]*-uniform* $\mathrm{CIRCUIT}(\mathcal{B}, n^{O(1)}, O(1))$*, and*
- *bounded fan-in multiplication can be computed by a circuit over $\mathcal{B}$.*

Then $\mathfrak{C} \subseteq$ (FO[BIT]-uniform $\mathfrak{C}$)/poly.

Proof. This can be proven analogously to Theorem 7.4. Similar to the decision version, we describe the construction using unbounded fan-in addition and bounded fan-in multiplication for simplicity. These gates ultimately have to be replaced by the corresponding subcircuits. Let $F \in \mathfrak{C}$ via the circuit family $C = (\mathbf{C}_n)_{n\in\mathbb{N}}$ of size s and depth d. Without loss of generality assume that $d \leq s$ and both s and d are FO[BIT]-definable. Let $C' = (\mathbf{C}'_n)_{n\in\mathbb{N}}$ be a weakly layered circuit family computing F. Note that C' has size $s \cdot (d+1)$ and depth d, and each layer contains exactly $s(n)$ nodes.

Let $n \in \mathbb{N}$ and g be a gate in $\mathbf{C}'_n$. Let $L_0, \ldots, L_{d(n)}$ be the layers of $\mathbf{C}'_n$ and let $0 \leq k \leq d(n)$ be such that $g \in L_k$. Assume that $k \geq 1$ for now. As in the proof of the decision version, a circuit $\mathbf{C}$ with output gate g is constructed that takes as inputs all input gates of the circuit $\mathbf{C}'_n$, all gates in L_{k-1} and an advice $a \in \{0,1\}^*$, and has the following properties: For any $f \in \mathcal{B}$ and for any combination of gates $g_1, \ldots, g_r \in L_{k-1}$, where r is the arity of f if f has bounded fan-in, and $1 \leq r \leq s(n)$ if f has unbounded fan-in, there is an

advice $a \in \{0,1\}^*$ such that $\mathrm{val}(\mathbf{C}, g, x \circ y \circ a) = f(\mathrm{val}(\mathbf{C}, g_1, x \circ y \circ a), \ldots, \mathrm{val}(\mathbf{C}, g_r, x \circ y \circ a))$ for all $x \in \{0,1\}^n$ and $y \in \{0,1\}^{|L_{k-1}|}$. Also, for any input position $0 \leq i \leq n-1$ there is an advice $a \in \{0,1\}^*$ such that $\mathrm{val}(\mathbf{C}, g, x \circ y \circ a) = x_i$ for all $x = x_0 \ldots x_{n-1} \in \{0,1\}^n$ and $y \in \{0,1\}^{|L_{k-1}|}$. Finally, for any constant $b \in \{0,1\}$, there is again an advice $a \in \{0,1\}^*$ such that $\mathrm{val}(\mathbf{C}, g, x \circ y \circ a) = b$ for all $x \in \{0,1\}^n$ and $y \in \{0,1\}^{|L_{k-1}|}$. Here, val is defined analogously to how it is defined for Boolean circuits, i.e., $\mathrm{val}(\mathbf{C}, g, x)$ is the value of gate g in circuit $\mathbf{C}$, when x is given to $\mathbf{C}$ as input. Note that in the proof of the decision version, $\mathbf{C}$ is constructed in such a way that for any newly added conjunction gate, one of its two inputs is an advice bit, and for an adequate choice of the advice bits, at most one predecessor of any newly added disjunction gate is non-zero. Hence, we can simply use summation-gates instead of disjunction-gates and multiplication-gates instead of conjunction-gates to obtain generalized counting arithmetic circuits with the same functionality. The case that $k = 0$, i.e., $g \in L_0$, can be handled analogously again.

The desired circuit family can than be constructed by repeating the construction for all gates. The arguments concerning size, depth, and uniformity of the resulting circuit family are identical to the corresponding arguments for the decision version, finishing the proof. □

Next, we cover the case of classes from the #NC-hierarchy. We begin by transferring the normal form from Lemma 7.5 to the counting setting. This normal form together with the previous theorem will then allow us to prove the counting version of Theorem 7.6.

Lemma 7.12. *Let $i \in \mathbb{N}$ with $i \geq 1$ and let $C = (\mathbf{C}_n)_{n \in \mathbb{N}}$ be a $\#\mathrm{NC}^i$ circuit family. Then there is an $\#\mathrm{NC}^i$ circuit family $C' = (\mathbf{C}'_n)_{n \in \mathbb{N}}$ of some depth d accepting the same language with the following properties:*

1. *C' is weakly layered and alternating with the output gates being addition gates,*
2. *for all n, $d(n)$ is divisible by the smallest even number that is at least $\log_2 n$, and*
3. *for all n and all input gates g in $\mathbf{C}'_n$, either there is no path from g to the output gate of $\mathbf{C}'_n$, or the length of any such path is exactly $d(n)$.*

Proof. By Lemma 7.10 we can assume without loss of generality that C is weakly layered. As C is a counting arithmetic circuit, it is in input normal form. Now, the same construction as in the proof of Lemma 7.5 can be used. Here, conjunction and disjunction are replaced by multiplication and addition, respectively. The construction is based on the idea of adding new dummy-gates between gates, which only have one input in order to not change the output. To ensure that $\wedge$- and $\vee$-gates have fan-in two, additional constant gates were added as predecessors to gates that would otherwise only have one predecessor. This construction obviously still works in the counting setting, as 0 and 1 are the neutral elements not only for $\vee$ and $\wedge$, respectively, but also for $+$ and $\times$, respectively. □

Finally, we are in a position to show the counting version of Theorem 7.6, i.e., $\#\mathrm{NC}^i \subseteq (\mathrm{FO[BIT]}\text{-uniform } \#\mathrm{NC}^i)/\mathrm{poly}$. As mentioned earlier, we proved Theorem 7.11

for generalized counting arithmetic circuits as this will allow us to apply it in the following proof. The reason is that similar to the proof of Theorem 7.6, parts of circuits are grouped into batches and can then be computed by individual gates, which compute the function computed by an alternating, tree-shape circuit of the respective depth. By applying this transformation to the whole circuit family, we obtain a family of generalized counting arithmetic circuits, for which we can apply Theorem 7.11.

Theorem 7.13. $\#\mathrm{NC}^i \subseteq (\mathrm{FO[BIT]}\text{-uniform } \#\mathrm{NC}^i)/\mathrm{poly}$ *for all* $i \geq 1$.

Proof. We proceed analogously to the proof of Theorem 7.6. Let $L \in \mathfrak{C} = \#\mathrm{NC}^i$ via the circuit family $C = (\mathbf{C}_n)_{n\in\mathbb{N}}$ of depth d. By Lemma 7.12 we can assume that C has properties (1)-(3) from the statement of that lemma. Fix some $n \in \mathbb{N}$ and the corresponding circuit $\mathbf{C}_n$ from C. Define $d_{\mathrm{batch}}(n)$, $L_0, \ldots, L_{d(n)}$ and batches 0 to $d(n)/d_{\mathrm{batch}}(n)$ in the same way as in the proof of Theorem 7.6.

Now consider some $0 \leq j \leq d(n)/d_{\mathrm{batch}}(n)$ and a non-constant gate $g \in L_{(j+1)\cdot d_{\mathrm{batch}}(n)}$. By the assumptions, g is an $\times$-gate. Let $\mathbf{C}_g$ be the subcircuit of $\mathbf{C}_n$ rooted in g and let $\mathbf{C}'_g$ be the substructure of $\mathbf{C}_g$ induced by $L_{j\cdot d_{\mathrm{batch}}(n)} \cup \cdots \cup L_{(j+1)\cdot d_{\mathrm{batch}}(n)}$. As in the proof of the decision version, the gates on the lowest layer of $\mathbf{C}'_g$ can be treated as input gates, basically associating the ℓ-th gate in $L_{j\cdot d_{\mathrm{batch}}(n)}$ with the ℓ-th input gate. Let $\mathbf{C}''_g$ be the circuit $\mathbf{C}'_g$ unfolded into tree-shape and extended to a full binary tree of depth $d_{\mathrm{batch}}(n)$. For the latter, constant gates are again replaced by full binary trees of adequate depth. The size of $\mathbf{C}''_g$ is $2^{d_{\mathrm{batch}}(n)}$, which is polynomial in n (when considering the whole family C). Now, the subcircuit $\mathbf{C}''_g$ is a fixed circuit apart from the input gates. This means that the function computed by g is the fixed function h_n computed by such a circuit with inputs being gates from $L_{j\cdot d_{\mathrm{batch}}(n)}$ or constant gates.

This allows us to construct a circuit $\mathbf{C}'_n$ of depth $d(n)/d_{\mathrm{batch}}(n)$ using gates computing the function h_n or constants. Due to possible multiplicities of edges, we again have to introduce a polynomial number of copies of gates. Ultimately, this results in a circuit family C' of polynomial size and depth $d(n)/d_{\mathrm{batch}}(n)$ over basis $\{(h_n)_{n\in\mathbb{N}}, f_0, f_1\} \subseteq \{+, \times_2, (h_n)_{n\in\mathbb{N}}, f_0, f_1\}$ such that for all $x \in \{0,1\}^*$:

$$f(x) = C'(x).$$

By applying Theorem 7.11, we obtain an FO[BIT]-uniform circuit family C'' over basis $\{+, \times_2, (h_n)_{n\in\mathbb{N}}, f_0, f_1\}$ of polynomial size and depth $O((\log n)^{i-1})$ and a polynomially length-bounded advice function $A\colon \mathbb{N} \to \{0,1\}^*$ such that for all $x \in \{0,1\}^+$,

$$f(x) = C''(x \circ A(|x|)).$$

Finally, gates computing either an unbounded fan-in sum or a function from $(h_n)_{n\in\mathbb{N}}$ can be FO[BIT]-uniformly replaced by subcircuits of depth $O(\log n)$ over basis $\{+_2, \times_2\}$, yielding a FO[BIT]-uniform $\#\mathrm{NC}^i$ circuit family C'''. □

In the following corollary, we summarize the results on the relationship of non-uniform classes based on different forms of non-uniformity in the counting setting.

Corollary 7.14. *Let $\mathcal{B}$ be a generalized counting arithmetic basis, and D be a set of functions on $\mathbb{N}$. Let $\mathfrak{C}$ be either $\#\mathrm{NC}^i$ for some $i \geq 1$, or $\mathrm{CIRCUIT}(\mathcal{B}, n^{O(1)}, D)$ with the following properties:*

- *$O(1) \subseteq O(D)$, and for any $d \in O(D)$ and polynomial p, we have $d \circ p \in O(D)$,*
- *for any $d \in O(D)$ there is a $d' \in O(D)$ with $d \leq d'$ such that $d'(|w|)$ is definable in $\mathrm{FO}[\mathrm{BIT}]$ from $\mathbf{A}_w \in \mathrm{STRUC}[\tau_{string}]$,*
- *unbounded fan-in summation is in $\mathrm{FO}[\mathrm{BIT}]$-uniform $\mathrm{CIRCUIT}(\mathcal{B}, n^{O(1)}, O(1))$, and*
- *bounded fan-in multiplication can be computed by a circuit over $\mathcal{B}$.*

Then $\mathfrak{C} = (\mathrm{FO}[\mathrm{BIT}]\text{-uniform } \mathfrak{C})/\mathrm{poly}$.

7.2 Non-uniformity Via Advice Functions in Logics

We will now turn our attention to complexity classes based on logics for which valuations are first-order structures. It turns out that due to the closer similarity of built-in predicates in first-order logic and a polynomially length-bounded advice function, the two notions coincide for a wide variety of classes. While we do require that the logic defines properties of first-order structures and will also need a certain closure property, we will not make any assumptions about what kind of objects formulae of the logic are.

We will first cover the case of decision classes and will then argue that the proofs directly transfer to the counting setting.

7.2.1 Decision Classes

Here, we will show that for a wide variety of model-theoretic decision classes, the different types of non-uniformity coincide. For this, we use a very general notion of logic, which was used by Lück [Lüc20], identifying sufficient assumptions that allow us to prove that the different kinds of uniformity coincide.

We start by introducing the general notion of logic.

Definition 7.15. A *logic* $\mathcal{L}$ is a triple $\mathcal{L} = (\Phi_{\mathcal{L}}, \mathfrak{A}_{\mathcal{L}}, \models_{\mathcal{L}})$, where $\Phi_{\mathcal{L}}$ and $\mathfrak{A}_{\mathcal{L}}$ are classes and $\models_{\mathcal{L}} \subseteq \mathfrak{A}_{\mathcal{L}} \times \Phi_{\mathcal{L}}$ is a relation, called the *modeling relation of $\mathcal{L}$*. We call elements of $\Phi_{\mathcal{L}}$ *formulae of $\mathcal{L}$* and elements of $\mathfrak{A}_{\mathcal{L}}$ *valuations of $\mathcal{L}$*.

To illustrate the definition, we describe how first-order logic FO over finite, relational structures and restricted to sentences, as usually used in the context of descriptive complexity, fits into this framework.

Example 7.16. First-order logic over finite, relational structures and restricted to sentences can be defined in the above framework by setting $\mathrm{FO} := (\Phi_{\mathrm{FO}}, \mathfrak{A}_{\mathrm{FO}}, \models_{\mathrm{FO}})$, where Φ_{FO} is the class of all first-order sentences over arbitrary vocabularies, $\mathfrak{A}_{\mathrm{FO}}$ is the class of all finite first-order structures over arbitrary vocabularies with domains of the form $\{0, \ldots, n-1\}$ for some $n \in \mathbb{N}$, and $\models_{\mathrm{FO}}$ is the usual modeling relation of first-order logic.

For simplicity, we assume that first-order vocabularies are described by the number of their relation symbols and a tuple of the arities of these symbols. We can then refer to the symbols using indices. Further, every formula $\varphi \in \Phi_{\mathrm{FO}}$ contains information about the associated vocabulary, and for a formula φ with associated vocabulary σ, we only allow structures from $\mathrm{STRUC}[\sigma]$ to be models of φ.

For any vocabulary σ we denote by $\Phi^\sigma_{\mathrm{FO}}$ the class of σ-sentences and by $\mathfrak{A}^\sigma_{\mathrm{FO}} := \mathrm{STRUC}[\sigma]$ the class of σ-structures. Accordingly, Φ_{FO} and $\mathfrak{A}_{\mathrm{FO}}$ are the disjoint unions of all classes $\Phi^\sigma_{\mathrm{FO}}$ and $\mathfrak{A}^\sigma_{\mathrm{FO}}$, respectively. It then holds that for any vocabulary σ, $\varphi \in \Phi^\sigma_{\mathrm{FO}}$, and $\mathbf{A} \in \mathfrak{A}_{\mathrm{FO}}$, we have

$$\mathbf{A} \models \varphi \Rightarrow \mathbf{A} \in \mathfrak{A}^\sigma_{\mathrm{FO}}.$$

Note that this framework is very general, allowing many objects that would usually not be considered logics. As we are only interested in a framework to transfer given results from the FO[BIT]-uniform setting to the non-uniform setting, this is not an issue here. It is not the goal of this chapter to contribute to the discussion on what objects should be considered logics.

In the following, we are interested in logics where valuations are first-order structures. The reason is that by associating first-order structures with their binary encodings, such logics define classes of languages in the usual sense of descriptive complexity, and thus give rise to model-theoretic characterizations of complexity classes. Furthermore, in order to define built-in predicates, we need to be able to distinguish between formulae (and structures) over different first-order vocabularies. For this reason, we introduce the following notion of *logics over* FO-*structures*.

Definition 7.17. A logic $\mathcal{L} = (\Phi_{\mathcal{L}}, \mathfrak{A}_{\mathcal{L}}, \models_{\mathcal{L}})$ is called a *logic over* FO-*structures*, if

$$\mathfrak{A}_{\mathcal{L}} = \mathfrak{A}_{\mathrm{FO}} \quad \text{and} \quad \Phi_{\mathcal{L}} = \bigsqcup_{\substack{\text{FO-voc-}\\ \text{abulary } \sigma}} \Phi^\sigma_{\mathcal{L}}$$

for subclasses $\Phi^\sigma_{\mathcal{L}}$ of $\Phi_{\mathcal{L}}$ such that for any FO-vocabulary σ, $\varphi \in \Phi^\sigma_{\mathcal{L}}$, and $\mathbf{A} \in \mathfrak{A}_{\mathrm{FO}}$, we have

$$\mathbf{A} \models_{\mathcal{L}} \varphi \Rightarrow \mathbf{A} \in \mathfrak{A}^\sigma_{\mathrm{FO}} = \mathrm{STRUC}[\sigma].$$

Now, for any logic $\mathcal{L}$ over FO-structures and any set of numerical predicates $\mathfrak{R}$, we can define the complexity class $\mathcal{L}[\mathfrak{R}]$ of languages that are exactly sets of encodings of the models of $\mathcal{L}$-formulae with built-in predicate symbols that are interpreted by predicates from $\mathfrak{R}$.

Definition 7.18. Let $\mathcal{L} = (\Phi_{\mathcal{L}}, \mathfrak{A}_{\mathcal{L}}, \models_{\mathcal{L}})$ be a logic over FO-structures and $\mathfrak{R}$ be a set of relations over $\mathbb{N}$. Then $\mathcal{L}[\mathfrak{R}]$ is the class of languages $L \subseteq \{0,1\}^+$ for which there are vocabularies σ and $\tau = (\mathsf{N}_1, \ldots, \mathsf{N}_k)$, a formula $\varphi \in \Phi^{\sigma\cup\tau}_{\mathcal{L}}$, and a non-uniform family $\mathcal{I} = (\mathbf{I}_n)_{n\in\mathbb{N}}$ of interpretations of symbols in τ by relations from $\mathfrak{R}$ such that for all $\mathbf{A} \in \mathrm{STRUC}[\sigma]$, we have

$$\mathrm{enc}_\sigma(\mathbf{A}) \in L \iff \mathbf{A} \cup \mathbf{I}_{|\mathrm{dom}(\mathbf{A})|}(\tau) \models \varphi,$$

and $x \notin L$ if x is not the encoding of a σ-structure. Here, $\mathbf{I}_{|\mathrm{dom}(\mathbf{A})|}(\tau)$ is a shorthand for $(\mathbf{I}_{|\mathrm{dom}(\mathbf{A})|}(\mathsf{N}_1), \ldots, \mathbf{I}_{|\mathrm{dom}(\mathbf{A})|}(\mathsf{N}_k))$.

As desired, when using the definition of first-order logic in this framework given in Example 7.16, the corresponding complexity class FO[$\mathfrak{R}$] arising from Definition 7.18 coincides with our earlier definition of this class for any set of built-in relations $\mathfrak{R}$. Similar as for FO[$\mathfrak{R}$], we will also by abuse of notation identify numerical predicate symbols with numerical predicates, as the definition asks for the existence of a corresponding interpretation of the symbols and hence we can always assume that specific symbols are interpreted as intended.

Recall that the goal of this section is to show $\mathcal{L}[\mathrm{Arb}] = \mathcal{L}[\mathrm{BIT}]/\mathrm{poly}$ for a wide variety of logics $\mathcal{L}$. Hence, we will exclusively be interested in classes $\mathcal{L}[\mathfrak{R}]$ where $\mathfrak{R}$ is either {BIT} or Arb. In order to show that the two classes coincide, the logic has to be powerful enough to encode and decode non-uniform information in the different forms, i.e., either given as part of the input structure as in the definition of $\mathcal{L}[\mathrm{Arb}]$, or as part of the encoding of the input structure as in the definition of $\mathcal{L}[\mathrm{BIT}]/\mathrm{poly}$. As FO-queries are powerful enough for this, we can formalize this by requiring that the logic is closed under application of FO-queries. This property corresponds to the property of #Win-FO[BIT] described in Lemma 3.4, generalized to arbitrary sets of built-in predicates. For technical reasons, we will need a slightly more general notion of this closure property. In order to simplify the statement of our main theorem, we first precisely define this notion.

Definition 7.19. Let $\mathcal{L} = (\Phi_{\mathcal{L}}, \mathfrak{A}_{\mathrm{FO}}, \models_{\mathcal{L}})$ be a logic over FO-structures and $\mathfrak{R}$ be a set of relations over $\mathbb{N}$. We say that $\mathcal{L}[\mathfrak{R}]$ *is closed under applying partial* FO[$\mathfrak{R}$]*-queries*, if for any

- vocabularies σ, σ', τ' with $\tau' = (\mathsf{N}_1, \ldots, \mathsf{N}_k)$,
- formula $\varphi \in \Phi^{\sigma}_{\mathcal{L}}$,
- FO-query $\mathcal{I}$: STRUC$[\sigma' \cup \tau'] \to$ STRUC$[\sigma]$, and
- FO-formula ψ over $\sigma' \cup \tau'$,

there is a formula $\varphi' \in \Phi^{\sigma' \cup \tau'}_{\mathcal{L}}$ such that for all $\mathbf{A} \in$ STRUC$[\sigma']$ and non-uniform families of interpretations $\mathcal{I} = (\mathbf{I}_n)_{n \in \mathbb{N}}$ of the symbols in τ' by relations from $\mathfrak{R}$, we have

$$\mathbf{A} \cup \mathbf{I}_{|\mathrm{dom}(\mathbf{A})|}(\tau') \models_{\mathcal{L}} \varphi' \iff \mathcal{I}(\mathbf{A} \cup \mathbf{I}_{|\mathrm{dom}(\mathbf{A})|}(\tau')) \models_{\mathcal{L}} \varphi,$$

if $\mathbf{A} \cup \mathbf{I}_{|\mathrm{dom}(\mathbf{A})|}(\tau') \models_{\mathrm{FO}} \psi$, and $\mathbf{A} \cup \mathbf{I}_{|\mathrm{dom}(\mathbf{A})|}(\tau') \not\models_{\mathcal{L}} \varphi'$ otherwise.

We are now in a position to prove the main theorem of this section, i.e., the fact that $\mathcal{L}[\mathrm{Arb}]$ coincides with $\mathcal{L}[\mathrm{BIT}]/\mathrm{poly}$ for a wide range of logics $\mathcal{L}$. The premise will be that $\mathcal{L}[\mathrm{Arb}]$ is closed under applying partial FO[Arb]-queries. The proof for $\mathcal{L}[\mathrm{BIT}]/\mathrm{poly} \subseteq \mathcal{L}[\mathrm{Arb}]$ will be similar to the proof of Corollary 3.6, as we apply the composition of several FO-queries to a formula, using the closure property to argue that a given language from $\mathcal{L}[\mathrm{BIT}]/\mathrm{poly}$ is also in $\mathcal{L}[\mathrm{Arb}]$.

Theorem 7.20. *Let* $\mathcal{L} = (\Phi_{\mathcal{L}}, \mathfrak{A}_{\mathcal{L}}, \models_{\mathcal{L}})$ *be a logic over* FO*-structures such that* $\mathcal{L}[\mathrm{Arb}]$ *is closed under applying partial* FO[Arb]*-queries.*

Then $\mathcal{L}[\mathrm{Arb}] = \mathcal{L}[\mathrm{BIT}]/\mathrm{poly}$.

Proof. **$\mathcal{L}[\mathrm{Arb}] \subseteq \mathcal{L}[\mathrm{BIT}]/\mathrm{poly}$:** Let $L \in \mathcal{L}[\mathrm{Arb}]$, i.e., there are vocabularies σ and τ, a formula $\varphi \in \Phi_{\mathcal{L}}^{\sigma\cup\tau}$, and a non-uniform family $\mathcal{I} = (\mathbf{I}_n)_{n\in\mathbb{N}}$ of interpretations of the symbols in τ such that for all $\mathbf{A} \in \mathrm{STRUC}[\sigma]$ we have

$$\mathrm{enc}_\sigma(\mathbf{A}) \in L \iff \mathbf{A} \cup \mathbf{I}_{|\mathrm{dom}(\mathbf{A})|}(\tau) \models_{\mathcal{L}} \varphi,$$

and $x \notin L$ if x is not the encoding of a σ-structure. The formula φ also defines a language L' as follows: For all $x \in \{0,1\}^+$,

$$x \in L' \iff \mathrm{enc}^{-1}_{\sigma\cup\tau}(x) \models_{\mathcal{L}} \varphi,$$

if x is the encoding of a $\sigma\cup\tau$-structure and $x \notin L'$ otherwise. By definition, $L' \in \mathcal{L}[\mathrm{BIT}]$ via φ, using the empty vocabulary for numerical predicate symbols.

Now by definition of the encoding function $\mathrm{enc}_{\sigma\cup\tau}$, for any cardinality n of domains and any interpretation $\mathbf{B}$ of the symbols in τ as relations over $\{0,\dots,n-1\}$, there is a suffix $\mathrm{enc}'(\mathbf{B}) \in \{0,1\}^+$ such that for any $\mathbf{A} \in \mathrm{STRUC}[\sigma]$ with $|\mathrm{dom}(\mathbf{A})| = n$ we have

$$\mathrm{enc}^{-1}_{\sigma\cup\tau}(\mathrm{enc}_\sigma(\mathbf{A}) \circ \mathrm{enc}'(\mathbf{B})) = \mathbf{A} \cup \mathbf{B}.$$

Furthermore, there is a polynomial p such that $\mathrm{enc}'(\mathbf{B})$ is bounded by a polynomial in the cardinality n of the domain. This means that we can define a polynomially length-bounded advice function $A\colon \mathbb{N} \to \{0,1\}^*$ by

$$A(n) \coloneqq \mathrm{enc}'(\mathbf{I}_n(\tau)),$$

if n is the length of encodings of σ-structures and $A(n) \coloneqq 1^{\mathrm{error}(n)}$ otherwise, where $\mathrm{error}(n)$ simply is a number such that $n + \mathrm{error}(n)$ is not the length of any encoding of a $\sigma\cup\tau$-structure. By definition of A, for all $x \in \{0,1\}^+$ that are encodings of σ-structures we have

$$\mathrm{enc}^{-1}_{\sigma\cup\tau}(x \circ A(|x|)) = \mathrm{enc}^{-1}_\sigma(x) \cup \mathbf{I}_{|\mathrm{dom}(\mathrm{enc}^{-1}_\sigma(x))|}(\tau).$$

It now holds that for all $x \in \{0,1\}^+$,

$$\begin{aligned} x \in L &\iff \mathrm{enc}^{-1}_\sigma(x) \cup \mathbf{I}_{|\mathrm{dom}(\mathrm{enc}^{-1}_\sigma(x))|}(\tau) \models_{\mathcal{L}} \varphi \\ &\iff \mathrm{enc}^{-1}_{\sigma\cup\tau}(x \circ A(|x|)) \models_{\mathcal{L}} \varphi \\ &\iff x \circ A(|x|) \in L', \end{aligned}$$

if x is the encoding of a σ-structure. On the other hand, if x is not the encoding of a σ-structure, then $x \notin L$ and $x \circ A(|x|) \notin L'$ by definition of A. Hence, $L \in \mathcal{L}[\mathrm{BIT}]/\mathrm{poly}$.

$\mathcal{L}[\mathrm{BIT}]/\mathrm{poly} \subseteq \mathcal{L}[\mathrm{Arb}]$: Let $L \in \mathcal{L}[\mathrm{BIT}]/\mathrm{poly}$, i.e., there is a language $L' \in \mathcal{L}[\mathrm{BIT}]$ and a polynomially length-bounded advice function $A\colon \mathbb{N} \to \{0,1\}^*$ such that for all $x \in \{0,1\}^+$,

$$x \in L \iff x \circ A(|x|) \in L'.$$

By definition of $\mathcal{L}[\mathrm{BIT}]$, there is a vocabulary σ and a formula $\varphi \in \Phi_{\mathcal{L}}^{\sigma\cup(\mathrm{BIT})}$ such that for all $\mathbf{A} \in \mathrm{STRUC}[\sigma]$ we have

$$\mathrm{enc}_\sigma(\mathbf{A}) \in L' \iff \mathbf{A} \cup (\mathrm{BIT}_{|\mathrm{dom}(\mathbf{A})|-1}) \models_{\mathcal{L}} \varphi,$$

and $w \notin L'$ if w is not the encoding of a σ-structure. Here, by BIT_n we denote the restriction of the BIT-predicate to numbers $\leq n$. To be precise, the vocabulary of built-in predicate symbols could also contain several symbols that are all interpreted as BIT_n. The proof easily generalizes to this case.

As mentioned in Remark 2.28, there is an FO[BIT]-query $\mathrm{bin}_\sigma^{-1}\colon \mathrm{STRUC}[\tau_{\mathrm{string}} \cup (\mathrm{BIT})] \to \mathrm{STRUC}[\sigma \cup (\mathrm{BIT})]$ such that for all $w \in \{0,1\}^+$ that are encodings of σ-structures we have

$$\mathrm{bin}_\sigma^{-1}(\mathbf{A}_w \cup (\mathrm{BIT}_{|w|})) = \mathrm{enc}_\sigma^{-1}(w) \cup (\mathrm{BIT}_{|\mathrm{dom}(\mathrm{enc}_\sigma^{-1}(w))|}).$$

Let p be a polynomial such that $|A(n)| \leq p(n)$ for all $n \in \mathbb{N}$. From this we can define a vocabulary $\sigma_A = (\mathsf{A}_1, \ldots, \mathsf{A}_k, \mathsf{A}_{\mathrm{len},1}, \ldots, \mathsf{A}_{\mathrm{len},k})$ such that for all cardinalities of domains $n \in \mathbb{N}$, $p(n) \leq \sum_{i=1}^k n^{\mathrm{ar}(\mathsf{A}_i)}$. This means that there is a non-uniform family of interpretations $\mathcal{I} = (\mathbf{I}_n)_{n\in\mathbb{N}}$ such that $\mathbf{I}_n$ interprets the symbols $\mathsf{A}_1, \ldots, \mathsf{A}_k$ such that their encoding in binary is $A(n)$ and the symbols $\mathsf{A}_{\mathrm{len},1}, \ldots, \mathsf{A}_{\mathrm{len},k}$ such that their encoding in binary is $1^{|A(n)|}0^{\sum_{i=1}^k n^{\mathrm{ar}(\mathsf{A}_i)}-|A(n)|}$, i.e., a unary encoding of $|A(n)|$. For this, assume the encoding function enc′ used in the proof of $\mathcal{L}[\mathrm{Arb}] \subseteq \mathcal{L}[\mathrm{BIT}]/\mathrm{poly}$ above, defined in accordance with the usual encoding function for first-order structures. Now it is easy to see that there is an FO-query $\mathcal{J}\colon \mathrm{STRUC}[\tau_{\mathrm{string}} \cup \sigma_A \cup (\mathrm{BIT})] \to \mathrm{STRUC}[\tau_{\mathrm{string}} \cup (\mathrm{BIT})]$ such that for all $x \in \{0,1\}^+$,

$$\mathcal{J}(\mathbf{A}_x \cup \mathbf{I}_{|x|}(\sigma_A) \cup \mathrm{BIT}_{|x|}) = \mathbf{A}_{x\circ A(|x|)} \cup (\mathrm{BIT}_{|x\circ A(|x|)|-1}).$$

Now consider the composition $\mathcal{J}' := \mathrm{bin}_\sigma^{-1} \circ \mathcal{J}$, which is again an FO-query. We have for all $x \in \{0,1\}^+$:

$$\mathcal{J}'(\mathbf{A}_x \cup \mathbf{I}_{|x|}(\sigma_A) \cup \mathrm{BIT}_{|x|}) = \mathrm{enc}_\sigma^{-1}(x \circ A(|x|)) \cup (\mathrm{BIT}_{|\mathrm{dom}(\mathrm{enc}_\sigma^{-1}(x\circ A(|x|)))|-1}),$$

if $x \circ A(|x|)$ is the encoding of a σ-structure.

The formula φ shows that $L' \in \mathcal{L}[\mathrm{BIT}]$ and since $\{\mathrm{BIT}\} \subseteq \mathrm{Arb}$ it also shows $L' \in \mathcal{L}[\mathrm{Arb}]$. Also, the property that $x \circ A(|x|)$ is the encoding of a σ-structure is definable from the τ_{string}-structure $\mathbf{A}_x$ by an FO[Arb]-formula. Therefore, we can now use the closure of $\mathcal{L}[\mathrm{Arb}]$ under partial FO[Arb]-queries to obtain a formula φ' such that for all $x \in \{0,1\}^+$:

$$\begin{aligned}\mathbf{A}_x \cup \mathbf{I}_{|x|}(\sigma_A) \cup (\mathrm{BIT}_{|x|}) \models_{\mathcal{L}} \varphi' &\iff \mathcal{J}'(\mathbf{A}_x \cup \mathbf{I}_{|x|}(\sigma_A) \cup (\mathrm{BIT}_{|x|})) \models_{\mathcal{L}} \varphi \\ &\iff x \circ A(|x|) \in L' \\ &\iff x \in L,\end{aligned}$$

if $x \circ A(|x|)$ is the encoding of a σ-structure, and $x \notin L$ and $\mathbf{A}_x \cup \mathbf{I}_{|x|}(\sigma_A) \cup (\mathrm{BIT}_{|x|}) \not\models_{\mathcal{L}} \varphi'$ otherwise. □

The closure property required by the above result is obviously somewhat strong, as it requires that the logic can express arbitrary FO-definable properties—more precisely, its non-uniform version can express arbitrary FO[Arb]-definable properties. The reason we chose this closure property is that we are mainly interested in transferring characterizations of classes from circuit complexity. In this setting, weaker classes than $\mathrm{AC}^0 = \mathrm{FO}$ are rarely considered. Furthermore, most known logics used in model-theoretic characterizations of complexity classes have this closure property, see Subsection 7.2.3 for some examples. One could attempt to generalize the result by requiring a weaker closure property. For this it might be necessary to use a different pairing function to define classes of the form $\mathfrak{C}/\mathrm{poly}$ similar as in the case of weaker uniformities, discussed below Theorem 7.6.

7.2.2 Counting Classes

Next, we will transfer the results above from the decision setting to the counting setting. For this, we first introduce a general framework for logic-based counting functions similar to the framework for logics and logic-based decision classes used above. We will then see that by adequately transferring the required closure property to the counting setting, we obtain a wide range of counting classes for which the different kinds of non-uniformity coincide.

In the spirit of the logic QSO, introduced by Arenas et al. [AMR20], we call logics in our framework for logic-based counting functions *quantitative logics*.

Definition 7.21. A *quantitative logic* $\mathcal{L}$ is a triple $\mathcal{L} = (\Phi_{\mathcal{L}}, \mathfrak{A}_{\mathcal{L}}, \#\mathcal{M}_{\mathcal{L}})$, where $\Phi_{\mathcal{L}}$ and $\mathfrak{A}_{\mathcal{L}}$ are classes and $\#\mathcal{M}_{\mathcal{L}} \colon \mathfrak{A}_{\mathcal{L}} \times \Phi_{\mathcal{L}} \to \mathbb{N}$ is a function, called the *counting function of* $\mathcal{L}$. We call elements of $\Phi_{\mathcal{L}}$ *formulae of* $\mathcal{L}$ and elements of $\mathfrak{A}$ *valuations of* $\mathcal{L}$.

To illustrate this definition, we go over how #Win-FO fits into this framework next. As this is very similar to how FO fits into the corresponding framework for (non-quantitative) logics, we will do this very briefly.

Example 7.22. The quantitative version of first-order logic can be defined in the above framework by setting $\#\text{Win-FO} := (\Phi_{\#\text{Win-FO}}, \mathfrak{A}_{\#\text{Win-FO}}, \#\mathcal{M}_{\#\text{Win-FO}})$, where $\Phi_{\#\text{Win-FO}} := \Phi_{\mathrm{FO}}$, $\mathfrak{A}_{\#\text{Win-FO}} := \mathfrak{A}_{\mathrm{FO}}$ and $\#\mathcal{M}_{\#\text{Win-FO}}$ is the generalization of #Win used earlier to arbitrary vocabularies (but without built-in numerical predicates), i.e., for any formula $\varphi \in \Phi_{\mathrm{FO}}$ and structure $\mathbf{A} \in \mathfrak{A}_{\mathrm{FO}}$, we define

$$\#\mathcal{M}_{\#\text{Win-FO}}(\mathbf{A}, \varphi) := \#\mathrm{Win}(\mathbf{A}, \varphi),$$

if there is a vocabulary σ such that $\varphi \in \Phi^{\sigma}_{\mathrm{FO}}$ and $\mathbf{A} \in \mathfrak{A}^{\sigma}_{\mathrm{FO}}$, and $\#\mathcal{M}_{\#\text{Win-FO}}(\mathbf{A}, \varphi) := 0$ otherwise. The subclasses $\mathfrak{A}^{\sigma}_{\#\text{Win-FO}}$ and $\Phi^{\sigma}_{\#\text{Win-FO}}$ for vocabularies σ are defined in the same way as for FO as well, i.e., $\mathfrak{A}^{\sigma}_{\#\text{Win-FO}} := \mathfrak{A}^{\sigma}_{\mathrm{FO}}$ and $\Phi^{\sigma}_{\#\text{Win-FO}} := \Phi^{\sigma}_{\mathrm{FO}}$.

We will now transfer the notions defined for (non-quantitative) logics in the previous section to the setting of quantitative logics. As the new definitions are completely analogous to their counterparts in the decision setting, see the previous section for explanations and motivation.

Definition 7.23. A quantitative logic $\mathcal{L} = (\Phi_\mathcal{L}, \mathfrak{A}_\mathcal{L}, \#\mathcal{M}_\mathcal{L})$ is called a *quantitative logic over* FO*-structures*, if

$$\mathfrak{A}_\mathcal{L} = \mathfrak{A}_{\mathrm{FO}} \quad \text{and} \quad \Phi_\mathcal{L} = \bigsqcup_{\substack{\text{FO-voc-}\\ \text{abulary } \sigma}} \Phi^\sigma_\mathcal{L}$$

for subclasses $\Phi^\sigma_\mathcal{L}$ of $\Phi_\mathcal{L}$ such that for any FO-vocabulary σ, $\varphi \in \Phi^\sigma_\mathcal{L}$, and $\mathbf{A} \in \mathfrak{A}_{\mathrm{FO}}$, we have

$$\#\mathcal{M}_\mathcal{L}(\mathbf{A}, \varphi) > 0 \Rightarrow \mathbf{A} \in \mathfrak{A}^\sigma_{\mathrm{FO}} = \mathrm{STRUC}[\sigma].$$

Definition 7.24. Let $\mathcal{L} = (\Phi_\mathcal{L}, \mathfrak{A}_\mathcal{L}, \#\mathcal{M}_\mathcal{L})$ be a quantitative logic over FO-structures and $\mathfrak{R}$ be a set of relations over $\mathbb{N}$. Then $\mathcal{L}[\mathfrak{R}]$ is the class of counting functions f for which there are vocabularies σ and $\tau = (\mathsf{N}_1, \ldots, \mathsf{N}_k)$, a formula $\varphi \in \Phi^{\sigma\cup\tau}_\mathcal{L}$, and a non-uniform family $\mathcal{I} = (\mathbf{I}_n)_{n\in\mathbb{N}}$ of interpretations of symbols in τ by relations from $\mathfrak{R}$ such that for all $\mathbf{A} \in \mathrm{STRUC}[\sigma]$, we have

$$f(\mathrm{enc}_\sigma(\mathbf{A})) = \#\mathcal{M}_\mathcal{L}(\mathbf{A} \cup \mathbf{I}_{|\mathrm{dom}(\mathbf{A})|}(\tau), \varphi),$$

and $f(x) = 0$ if x is not the encoding of a σ-structure.

Definition 7.25. Let $\mathcal{L} = (\Phi_\mathcal{L}, \mathfrak{A}_\mathcal{L}, \#\mathcal{M}_\mathcal{L})$ be a quantitative logic over FO-structures and $\mathfrak{R}$ be a set of relations over $\mathbb{N}$. We say that $\mathcal{L}[\mathfrak{R}]$ *is closed under applying partial* FO$[\mathfrak{R}]$*-queries*, if for any

- vocabularies σ, σ', τ' with $\tau' = (\mathsf{N}_1, \ldots, \mathsf{N}_k)$,
- formula $\varphi \in \Phi^\sigma_\mathcal{L}$,
- FO-query $\mathcal{I}\colon \mathrm{STRUC}[\sigma' \cup \tau'] \to \mathrm{STRUC}[\sigma]$, and
- FO-formula ψ over $\sigma' \cup \tau'$,

there is a formula $\varphi' \in \Phi^{\sigma'\cup\tau'}_\mathcal{L}$ such that for all $\mathbf{A} \in \mathrm{STRUC}[\sigma']$ and non-uniform families of interpretations $\mathcal{I} = (\mathbf{I}_n)_{n\in\mathbb{N}}$ of the symbols in τ' by relations from $\mathfrak{R}$, we have

$$\#\mathcal{M}_\mathcal{L}(\mathbf{A} \cup \mathbf{I}_{|\mathrm{dom}(\mathbf{A})|}(\tau'), \varphi') \Longleftrightarrow \#\mathcal{M}_\mathcal{L}(\mathcal{I}(\mathbf{A} \cup \mathbf{I}_{|\mathrm{dom}(\mathbf{A})|}(\tau')), \varphi),$$

if $\mathbf{A} \cup \mathbf{I}_{|\mathrm{dom}(\mathbf{A})|}(\tau') \models_{\mathrm{FO}} \psi$, and $\#\mathcal{M}_\mathcal{L}(\mathbf{A} \cup \mathbf{I}_{|\mathrm{dom}(\mathbf{A})|}(\tau'), \varphi') = 0$ otherwise.

We can now state the counting version of the main theorem of this section, i.e., also for a wide range of quantitative logics, the different kinds of non-uniformity coincide.

Theorem 7.26. *Let* $\mathcal{L} = (\Phi_\mathcal{L}, \mathfrak{A}_\mathcal{L}, \#\mathcal{M}_\mathcal{L})$ *be a quantitative logic over* FO*-structures such that* $\mathcal{L}[\mathrm{Arb}]$ *is closed under applying partial* FO[Arb]*-queries.*

Then $\mathcal{L}[\mathrm{Arb}] = \mathcal{L}[\mathrm{BIT}]/\mathrm{poly}$.

Proof. It is easy to see that the assumed closure property allows us to make the same arguments with regard to $\#\mathcal{M}_\mathcal{L}$ that were made with regard to $\models_\mathcal{L}$ for the (non-quantitative) logic $\mathcal{L}$ in the proof of the decision version. Hence, the proof is completely analogous to the proof of Theorem 7.20. □

7.2.3 Examples

The results with regard to circuit complexity classes in Section 7.1 are directly applicable to many important classes in this setting, and in particular to all classes in the NC-, AC-, SAC-, and TC-hierarchies apart from the obvious exception NC^0. In contrast, due to the unusual framework we use in this section, it is not as obvious that our results with regard to model-theoretic complexity classes are widely applicable. Hence, we will use this Subsection to argue that for many important model-theoretic classes that are used in descriptive complexity, our results are directly applicable. As this is not the focus of our research and the proofs are relatively straightforward, we will focus on briefly stating how the different classes fit into the framework we use and only present proof sketches for the required closure property. We will cover the class FO, the classes #Win-FO and FOCW from Chapters 3 and 4, the classes ESO and #FO, as well as extensions of FO and #Win-FO obtained by adding the different types of GPR-operators introduced in Chapter 6.

FO, #Win-FO and FOCW

In Examples 7.16 and 7.22 we have already seen how the classes FO and #Win-FO can be defined in our framework for logics and quantitative logics, respectively. We will now see that both classes have the required closure property. Subsequently, we will briefly cover the class FOCW introduced in Section 4.2.

The fact that #Win-FO[Arb] is closed under applying partial FO[Arb]-queries is a generalization of the non-uniform version of Lemma 3.4 with a similar proof. We briefly sketch the proof next.

Lemma 7.27. #Win-FO[Arb] *is closed under applying partial* FO[Arb]*-queries.*

Proof sketch. Let φ be a σ-formula for some vocabulary σ. Let $\mathcal{I}\colon \mathrm{STRUC}[\sigma' \cup \tau'] \to \mathrm{STRUC}[\sigma]$ be an FO-query and ψ an FO-sentence over $\sigma' \cup \tau'$ for additional vocabularies σ' and τ'.

By Lemma 3.3, we can assume without loss of generality that the number of winning strategies of the verifier on formulae from $\mathcal{I}$ and on the formula ψ in arbitrary structures and for arbitrary non-uniform families of interpretations of the symbols in τ' by relations from $\mathfrak{R}$ (as well as for arbitrary assignments to the free variables in case of formulae from $\mathcal{I}$) is at most 1. Now, we plug the formulae from $\mathcal{I}$ into φ, replacing variables by tuples of adequate arity and relativizing quantifiers according to the formula φ_0 in $\mathcal{I}$, yielding an FO[Arb]-formula φ' over σ' with built-in predicate symbols from τ'. Finally, the formula $\varphi' \wedge \psi$ has the desired properties, i.e., for all $\mathbf{A} \in \mathrm{STRUC}[\sigma']$ and all non-uniform families of interpretations $\mathcal{I} = (\mathbf{I}_n)_{n\in\mathbb{N}}$ of the symbols in τ' by relations from $\mathfrak{R}$, we have

$$\#\mathrm{Win}(\mathbf{A} \cup \mathbf{I}_{|\mathrm{dom}(\mathbf{A})|}(\tau'), \varphi' \wedge \psi) = \#\mathrm{Win}(\mathcal{I}(\mathbf{A}, \mathbf{I}_{|\mathrm{dom}(\mathbf{A})|}(\tau')), \varphi),$$

if $\mathbf{A} \cup \mathbf{I}_{|\mathrm{dom}(\mathbf{A})|}(\tau') \models \psi$, and $\#\mathrm{Win}(\mathbf{A} \cup \mathbf{I}_{|\mathrm{dom}(\mathbf{A})|}(\tau'), \varphi' \wedge \psi) = 0$ otherwise. □

The fact that the decision class FO[Arb] is closed under applying partial FO[Arb]-queries now follows immediately from the previous lemma together with the fact that FO and #Win-FO are defined with the same class of formulae and for all $\varphi \in \Phi_{\mathrm{FO}}$ and $\mathbf{A} \in \mathfrak{A}_{\mathrm{FO}}$ it holds that

$$\mathbf{A} \models_{\mathrm{FO}} \varphi \iff \#\mathcal{M}_{\#\text{Win-FO}}(\mathbf{A}, \varphi) \geq 1.$$

We will now cover the case of FOCW. First, we need to describe how FOCW fits into our framework. For this, define $\mathrm{FOCW} := (\Phi_{\mathrm{FOCW}}, \mathfrak{A}_{\mathrm{FO}}, \models_{\mathrm{FOCW}})$, where Φ_{FOCW} is the set of FOCW-formulae and $\models_{\mathrm{FOCW}}$ is defined as follows. For any $\mathbf{A} \in \mathfrak{A}_{\mathrm{FO}}$ and any $\varphi \in \Phi_{\mathrm{FOCW}}$, we have

$$\mathbf{A} \models_{\mathrm{FOCW}} \varphi \iff \mathbf{A}' \models \varphi,$$

where $\mathbf{A}'$ is the two-sorted structure obtained from $\mathbf{A}$ by adding a second sort of cardinality $|\mathrm{dom}(\mathbf{A})|$ as well as the numerical predicates + and × on the second sort. The class Φ_{FOCW} can naturally be split into subclasses $\Phi^{\sigma}_{\mathrm{FOCW}}$ for vocabularies σ. We now show that using this definition, FOCW[Arb] is closed under applying partial FO[Arb]-queries.

Lemma 7.28. FOCW[Arb] *is closed under applying partial* FO[Arb]*-queries.*

Proof sketch. Let σ be a vocabularies and $\varphi \in \Phi^{\sigma}_{\mathrm{FOCW}}$. In the non-uniform setting, we have access to the numerical predicates + and × on the first sort. This allows us to define the m-th element in the first sort from the m-th element in the second sort and vice versa for all $m \in \mathbb{N}$. Hence, we can replace all occurrences of second-sort variables i by first-sort variables x_i and occurrences of relations on the second sort by adequate built-in predicates symbols apart from predicates $\#_\xi(\bar{i})$. Each such occurrence can be replaced by a subformula $\varphi_{\#_\xi(\bar{i})} := \varphi_{\overline{x_{\bar{i}}}=\bar{i}} \wedge \#_\xi(\bar{i})$, where $\varphi_{\overline{x_{\bar{i}}}=\bar{i}}$ expresses that $\overline{x_{\bar{i}}}$ and $\bar{i}$ represent the same natural number over the different sorts.

We can now use the closure of FO[Arb] under application of partial FO[Arb]-queries to modify the formula according to any FO[Arb]-query $\mathcal{I}\colon \mathrm{STRUC}[\sigma' \cup \tau'] \to \mathrm{STRUC}[\sigma]$ except for subformulae $\varphi_{\overline{x_{\bar{i}}}=\bar{i}} \wedge \varphi_{\#_\xi(\bar{i})}$, as these are the only subformulae containing second-sort variables. For each such formula, do the following: First, modify the subformula $\varphi_{\overline{x_{\bar{i}}}=\bar{i}}$ according to $\mathcal{I}$. Also extend the tuple size of $\bar{i}$ accordingly without otherwise changing occurrences of second-sort relation symbols. Possibly, numerical predicates for tuples of higher arity over the second sort have to be expressed in terms of predicates on individual elements for this. This way, the formula expresses for any σ'-structure $\mathbf{A}$ and any non-uniform family of interpretations $\mathcal{I} = (\mathbf{I}_n)_{n \in \mathbb{N}}$ the required property of $\overline{x_{\bar{i}}}$ and $\bar{i}$ with respect to $\mathcal{I}(\mathbf{A} \cup \mathbf{I}_{|\mathrm{dom}(\mathbf{A})|})$. Second, modify the formula ξ according to $\mathcal{I}$ as well, which is possible as #Win-FO is closed under partial FO[Arb]-queries.

Let φ' be the formula obtained in the previous step, which is a formula over $\sigma' \cup \tau'$. Now, for any FO-formula ψ over $\sigma' \cup \tau'$, the formula $\varphi' \wedge \psi$ is still a formula in $\Phi^{\sigma' \cup \tau'}_{\mathrm{FOCW}}$ and it has the desired properties, finishing the proof. □

ESO and #FO

We will next cover the case of the logics ESO and #FO. Again, we will begin with the counting version as this allows us to obtain the corresponding result for the decision version as a direct corollary.

We begin by defining #FO in the sense of quantitative logics introduced above. For this, let $\#\mathrm{FO} := (\Phi_{\#\mathrm{FO}}, \mathfrak{A}_{\mathrm{FO}}, \#\mathcal{M}_{\#\mathrm{FO}})$, where $\Phi_{\#\mathrm{FO}}$ is the class of FO-formulae over arbitrary vocabularies with additional relation variables that are marked as being free variables. The free relation variables can, similar to the underlying vocabulary, be described by the tuple of arities of the relations. Then for any vocabulary σ, $\Phi^{\sigma}_{\#\mathrm{FO}}$ contains FO-formulae over vocabulary σ with additional free relation variables and $\#\mathcal{M}_{\#\mathrm{FO}}$ is defined as follows: For any $\mathbf{A} \in \mathfrak{A}_{\#\mathrm{FO}}$ and any $\varphi \in \Phi_{\#\mathrm{FO}}$ containing free relation variables $\{\mathsf{R}_1, \dots, \mathsf{R}_k\}$, define

$$\#\mathcal{M}_{\#\mathrm{FO}}(\mathbf{A}, \varphi) := |\{(R_1, \dots, R_k) \mid \mathbf{A} \models \varphi(R_1, \dots, R_k)\}|.$$

For simplicity, we do not allow free first-order variables here. The definition can be modified to allow such variables and the proof below can be generalized accordingly. We now show that #FO[Arb] has the required closure property.

Lemma 7.29. #FO[Arb] *is closed under applying partial* FO[Arb]*-queries.*

Proof sketch. Let $\varphi(\mathsf{R}_1, \dots, \mathsf{R}_k)$ be an FO-formula over some vocabulary σ with free relation symbols $\mathsf{R}_1, \dots, \mathsf{R}_k$. Without loss of generality, we can assume that for any occurrence $\mathsf{R}_i(t_1, \dots, t_{\mathrm{ar}(\mathsf{R}_i)})$, the terms $t_1, \dots, t_{\mathrm{ar}(\mathsf{R}_i)}$ are actually variables. Let $\mathcal{I}\colon \mathrm{STRUC}[\sigma' \cup \tau'] \to \mathrm{STRUC}[\sigma]$ be an FO-query and $\varphi_0(x_1, \dots, x_\ell)$ the formula defining the new universe in $\mathcal{I}$.

We can first proceed in the same way as for showing that FO[Arb] is closed under applying FO[Arb]-queries. More precisely, we can replace occurrences of symbols from σ by the corresponding formulae from $\mathcal{I}$, replace individual variables by tuples, and relativize quantifiers according to φ_0. Let φ' be the formula obtained after this step. What remains is to handle occurrences of the free relation symbols $\mathsf{R}_1, \dots, \mathsf{R}_k$. We replace each R_i by a new symbol R'_i and define $\mathrm{ar}(\mathsf{R}'_i) := \ell \cdot \mathrm{ar}(\mathsf{R}_i)$. Now, we can replace any occurrence $\mathsf{R}_i(z_1, \dots, z_{\mathrm{ar}(\mathsf{R}_i)})$ in φ' by $\mathsf{R}'_i(z_1^1, \dots, z_1^\ell, \dots, z_{\mathrm{ar}(\mathsf{R}_i)}^1, \dots, z_{\mathrm{ar}(\mathsf{R}_i)}^\ell)$, that is, replace variables by tuples of adequate arities in these atoms. Let φ'' be the formula obtained after this modification. Next, we need to ensure that by any satisfying assignment to the free relation variables, each free relation variable R'_i is actually mapped to a relation over tuples satisfying φ_0. For this, let

$$\varphi''' := \varphi'' \wedge \bigwedge_{i=1}^{k} \forall \overline{y} = y_1^1 \dots y_k^{\ell \cdot \mathrm{ar}(\mathsf{R}_k)}\, \mathsf{R}_i(\overline{y}) \to \left(\bigwedge_{j=1}^{\mathrm{ar}(\mathsf{R}_j)} \varphi_0\left(y_j^1, \dots, y_j^{\ell \cdot \mathrm{ar}(\mathsf{R}_1)}\right) \right).$$

We now have for all $\mathbf{A} \in \mathrm{STRUC}[\sigma']$ and all non-uniform families $\mathcal{I} = (\mathbf{I}_n)_{n \in \mathbb{N}}$ of interpretations of symbols in τ' that $|\{(R'_1, \dots, R'_k) \mid \mathbf{A} \cup \mathbf{I}_{|\mathrm{dom}(\mathbf{A})|}(\tau') \models \varphi'''(R'_1, \dots, R'_k)\}|$ equals $|\{(R_1, \dots, R_k) \mid \mathcal{I}(\mathbf{A} \cup \mathbf{I}_{|\mathrm{dom}(\mathbf{A})|}(\tau')) \models \varphi(R_1, \dots, R_k)\}|$.

Finally, partial FO[Arb]-queries can be handled by simply using the conjunction of φ''' with any additionally given FO-formula ψ over $\sigma' \cup \tau'$: The formula $\varphi''' \wedge \psi$ yields

the same number of satisfying assignments of the free relation variables for structures satisfying ψ, but yields no satisfying assignments for structures that do not satisfy ψ. □

The case of ESO can be handled completely analogously. First, ESO can be defined in the framework as $\mathrm{ESO} := (\Phi_{\mathrm{ESO}}, \mathfrak{A}_{\mathrm{FO}}, \models_{\mathrm{ESO}})$, where $\Phi_{\mathrm{ESO}} := \Phi_{\#\mathrm{FO}}$ and for any $\mathbf{A} \in \mathfrak{A}_{\mathrm{FO}}$ and $\varphi \in \Phi_{\mathrm{ESO}}$, we define

$$\mathbf{A} \models_{\mathrm{ESO}} \varphi \iff \#\mathcal{M}_{\#\mathrm{FO}}(\mathbf{A}, \varphi) \geq 1.$$

The fact that ESO[Arb] is closed under applying partial FO[Arb]-queries then directly follows from the fact that #FO has this closure property.

FO + GPR

Here, we want to show that the logics introduced in Chapter 6 have the required closure property as well, so the different kinds of non-uniformity also coincide for these logics. We show this for the logic #Win-FO + GPR and argue that it immediately transfers to FO + GPR, but the result can be transferred to the other logics, i.e., the versions based on bounded and semi-unbounded GPR. The definition of #Win-FO + GPR in our framework for quantitative logics is straightforward: Define $\#\text{Win-FO+GPR} := (\Phi_{\#\text{Win-FO+GPR}}, \mathfrak{A}_{\mathrm{FO}}, \#\mathcal{M}_{\#\text{Win-FO+GPR}})$, where $\Phi_{\#\text{Win-FO+GPR}}$ is the class of #Win-FO + GPR formulae over arbitrary vocabularies as introduced in Chapter 6 and $\#\mathcal{M}_{\#\text{Win-FO+GPR}}(\mathbf{A}, \varphi)$ is the number of winning strategies of the verifier in the model-checking game for $\mathbf{A} \models \varphi$, again as introduced in Chapter 6. Note that strictly following the notation of this chapter, the class #Win-FO[Arb] + GPR in the following lemma is called (#Win-FO + GPR)[Arb].

Lemma 7.30. #Win-FO[Arb] + GPR *is closed under applying partial* FO[Arb]*-queries.*

Proof sketch. This can be proven similarly to proving that #Win-FO[Arb] has the desired closure property. Let

$$[\mathsf{P}(\overline{y}) \equiv \theta(\overline{y}, \mathsf{P}^+)]\, \varphi(\mathsf{P}^+) \in \Phi_{\#\text{Win-FO+GPR}}$$

be a formula over some vocabulary σ. For this proof sketch we restrict ourselves to FO[Arb] + GPR-sentences of the above form, i.e., sentences containing only a single GPR-operator in the front and using a predicate symbol P without additional arguments beside the recursion variables. It is straightforward to generalize the proof to arbitrary #Win-FO[Arb] + GPR-formulae. Let $\mathcal{I}\colon \mathrm{STRUC}[\sigma' \cup \tau'] \to \mathrm{STRUC}[\sigma]$ be an FO-query for vocabularies σ', τ'. Without loss of generality we can assume that on all formulae in $\mathcal{I}$ the verifier has at most one winning strategy in the model-checking game for any input structure and any assignment to the free variables.

As φ is an #Win-FO-formula except for occurrences of the free relation variable P, we can apply $\mathcal{I}$ to this formula as usual by replacing variables by adequate tuples, relativizing quantifiers and replacing occurrences of symbols from σ by the corresponding formulae from $\mathcal{I}$. This is possible due to #Win-FO[Arb] being closed under partial

FO[Arb]-queries. Let φ' be the resulting formula after also replacing P by a new symbol P' of adequate arity. Note that P' only occurs positively in φ' as no additional negations are introduced around occurrences of P'. Now what remains is to modify the GPR-operator defining P in such a way that for atoms involving P in φ' we obtain the same number of winning strategies as for the corresponding atom in φ.

First, replace P by P' in the GPR-operator. Now, note that the formula θ that describes the recursive definition of the interpretation of P is an #Win-FO-formula as well, apart from guarded quantifiers $Q\overline{z}.(\overline{z} < \overline{y}/2 \wedge \xi(\overline{y}, \overline{z}))$. We can therefore apply $\mathcal{I}$ to θ in the same way as to φ apart from these quantifiers. Now for any such quantifier, we can replace $\overline{z}$ by a tuple of adequate arity and apply $\mathcal{I}$ to the FO-formula $\xi(\overline{y}, \overline{z})$. Regarding $\overline{z} < \overline{y}/2$, recall that this is a shorthand for the formula $\exists \overline{u}\,(\overline{z} + \overline{z} = \overline{u} \wedge \overline{u} < \overline{y})$. As $\overline{u}$ is bounded by $\overline{z}$, we do not need to further restrict $\overline{u}$ to be contained in the universe of the structure $\mathcal{I}$ maps to. Hence, we only need to adapt the arities of the tuples in this formula, still obtaining a formula of the form $\overline{z} < \overline{y}/2$. This means that the guards still have the required properties. Let θ' be the formula obtained after these modifications. It is easy to see that P' only occurs positively in θ' as no new negations around occurrences of P' were introduced. Furthermore, none of the added quantifiers have an occurrence of P' in their scope. As we only applied $\mathcal{I}$ to FO- and #Win-FO-formulae above, these modifications preserve the number of winning strategies of the verifier in arbitrary structures and for arbitrary non-uniform families of interpretations.

In order to argue that the FO[Arb]-query may also be partial, simply use the fact that for any $\sigma' \cup \tau'$-formula ψ, the formula $[\mathsf{P}'(\overline{y}) \equiv \theta'(\overline{y}, \mathsf{P}'^{+})]\varphi(\mathsf{P}^{+}) \wedge \psi$ has the desired properties. For this, assume without loss of generality that the number of winning strategies of the verifier in the model-checking game on ψ for arbitrary input structures is at most 1. □

The (non-quantitative) logic FO + GPR can be handled analogously to the logics FO and ESO. Define $\mathrm{FO}+\mathrm{GPR} := (\Phi_{\mathrm{FO+GPR}}, \mathfrak{A}_{\mathrm{FO}}, \models_{\mathrm{FO+GPR}})$, where $\Phi_{\mathrm{FO+GPR}} := \Phi_{\#\text{Win-FO+GPR}}$ and for all $\mathbf{A} \in \mathfrak{A}_{\mathrm{FO}}$ and $\varphi \in \Phi_{\mathrm{FO+GPR}}$, define

$$\mathbf{A} \models_{\mathrm{FO+GPR}} \varphi \iff \#\mathcal{M}_{\#\text{Win-FO+GPR}}(\mathbf{A}, \varphi) \geq 1.$$

Closure under partial FO[Arb]-queries immediately follows from the corresponding closure property of #Win-FO[Arb] + GPR.

7.3 Conclusion and Outlook

In this chapter we explored the direct transfer of model-theoretic characterizations from the uniform to the non-uniform setting, only using the uniform characterization in a black-box manner. As we focus on FO[BIT]-uniform classes in this thesis, we restricted ourselves to this type of uniformity. We identified an approach to make this possible: If one can show that for both the circuit complexity class and the model-theoretic class, the non-uniform version coincides with the uniform version with a polynomial advice function, the characterization can directly be transferred. From this starting point we

then went on to identify a wide range of classes with this property, separately treating circuit complexity classes and logical classes. In both cases, we also covered the case of counting classes.

Regarding circuit complexity classes, we used a layered normal form, which ensures that even when adding all possible edges, i.e., all edges between subsequent layers, the underlying graph does not contain cycles. This made it possible to use advice bits to determine which of the edges are actually present. We showed that this approach works in a very general setting, our results being applicable, in particular, to all classes from the NC-, SAC-, AC-, and TC-hierarchies except for the very restrictive class NC^0. By transferring our proofs to the counting setting, we have also seen that our approach is applicable to the counting versions of all of the above classes.

Regarding classes of languages defined in terms of logics, we obtained a result that could be considered even more general. The only restriction is that the logic should be closed under the application of (partial) FO-queries, which is a natural property for logics based on first-order logic. In a sense, this is not surprising, as non-uniformity given by built-in predicates is very similar to non-uniformity given by an advice function in the first place. In a sense, the advice is simply encoded in different ways, meaning that our logic just needs to be powerful enough to encode and decode the different ways of pairing the input with the advice. We then considered a general form of quantitative logics, allowing us to transfer the above result to the counting setting.

There are several interesting open questions and directions for future research opened by our results, which we will discuss next. It could be interesting to further generalize our results or identify limits to our approach. One caveat of our results in circuit complexity is that they do not immediately apply to U_E^*-uniform NC^1 and $\#NC^1$. This also means that they are not applicable to the characterization of this class by Compton and Laflamme [CL90]. It would be interesting to generalize our results to this setting. It could also be interesting to look at even more restrictive forms of uniformity such as FO[<]-uniformity [BL06].

Furthermore, we only used circuit complexity classes defined by restricting size and depth of circuits. While these are the most common resource bounds in circuit complexity, sometimes other resource bounds are used. As an example, recall that the characterization of #P in terms of circuit complexity uses circuits with polynomial degree. It could be interesting to generalize our results in this direction.

Finally, beyond purely theoretical interest, it would be interesting to see applications of our results. Most directly, it would be interesting to use them to transfer model-theoretic characterizations, or possibly other kinds of characterizations, from the uniform to the non-uniform setting. Additionally, our results for circuit complexity classes could potentially be used to transfer structural properties from the uniform to the non-uniform setting. A simple example would be a property such as monotonicity. When proving such characterizations or structural properties of circuit complexity classes directly, one often starts by showing the non-uniform version and then refines the proof to obtain the uniform version. Thus, applying our results could be especially useful in case of circuit complexity classes that already have characterizations in terms of some other model, e.g. machine-based. In that case, a model-theoretic character-

ization of the corresponding machine-based class could be directly transferred to a characterization of the non-uniform version of the circuit complexity class.

Bibliography

[AAB+99] Eric Allender, Andris Ambainis, David A. Mix Barrington, Samir Datta, and Huong LeThanh. Bounded depth arithmetic circuits: Counting and closure. In *Automata, Languages and Programming, 26th International Colloquium, ICALP'99, Prague, Czech Republic, July 11-15, 1999, Proceedings*, volume 1644 of *Lecture Notes in Computer Science*, pages 149–158. Springer, 1999. (cited on page 31.)

[AAD00] Manindra Agrawal, Eric Allender, and Samir Datta. On TC^0, AC^0, and arithmetic circuits. *J. Comput. Syst. Sci.*, 60(2):395–421, 2000. (cited on pages 10, 29, 30, 53, 78, 83, and 86.)

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009. (cited on page 17.)

[ABL98] Andris Ambainis, David A. Mix Barrington, and Huong LeThanh. On counting AC^0 circuits with negative constants. In *Mathematical Foundations of Computer Science 1998, 23rd International Symposium, MFCS'98, Brno, Czech Republic, August 24-28, 1998, Proceedings*, volume 1450 of *Lecture Notes in Computer Science*, pages 409–417. Springer, 1998. (cited on pages 10, 30, 53, 78, 84, and 86.)

[ÀJ93] Carme Àlvarez and Birgit Jenner. A very hard log-space counting class. *Theor. Comput. Sci.*, 107(1):3–30, 1993. (cited on pages 5 and 20.)

[Ajt83] Miklós Ajtai. Σ^1_1-formulae on finite structures. *Ann. Pure Appl. Log.*, 24(1):1–48, 1983. (cited on page 11.)

[All99] Eric Allender. The permanent requires large uniform threshold circuits. *Chic. J. Theor. Comput. Sci.*, article 7, 1999. (cited on page 24.)

[AMR17] Marcelo Arenas, Martin Muñoz, and Cristian Riveros. Descriptive complexity for counting complexity classes. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017. (cited on page 13.)

[AMR20] Marcelo Arenas, Martin Muñoz, and Cristian Riveros. Descriptive complexity for counting complexity classes. *Log. Methods Comput. Sci.*, 16(1), article 9, 2020. (cited on pages 13, 85, and 154.)

[Bar89] David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comput. Syst. Sci.*, 38(1):150–164, 1989. (cited on pages 9 and 51.)

[BCGR92] Samuel R. Buss, Stephen A. Cook, A. Gupta, and V. Ramachandran. An optimal parallel algorithm for formula evaluation. *SIAM J. Comput.*, 21(4):755–780, 1992. (cited on pages 10 and 31.)

[BI94] David A. Mix Barrington and Neil Immerman. Time, hardware, and uniformity. In *Proceedings of the Ninth Annual Structure in Complexity Theory Conference, Amsterdam, The Netherlands, June 28 - July 1, 1994*, pages 176–185. IEEE Computer Society, 1994. (cited on pages 8, 51, and 54.)

[BIS90] David A. Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within NC^1. *J. Comput. Syst. Sci.*, 41(3):274–306, 1990. (cited on pages 8, 12, 51, 52, 54, 55, and 111.)

[BJ87] George Boolos and Richard C. Jeffrey. *Computability and Logic (2. ed.)*. Cambridge University Press, 1987. (cited on page 4.)

[BL06] Christoph Behle and Klaus-Jörn Lange. FO[<]-uniformity. In *21st Annual IEEE Conference on Computational Complexity (CCC 2006), 16-20 July 2006, Prague, Czech Republic*, pages 183–189. IEEE Computer Society, 2006. (cited on pages 51 and 161.)

[Bor77] Allan Borodin. On relating time and space to size and depth. *SIAM J. Comput.*, 6(4):733–744, 1977. (cited on pages 7, 8, and 24.)

[Bus87] Samuel R. Buss. The Boolean formula value problem is in ALOGTIME. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 123–131. ACM, 1987. (cited on page 55.)

[CG96] Kevin J. Compton and Erich Grädel. Logical definability of counting functions. *J. Comput. Syst. Sci.*, 53(2):283–297, 1996. (cited on page 13.)

[CL90] Kevin J. Compton and Claude Laflamme. An algebra and a logic for NC^1. *Inf. Comput.*, 87(1/2):240–262, 1990. (cited on pages 12, 14, 52, 111, 112, 131, and 161.)

[Cla] Clay Mathematics Institute. Millenium problems. `http://www.claymath.org/millennium-problems`. last checked: 08.05.2021. (cited on page 2.)

[CMTV98] Hervé Caussinus, Pierre McKenzie, Denis Thérien, and Heribert Vollmer. Nondeterministic NC^1 computation. *J. Comput. Syst. Sci.*, 57(2):200–212, 1998. (cited on pages 10, 29, 30, 31, 53, and 56.)

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May*

3-5, 1971, Shaker Heights, Ohio, USA, pages 151–158. ACM, 1971. (cited on pages 2 and 95.)

[Coo73] Stephen A. Cook. A hierarchy for nondeterministic time complexity. *J. Comput. Syst. Sci.*, 7(4):343–353, 1973. (cited on pages 89 and 105.)

[CT90] Ashok K. Chandra and Martin Tompa. The complexity of short two-person games. *Discret. Appl. Math.*, 29(1):21–33, 1990. (cited on page 25.)

[Daw15] Anuj Dawar. The nature and power of fixed-point logic with counting. *ACM SIGLOG News*, 2(1):8–21, 2015. (cited on page 42.)

[DHK05] Arnaud Durand, Miki Hermann, and Phokion G. Kolaitis. Subtractive reductions and complete problems for counting complexity classes. *Theor. Comput. Sci.*, 340(3):496–513, 2005. (cited on pages 4, 95, and 100.)

[DHKV16] Arnaud Durand, Anselm Haak, Juha Kontinen, and Heribert Vollmer. Descriptive complexity of #AC^0 functions. In *25th EACSL Annual Conference on Computer Science Logic, CSL 2016, August 29 - September 1, 2016, Marseille, France*, volume 62 of *LIPIcs*, pages 20:1–20:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. (cited on page 15.)

[DHKV21] Arnaud Durand, Anselm Haak, Juha Kontinen, and Heribert Vollmer. Descriptive complexity of #P functions: A new perspective. *J. Comput. Syst. Sci.*, 116:40–54, 2021. (cited on page 15.)

[DHV18] Arnaud Durand, Anselm Haak, and Heribert Vollmer. Model-theoretic characterization of Boolean and arithmetic circuit classes of small depth. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 354–363. ACM, 2018. (cited on page 15.)

[EFT94] Heinz-Dieter Ebbinghaus, Jörg Flum, and Wolfgang Thomas. *Mathematical Logic (2. ed.)*. Undergraduate texts in mathematics. Springer, 1994. (cited on page 35.)

[Fag74] Ronald Fagin. Generalized first-order spectra and polynomial-time recognizable sets. In *Complexity of computation, SIAM-AMS Proceedings*, volume 7, pages 43–73, 1974. (cited on pages 11 and 49.)

[FSS84] Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Math. Syst. Theory*, 17(1):13–27, 1984. (cited on pages 7 and 11.)

[FVB94] Gudmund Skovbjerg Frandsen, Mark Valence, and David A. Mix Barrington. Some results on uniform arithmetic circuit complexity. *Math. Syst. Theory*, 27(2):105–124, 1994. (cited on pages 12 and 85.)

[Gil77] John Gill. Computational complexity of probabilistic turing machines. *SIAM J. Comput.*, 6(4):675–695, 1977. (cited on page 19.)

[GLS01] Georg Gottlob, Nicola Leone, and Francesco Scarcello. The complexity of acyclic conjunctive queries. *J. ACM*, 48(3):431–498, 2001. (cited on pages 7 and 25.)

[GO04] Etienne Grandjean and Frédéric Olive. Graph properties checkable in linear time in the number of vertices. *J. Comput. Syst. Sci.*, 68(3):546–597, 2004. (cited on pages 14, 89, 104, 105, 108, and 110.)

[Grä92] Erich Grädel. Capturing complexity classes by fragments of second-order logic. *Theor. Comput. Sci.*, 101(1):35–57, 1992. (cited on page 12.)

[Grä13] Erich Grädel. Model-checking games for logics of imperfect information. *Theor. Comput. Sci.*, 493:2–14, 2013. (cited on pages 48 and 63.)

[Gro08] Martin Grohe. The quest for a logic capturing PTIME. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 267–271. IEEE Computer Society, 2008. (cited on page 11.)

[Gro17] Martin Grohe. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*, volume 47 of *Lecture Notes in Logic*. Cambridge University Press, 2017. (cited on page 42.)

[Hes01] William Hesse. Division is in uniform TC^0. In *Automata, Languages and Programming, 28th International Colloquium, ICALP 2001, Crete, Greece, July 8-12, 2001, Proceedings*, volume 2076 of *Lecture Notes in Computer Science*, pages 104–114. Springer, 2001. (cited on pages 7 and 25.)

[Hin82] Jaakko Hintikka. Game-theoretical semantics: insights and prospects. *Notre Dame J. Formal Log.*, 23(2):219–241, 1982. (cited on pages 13 and 45.)

[HKM+19] Anselm Haak, Juha Kontinen, Fabian Müller, Heribert Vollmer, and Fan Yang. Counting of teams in first-order team logics. In *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany*, volume 138 of *LIPIcs*, pages 19:1–19:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. (cited on pages 13 and 110.)

[HMU07] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation, 3rd Edition*. Pearson international edition. Addison-Wesley, 2007. (cited on pages 17 and 24.)

[HV16] Anselm Haak and Heribert Vollmer. A model-theoretic characterization of constant-depth arithmetic circuits. In *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. (cited on pages 13 and 15.)

[HV19] Anselm Haak and Heribert Vollmer. A model-theoretic characterization of constant-depth arithmetic circuits. *Ann. Pure Appl. Log.*, 170(9):1008–1029, 2019. (cited on page 15.)

[Imm82] Neil Immerman. Relational queries computable in polynomial time (extended abstract). In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 147–152. ACM, 1982. (cited on page 11.)

[Imm86] Neil Immerman. Relational queries computable in polynomial time. *Inf. Control.*, 68(1-3):86–104, 1986. (cited on page 11.)

[Imm87] Neil Immerman. Languages that capture complexity classes. *SIAM J. Comput.*, 16(4):760–778, 1987. (cited on page 12.)

[Imm89] Neil Immerman. Expressibility and parallel complexity. *SIAM J. Comput.*, 18(3):625–638, 1989. (cited on pages 12, 13, 51, and 54.)

[Imm99] Neil Immerman. *Descriptive Complexity*. Graduate texts in computer science. Springer, 1999. (cited on pages 40, 42, 49, 50, 51, 75, 77, 111, and 123.)

[JLL76] Neil D. Jones, Y. Edmund Lien, and William T. Laaser. New problems complete for nondeterministic log space. *Math. Syst. Theory*, 10:1–17, 1976. (cited on page 12.)

[JVV86] Mark Jerrum, Leslie G. Valiant, and Vijay V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theor. Comput. Sci.*, 43:169–188, 1986. (cited on page 98.)

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. (cited on page 2.)

[KLM89] Richard M. Karp, Michael Luby, and Neal Madras. Monte-carlo approximation algorithms for enumeration problems. *J. Algorithms*, 10(3):429–448, 1989. (cited on page 109.)

[KLM12] Andreas Krebs, Nutan Limaye, and Meena Mahajan. Counting paths in VPA is complete for $\#\mathrm{NC}^1$. *Algorithmica*, 64(2):279–294, 2012. (cited on page 31.)

[Kon09] Juha Kontinen. A logical characterization of the counting hierarchy. *ACM Trans. Comput. Log.*, 10(1):7:1–7:21, 2009. (cited on page 12.)

[KS14] Neeraj Kayal and Ramprasad Saptharishi. A selection of lower bounds for arithmetic circuits. In *Perspectives in Computational Complexity: The Somenath Biswas Anniversary Volume*, pages 77–116. Birkhäuser, 2014. (cited on page 10.)

[Lad75] Richard E. Ladner. The circuit value problem is log space complete for P. *SIGACT News*, 7(1):18–20, 1975. (cited on page 9.)

[Lev73] Leonid Anatolievich Levin. Universal sequential search problems. *Problems Inform. Transmission*, 9(3):265–266, 1973. (cited on page 2.)

[Lib04] Leonid Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004. (cited on pages 49 and 79.)

[LL78] Paul Lorenzen and Kuno Lorenz. *Dialogische Logik*. Darmstadt: Wissenschaftliche Buchgesellschaft, 1978. (cited on page 45.)

[LMSV01] Clemens Lautemann, Pierre McKenzie, Thomas Schwentick, and Heribert Vollmer. The descriptive complexity approach to LOGCFL. *J. Comput. Syst. Sci.*, 62(4):629–652, 2001. (cited on pages 12, 52, and 111.)

[LS92] Thomas W. Lynch and Earl E. Swartzlander, Jr. A spanning tree carry lookahead adder. *IEEE Trans. Computers*, 41(8):931–939, 1992. (cited on page 7.)

[Lüc20] Martin Lück. *Team Logic: Axioms, Expressiveness, Complexity*. PhD thesis, University of Hanover, Hannover, Germany, 2020. (cited on page 149.)

[Lup58] Oleg B. Lupanov. A method of circuit synthesis. *Izvestia VUZ Radiofizika*, 1:120–140, 1958. (cited on page 5.)

[Lyn82] James F. Lynch. Complexity classes and theories of finite models. *Math. Syst. Theory*, 15(2):127–144, 1982. (cited on page 11.)

[Mah14] Meena Mahajan. Algebraic complexity classes. In *Perspectives in Computational Complexity: The Somenath Biswas Anniversary Volume*, pages 51–75. Birkhäuser, 2014. (cited on page 10.)

[MP08] Guillaume Malod and Natacha Portier. Characterizing Valiant's algebraic complexity classes. *J. Complex.*, 24(1):16–38, 2008. (cited on page 29.)

[MU17] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis*. Cambridge University Press, 2nd edition, 2017. (cited on page 102.)

[MV97] Meena Mahajan and V. Vinay. Determinant: Combinatorics, algorithms, and complexity. *Chic. J. Theor. Comput. Sci.*, 1997, 1997. (cited on pages 5 and 20.)

[MW14] Martin Mundhenk and Felix Weiss. An AC^1-complete model checking problem for intuitionistic logic. *Comput. Complex.*, 23(4):637–669, 2014. (cited on pages 7 and 25.)

[PF79] Nicholas Pippenger and Michael J. Fischer. Relations among complexity measures. *J. ACM*, 26(2):361–381, 1979. (cited on page 9.)

[Pip79] Nicholas Pippenger. On simultaneous resource bounds (preliminary version). In *20th Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 29-31 October 1979*, pages 307–311. IEEE Computer Society, 1979. (cited on page 7.)

[Ros10] Benjamin Rossman. *Average-Case Complexity of Detecting Cliques*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2010. (cited on page 109.)

[RS42] John Riordan and Claude E. Shannon. The number of two-terminal series-parallel networks. *Journal of Mathematics and Physics*, 21(1-4):83–93, 1942. (cited on page 5.)

[Ruz81] Walter L. Ruzzo. On uniform circuit complexity. *J. Comput. Syst. Sci.*, 22(3):365–383, 1981. (cited on pages 8, 9, 24, 54, 55, and 56.)

[Sap15] Ramprasad Saptharishi. A survey of known lower bounds in arithmetic circuits. `https://github.com/dasarpmar/lowerbounds-survey`, 2015. last checked: 05.05.2021. (cited on page 10.)

[Sav76] John E. Savage. *The Complexity of Computing*. John Wiley & Sons, New York, 1976. (cited on page 6.)

[Sha38] Claude E. Shannon. A symbolic analysis of relay and switching circuits. *Transactions of the American Institute of Electrical Engineers*, 57(12):713–723, December 1938. (cited on page 5.)

[Sha49] Claude E. Shannon. The synthesis of two-terminal switching circuits. *Bell Syst. Tech. J.*, 28(1):59–98, 1949. (cited on page 5.)

[Sip83] Michael Sipser. Borel sets and circuit complexity. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 61–69. ACM, 1983. (cited on page 109.)

[Sip97] Michael Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 1997. (cited on page 17.)

[Soa87] Robert I. Soare. *Recursively Enumerable Sets and Degrees - a Study of Computable Functions and Computability Generated Sets*. Perspectives in mathematical logic. Springer, 1987. (cited on page 12.)

[SST95] Sanjeev Saluja, K. Venkata Subrahmanyam, and Madhukar N. Thakur. Descriptive complexity of #P functions. *J. Comput. Syst. Sci.*, 50(3):493–505, 1995. (cited on pages 12, 14, 52, 85, 87, 90, 98, 99, 102, 107, 108, and 109.)

[Sto76] Larry J. Stockmeyer. The polynomial-time hierarchy. *Theor. Comput. Sci.*, 3(1):1–22, 1976. (cited on page 11.)

[SV84] Larry J. Stockmeyer and Uzi Vishkin. Simulation of parallel random access machines by circuits. *SIAM J. Comput.*, 13(2):409–422, 1984. (cited on pages 9 and 24.)

[Tod91a] Seinosuke Toda. Counting problems computationally equivalent to computing the determinant. 1991. (cited on pages 5 and 20.)

[Tod91b] Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991. (cited on page 5.)

[Tod92] Seinosuke Toda. Classes of arithmetic circuits capturing the complexity of computing the determinant. *IEICE Transactions on Information and Systems*, 75:116–124, 1992. (cited on page 10.)

[Val79a] Leslie G. Valiant. Completeness classes in algebra. In *Proceedings of the 11h Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1979, Atlanta, Georgia, USA*, pages 249–261. ACM, 1979. (cited on page 10.)

[Val79b] Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979. (cited on pages 5 and 20.)

[Val79c] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979. (cited on page 95.)

[Val92] Leslie G. Valiant. Why is Boolean complexity theory difficult? In *Proceedings of the London Mathematical Society Symposium on Boolean Function Complexity*, pages 84–94, 1992. (cited on pages 5 and 20.)

[Var82] Moshe Y. Vardi. The complexity of relational query languages (extended abstract). In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 137–146. ACM, 1982. (cited on page 11.)

[Ven91] H. Venkateswaran. Properties that characterize LOGCFL. *J. Comput. Syst. Sci.*, 43(2):380–404, 1991. (cited on pages 9, 24, and 55.)

[Ven92] H. Venkateswaran. Circuit definitions of nondeterministic complexity classes. *SIAM J. Comput.*, 21(4):655–670, 1992. (cited on pages 9, 10, 29, and 32.)

[Vin91] V. Vinay. Counting auxiliary pushdown automata and semi-unbounded arithmetic circuits. In *Proceedings of the Sixth Annual Structure in Complexity Theory Conference, Chicago, Illinois, USA, June 30 - July 3, 1991*, pages 270–284. IEEE Computer Society, 1991. (cited on pages 5, 9, 10, 20, 29, 30, 31, and 53.)

[Vol99] Heribert Vollmer. *Introduction to Circuit Complexity - A Uniform Approach*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 1999. (cited on pages 20 and 24.)

[Woe16] Gerhard J. Woeginger. The P-versus-NP page. `https://www.win.tue.nl/~gwoegi/P-versus-NP.htm`, 2016. last checked: 08.05.2021. (cited on page 2.)

Page left intentionally blank to have matching page numbers with the printed version.

Curriculum Vitae

Name	Anselm Haak
Geburtsdatum	16.04.1990

Ausbildung

1996–2000	Astrid-Lindgren Grundschule Burgdorf.
2000–2002	Orientierungsstufe I Burgdorf.
2002–2006	Gymnasium Burgdorf.
2006–2009	BBS Burgdorf Lehrte, Fachgymnasium Technik.
2009	Allgemeine Hochschulreife, *Note: 1.5*. Erhöhtes Anforderungsniveau: Technik-Informationstechnik, Mathematik, Englisch.
2010–2013	Studium Bachelor Informatik, Nebenfach Mathematik, Leibniz Universität Hannover, *Note: 1.1*, Abschlussthema: Komplexität der Matrizenmultiplikation.
2013–2015	Studium Master Informatik, Nebenfach Mathematik, Leibniz Universität Hannover, *Note: 1.0*, Abschlussthema: Complexity of Parameterized Counting.

Bisherige Tätigkeiten

2009–2010	Zivildienst, Werkstatt für behinderte Menschen Burgdorf.
2011–2015	Studentische Hilfskraft, Leibniz Universität Hannover, an den Instituten für Praktische Informatik, Mikroelektronische Systeme, Mensch-Computer-Interaktion, Systems Engineering und Theoretische Informatik.
10/2015–11/2015	Wissenschaftliche Hilfskraft am Institut für Theoretische Informatik, Leibniz Universität Hannover.
seit 11/2015	Wissenschaftlicher Mitarbeiter am Institut für Theoretische Informatik, Leibniz Universität Hannover.

Page left intentionally blank to have matching page numbers with the printed version.

List of Publications

[1] Arnaud Durand, Anselm Haak, Juha Kontinen, and Heribert Vollmer. Descriptive complexity of #P functions: A new perspective. *J. Comput. Syst. Sci.*, 116:40–54, 2021. (not cited.)

[2] Anselm Haak, Arne Meier, Om Prakash, and B. V. Raghavendra Rao. Parameterised counting in logspace. In *STACS*, volume 187 of *LIPIcs*, pages 40:1–40:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. (not cited.)

[3] Anselm Haak and Heribert Vollmer. A model-theoretic characterization of constant-depth arithmetic circuits. *Ann. Pure Appl. Log.*, 170(9):1008–1029, 2019. (not cited.)

[4] Anselm Haak, Juha Kontinen, Fabian Müller, Heribert Vollmer, and Fan Yang. Counting of teams in first-order team logics. In *MFCS*, volume 138 of *LIPIcs*, pages 19:1–19:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. (not cited.)

[5] Arnaud Durand, Anselm Haak, and Heribert Vollmer. Model-theoretic characterization of boolean and arithmetic circuit classes of small depth. In *LICS*, pages 354–363. ACM, 2018. (not cited.)

[6] Arnaud Durand, Anselm Haak, Juha Kontinen, and Heribert Vollmer. Descriptive complexity of $\#AC^0$ functions. In *CSL*, volume 62 of *LIPIcs*, pages 20:1–20:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. (not cited.)

[7] Anselm Haak and Heribert Vollmer. A model-theoretic characterization of constant-depth arithmetic circuits. In *WoLLIC*, volume 9803 of *Lecture Notes in Computer Science*, pages 234–248. Springer, 2016. (not cited.)